1983

Stability analysis of fixed-point digital filters using a constructive algorithm

Kelvin Todd Erickson
Iowa State University

Erickson, Kelvin Todd

STABILITY ANALYSIS OF FIXED-POINT DIGITAL FILTERS USING A CONSTRUCTIVE ALGORITHM

Iowa State University PH.D. 1983

Stability analysis of fixed-point digital filters using a constructive algorithm

by

Kelvin Todd Erickson

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of the
Requirements for the Degree of
DOCTOR OF PHILOSOPHY

Department: Electrical Engineering
Major: Electrical Engineering (Control Systems)

Approved:

Members of the Committee

Signature was redacted for privacy.

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa

1983

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

Due to recent advances in semiconductor technology, there has been increasing interest in the digital processing of signals. The basic element of almost every digital processing system is a digital filter. These digital filters are often implemented using a microprocessor with fixed-point arithmetic. However, the implemented digital filter will not behave exactly like the desired filter because the microprocessor introduces quantization and overflow nonlinearities due to the fixed-point arithmetic. In the recursive part of the filter, these nonlinearities may cause the filter to exhibit limit cycle oscillations. If limit cycles do exist for a given digital filter, then the output of the filter may not return to zero when the input is zero. Since these limit cycles are undesirable, the digital filter designer needs guidance to determine for a particular application which filters do not exhibit limit cycles. This dissertation addresses this problem by giving the region of allowable parameters that insures the absence of these limit cycles for a given digital filter structure. These parameters are the digital multiplier gains in the filter structure. Since many digital filters are formed by combining second order filters, we will restrict our attention to second order filters. Also, since this dissertation only treats the existance or absence of limit cycles, we are not concerned about their amplitude if they do exist.

In recent papers, Brayton and Tong [1], [2] established some elegant results which are the basis of a constructive approach in the stability analysis of dynamical systems. In this dissertation, we apply this constructive algorithm to the stability analysis of digital filters. Specifically, we find the regions in the parameter plane where a given second order fixed-point digital filter is globally asymptotically stable, using the constructive algorithm of Brayton and Tong. In these regions, the absence of limit cycles is insured.

In Chapter II, we present the necessary background material concerning the constructive stability algorithm. In Chapter III, we show how the constructive algorithm is applied to the stablity analysis of four second order fixed-point digital filters: direct form, coupled form, wave filters and lattice filters. Chapter IV presents the stability results obtained by the constructive algorithm and compares them with existing stability results. In Chapter V, we present our conclusions and suggestions for further work. Descriptions and listings of the computer programs used in our investigation are contained in the appendices.

II. PRELIMINARY MATERIAL

This chapter presents the necessary background material concerning the constructive stability algorithm that we use in the stability analysis of digital filters. In Section A, the notation that will be used throughout this dissertation will be presented. In Section B, aspects of stability analysis of general systems described by difference equations will be discussed. Specifically, only the Lyapunov stability results that are required in subsequent sections will be presented. In Section C, significant results of the constructive stability algorithm due to Brayton and Tong [1], [2] will be presented. In Section D, we introduce the concept of the extreme matrices of a convex set of matrices. This concept is used in the next chapter where the constructive algorithm is applied to fixed-point digital filters.

A. Notation

Let U and V be arbitrary sets. If u is an element of U, we write $u \in U$. If U is a subset of V, we write $U \subset V$. Let $U \times V$ denote the cartesian product of U and V. The boundary of U is denoted by ∂U. If W is a convex polyhedral region, then the elements of the set E(W) denote its extreme vertices and $\kappa[W] = W \cup \partial W$ denotes its convex hull.

Let R denote the real line, let $R^+ = [0,\infty)$ and let R^n denote the set of real-valued n-tuples. The symbol $|\cdot|$ denotes a vector norm on R^n. If f is a function or mapping of a set X into a set Y, we write $f : X \to Y$. Also, $B(r) = \{x \in R^n : |x| < r\}$.

Matrices are usually assumed to be real and we denote them by upper case letters. If $A = [a_{ij}]$ is an arbitrary n x n matrix, then A^T denotes the transpose of A. Also, $\|A\|$ is used to denote the matrix norm of A induced by some vector norm. A set of matrices is denoted by an underlined upper case letter, e.g., $\underline{A}$. The set of extreme matrices of a convex set of matrices $\underline{A}$ is denoted by $E(\underline{A})$.

A continuous function $\phi = R^+ \to R^+$ is said to belong to class K, i.e., $\phi \in K$, if $\phi(0) = 0$ and if ϕ is strictly increasing on R^+. If $\phi = R^+ \to R^+$, if $\phi \in K$, and if $\lim_{r \to \infty} \phi(r) = \infty$, then ϕ is said to belong to class KR.

Let $\tau \in I \triangleq \{t_o + k\}$, $k = 0,1,2, \cdots$. In a digital filter structure block diagram, z^{-1} represents a unit delay.

The function $f(\cdot)$ is said to belong to the sector $[k_1, k_2]$, if i) $f(0) = 0$ and ii) $k_1 < \frac{f(x)}{x} < k_2$, $x \neq 0$, for all $x \in R$. A general sector $[k_1, k_2]$ is represented by the hatched region of Figure 2.1.

Finally, when we write a complex number, let $j = \sqrt{-1}$.

B. Stability of Systems Described by Difference Equations

In the present section, we consider systems described by ordinary autonomous difference equations of the form

$$x(\tau+1) = g[x(\tau)] \qquad (2.1)$$

where $x \in R^n$, $g = R^n \to R^n$ and $\tau \in I$. We denote unique solutions of (2.1) by $x(\tau; x_o, \tau_o)$, where $x_o = x(\tau_o; x_o, \tau_o)$. Since we are dealing

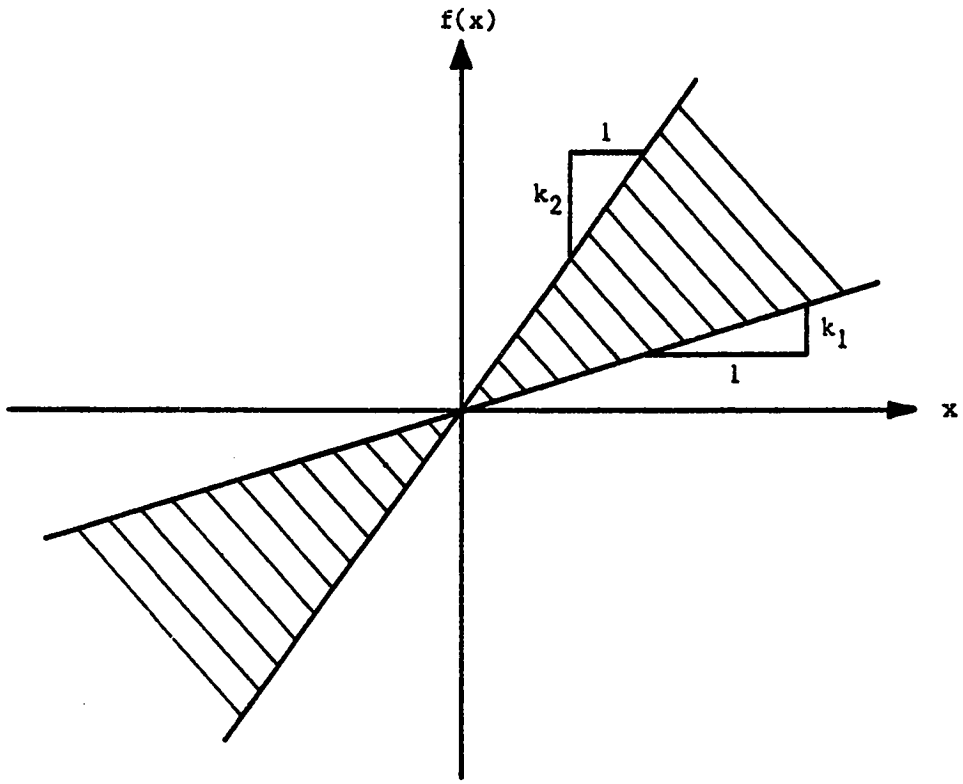


Figure 2.1. A general sector $[k_1, k_2]$

with autonomous equations, we shall assume without loss of generality that $\tau_o = 0$. Any point $x_e \in R^n$ for which it is true that $x_e = g(x_e)$ is called an equilibrium point of (2.1). We will henceforth assume that $x = 0$ is an isolated equilibrium of (2.1), i.e., that there exists a constant $r > 0$ such that $B(r)$ contains no equilibrium points of (2.1) other than the origin. Thus, we have in particular $g(0) = 0$.

We will call any nontrivial periodic solution of (2.1) a limit cycle. It is customary in the study of digital filters to include nonzero equilibrium points as limit cycles. Unless otherwise stated, we will follow this practice.

Since (2.1) is a system of nonlinear equations, it is in general not possible to generate a closed-form solution for (2.1). For this reason, the qualitative analysis of the equilibrium $x = 0$ of (2.1) is of great importance, especially the stability analysis of $x = 0$ in the sense of Lyapunov. The concepts of Lyapunov stability that we shall require are given next.

Definition 2.1 The equilibrium $x = 0$ of (2.1) is said to be stable (in the sense of Lyapunov) if for every $\varepsilon > 0$, there exists a $\delta(\varepsilon) > 0$ such that $|x(\tau; x_o, 0)| < \varepsilon$ for all $\tau \geq 0$ whenever $|x_o| < \delta$.

Definition 2.2 The equilibrium $x = 0$ of (2.1) is said to be asymptotically stable (in the sense of Lyapunov) if (i) it is stable and (ii) there exists a number $\eta > 0$ having the property that $\lim_{\tau \to \infty} x(\tau; x_o, 0) = 0$ whenever $|x_o| < \eta$. If in particular, condition (ii) is true for all $x_o \in R^n$, then the equilibrium $x = 0$ of (2.1) is

said to be globally asymptotically stable (g. a. s.) or asymptotically stable in the large.

Definition 2.3 The equilibrium $x = 0$ of (2.1) is unstable if it is not stable.

The principal Lyapunov results which yield conditions for stability, asymptotic stability or instability in the sense of definitions 2.1, 2.2 and 2.3 involve the existence of functions, $v : R^n \to R$. Such functions have certain definiteness properties which we enumerate next.

Definition 2.4 A function $v : R^n \to R$ is said to be positive definite if there exists a function $\psi \in K$ such that $v(0) = 0$ and $v(x) > \psi(|x|)$ for all $x \in B(r)$ for some $r > 0$.

Definition 2.5 A function $v : R^n \to R$ is said to be negative definite if $-v$ is positive definite.

Definition 2.6 A function $v : R^n \to R$ is said to be radially unbounded if there exists a function $\psi \in KR$ such that $v(0) = 0$ and $v(x) > \psi(|x|)$ for all $x \in R^n$.

The first forward difference of a function $v : R^n \to R$ along the solutions of equation (2.1) is given by

$$\begin{aligned} Dv_{(2.1)}[x(\tau)] &\triangleq v[x(\tau+1)] - v[x(\tau)] \\ &= v[g(x(\tau))] - v[x(\tau)] \end{aligned} \tag{2.2}$$

for all $\tau > 0$. Henceforth, we shall use the notation

$$Dv_{(2.1)}(x) = v(g(x)) - v(x). \tag{2.3}$$

Furthermore, we shall assume that v is continuous and that it satisfies a Lipschitz condition with respect to x.

We are now in a position to state three Lyapunov theorems which will be of interest to us.

Theorem 2.1 The equilibrium $x = 0$ of (2.1) is stable if there exists a function $v : R^n \to R$ such that (i) v is positive definite and (ii) $Dv_{(2.1)}(x) \leq 0$ for all $x \in B(r)$ for some $r > 0$.

Theorem 2.2 The equilibrium $x = 0$ of (2.1) is asymptotically stable if there exists a function $v : R^n \to R$ such that (i) v is positive definite and (ii) $Dv_{(2.1)}(x)$ is negative definite.

Theorem 2.3 The equilibrium $x = 0$ of (2.1) is globally asymptotically stable if there exists a function $v : R^n \to R$ such that (i) v is radially unbounded and (ii) $Dv_{(2.1)}(x)$ is negative definite for all $x \in R^n$.

Note that if it is possible to find a v function for (2.1) which satisfies the conditions of Theorem 2.3, then

1.) System (2.1) has only one equilibrium;

2.) This equilibrium will be $x = 0$;

3.) No limit cycles will exist for system (2.1).

C. Constructive Stability Algorithm

In two papers [1] and [2], Brayton and Tong present an algorithm to construct a Lyapunov function to establish the stability and global asymptotic stability of the equilibrium $x = 0$ of dynamical systems

described by ordinary differential equations and also by difference equations. Their algorithm is applicable to systems having only one equilibrium point. The basic philosophy of their method is to determine the stability of an appropriate set of matrices associated with the system in question. The stability and asymptotic stability of this set of matrices is used to deduce the stability and asymptotic stability of the equilibrium of the given system. (We will make precise definitions of these terms later in this section.) To utilize this constructive stability algorithm, we rewrite the given system equation,

$$x(k+1) = g[x(k)] \tag{2.4}$$

as

$$x(k+1) = M(x(k))x(k) \tag{2.5}$$

where $M(x(k))$ is chosen so that $M(x(k))x(k) = g[x(k)]$. For every $x(k) \in R^n$, $M(x(k))$ will be a real $n \times n$ matrix. If we let $\underline{M}$ denote the set of all matrices, M, obtained by varying $x(k)$ over all allowable values, then we can rewrite (2.5) equivalently as

$$x(k+1) = M_k x(k) \ , \ M_k \in \underline{M} \ . \tag{2.6}$$

In [1] and [2], it is shown that the equilibrium $x = 0$ of (2.4) is stable (globally asymptotically stable) if the set of matrices $\underline{M}$ is stable (asymptotically stable). (The precise definitions of these two terms are given in the next paragraphs.) A summary of the results of

Brayton and Tong is presented next. Refer to [1] and [2] for further details concerning these results. The results which follow are phrased in terms of real matrices. The interested reader is referred to [1] for details concerning the extension of these results to complex matrices.

We call a set $\underline{A}$ of n x n real matrices stable if for every neighborhood of the origin $U \subset R^n$, there exists another neighborhood of the origin $V \subset R^n$ such that for every $M \in \underline{A}'$, we have $MV \subseteq U$. Here, $\underline{A}'$ denotes the multiplicative semigroup generated by $\underline{A}$ and $MV = \{u \in R^n: u = Mv, v \in V\}$.

In [1], it is shown that the following statements (which characterize the properties of a class of stable matrices) are equivalent.

a) $\underline{A}$ is stable.

b) $\underline{A}'$ is bounded.

c) There exists a bounded neighborhood of the origin $W \subset R^n$ such that $MW \subseteq W$ for every $M \in \underline{A}$. Furthermore, W can be chosen to be convex and balanced.

d) There exists a vector norm $|\cdot|_W$ such that $|Mx|_W \leq |x|_W$ for all $M \in \underline{A}$ and for all $x \in R^n$.

Now, let $\alpha \in R$ and let $W \subset R^n$. Let $\alpha W = \{u \in R^n: u = \alpha w, w \in W\}$. Since statements c) and d) above are related by

$$|x|_W = \inf\{\alpha: \alpha > 0, x \in \alpha W\} \qquad (2.7)$$

it follows that $|x|_W$ defines a Lyapunov function for $\underline{A}$, i.e., it defines

a function v with the property

$$v(Mx) \leq v(x), \text{ for all } M \in \underline{A} \text{ and } x \in R^n. \qquad (2.8)$$

Next, we call a set of matrices $\underline{A}$ asymptotically stable if there exists a number $\rho > 1$ such that $\rho\underline{A}$ is stable. (The set $\rho\underline{A}$ is obtained by multiplying every member of $\underline{A}$ by ρ.) In [2], it is shown that the following statements (which characterize the properties of a class of asymptotically stable matrices) are equivalent.

a) $\underline{A}$ is asymptotically stable.

b) There exists a convex, balanced, and polyhedral neighborhood of the origin W and a positive number $\gamma < 1$ such that for each $M \in A$, we have $MW \subseteq \gamma W$. (Here $\gamma W = \{u \in R^n: u = \gamma w, w \in W\}$.)

c) $\underline{A}$ is stable and there exists a positive constant K such that for all $M \in \underline{A}'$, $|\lambda_i(M)| \leq K < 1$, $i=1,\cdots,n$, where $\lambda_i(M)$ denotes the i^{th} eigenvalue of M.

Note that if $\underline{A}$ is stable, then $\gamma\underline{A}$ is asymptotically stable for all positive $\gamma < 1$.

In [1] and [2], a constructive algorithm is given to determine whether a set of m n x n real matrices $\underline{A} = \{M_0, \cdots, M_{m-1}\}$ is stable by starting with an initial polyhedral neighborhood of the origin W_o and by defining a sequence of regions $\{W_{k+1}\}$ by

$$W_{k+1} \triangleq \kappa\left[\bigcup_{j=0}^{\infty} M_k^j W_k\right], \text{ where } k'=(k-1) \bmod m \qquad (2.9)$$

and where $\kappa[\cdot]$ denotes the convex hull of a set. Now, $\underline{A}$ is stable if and only if

$$W^* = \bigcup_{k=0}^{\infty} W_k \tag{2.10}$$

is bounded. Note that W* is also given by

$$W^* = \kappa[\cup MW_o,\ M \in \underline{A}']. \tag{2.11}$$

Since all extreme points z of W_{k+1} are of the form $z = M_i^j u$, where u is an extreme point of W_k, we need only deal with the extreme points of W_k in order to obtain

$$W_{k+1} = \kappa[M_k^j, u:\ u \in E(W_k)] \tag{2.12}$$

where $E(W_k)$ denotes the set of extreme points of W_k. Clearly, the new extreme points $E(W_{k+1})$ are images of $E(W_k)$. If $|\lambda(M_{k'})| < 1$ for $M_{k'} \in \underline{A}$, then there exists an integer $J_{k'}$ such that

$$\kappa\left[\bigcup_{j=0}^{\infty} M_k^j, W_k\right] = \kappa\left[\bigcup_{j=0}^{J_{k'}} M_k^j, W_k\right] \tag{2.13}$$

since W_k is a bounded neighborhood of the origin. Notice that $J_{k'}$ can be recognized since it is the smallest J_k to satisfy

$$M_{k'}\, \kappa\left[\bigcup_{j=0}^{J_k} M_k^j, W_k\right] \subseteq \kappa\left[\bigcup_{j=0}^{J_{k'}} M_k^j, W_k\right]. \tag{2.14}$$

Thus, W_{k+1} will be formed in a finite number of steps, since W_k is expressed as the convex hull of a finite set of points.

In practice, W_o above is usually chosen as simple as possible, i.e., it is chosen as the region defined by

$$E(W_o) = \{w_i \in R^n: x_{ii}=1,\ x_{ij}=0,\ j\neq i,\ i=1,\cdots,n\} \quad (2.15)$$

where $w_i = (x_{i1}, x_{i2}, \cdots, x_{in}) \in R^n$.

Note that W_o determined in this way is symmetric, and of all symmetric polyhedral regions, it possesses a minimal number of extreme points, namely 2n.

We call a set of matrices $\underline{A}$ unstable if $\underline{A}$ is not stable. In [1], the following instability criterion is established: $\underline{A}$ is unstable if there exists a k such that $\partial W_o \cap \partial W_k = \phi$, where ϕ denotes the null set. For additional (and improved) instability criteria, refer to [2].

In [2] it is also shown that if a set $\underline{A}$ of matrices, with $E(\underline{A})$ finite, is asymptotically stable, then the constructive algorithm given above will terminate "stable" in a finite number of steps. Thus, a set $\underline{A}$ can be determined stable in a finite number of steps if $\underline{A}$ is asymptotically stable. We have no way of knowing, by means of the constructive algorithm alone, that $\underline{A}$ is asymptotically stable at the termination of the algorithm. However, we can show that $\underline{A}$ is asymptotically stable by choosing a $\rho > 1$ sufficiently small and then showing that $\rho\underline{A}$ is stable by using the constructive algorithm.

Next, we observe that the set $\underline{M}$ given in Equation (2.6) consists in general of infinitely many matrices. However, the following result, established in [1], reduces the stability analysis of the equilibrium $x = 0$ of (2.6) to a finite set of matrices: let $\underline{A}$ be a set of matrices

in the linear space of n x n matrices and let E($\underline{A}$) be the set of extreme matrices of $\underline{A}$ (the precise definition of an extreme matrix will be given in the next section). Then, $\kappa(\underline{A})$ is stable if and only if E($\underline{A}$) is stable. Thus, if E($\underline{A}$) happens to be finite, then the stability analysis of $\underline{A}$ (and hence, of (2.6)) can be accomplished in a finite number of steps.

D. Extreme Matrices of a Convex Set of Matrices

In this section, we introduce the concepts of a convex set of matrices, an extreme subset and an extreme matrix. We phrase our definitions in terms of a linear vector space of real n x n matrices over R. For general definitions of these concepts, see Dunford and Schwartz [3].

Definition 2.7 Let $(R^{n \times n}, R)$ denote the linear space of real n x n matrices over R. A set $\underline{A} \subset R^{n \times n}$ is convex if $X, Y \in \underline{A}$, $k \in R$, and $0 \leq k \leq 1$, imply $kX + (1-k)Y \in \underline{A}$.

Definition 2.8 Let $A_1, A_2 \in \underline{A}$ and $k \in R$. A non-void subset $\underline{B} \subseteq \underline{A}$ is said to be an extreme subset of $\underline{A}$ if a proper convex combination $kA_1 + (1-k)A_2$, $0 < k < 1$, is in $\underline{B}$ only if $A_1, A_2 \in \underline{B}$. An extreme subset of $\underline{A}$ consisting of just one matrix is called an extreme matrix of $\underline{A}$. The set of extreme matrices of $\underline{A}$ is denoted by E($\underline{A}$).

To apply these definitions to our problem, consider the set of 2 x 2 matrices with one parameter varying,

$$\underline{A}_1 = \left\{ \begin{bmatrix} a & b \\ c & d \end{bmatrix} \right\} \qquad (2.16)$$

where b, c and d are constants and

$$\alpha_1 \leq a \leq \alpha_2 \tag{2.17}$$

where α_1 and α_2 are constants. $\underline{A}_1$ is a convex set of matrices by Definition 2.7. The set of extreme matrices of $\underline{A}_1$ is

$$E(\underline{A}_1) = \{B_{11}, B_{12}\} \tag{2.18}$$

where

$$B_{11} = \begin{bmatrix} \alpha_1 & b \\ c & d \end{bmatrix} \quad B_{12} = \begin{bmatrix} \alpha_2 & b \\ c & d \end{bmatrix} . \tag{2.19}$$

In order to demonstrate that B_1 and B_2 are extreme matrices of $\underline{A}_1$, we examine the convex combination of any two matrices in $\underline{A}_1$. Let A_{11}, $A_{12} \in \underline{A}_1$ be given by

$$A_{11} = \begin{bmatrix} a_1 & b \\ c & d \end{bmatrix} \qquad A_{12} = \begin{bmatrix} a_2 & b \\ c & d \end{bmatrix} \tag{2.20}$$

where

$$\begin{aligned} &\alpha_1 \leq a_1 \leq \alpha_2 \\ &\alpha_1 \leq a_2 \leq \alpha_2 . \end{aligned} \tag{2.21}$$

A convex combination of these two matrices is

$$kA_{11} + (1-k)A_{12} = \begin{bmatrix} ka_1 + (1-k)a_2 & b \\ c & d \end{bmatrix} . \tag{2.22}$$

Clearly, $kA_{11} + (1-k)A_{12} = B_{11}$ for $0 < k < 1$ only if $A_{11} = A_{12} = B_{11}$. Hence, B_{11} is an extreme matrix by Definition 2.8. Similarly, $kA_{11} + (1-k)A_{12} = B_{12}$ for $0 < k < 1$ only if $A_{11} = A_{12} = B_{12}$. Therefore, B_{12} is also an extreme matrix. The matrices B_{11} and B_{12} are the only extreme matrices since they are the only matrices in $\underline{A}_1$ that satisfy Definition 2.8.

Now suppose we have a set of 2 x 2 matrices with two varying parameters,

$$\underline{A}_2 = \left\{ \begin{bmatrix} a & b \\ c & d \end{bmatrix} \right\} \tag{2.23}$$

where b and d are constants and

$$\begin{aligned} \alpha_1 \leq a \leq \alpha_2 \\ \gamma_1 \leq c \leq \gamma_2 \end{aligned} \tag{2.24}$$

where α_1, α_2, γ_1 and γ_2 are constants. $\underline{A}_2$ is a convex set of matrices by Definition 2.7. The set of extreme matrices of $\underline{A}_2$ is

$$E(\underline{A}_2) = \{B_{21}, B_{22}, B_{23}, B_{24}\} \tag{2.25}$$

where

$$\begin{aligned} B_{21} &= \begin{bmatrix} \alpha_1 & b \\ \gamma_1 & d \end{bmatrix} \qquad B_{22} = \begin{bmatrix} \alpha_2 & b \\ \gamma_1 & d \end{bmatrix} \\ B_{23} &= \begin{bmatrix} \alpha_1 & b \\ \gamma_2 & d \end{bmatrix} \qquad B_{24} = \begin{bmatrix} \alpha_2 & b \\ \gamma_2 & d \end{bmatrix} \end{aligned} \tag{2.26}$$

To demonstrate that the extreme matrices of $\underline{A}_2$ are given by Equation (2.26), we examine the convex combination of any two matrices in $\underline{A}_2$. Let A_{21}, $A_{22} \in \underline{A}_2$ be defined by

$$A_{21} = \begin{bmatrix} a_1 & b \\ c_1 & d \end{bmatrix} \qquad A_{22} = \begin{bmatrix} a_2 & b \\ c_2 & d \end{bmatrix} \tag{2.27}$$

where

$$\begin{aligned} \alpha_1 &\leq a_1 \leq \alpha_2 \\ \alpha_1 &\leq a_2 \leq \alpha_2 \\ \gamma_1 &\leq c_1 \leq \gamma_2 \\ \gamma_1 &\leq c_2 \leq \gamma_2 \, . \end{aligned} \tag{2.28}$$

A convex combination of A_{21} and A_{22} is

$$kA_{21} + (1-k)A_{22} = \begin{bmatrix} ka_1 + (1-k)a_2 & b \\ kc_1 + (1-k)c_2 & d \end{bmatrix} \, . \tag{2.29}$$

Clearly, $kA_{21} + (1-k)A_{22} = B_{21}$ for $0 < k < 1$ only if $A_{21} = A_{22} = B_{21}$. A similar argument shows that B_{22}, B_{23} and B_{24} are also extreme matrices of $\underline{A}_2$. The matrices in Equation (2.26) are the only extreme matrices of $\underline{A}_2$ since they are the only matrices in $\underline{A}_2$ that satisfy Definition 2.8.

For a general second order digital filter, we will consider matrices of the form

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \tag{2.30}$$

where the elements of A satisfy the inequalities

$$\begin{aligned} \alpha_1 &\leq a \leq \alpha_2 \\ \beta_1 &\leq b \leq \beta_2 \\ \gamma_1 &\leq c \leq \gamma_2 \\ \delta_1 &\leq d \leq \delta_2 \end{aligned} \tag{2.31}$$

where α_i, β_i, γ_i and δ_i, $i = 1, 2$ are constants. Let $\underline{A}$ be the set of all matrices obtained by varying a, b, c and d over all allowable values. The set of extreme matrices is obtained in a manner similar to the preceding examples and is given by

$$E(\underline{A}) = \left\{ \begin{bmatrix} \alpha_i & \beta_j \\ \gamma_k & \delta_\ell \end{bmatrix}, \quad i, j, k, \ell = 1,2 \right\}. \tag{2.32}$$

III. APPLICATION OF THE CONSTRUCTIVE ALGORITHM TO THE STABILITY ANALYSIS OF DIGITAL FILTERS

In this chapter, we show how to apply the Brayton-Tong constructive algorithm to the stability analysis of digital filters. In Section A, the types of nonlinearities that occur in fixed-point digital filters will be presented. In Section B, we present the procedure used to determine the extreme matrices for a general second order digital filter. In Section C, this procedure is applied to four second order digital filter structures: direct form, coupled form, wave filters, and lattice filters.

In Chapter IV, the stability results obtained by the constructive algorithm for these four filter structures are compared with existing stability results.

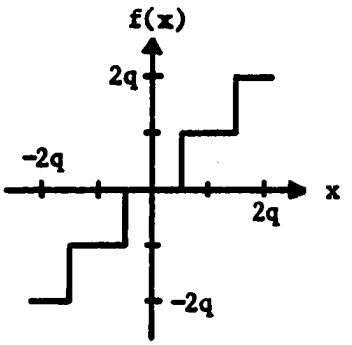
A. Nonlinearities in Digital Filters

In digital filters, the representation of signals must necessarily have finite precision. The finite precision or wordlength is a consequence of the encoding of the signals in a particular format (e.g., fixed- or floating-point) and of the storage of these signals in registers which have finite wordlength. Multiplications and additions performed in the digital filter generally lead to an increase in the wordlength required for the result of the operation. If the number of operations performed on a signal remains finite, as in a nonrecursive filter, the increasing

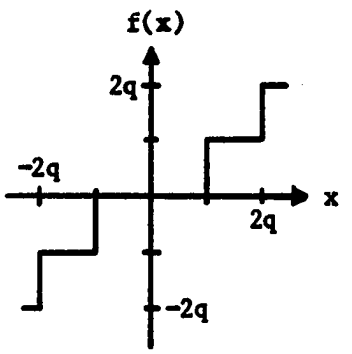
wordlength can be handled by using larger registers for storing the results of the arithmetic operations. However, in a recursive digital filter, a wordlength reduction is necessary to prevent the wordlength of the signals from increasing indefinitely.

In this dissertation, we assume that the digital filters use fixed-point arithmetic. In fixed-point arithmetic, each number is represented by a sign bit and a magnitude. Thus, the magnitude of any number is represented by a string of binary digits of fixed length B. When two B-bit numbers are multiplied, the result is a 2B-bit number. A quantization nonlinearity is produced when the 2B-bit number is reduced in wordlength to B bits. Quantization only affects the least significant bits. Addition also poses a problem when the sum of two numbers falls outside the representable range. An overflow nonlinearity results when this number is modified so that it falls back within the representable range. In general, the overflow nonlinearity changes the most significant bits as well as the least significant bits of a fixed-point number. These two types of nonlinearities are well described in the literature (see e.g., [4], [5]) and therefore, will only be briefly discussed here.

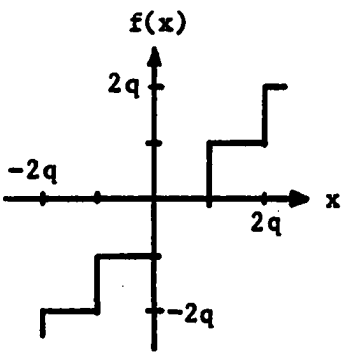
Quantization can be performed by substituting the nearest possible number that can be represented by the limited number of bits. This type of nonlinear operation is called a roundoff quantizer and its characteristic is shown in Figure 3.1(a). Another possibility consists of discarding the least significant bits in the



(a) Roundoff



(b) Magnitude truncation



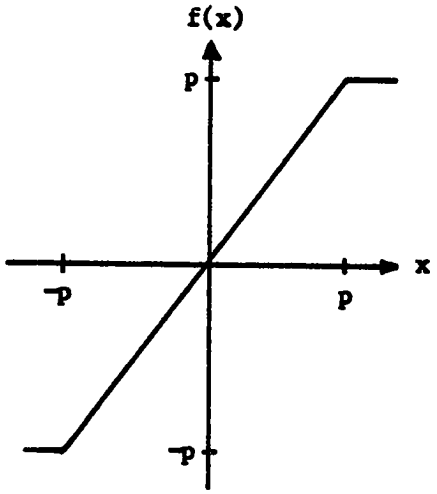
(c) Value truncation

Figure 3.1. Fixed-point quantization characteristics

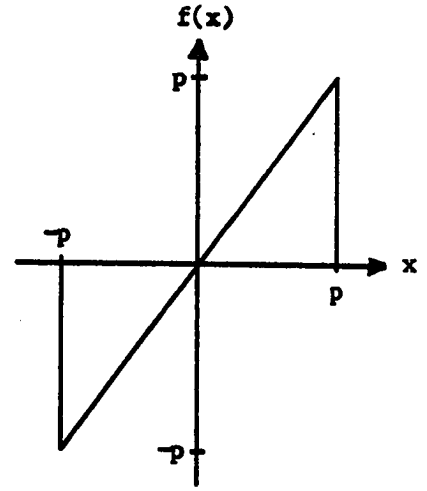
number. If the signals are represented by sign and magnitude then we have a magnitude truncation quantization characteristic as depicted in Figure 3.1(b). If the signals are represented in a two's complement format, the nonlinearity is a two's complement or value truncation quantization as shown in Figure 3.1(c). In this dissertation, value truncation is not considered. Thus, the term truncation will always refer to magnitude truncation in this dissertation.

If an overflow occurs, a number of different actions may be taken. If the number that caused the overflow is replaced by a number having the same sign, but with a magnitude corresponding to the overflow level, a saturation overflow characteristic shown in Figure 3.2(a) is obtained. Zeroing overflow substitutes the number zero in case of an overflow (see Figure 3.2(b)). In two's complement arithmetic, the most significant bits that caused the overflow are discarded. In this case, overflows in intermediate results do not cause errors, as long as the final result does not have overflow. This two's complement overflow characteristic is illustrated in Figure 3.2(c). Another way of dealing with overflow is the triangular overflow characteristic (see Figure 3.2(d)) as proposed by Eckhardt and Winkelnkemper [6].

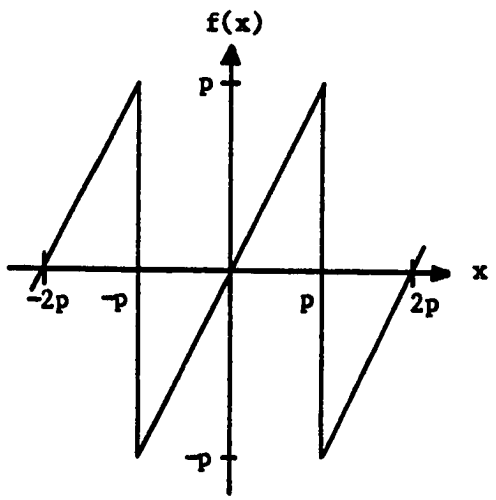
It is possible to have different wordlengths for the various signals in the filter, resulting in different quantization stepsizes and/or different overflow levels. We will assume throughout this



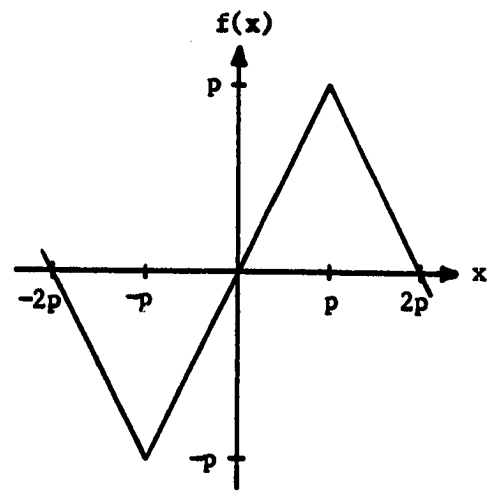
(a) Saturation



(b) Zeroing



(c) Two's complement



(d) Triangular

Figure 3.2. Overflow characteristics

dissertation that all quantizers in a filter have the same quantization stepsize, q, and are the same type, e.g., roundoff or truncation. Similarly, we will assume that all overflow nonlinearities in a filter have the same overflow level, p, and are the same type.

The above nonlinearities will be viewed in this dissertation as belonging to a sector $[k_m, k_M]$, where

$$k_m \leq \frac{f(\sigma)}{\sigma} \leq k_M \text{ for all } \sigma \in R. \tag{3.1}$$

The function $f(\cdot)$ represents the nonlinearity and the only restrictions on k_m and k_M are $-\infty < k_m \leq k_M < \infty$.

Under the above assumptions, we view the quantization nonlinearities as belonging to the sector $[0, k_q]$ where

$$k_q = \begin{cases} 1 & \text{truncation} \\ 2 & \text{roundoff} . \end{cases} \tag{3.2}$$

Henceforth, k_q will represent the upper slope of the sector that contains the quantization nonlinearity. The overflow nonlinearities are represented as belonging to the sector $[k_o, 1]$ where

$$k_o = \begin{cases} 0 & \text{saturation or zeroing} \\ -\frac{1}{3} & \text{triangular} \\ -1 & \text{two's complement} . \end{cases} \tag{3.3}$$

Henceforth, k_o will represent the lower slope of the sector that contains the overflow nonlinearity.

When these two nonlinear operations are combined, i.e., quantization and overflow functions are executed simultaneously, then

the nonlinear operation is represented as belonging to the sector $[k_o, k_q]$. The constant k_o is determined by the type of overflow being performed and the constant k_q is determined by the type of quantization operation.

Our representation of a fixed-point digital filter is not an exact description of an actual realization of a digital filter. Due to the finite number of values that a signal in a digital filter can attain, actual realizations of digital filters are finite state machines. The digital filters which we analyze are still idealizations in the sense that they are not finite state machines. This difficulty is not a serious problem since we assume that a filter operates in its designated range.

B. General Digital Filter

To apply the constructive stability method, we represent a digital filter as a system of difference equations,

$$x(k+1) = g[x(k)] \tag{3.4}$$

where $k = 0, 1, 2, \cdots$. Following the procedure outlined in the previous chapter, we rewrite the given system equations as

$$x(k+1) = M(x(k))x(k) \tag{3.5}$$

where $M(x(k))$ is chosen so that $M(x(k))x(k) = g[x(k)]$. Since we consider only second order systems in this dissertation, the matrix M may be rewritten as

$$M(x(k)) = \begin{bmatrix} a(x(k)) & b(x(k)) \\ c(x(k)) & d(x(k)) \end{bmatrix}. \tag{3.6}$$

We assume that the elements of M satisfy the inequalities

$$\begin{aligned} \alpha_1 &\leq a(x(k)) \leq \alpha_2 \\ \beta_1 &\leq b(x(k)) \leq \beta_2 \\ \gamma_1 &\leq c(x(k)) \leq \gamma_2 \\ \delta_1 &\leq d(x(k)) \leq \delta_2 \end{aligned} \tag{3.7}$$

where α_i, β_i, γ_i and δ_i, $i = 1, 2$ are constants.

Let $\underline{M}$ be the set of all matrices obtained by varying x(k) in M(x(k)) over all allowable values. The extreme matrices of set $\underline{M}$ are obtained as

$$E(\underline{M}) = \left\{ \begin{bmatrix} \alpha_i & \beta_j \\ \gamma_k & \delta_\ell \end{bmatrix}, \; i,j,k,\ell = 1,2 \right\}. \tag{3.8}$$

By the results of the previous chapter, the set $\underline{M}$ is stable (asymptotically stable) if and only if $E(\underline{M})$ is stable (asymptotically stable). So we need only determine the stability (asymptotic stability) properties of $E(\underline{M})$ to determine the stability (global asymptotic stability) of the digital filter described by (3.4). If the set $\underline{M}$ is unstable then we can draw no conclusion about the stability of the digital filter described by (3.4).

Using the results of the previous chapter, we show that $\underline{M}$ is asymptotically stable by choosing a $\rho > 1$ sufficiently small and then showing that ρM is stable by using the constructive algorithm. For the digital filters considered in this dissertation, the choice of $\rho = 1.0000001$ will be used to show the asymptotic stability of the set of extreme matrices in all of the cases we will consider.

Since the constructive algorithm is used to show that the equilibrium $x = 0$ of a given digital filter (3.4) is globally asymptotically stable, then in particular, no limit cycles will exist in this digital filter.

The appendices contain descriptions and listings of the computer programs that implement the Brayton-Tong constructive algorithm when applied to the stability analysis of digital filters.

C. Specific Digital Filters

In this section, we give the details of the application of the Brayton-Tong constructive algorithm to the stability analysis of the following four digital filter structures:

a) Direct form digital filter

b) Coupled form digital filter

c) Wave digital filter

d) Lattice digital filter.

For each of the digital filter structures considered, the region, in terms of the filter parameters, where the linear filter

(i.e., the filter without quantization or overflow) is globally asymptotically stable is given. Since the linear filter is unstable outside of this region, we are not interested in the nonlinear filter whose parameters fall outside of this region. Next we present the particular nonlinear structures for each type of filter that we consider. Finally, for each nonlinear filter structure we derive the set of extreme matrices used by the constructive algorithm.

1. Direct form digital filter

The second order direct form digital filter, which directly implements a filter transfer function, has been investigated extensively [7]. Since we only consider filters with zero input, the recursive parts of the direct form 1 structure and the direct form 2 structure are equivalent. The linear recursive part of this digital filter is shown in Figure 3.3.

The region where this linear filter is globally asymptotically stable in terms of the parameters a and b is derived by considering the transfer function of the linear filter,

$$H(z) = \frac{z^2}{z^2 - az - b} . \qquad (3.9)$$

Using Jury's criterion [8], it can be shown that the ideal second order digital filter is globally asymptotically stable if and only if

$$\begin{aligned} |b| &< 1 \\ |a| + b &< 1 . \end{aligned} \qquad (3.10)$$

This stability region corresponds to the triangular region shown in Figure 3.4. The linear filter is globally asymptotically stable for all coefficients inside this region.

When the linear second order direct form digital filter is implemented in fixed-point arithmetic, there are two possible ways of placing the quantization nonlinearity. Quantization can be performed immediately after each multiplication. This nonlinear second order digital filter structure is shown in Figure 3.5, with Q representing a quantizer. Alternatively, the results of the two multiplications may be added with full precision and only one quantization is needed. This structure is shown in Figure 3.6. For both possible quantizer configurations, the overflow nonlinearity, P, is placed after the adder as shown. We next develop the set of extreme matrices for each structure.

a. One quantizer The structure for the second order direct form digital filter with one quantizer is shown in Figure 3.6. We will consider the quantization and overflow nonlinearities together. With this assumption, the state equations are

$$\begin{aligned} x_1(k+1) &= f[ax_1(k) + bx_2(k)] \\ x_2(k+1) &= x_1(k) \end{aligned} \tag{3.11}$$

where $f(\cdot)$ is the combined quantization and overflow nonlinearity.

Following the technique outlined in Section B of this chapter, the state equations are written as

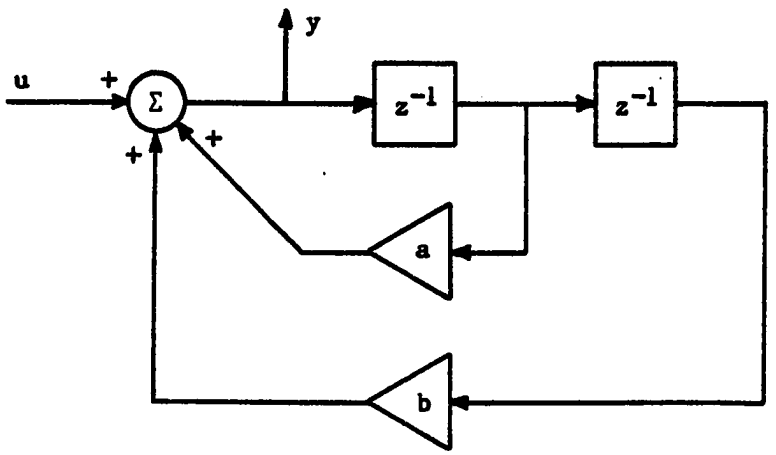


Figure 3.3. Linear second order direct form digital filter

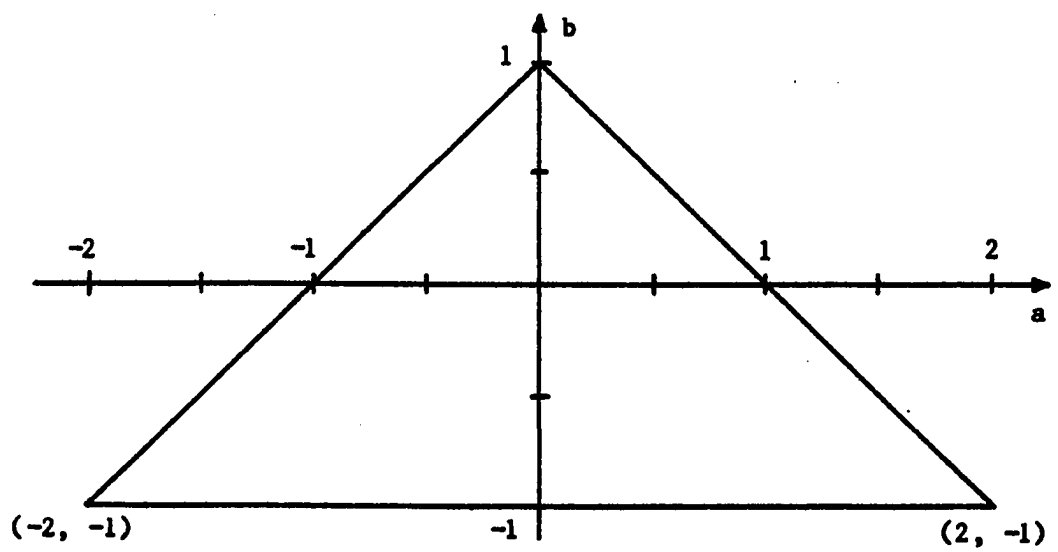


Figure 3.4. Region in the parameter plane where a linear second order direct form filter is globally asymptotically stable

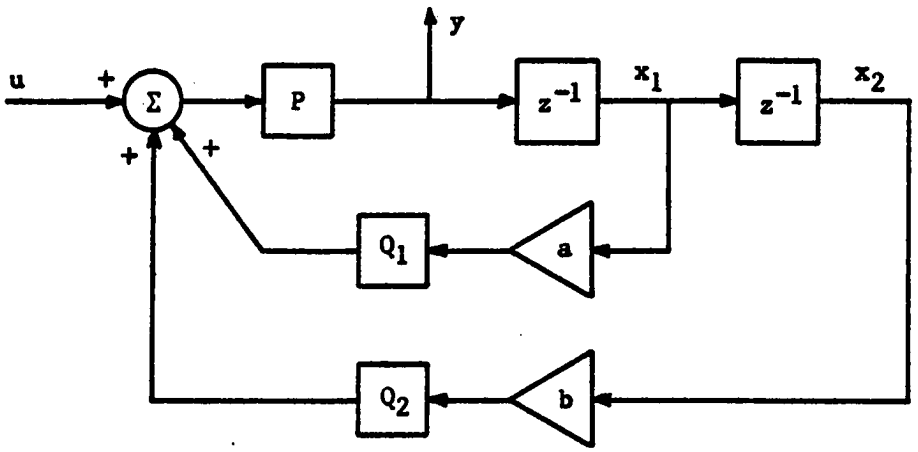


Figure 3.5. Direct form digital filter with two quantizers

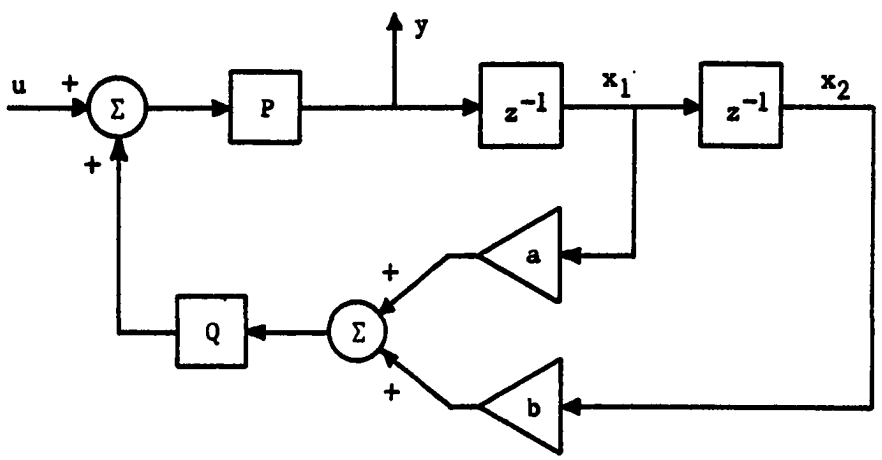


Figure 3.6. Direct form digital filter with one quantizer

$$x(k+1) = M(x(k))x(k).$$

The matrix M(x(k)) is given by

$$M(x(k)) = \begin{bmatrix} \Phi(x)a & \Phi(x)b \\ 1 & 0 \end{bmatrix}, \tag{3.12}$$

where

$$\Phi(x) = \frac{f[ax_1 + bx_2]}{ax_1 + bx_2}. \tag{3.13}$$

Since we view the quantization and overflow nonlinearities as belonging to a sector, the function $\Phi(x)$ is bounded by constants, α_1 and α_2 such that

$$\alpha_1 \leq \Phi(x) \leq \alpha_2 . \tag{3.14}$$

For the particular nonlinearities which we consider, we have

$$\begin{aligned} \alpha_1 &= k_o \\ \alpha_2 &= k_q \end{aligned} \tag{3.15}$$

where k_o and k_q are defined by Equations (3.2) and (3.3), respectively. The extreme matrices of the set $\underline{M}$ are

$$E(\underline{M}) = \left\{ \begin{bmatrix} \alpha_i a & \alpha_i b \\ 1 & 0 \end{bmatrix}, i = 1,2 \right\}. \tag{3.16}$$

In this case, for each point (a,b) there are two extreme matrices given by A_1 and A_2, where

$$A_1 = \begin{bmatrix} k_o a & k_o b \\ 1 & 0 \end{bmatrix} \qquad A_2 = \begin{bmatrix} k_q a & k_q b \\ 1 & 0 \end{bmatrix} . \qquad (3.17)$$

If the overflow nonlinearity is absent, then $\alpha_1 = 0$ and the set of extreme matrices in this case is the same as for one saturation or zeroing overflow nonlinearity.

b. Two quantizers The structure of the second order direct form digital filter with two quantizers is shown in Figure 3.5. We cannot combine the quantization and overflow nonlinearities in this case. The state equations are

$$\begin{aligned} x_1(k+1) &= P\{Q_1[ax_1(k)] + Q_2[bx_2(k)]\} \\ x_2(k+1) &= x_1(k) . \end{aligned} \qquad (3.18)$$

Following the technique outlined in Section B of this chapter, the state equations are written as

$$x(k+1) = M(x(k))x(k)$$

where

$$M(x(k)) = \begin{bmatrix} \Phi_1(x)\Phi_3(x)a & \Phi_2(x)\Phi_3(x)b \\ 1 & 0 \end{bmatrix} \qquad (3.19)$$

and

$$\begin{aligned} \Phi_1(x) &= \frac{Q_1[ax_1]}{ax_1} \qquad \Phi_2(x) = \frac{Q_2[bx_2]}{bx_2} \\ \Phi_3(x) &= \frac{P\{Q_1[ax_1] + Q_2[bx_2]\}}{Q_1[ax_1] + Q_2[bx_2]} . \end{aligned} \qquad (3.20)$$

When the M(x(k)) given by (3.19) and (3.20) is multiplied by x(k), the state equations are obtained.

Since we view the quantization and overflow nonlinearities as belonging to a sector, the functions $\Phi_1(x)$, $\Phi_2(x)$ and $\Phi_3(x)$ are bounded by constants such that

$$\alpha_{i1} \leq \Phi_i(x) \leq \alpha_{i2}\ , \qquad i = 1,2,3 \tag{3.21}$$

where

$$\begin{aligned} \alpha_{11} &= \alpha_{21} = 0 \\ \alpha_{12} &= \alpha_{22} = k_q \\ \alpha_{31} &= k_o,\ \alpha_{32} = 1. \end{aligned} \tag{3.22}$$

The functions $\Phi_1(x)\Phi_3(x)$ and $\Phi_2(x)\Phi_3(x)$ are also bounded by constants, β_i and γ_i, $i = 1,2$ such that

$$\begin{aligned} \beta_1 &\leq \Phi_1(x)\Phi_3(x) \leq \beta_2 \\ \gamma_1 &\leq \Phi_2(x)\Phi_3(x) \leq \gamma_2 \end{aligned} \tag{3.23}$$

where

$$\begin{aligned} \beta_1 &= \min(\alpha_{11}\alpha_{31},\ \alpha_{11}\alpha_{32},\ \alpha_{12}\alpha_{31},\ \alpha_{12}\alpha_{32}) = k_q k_o \\ \beta_2 &= \max(\alpha_{11}\alpha_{31},\ \alpha_{11}\alpha_{32},\ \alpha_{12}\alpha_{31},\ \alpha_{12}\alpha_{32}) = k_q \\ \gamma_1 &= \min(\alpha_{21}\alpha_{31},\ \alpha_{21}\alpha_{32},\ \alpha_{22}\alpha_{31},\ \alpha_{22}\alpha_{32}) = k_q k_o \\ \gamma_2 &= \max(\alpha_{21}\alpha_{31},\ \alpha_{21}\alpha_{32},\ \alpha_{22}\alpha_{31},\ \alpha_{22}\alpha_{32}) = k_q\ . \end{aligned} \tag{3.24}$$

The extreme matrices of the set $\underline{M}$ are

$$E(\underline{M}) = \left\{ \begin{bmatrix} \beta_i a & \gamma_j b \\ 1 & 0 \end{bmatrix}, \quad i,j = 1,2 \right\}. \tag{3.25}$$

For each point (a,b) in the parameter plane, there are four extreme matrices used in the Brayton-Tong constructive algorithm. For this example, these extreme matrices are

$$A_1 = \begin{bmatrix} k_q k_o a & k_q k_o b \\ 1 & 0 \end{bmatrix} \quad A_2 = \begin{bmatrix} k_q k_o a & k_q b \\ 1 & 0 \end{bmatrix}$$
$$A_3 = \begin{bmatrix} k_q a & k_q k_o b \\ 1 & 0 \end{bmatrix} \quad A_4 = \begin{bmatrix} k_q a & k_q b \\ 1 & 0 \end{bmatrix}. \tag{3.26}$$

If the overflow nonlinearity is absent, then $\alpha_{31} = \beta_1 = \gamma_1 = 0$ and the set of extreme matrices in this case is the same as for the filter with a saturation or zeroing overflow nonlinearity.

2. Coupled form digital filter

The coupled or normal form digital filter was first proposed by Rader and Gold [9] as a digital filter structure whose pole locations were less sensitive than the direct form structure to parameter errors. However, the coupled form can only realize complex conjugate poles. With finite wordlength parameters this structure also has a uniform grid of possible pole locations [5]. The linear recursive part of a coupled form digital filter whose poles are at a ± jb and that has zero input is shown in Figure 3.7.

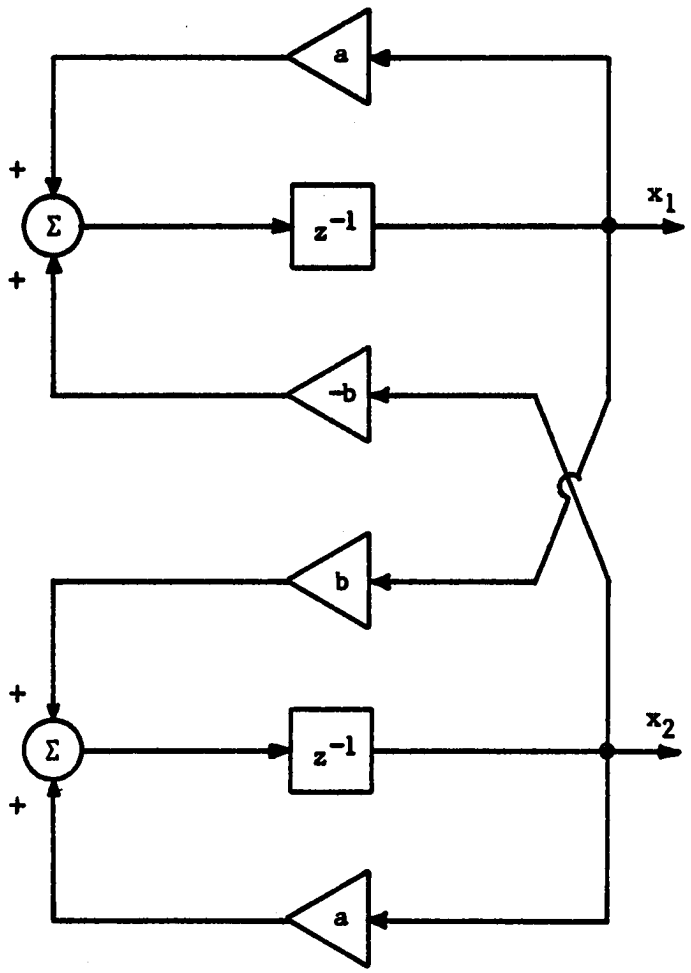


Figure 3.7. Linear second order coupled form digital filter

The linear filter is globally asymptotically stable if and only if its poles lie within the unit circle. Equivalently, the parameters a and b must satisfy

$$a^2 + b^2 < 1. \tag{3.27}$$

This region corresponds to the interior of a unit circle in the a-b parameter plane.

As in the direct form digital filter, there are two possible ways of placing the quantization nonlinearity. Quantization can be performed immediately after each multiplication and thus four quantizers will be needed. This filter structure is shown in Figure 3.8 with Q_i, i=1,•••,4 representing the quantizers. Alternatively, the results of two multiplications may be added with full precision and then quantized. This implementation uses two quantizers and is shown in Figure 3.9. For both possible placements of the quantization nonlinearity, the overflow nonlinearities, P_1 and P_2, must be placed after each addition as shown. We next develop the set of extreme matrices for each structure that will be used by the constructive algorithm.

a. Two quantizers The coupled form digital filter structure to be analyzed is shown in Figure 3.9. As in the direct form digital filter, we assume that the overflow and quantization nonlinearities before each delay are combined. With this assumption, the state equations for the filter are

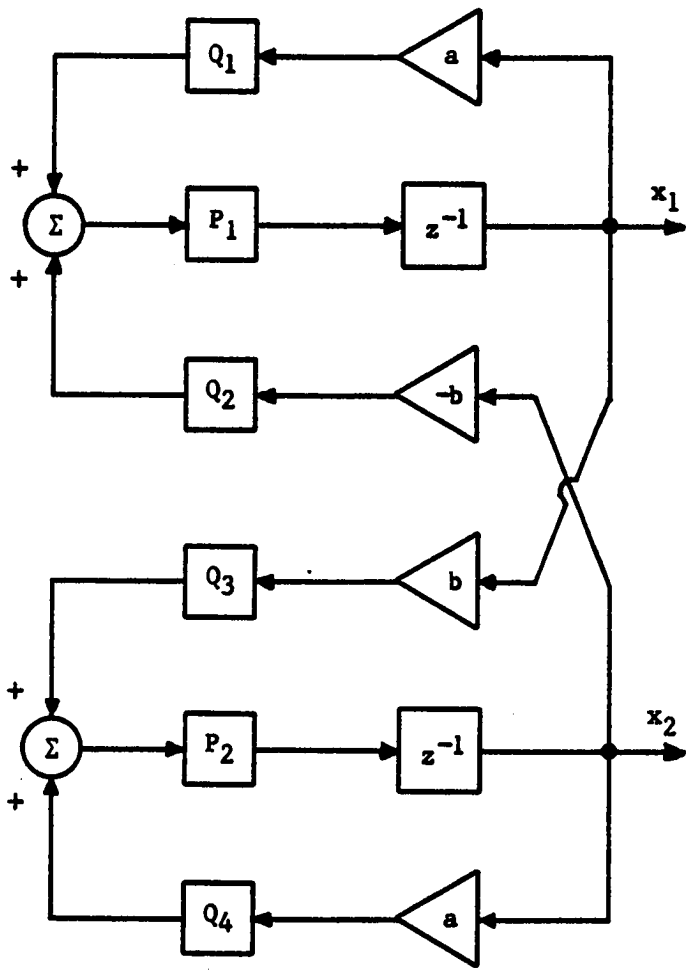


Figure 3.8. Coupled form digital filter with four quantizers

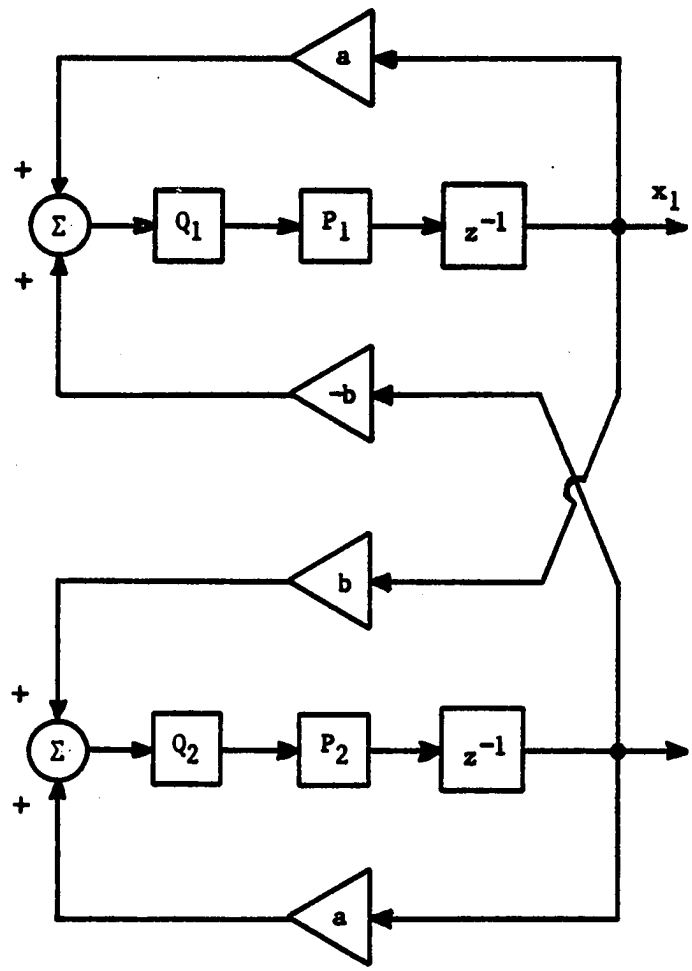


Figure 3.9. Coupled form digital filter with two quantizers

$$x_1(k+1) = f_1[ax_1(k) - bx_2(k)]$$
$$x_2(k+1) = f_2[bx_1(k) + ax_2(k)] \qquad (3.28)$$

where $f_1(\cdot)$ and $f_2(\cdot)$ are the combined quantization and overflow nonlinearities.

Following the technique outlined in Section B of this chapter, the state equations are written as

$$x(k+1) = M(x(k))x(k).$$

By defining

$$\Phi_1(x) = \frac{f_1[ax_1 - bx_2]}{ax_1 - bx_2}$$
$$\Phi_2(x) = \frac{f_2[bx_1 + ax_2]}{bx_1 + ax_2} \qquad (3.29)$$

the matrix M(x(k)) is given by

$$M(x(k)) = \begin{bmatrix} \Phi_1(x)a & -\Phi_1(x)b \\ \Phi_2(x)b & \Phi_2(x)a \end{bmatrix}. \qquad (3.30)$$

The functions $\Phi_1(x)$ and $\Phi_2(x)$ are bounded by constants

$$\alpha_1 \leq \Phi_1(x) \leq \alpha_2$$
$$\beta_1 \leq \Phi_2(x) \leq \beta_2 . \qquad (3.31)$$

For the particular nonlinearities we consider, these constants are

$$\alpha_1 = \beta_1 = k_o$$
$$\alpha_2 = \beta_2 = k_q . \qquad (3.32)$$

The extreme matrices of the set $\underline{M}$ are

$$E(\underline{M}) = \left\{ \begin{bmatrix} \alpha_i a & -\alpha_i b \\ \beta_j b & \beta_j a \end{bmatrix}, \; i,j=1,2 \right\}. \tag{3.33}$$

Therefore, for each point in the a-b parameter plane, the constructive algorithm uses four extreme matrices. If the overflow nonlinearities are absent, then $\alpha_i = \beta_1 = 0$ and the set of extreme matrices in this case is the same as for the filter with two saturation or zeroing overflow nonlinearities.

b. Four quantizers The structure of the coupled form digital filter with four quantizers is shown in Figure 3.8. The state equations are

$$\begin{aligned} x_1(k+1) &= P_1\{Q_1[ax_1(k)] + Q_2[-bx_2(k)]\} \\ x_2(k+1) &= P_2\{Q_3[bx_1(k)] + Q_4[ax_2(k)]\}. \end{aligned} \tag{3.34}$$

To apply the constructive stability algorithm, we write the state equations as

$$x(k+1) = M(x(k))x(k)$$

where

$$M(x(k)) = \begin{bmatrix} \Phi_1(x)\Phi_3(x)a & -\Phi_1(x)\Phi_4(x)b \\ \Phi_2(x)\Phi_5(x)b & \Phi_2(x)\Phi_6(x)a \end{bmatrix} \tag{3.35}$$

and

$$\Phi_1(x) = \frac{P_1\{Q_1[ax_1] + Q_2[-bx_2]\}}{Q_1[ax_1] + Q_2[-bx_2]}$$

$$\Phi_2(x) = \frac{P_2\{Q_3[bx_1] + Q_4[ax_2]\}}{Q_3[bx_1] + Q_4[ax_2]} \tag{3.36}$$

$$\Phi_3(x) = \frac{Q_1[ax_1]}{ax_1} \qquad \Phi_4(x) = \frac{Q_2[-bx_2]}{-bx_2}$$

$$\Phi_5(x) = \frac{Q_3[bx_1]}{bx_1} \qquad \Phi_6(x) = \frac{Q_4[ax_2]}{ax_2} .$$

The functions $\Phi_i(x)$, are bounded by constants α_{ij} such that

$$\alpha_{i1} \leq \Phi_i(x) \leq \alpha_{i2} , \quad i = 1,2,3,4,5 \tag{3.37}$$

where

$$\begin{aligned} \alpha_{11} &= \alpha_{21} = k_o \\ \alpha_{12} &= \alpha_{22} = 1 \\ \alpha_{31} &= \alpha_{41} = \alpha_{51} = \alpha_{61} = 0 \\ \alpha_{32} &= \alpha_{42} = \alpha_{52} = \alpha_{62} = k_q . \end{aligned} \tag{3.38}$$

Therefore, the functions $\Phi_1(x)\Phi_3(x)$, $\Phi_1(x)\Phi_4(x)$, $\Phi_2(x)\Phi_5(x)$ and $\Phi_2(x)\Phi_6(x)$ are bounded by constants β_i, γ_i, δ_i, ε_i, $i = 1,2$ such that

$$\begin{aligned} \beta_1 &\leq \Phi_1(x)\Phi_3(x) \leq \beta_2 \\ \gamma_1 &\leq \Phi_1(x)\Phi_4(x) \leq \gamma_2 \\ \delta_1 &\leq \Phi_2(x)\Phi_5(x) \leq \delta_2 \\ \varepsilon_1 &\leq \Phi_2(x)\Phi_6(x) \leq \varepsilon_2 \end{aligned} \tag{3.39}$$

where

$$\beta_1 = \gamma_1 = \delta_1 = \varepsilon_1 = k_q k_o$$
$$\beta_2 = \gamma_2 = \delta_2 = \varepsilon_2 = k_q. \qquad (3.40)$$

The extreme matrices of the set $\underline{M}$ are

$$E(\underline{M}) = \left\{ \begin{bmatrix} \beta_i a & -\gamma_j b \\ \delta_k b & \varepsilon_\ell a \end{bmatrix}, \; i,j,k,\ell = 1,2 \right\}. \qquad (3.41)$$

For this filter, there are sixteen extreme matrices for every point in the a-b parameter plane. If the overflow nonlinearities are absent, then $\beta_1 = \gamma_1 = \delta_1 = \varepsilon_1 = 0$ and the set of extreme matrices in this case is the same as for the filter with two saturation or zeroing overflow nonlinearities.

3. Wave digital filter

Wave digital filters are a class of low-sensitivity digital filter structures first advanced by Fettweis [10]. These structures can be synthesized from equally terminated LC analog filters by replacing the analog elements by appropriate digital realizations. Wave digital filters are either full-synchronic or half-synchronic. In a full-synchronic filter, the arithmetic operations are carried out, at least in principle, simultaneously at periodically recurring instants. In a half-synchronic filter, the various arithmetic operations are still carried out at the same rate, but do not take

place simultaneously, even in principle. We only consider full-synchronic wave digital filters, since most conventional digital filters are full-synchronic. A general wave digital filter is characterized by an n-port network as illustrated in Figure 3.10. Since wave digital filters are a class of filters, we only consider a specific example of a wave filter synthesized from an LC network in the next subsection.

a. Specific wave digital filter considered The wave digital filter structure which we will examine is based on a general second order lowpass LC filter shown in Figure 3.11. This section can represent many types of filters, e.g., Butterworth or Chebychev.

Following the synthesis procedure of Antoniou [11], we identify the series and parallel interconnection as shown in Figure 3.12(a). The wave digital filter is then formed with one parallel wire interconnection and one series wire interconnection, as in Figure 3.12(b). The resulting structure, in terms of delays, adders and multipliers is shown in Figure 3.13. The state equations for the linear wave digital filter with zero input ($a_1 = a_2 = 0$) are

$$\begin{aligned} x_1(k+1) &= c_{11}x_1(k) + c_{12}x_2(k) \\ x_2(k+1) &= c_{21}x_1(k) + c_{22}x_2(k) \end{aligned} \qquad (3.42)$$

where

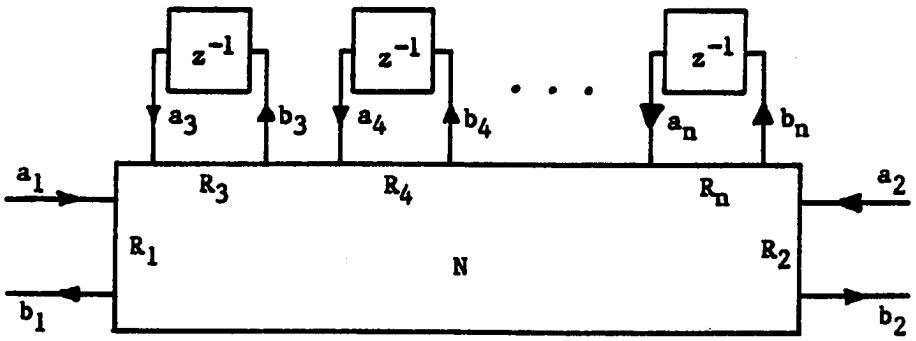


Figure 3.10. General full-synchronic wave digital filter

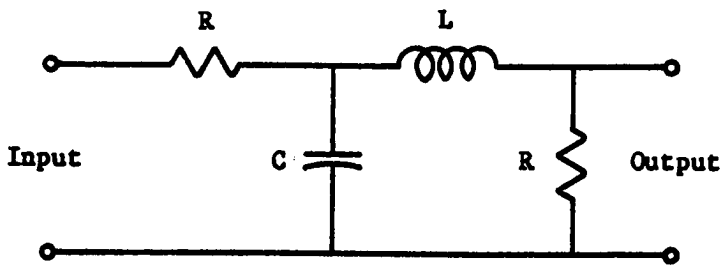
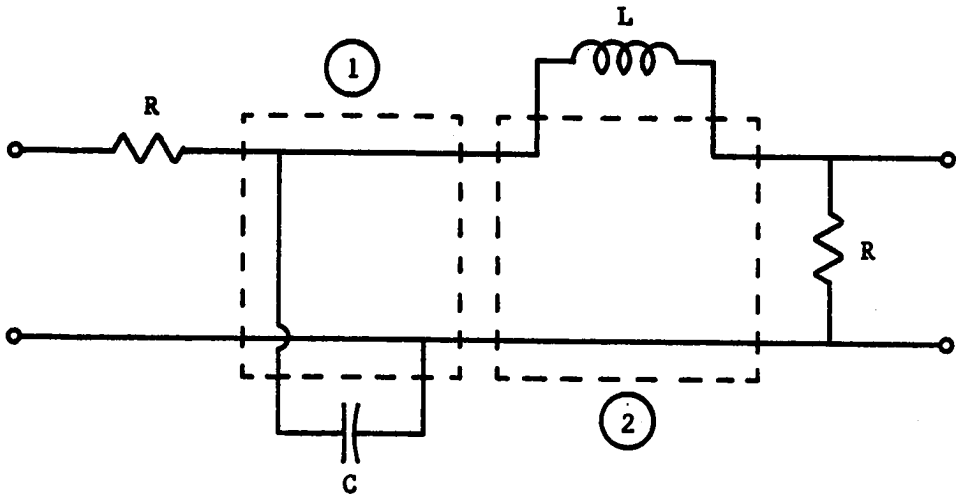
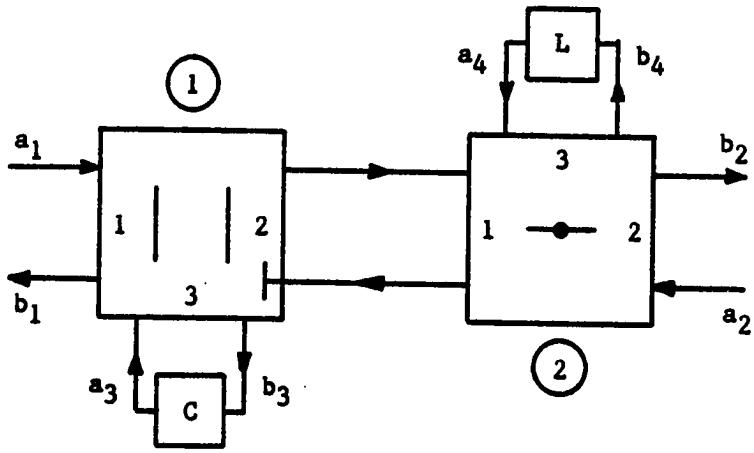


Figure 3.11. General second order LC lowpass analog filter



(a) Identification of wire interconnections



(b) Wave digital filter

Figure 3.12. Synthesis of second order LC lowpass wave digital filter

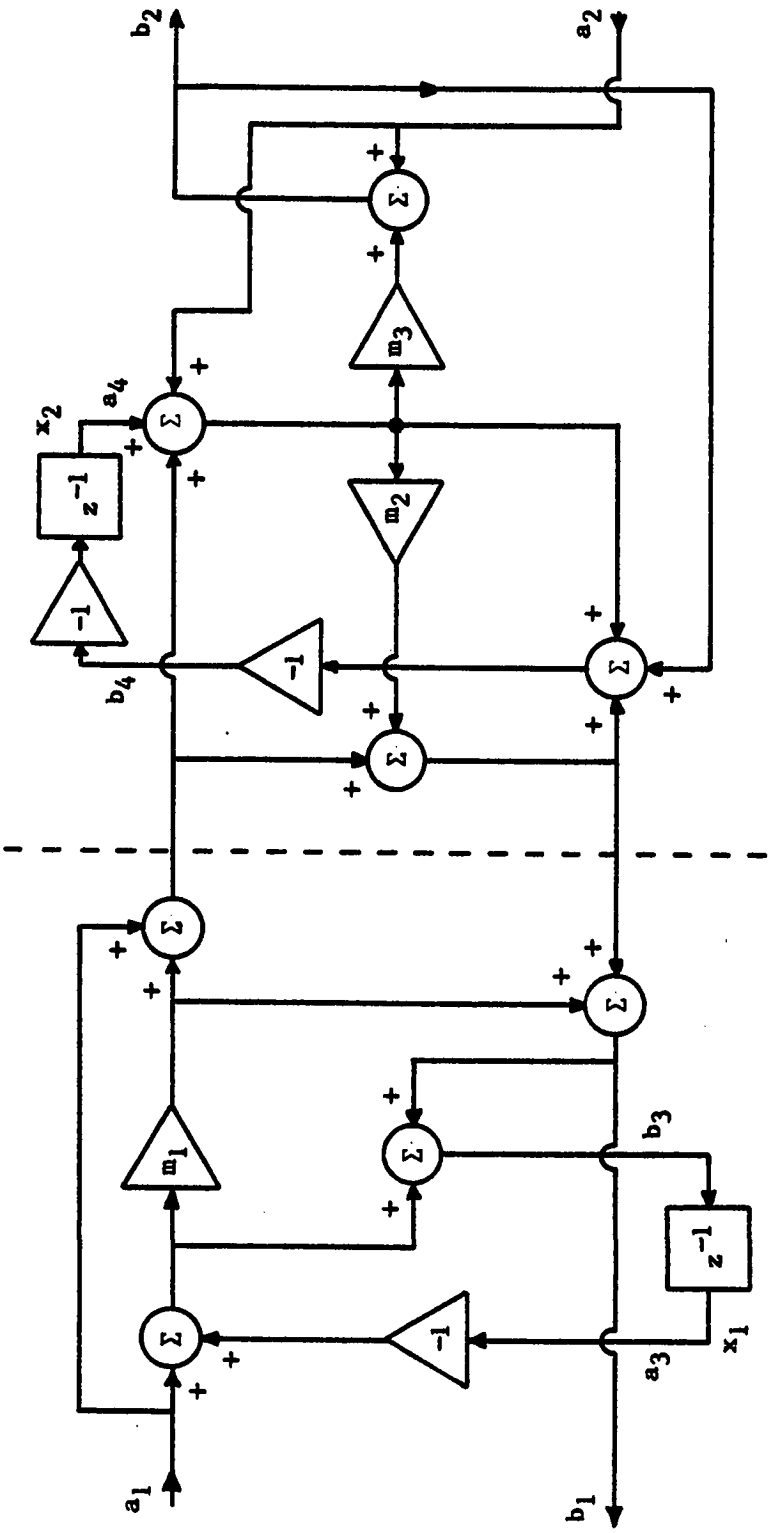


Figure 3.13. Linear wave digital filter structure for specific example

$$
\begin{aligned}
c_{11} &= -1 - m_1(2 + m_2) \\
c_{12} &= m_2 \\
c_{21} &= -m_1(2 + m_2 + m_3) \\
c_{22} &= 1 + m_2 + m_3 \,.
\end{aligned}
\tag{3.43}
$$

Let R_{ij} and G_{ij} represent the port resistance and conductance, respectively of the i^{th} port of the j^{th} interconnection. Let T represent the sample period of the filter. The calculation of the multiplier values proceeds as follows.

Interconnection 1 (P1 parallel adapter):

$$
\begin{aligned}
G_{11} &= \frac{1}{R} \\
G_{31} &= \frac{2C}{T} \\
G_{21} &= G_{11} + G_{31} = \frac{1}{R} + \frac{2C}{T} \\
m_1 &= \frac{G_{11}}{G_{21}} - 1 = \frac{\frac{1}{R}}{\frac{1}{R} + \frac{2C}{T}} - 1 = \frac{-2R(\frac{C}{T})}{1 + 2R\,(\frac{C}{T})} \,.
\end{aligned}
\tag{3.44}
$$

Interconnection 2 (S2 series adapter):

$$
\begin{aligned}
R_{12} &= \frac{1}{G_{21}} = \frac{1}{\frac{1}{R} + \frac{2C}{T}} \\
R_{22} &= R \\
R_{32} &= \frac{2L}{T}
\end{aligned}
\tag{3.45}
$$

$$m_2 = \frac{-2R_{12}}{R_{12} + R_{22} + R_{32}} = \frac{-\frac{2}{\frac{1}{R} + 2(\frac{C}{T})}}{\frac{1}{\frac{1}{R} + 2(\frac{C}{T})} + R + \frac{2L}{T}}$$

$$= \frac{-1}{1 + R(\frac{C}{T}) + \frac{1}{R}(\frac{L}{T}) + 2(\frac{L}{T})(\frac{C}{T})}$$

$$m_3 = \frac{-2R_{22}}{R_{12} + R_{22} + R_{32}} = \frac{-2R}{\frac{1}{\frac{1}{R} + 2(\frac{C}{T})} + R + \frac{2L}{T}}$$

$$= \frac{-[1 + R(\frac{C}{T})]}{1 + R(\frac{C}{T}) + \frac{1}{R}(\frac{L}{T}) + 2(\frac{L}{T})(\frac{C}{T})} \quad .$$

Given the analog filter of Figure 3.11, the actual values of R, L and C depend on the desired type of filter input impedance and cutoff frequency. Impendance scaling is used to change the input impedance of the filter. The magnitude scaling constant, k_m, is defined as

$$k_m = \frac{Z'}{Z} \tag{3.46}$$

where Z is the unscaled input impedance, and Z' is the desired impedance. If we apply impedance scaling to the filter, the new element values are R', C' and L' defined as

$$\begin{aligned} R' &= k_m R \\ L' &= k_m L \\ C' &= \frac{C}{k_m} \quad . \end{aligned} \tag{3.47}$$

However, impedance scaling does not change any of the multiplier values in the wave digital filter. Frequency scaling is used to change the cutoff frequency of the filter and the frequency scaling constant, k_f, is defined as

$$k_f = \frac{\omega'_{co}}{\omega_{co}} \tag{3.48}$$

where ω_{co} is the unscaled cutoff frequency and ω'_{co} is the desired cutoff frequency. If we apply frequency scaling to the filter, the new element values are R", C" and L" defined as

$$\begin{aligned} R" &= R \\ L" &= \frac{L}{k_f} \\ C" &= \frac{C}{k_f} \ . \end{aligned} \tag{3.49}$$

When frequency scaling is applied to the wave digital filter, the values of the multipliers change. As the cutoff frequency of the filter increases, the values of L and C decrease. However, as the cutoff frequency of the filter increases, the sample period of the digital filter should also decrease. Hence, we let $a = \frac{C}{T}$ and $b = \frac{L}{T}$. We also assume R = 1. With these assumptions, the values of the wave digital filter multipliers are

$$\begin{aligned} m_1 &= \frac{-2a}{1 + 2a} \\ m_2 &= \frac{-1}{1 + a + b + 2ab} \\ m_3 &= \frac{-(1 + a)}{1 + a + b + 2ab} = m_2(1 + a). \end{aligned} \tag{3.50}$$

For the passive LC network of Figure 3.11, $L > 0$ and $C > 0$ and so the wave digital filter parameters are also, $a > 0$ and $b > 0$. Fettweis [12] shows that all wave digital filters derived from classical LC networks are also pseudopassive, and therefore globally asymptotically stable, when infinite wordlength is used. Therefore, for the example considered here, the linear wave digital filter is globally asymptotically stable when $a > 0$ and $b > 0$.

We consider two possible structures for a nonlinear wave digital filter. Quantization and overflow nonlinearities can be applied at the states of the filter. This two quantizer structure is shown in Figure 3.14. This structure has received previous attention by other authors. We also consider quantization after each multiplication, as shown in Figure 3.15. In this case, there are three quantizers. This structure is a more realistic implementation of the actual filter using a microprocessor. We do not consider overflow nonlinearities due to the large number of adders. We next present the procedure used to generate the extreme matrices that are used by the constructive algorithm.

b. Two quantizers The wave digital filter structure we consider is shown in Figure 3.14. As in the direct form and coupled form filters, the quantization and overflow nonlinearities are considered together. Under this assumption, the state equations are

$$\begin{aligned} x_1(k+1) &= f_1[c_{11}x_1(k) + c_{12}x_2(k)] \\ x_2(k+1) &= f_2[c_{21}x_1(k) + c_{22}x_2(k)] \end{aligned} \qquad (3.51)$$

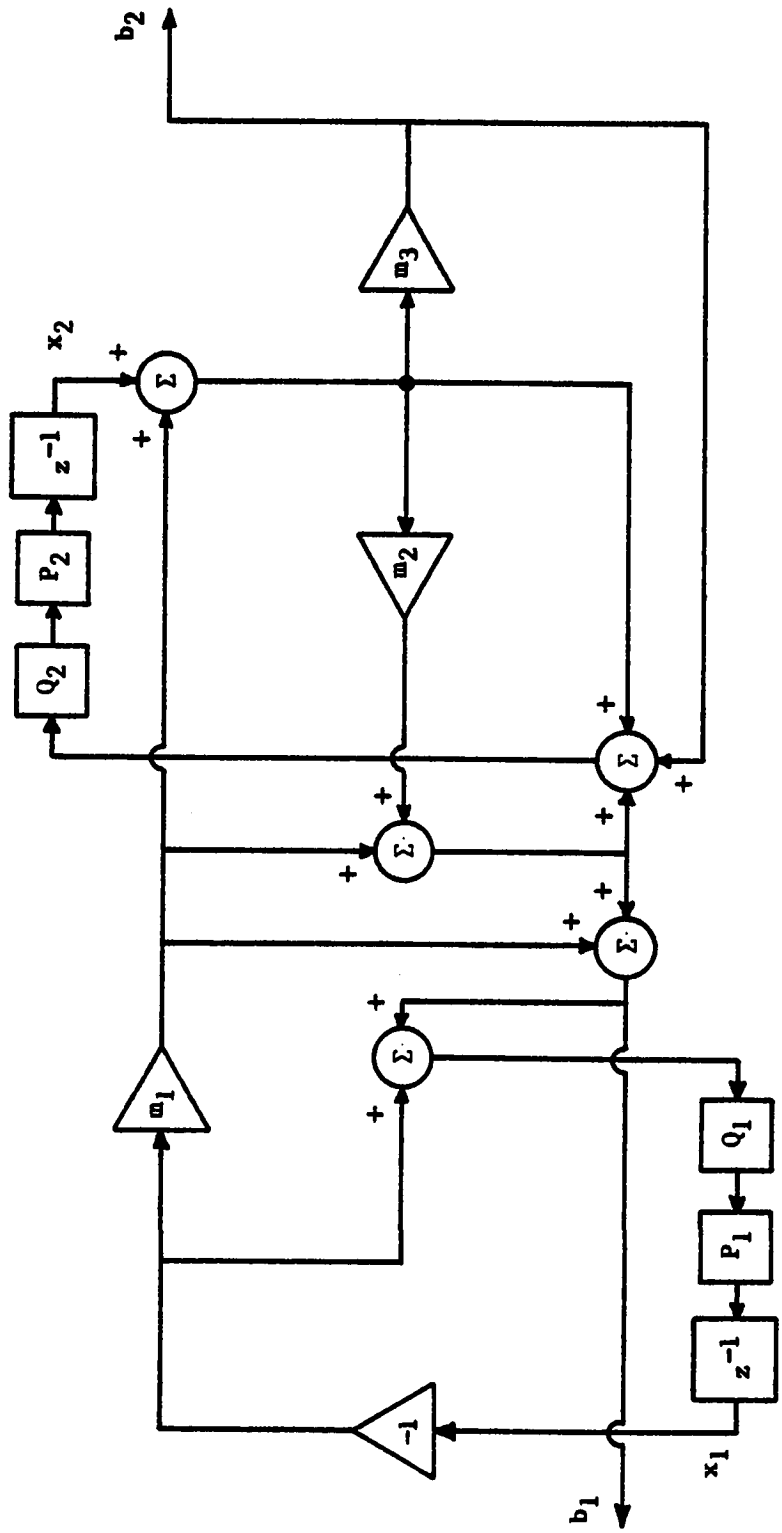


Figure 3.14. Wave digital filter with two quantizers

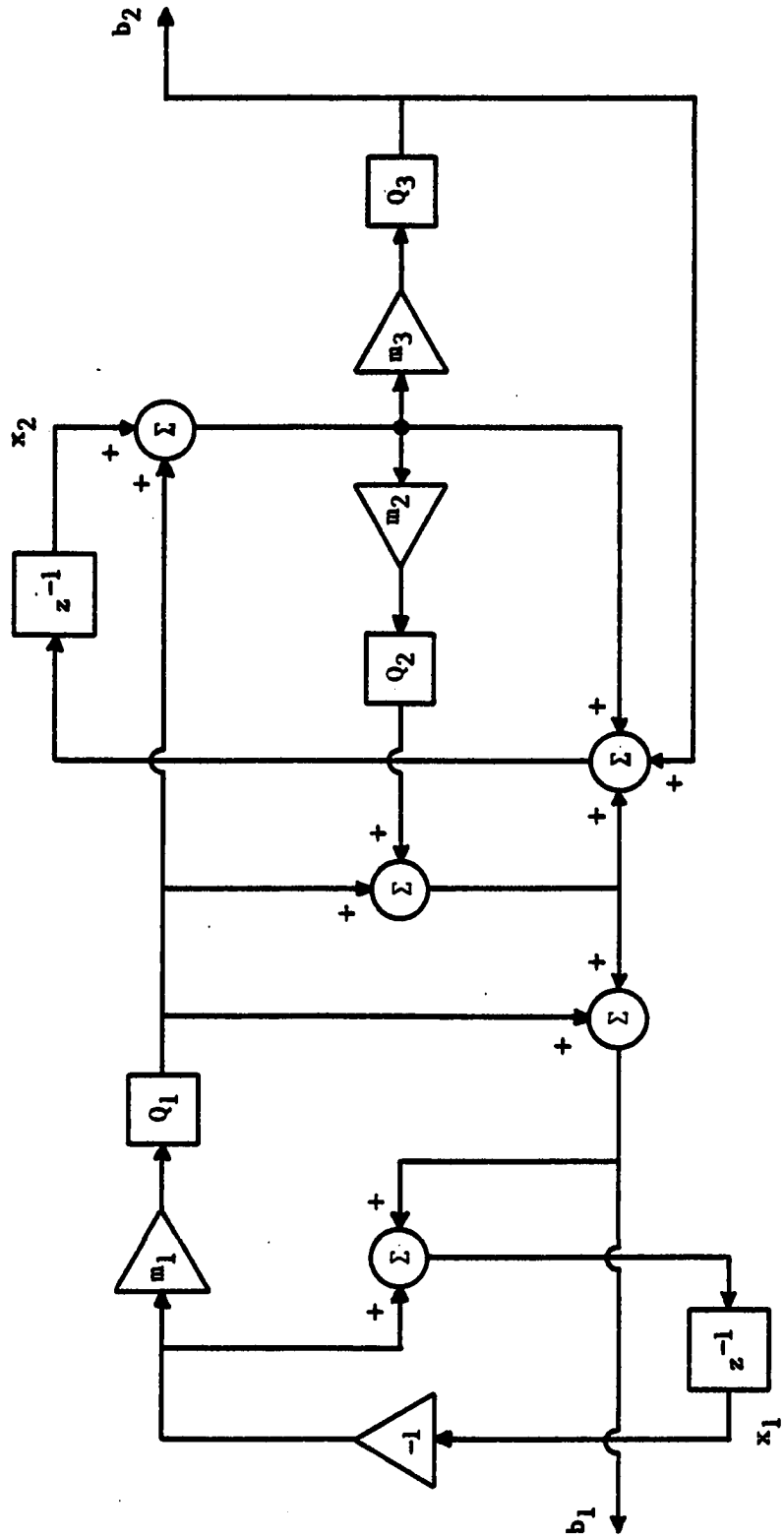


Figure 3.15. Wave digital filter with three quantizers

where $f_1(\cdot)$ and $f_2(\cdot)$ are the combined overflow and quantization nonlinearities. The coefficients c_{ij}, $i,j = 1,2$ are defined by Equations (3.43) and (3.50).

Following the technique outlined in Section B of this chapter, the state equations are written as

$$x(k+1) = M(x(k))x(k)$$

where

$$M(x(k)) = \begin{bmatrix} \Phi_1(x)c_{11} & \Phi_1(x)c_{12} \\ \Phi_1(x)c_{21} & \Phi_2(x)c_{22} \end{bmatrix} \tag{3.52}$$

and

$$\Phi_1(x) = \frac{f_1[c_{11}x_1 + c_{12}x_2]}{c_{11}x_1 + c_{12}x_2}$$
$$\Phi_2(x) = \frac{f_2[c_{21}x_1 + c_{22}x_2]}{c_{21}x_1 + c_{22}x_2} . \tag{3.53}$$

The function Φ_1 and Φ_2 are bounded by constants,

$$\alpha_1 \leq \Phi_1(x) \leq \alpha_2$$
$$\beta_1 \leq \Phi_2(x) \leq \beta_2 . \tag{3.54}$$

For our analysis,

$$\alpha_1 = \beta_1 = k_o$$
$$\alpha_2 = \beta_2 = k_q . \tag{3.55}$$

The extreme matrices of the set $\underline{M}$ are

$$E(\underline{M}) = \left\{ \begin{bmatrix} \alpha_i c_{11} & \alpha_i c_{12} \\ \beta_j c_{21} & \beta_j c_{22} \end{bmatrix}, \; i,j = 1,2 \right\} . \qquad (3.56)$$

Therefore, the constructive algorithm uses four extreme matrices for each point in the a-b parameter plane. If the overflow nonlinearities are absent, then $\alpha_1 = \beta_1 = 0$ and the set of extreme matrices for this case is the same as for the filter with saturation or zeroing overflow nonlinearities.

c. Three quantizers The wave digital filter structure we consider is shown in Figure 3.15. Note that only quantization nonlinearities are present in this filter. The state equations for this structure are

$$\begin{aligned} x_1(k+1) &= -x_1(k) + 2Q_1[-m_1 x_1(k)] + Q_2\{m_2 Q_1[-m_1 x_1(k)]\} + \\ &\quad + Q_2[m_2 x_2(k)] \\ x_2(k+1) &= 2Q_1[-m_1 x_1(k)] + Q_2\{m_2 Q_1[-m_1 x_1(k)]\} + \\ &\quad + Q_3\{m_3 Q_1[-m_1 x_1(k)]\} + Q_2[m_2 x_2(k)] + \\ &\quad + Q_3[m_3 x_2(k)] + x_2(k) . \end{aligned} \qquad (3.57)$$

To apply the constructive stability algorithm, we write the state equations as

$$x(k+1) = M(x(k))x(k).$$

By defining,

$$\Phi_1(x) = \frac{Q_1[-m_1x_1]}{-m_1x_1}$$

$$\Phi_2(x) = \frac{Q_2[m_2x_2]}{m_2x_2}$$

$$\Phi_3(x) = \frac{Q_3[m_3x_2]}{m_3x_2}$$

$$\Phi_4(x) = \frac{Q_2\{m_2Q_1[-m_1x_1]\}}{m_2Q_1[-m_1x_1]} \qquad (3.58)$$

$$\Phi_5(x) = \frac{Q_3\{m_3Q_1[-m_1x_1]\}}{m_3Q_1[-m_1x_1]}$$

$$\Phi_6(x) = -1 - 2m_1\Phi_1(x) - m_1m_2\Phi_1(x)\Phi_4(x)$$

$$\Phi_7(x) = -2m_1\Phi_1(x) - m_1m_2\Phi_1(x)\Phi_4(x) - m_1m_3\Phi_1(x)\Phi_5(x)$$

$$\Phi_8(x) = 1 + m_2\Phi_2(x) + m_3\Phi_3(x)$$

we can write M(x(k)) as

$$M(x(k)) = \begin{bmatrix} \Phi_6(x) & m_2\Phi_2(x) \\ \Phi_7(x) & \Phi_8(x) \end{bmatrix}. \qquad (3.59)$$

The functions $\Phi_i(x)$, $i = 1, \cdots, 5$, are bounded by constants,

$$\begin{aligned} \alpha_1 &< \Phi_1(x) < \alpha_2 \\ \beta_1 &< \Phi_2(x) < \beta_2 \\ \gamma_1 &< \Phi_3(x) < \gamma_2 \\ \beta_1 &< \Phi_4(x) < \beta_2 \\ \gamma_1 &< \Phi_5(x) < \gamma_2 \end{aligned} \qquad (3.60)$$

where

$$\alpha_1 = b_1 = \gamma_1 = 0$$
$$\alpha_2 = \beta_2 = \gamma_2 = k_q \ . \tag{3.61}$$

The functions $\Phi_1(x)\Phi_4(x)$ and $\Phi_1(x)\Phi_5(x)$ are also bounded by constants

$$\delta_1 \leq \Phi_1(x)\Phi_4(x) \leq \delta_2$$
$$\varepsilon_1 \leq \Phi_1(x)\Phi_5(x) \leq \varepsilon_2 \tag{3.62}$$

where

$$\delta_1 = \varepsilon_1 = 0$$
$$\delta_2 = \varepsilon_2 = k_q^2 \ . \tag{3.63}$$

The functions $\Phi_6(x)$, $\Phi_7(x)$ and $\Phi_8(x)$ are also bounded by constants,

$$\zeta_1 \leq \Phi_6(x) \leq \zeta_2$$
$$\eta_1 \leq \Phi_7(x) \leq \eta_2 \tag{3.64}$$
$$\theta_1 \leq \Phi_8(x) \leq \theta_2$$

where

$$\begin{aligned}
\zeta_1 &= \min\{[-1-2m_1\alpha_i-m_1m_2\delta_j],\ i,j = 1,2\} \\
\zeta_2 &= \max\{[-1-2m_1\alpha_i-m_1m_2\delta_j],\ i,j = 1,2\} \\
\eta_1 &= \min\{[-2m_1\alpha_i-m_1m_2\delta_j-m_1m_3\varepsilon_k],\ i,j,k = 1,2\} \\
\eta_2 &= \max\{[-2m_1\alpha_i-m_1m_2\delta_j-m_1m_3\varepsilon_k],\ i,j,k = 1,2\} \\
\theta_1 &= \min\{[1+m_2\beta_i+m_3\gamma_j],\ i,j = 1,2\} \\
\theta_2 &= \max\{[1+m_2\beta_i+m_3\gamma_j],\ i,j = 1,2\} \ .
\end{aligned} \tag{3.65}$$

Therefore, we write the exteme matrices of the set $\underline{M}$ as

$$E(\underline{M}) = \left\{ \begin{bmatrix} \zeta_i & m_2\beta_j \\ \eta_k & \theta_\ell \end{bmatrix}, \; i,j,k,\ell = 1,2 \right\} . \qquad (3.66)$$

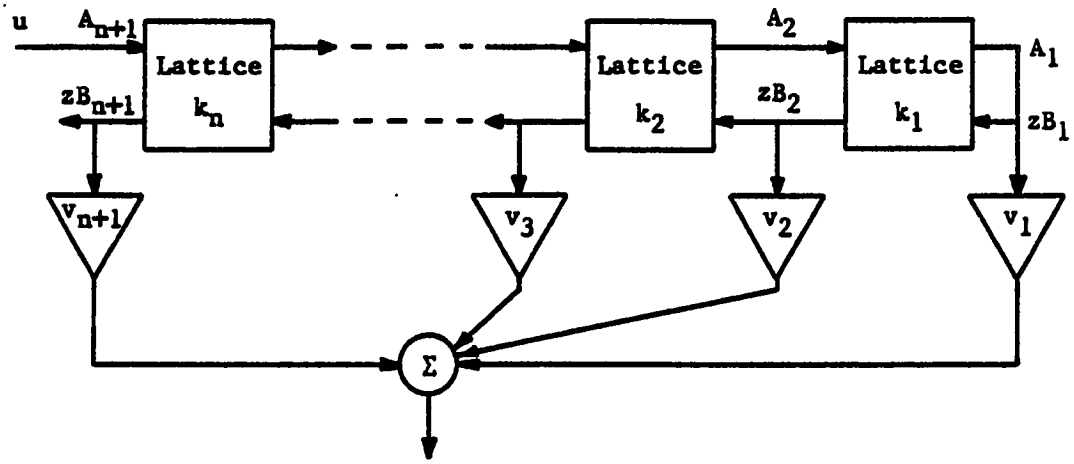
Thus, for each point in the a-b parameter plane, the constructive algorithm uses sixteen matrices.

4. Lattice digital filter

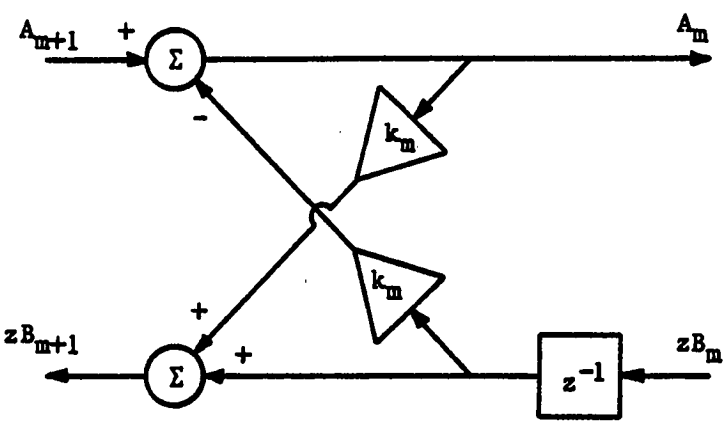
Since their introduction by Itakura and Saito [13], lattice digital filters have been used extensively in the area of speech and signal processing [14]. A general lattice filter is shown in Figure 3.16(a) as a cascade of lattice sections. The particular lattice structure we consider is the two multiplier lattice of Gray and Markel [15]. One section of this type of lattice filter is shown in Figure 3.16(b). Gray and Markel [15] have shown that the linear digital lattice filter will have all of its poles within the unit circle, and thus will be globally asymptotically stable, if and only if all of the k_m parameters satisfy

$$|k_m| < 1, \quad m = 1, 2, \cdots, n . \qquad (3.67)$$

We investigate two possible structures for the second order lattice digital filter. In the first structure, the quantization and overflow nonlinearities are applied at the states of the filter. We consider this first structure since it has been studied previously. This second order filter structure is shown in Figure 3.17. In the



a) General lattice structure



b) Two multiplier lattice section

Figure 3.16. General lattice digital filter structure

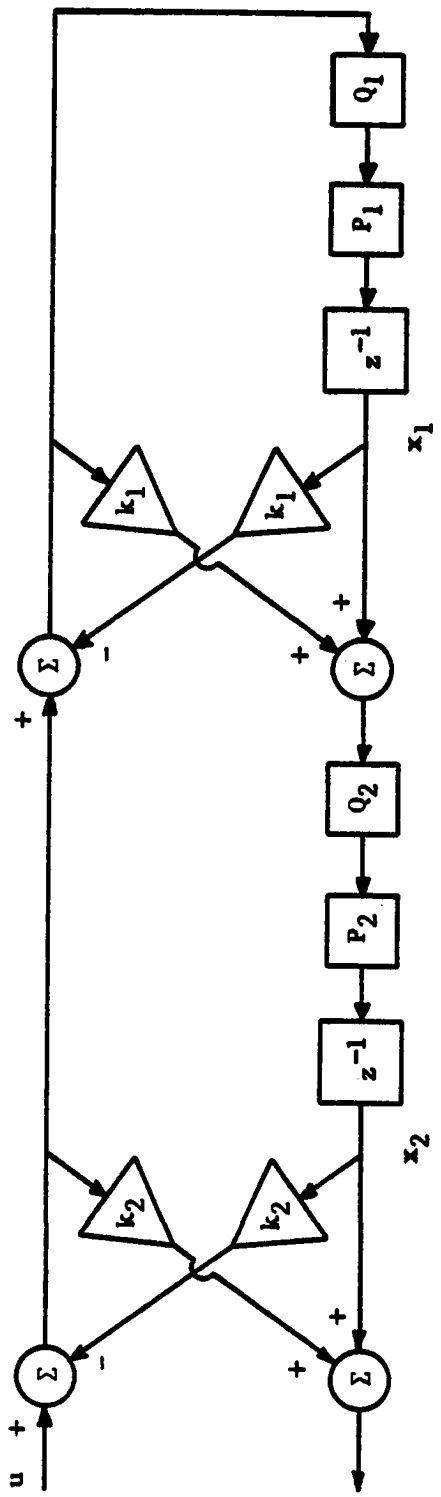


Figure 3.17. Lattice digital filter with two quantizers

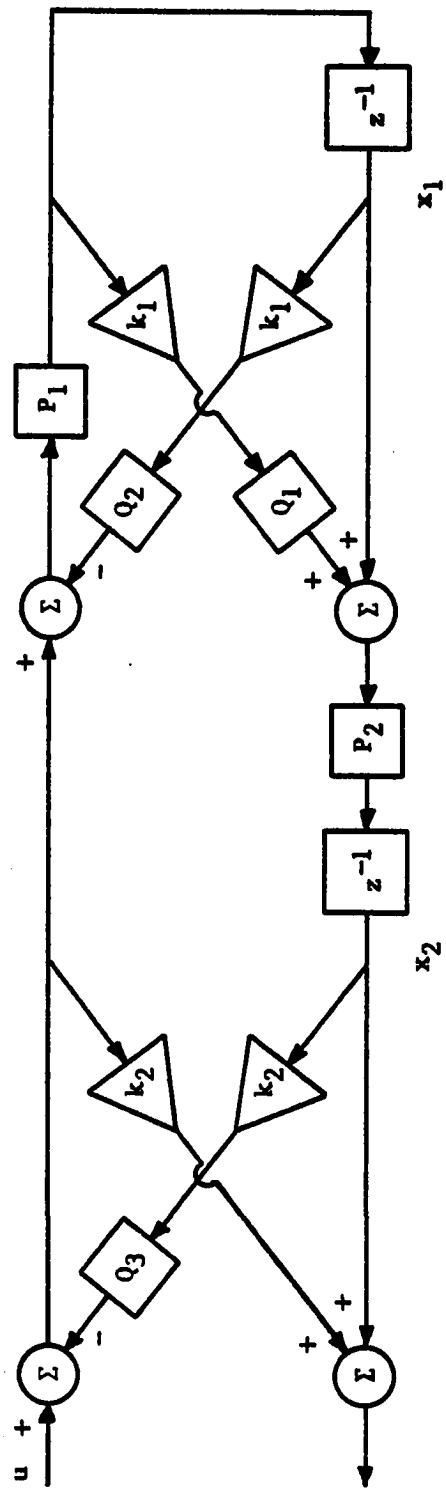


Figure 3.18. Lattice digital filter with three quantizers

second structure, quantization is assumed to take place after each multiplication and overflow is placed after each addition. This structure with three quantizers is a more realistic implementation of the actual filter using a fixed-point microprocessor and is shown in Figure 3.18.

a. Two quantizers The structure of the second order lattice digital filter with two quantizers is shown in Figure 3.17. Again, we consider the quantization and overflow nonlinearities together. The state equations for the structure are

$$\begin{aligned} x_1(k+1) &= f_1[-k_1x_1(k) - k_2x_2(k)] \\ x_2(k+1) &= f_2[(1-k_1^2)x_1(k) - k_1k_2x_2(k)] \end{aligned} \tag{3.68}$$

where $f_1(\cdot)$ and $f_2(\cdot)$ are the combined quantization and overflow nonlinearities.

Following the technique outlined in Section B of this chapter, the state equations are written as

$$x(k+1)' = M(x(k))x(k).$$

By defining

$$\begin{aligned} \Phi_1(x) &= \frac{f_1[-k_1x_1 - k_2x_2]}{-k_1x_1 - k_2x_2} \\ \Phi_2(x) &= \frac{f_2[(1-k_1^2)x_1 - k_1k_2x_2]}{(1-k_1^2)x_1 - k_1k_2x_2} \end{aligned} \tag{3.69}$$

the matrix M(x(k)) can be written as

$$M(x(k)) = \begin{bmatrix} -k_1\Phi_1(x) & -k_2\Phi_1(x) \\ (1-k_1^2)\Phi_2(x) & -k_1k_2\Phi_2(x) \end{bmatrix}. \qquad (3.70)$$

The function $\Phi_1(x)$ and $\Phi_2(x)$ are bounded by constants

$$\begin{aligned} \alpha_1 &\leq \Phi_1(k) \leq \alpha_2 \\ \beta_1 &\leq \Phi_2(x) \leq \beta_2 \end{aligned} \qquad (3.71)$$

where

$$\begin{aligned} \alpha_1 &= \beta_1 = k_o \\ \alpha_2 &= \beta_2 = k_q . \end{aligned} \qquad (3.72)$$

The extreme matrices of the set $\underline{M}$ are

$$E(\underline{M}) = \left\{ \begin{bmatrix} -k_1\alpha_i & -k_2\alpha_i \\ (1-k_1^2)\beta_j & -k_1k_2\beta_j \end{bmatrix}, \; i,j = 1,2 \right\}. \qquad (3.73)$$

Thus, for each point in the k_1-k_2 parameter plane, the constructive algorithm uses four extreme matrices. If the overflow nonlinearities are absent, then $\alpha_1 = \beta_1 = 0$ and the set of extreme matrices for this case is the same as for the filter with two saturation or zeroing overflow nonlinearities.

b. Three quantizers The lattice digital filter to be considered is shown in Figure 3.18. The state equations for this digital filter are

$$x_1(k+1) = P_1\{-Q_2[k_1x_1(k)] - Q_3[k_2x_2(k)]\}$$

$$x_2(k+1) = P_2\{x_1(k) + Q_1[k_1P_1(-Q_2[k_1x_1(k)] - Q_3[k_2x_1(k)])]\}. \tag{3.74}$$

To apply the constructive stability algorithm, we write the state equations as

$$x(k+1) = M(x(k))x(k).$$

The matrix M(x(k)) is given by

$$M(x(k)) = \begin{bmatrix} -k_1\Phi_1(x)\Phi_4(x) & -k_2\Phi_2(x)\Phi_4(x) \\ \Phi_5(x)[1-k_1^2\Phi_1(x)\Phi_3(x)\Phi_4(x)] & -k_1k_2\Phi_2(x)\Phi_3(x)\Phi_4(x)\Phi_5(x) \end{bmatrix} \tag{3.75}$$

where

$$\Phi_1(x) = \frac{Q_2[k_1x_1]}{k_1x_1}$$

$$\Phi_2(x) = \frac{Q_3[k_2x_2]}{k_2x_2}$$

$$\Phi_3(x) = \frac{Q_1[k_1P_1(-Q_2[k_1x_1] - Q_3[k_2x_2])]}{k_1P_1(-Q_2[k_1x_1] - Q_3[k_2x_2])} \tag{3.76}$$

$$\Phi_4(x) = \frac{P_1\{-Q_2[k_1x_1] - Q_3[k_2x_2]\}}{-Q_2[k_1x_1] - Q_3[k_2x_2]}$$

$$\Phi_5(x) = \frac{P_2\{x_1 + Q_1[k_1P_1(-Q_2[k_1x_1] - Q_3[k_2x_2])]\}}{x_1 + Q_1[k_1P_1(-Q_2[k_1x_1] - Q_3[k_2x_2])]}.$$

The functions $\Phi_i(x)$, $i = 1,\cdots, 5$ are bounded by constants

$$\begin{aligned}
\alpha_1 &\leq \Phi_1(x) \leq \alpha_2 \\
\beta_1 &\leq \Phi_2(x) \leq \beta_2 \\
\gamma_1 &\leq \Phi_3(x) \leq \gamma_2 \\
\delta_1 &\leq \Phi_4(x) \leq \delta_2 \\
\varepsilon_1 &\leq \Phi_5(x) \leq \varepsilon_2
\end{aligned} \tag{3.77}$$

where

$$\begin{aligned}
\alpha_1 &= \beta_1 = \gamma_1 = 0 \\
\alpha_2 &= \beta_2 = \gamma_2 = k_q \\
\delta_1 &= \varepsilon_1 = k_o \\
\delta_2 &= \varepsilon_2 = 1 \ .
\end{aligned} \tag{3.78}$$

Combinations of these functions are also bounded by constants, i.e.,

$$\begin{aligned}
\zeta_1 &\leq \Phi_1(x)\Phi_4(x) \leq \zeta_2 \\
\eta_1 &\leq \Phi_2(x)\Phi_4(x) \leq \eta_2 \\
\theta_1 &\leq \Phi_2(x)\Phi_3(x)\Phi_4(x)\Phi_5(x) \leq \theta_2 \\
\lambda_1 &\leq \Phi_5(x)[1-k_1^2\Phi_1(x)\Phi_3(x)\Phi_4(x)] \leq \lambda_2
\end{aligned} \tag{3.79}$$

where, for the nonlinearities which we consider,

$$\begin{aligned}
\zeta_1 &= \eta_1 = k_o k_q \\
\zeta_2 &= \eta_2 = k_q \\
\theta_1 &= k_o k_q^2 \\
\theta_2 &= k_q^2 \\
\lambda_1 &= \min[\varepsilon_i(1-k_1^2\alpha_j\gamma_k\delta_\ell),\ i,j,k,\ell = 1,2] \\
\lambda_2 &= \max[\varepsilon_i(1-k_1^2\alpha_j\gamma_k\delta_\ell),\ i,j,k,\ell = 1,2] \ .
\end{aligned} \tag{3.80}$$

Thus, the extreme matrices of the set $\underline{M}$ are

$$E(\underline{M}) = \left\{ \begin{bmatrix} -k_1\zeta_i & -k_2\eta_j \\ \lambda_k & -k_1k_2\theta_\ell \end{bmatrix}, \; i,j,k,\ell = 1,2 \right\}. \tag{3.81}$$

In this case, the constructive algorithm uses sixteen extreme matrices for every point in the k_1-k_2 parameter plane.

If the overflow nonlinearities are absent, the set of extreme matrices is not the same as for the filter with saturation or zeroing overflow nonlinearities. For this case, the constants that bound the combinations of the functions in (3.79) are

$$\begin{aligned} \zeta_1 &= \eta_1 = \theta_1 = 0 \\ \zeta_2 &= \eta_2 = k_q \\ \theta_2 &= k_q^2 \\ \lambda_1 &= 1 - k_1^2k_q^2 \\ \lambda_2 &= 1 \; . \end{aligned} \tag{3.82}$$

IV. COMPARISON OF STABILITY RESULTS BY THE CONSTRUCTIVE ALGORITHM WITH EXISTING STABILITY RESULTS

In this chapter, we present the stability results obtained by applying the Brayton-Tong constructive algorithm to different nonlinear digital filter structures. For each structure, we present the best known existing stability results. There are two categories of existing stability results for fixed-point digital filters. Some of the existing results are sufficient conditions for the absence of limit cycles in a digital filter. Other existing results are sufficient conditions for the global asymptotic stability of a digital filter. Results in the latter category are also sufficient conditions for the absence of limit cycles. We compare these existing stability results with the stability results due to the constructive algorithm. We will use the constructive algorithm to ascertain the global asymptotic stability of the equilibrium $x = 0$ of a digital filter which also guarantees the absence of limit cycles.

A. Direct Form Digital Filter

For a direct form digital filter, we consider the filter implemented with one or two quantizers. For both of these structures, we review existing results on the stability of the direct form filter and compare these results with the global asymptotic stability results obtained from the constructive algorithm.

1. One quantizer

a. Truncation quantizer For a second order direct form digital filter with one truncation quantizer and no overflow nonlinearity, the largest region in the a-b parameter plane where zero-input limit cycles are proven to be absent is shown in Claasen [16]. This result is also reported in [7]. This region where limit cycles do not exist is the same region in the parameter plane where it is shown by Claasen and Kristiansson [17] that the direct form digital filter with one saturation overflow nonlinearity is asymptotically stable with nonzero input. We note that their definition of asymptotic stability is different from our definition of asymptotic stability. (Our definition of asymptotic stability is the commonly used definition.) The interested reader is referred to [17] for the exact definitions of their terms. The result of Claasen [16] is now stated without proof.

Theorem 4.1 No zero-input limit cycles exist in the second order direct form digital filter of Figure 3.6 with one truncation quantizer and no overflow nonlinearity if the following is satisfied:

$$\max_{n>0} |q(n)| < 1. \tag{4.1}$$

If $\frac{a^2}{4} + b > 0$, $q(n)$ is defined as

$$q(n) = -\frac{\lambda_1 \lambda_2}{\lambda_1 - \lambda_2} (\lambda_1^n - \lambda_2^n) \tag{4.2}$$

and

$$\lambda_1, \lambda_2 = \frac{a}{2} \pm \sqrt{\frac{a^2}{4} + b}. \qquad (4.3)$$

If $\frac{a^2}{4} + b < 0$, then q(n) is rewritten as

$$q(n) = \frac{-r^{n+1}}{\sin(\beta)} \sin(\beta n) \qquad (4.4)$$

where

$$r = \sqrt{-b}\ , \ \beta = \arccos \frac{a}{2r}\ . \qquad (4.5)$$

This region in the a-b parameter plane where no limit cycles exist for a direct form filter with one truncation quantizer is shown as the unhatched region of Figure 4.1. Only half of the region is shown since it is symmetric with respect to the b axis.

Limit cycles in a second order direct form digital filter with only an overflow nonlinearity have been studied by Ebert, Mazo and Taylor [18]. They show that no overflow oscillations exist in the digital filter when saturation overflow or triangular overflow is used. For two's complement overflow, it is shown that a necessary and sufficient condition for the absence of limit cycles in the filter is that

$$|a| + |b| < 1\ . \qquad (4.6)$$

This region in the a-b parameter plane is depicted as the unhatched region in Figure 4.2. Ebert, Mazo and Taylor also state that zeroing overflow also leads to oscillations, but no analysis is presented in [18] to justify this assertion. There are no results known that investigate the limit cycles due to a zeroing overflow nonlinearity.

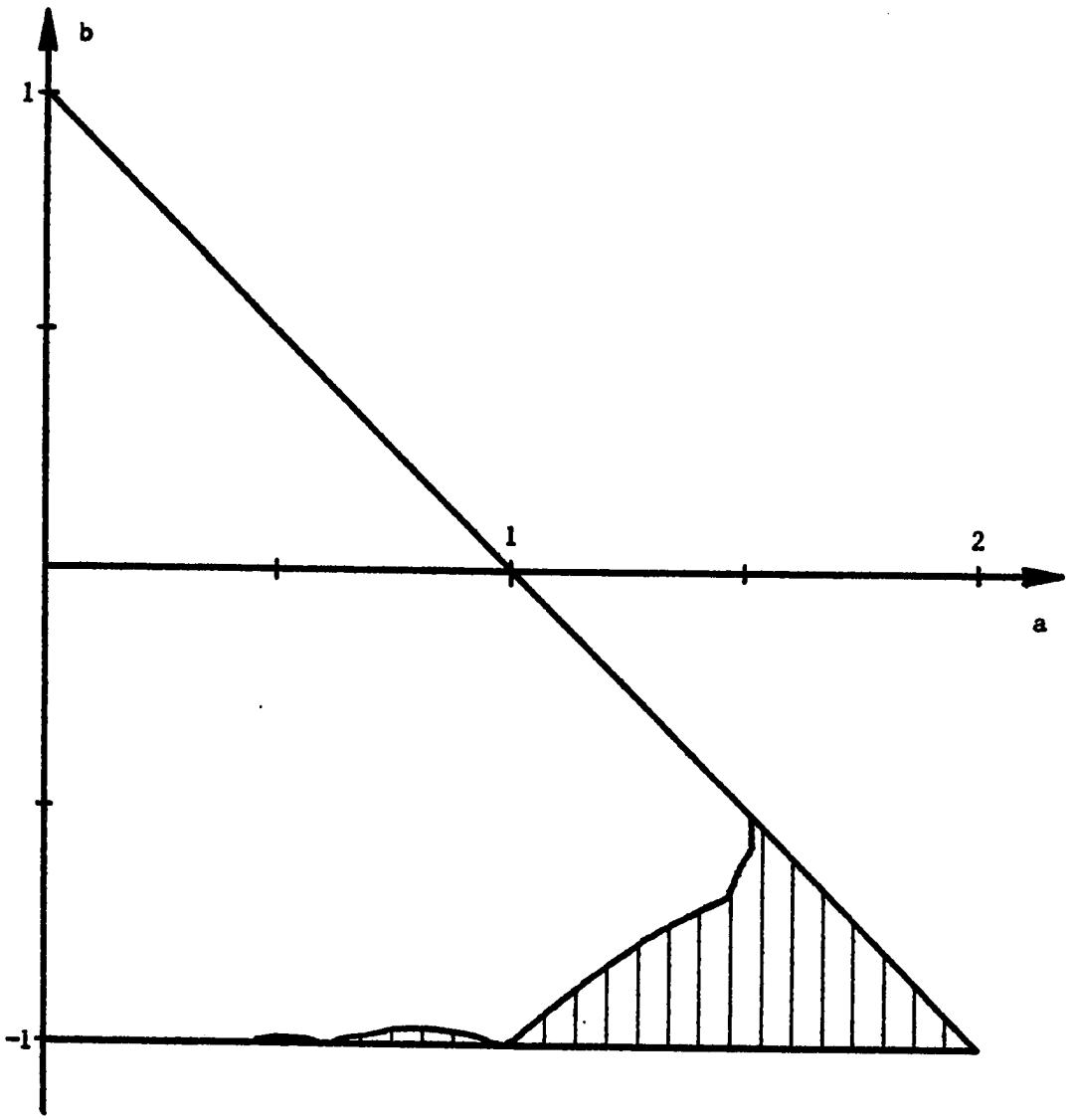


Figure 4.1. Region where a direct form filter with one truncation quantizer is free of limit cycles by Theorem 4.1

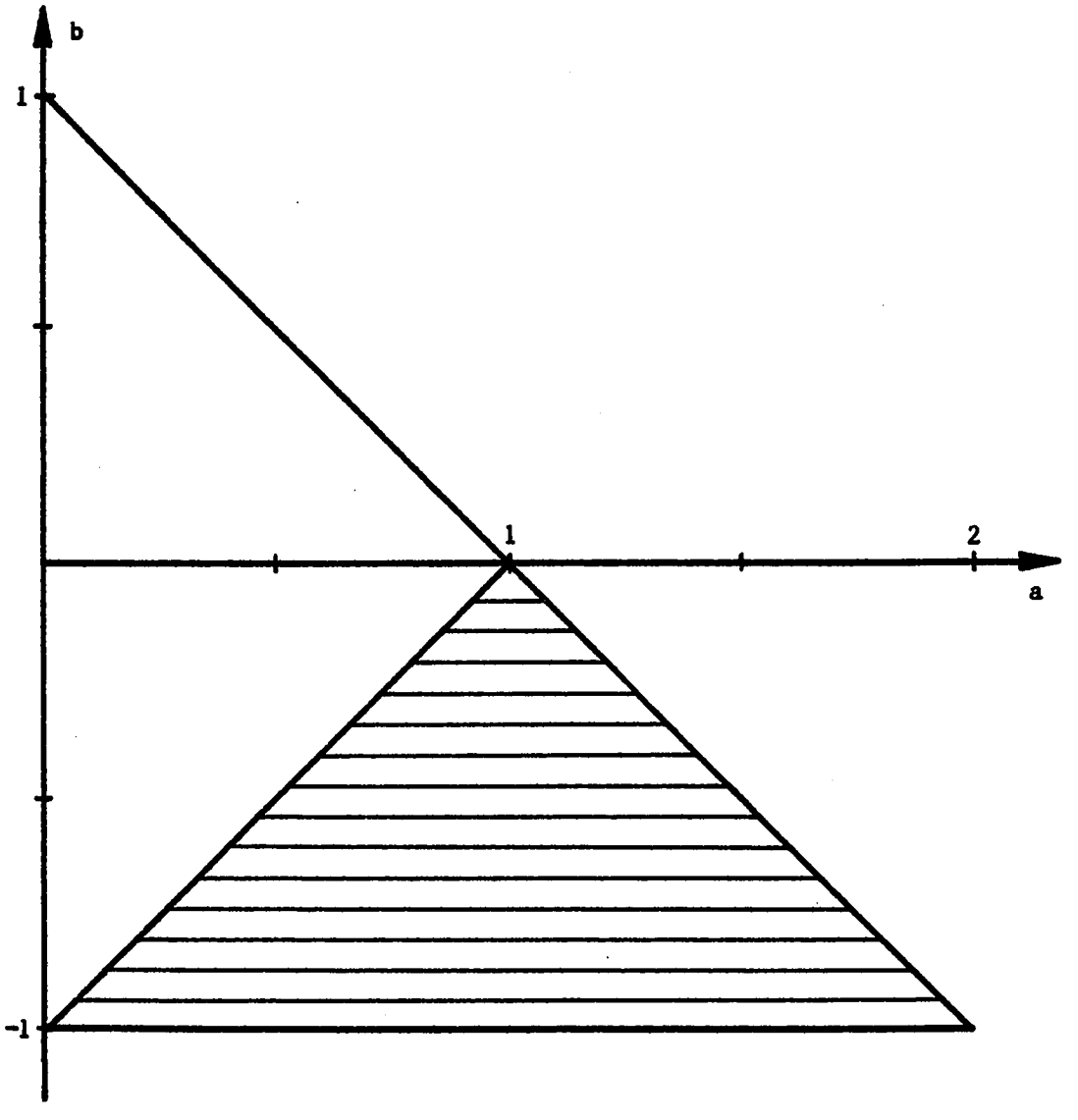


Figure 4.2. Region where a direct form filter with one two's complement overflow nonlinearity is free of limit cycles by Equation 4.6

To apply the constructive algorithm to the direct form filter with one truncation quantizer, we use the extreme matrices determined by Equation (3.16). The region of global asymptotic stability in the a-b parameter plane obtained by the constructive algorithm for this filter with a truncation quantizer and saturation, zeroing or no overflow is shown in Figure 4.3. The regions in the parameter plane where this digital filter is globally asymptotically stable with triangular and two's complement overflow are shown in Figures 4.4 and 4.5, respectively. These regions are symmetric about the b axis. The horizontally hatched region indicates the region where at least one of the eigenvalues of an extreme matrix has a magnitude greater than one. Vertical hatching indicates the rest of the region where we can draw no conclusion about the stability of the filter. The stability results by the constructive algorithm for the overflow nonlinearity without quantization are the same as the results for the overflow and truncation quantization nonlinearities combined.

When quantization is considered separately, the constructive algorithm yields the same region in the parameter plane where the filter is globally asymptotically stable as the result of Claasen [16] that deals with the absence of limit cycles. If we consider the overflow nonlinearity only, then the stability results by the constructive method are more conservative than those of Ebert, Mazo and Taylor [18] for saturation or triangular overflow. However, the constructive algorithm yields the same region where limit cycles are

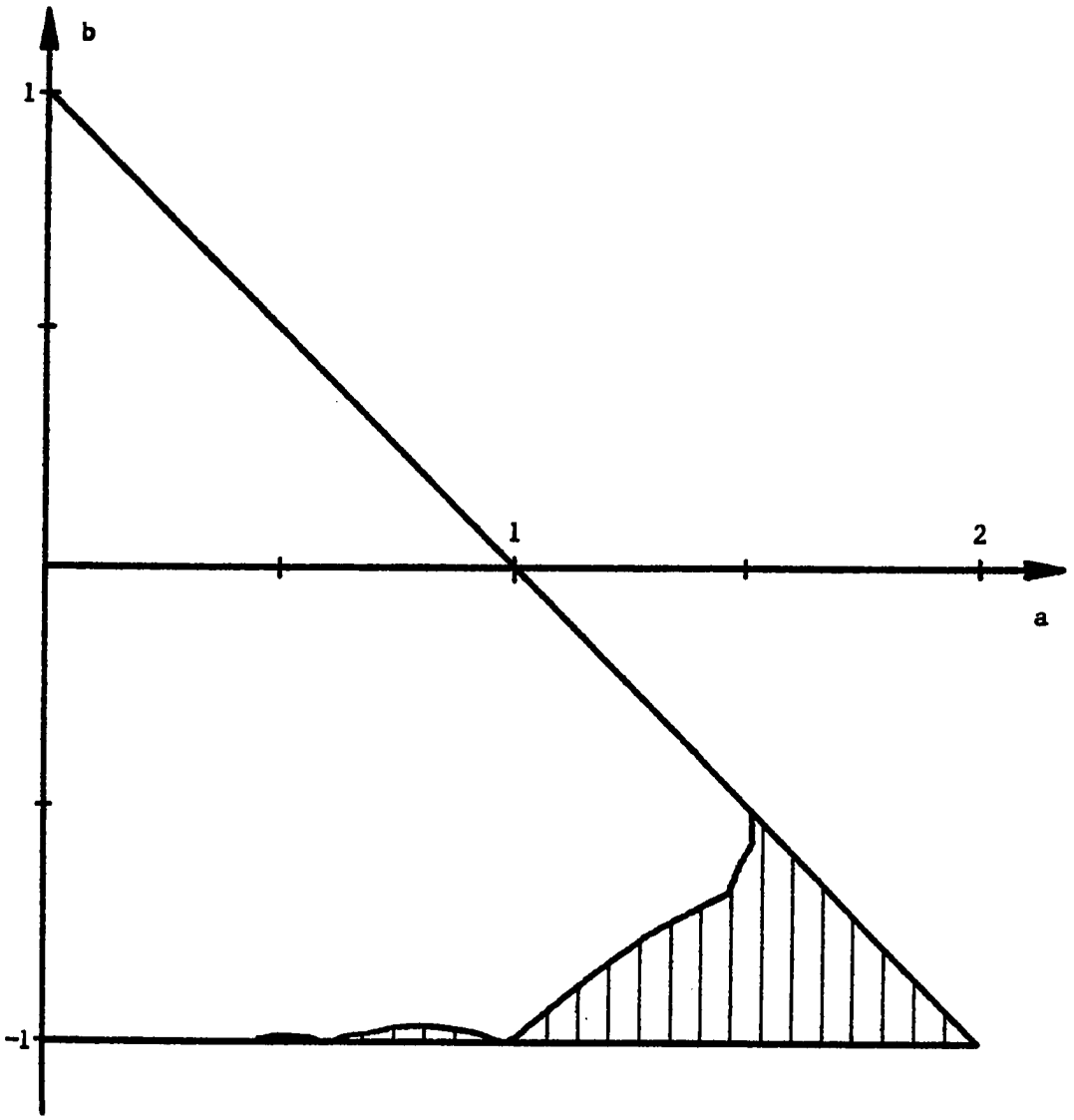


Figure 4.3. Region where a direct form filter with one truncation quantizer and saturation, zeroing or no overflow is g.a.s. by the constructive algorithm

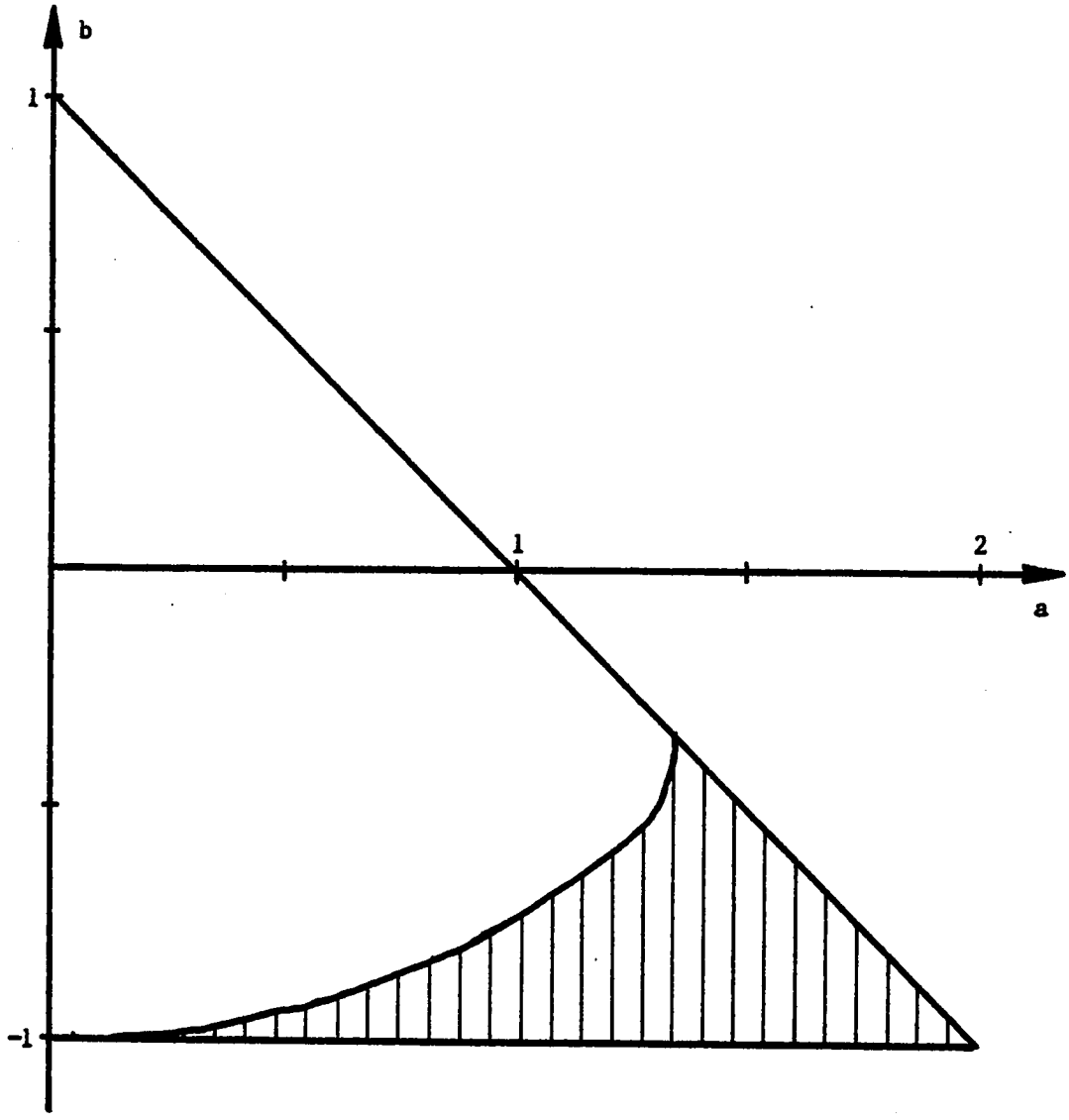


Figure 4.4. Region where a direct form filter with one truncation quantizer and triangular overflow is g.a.s. by the constructive algorithm

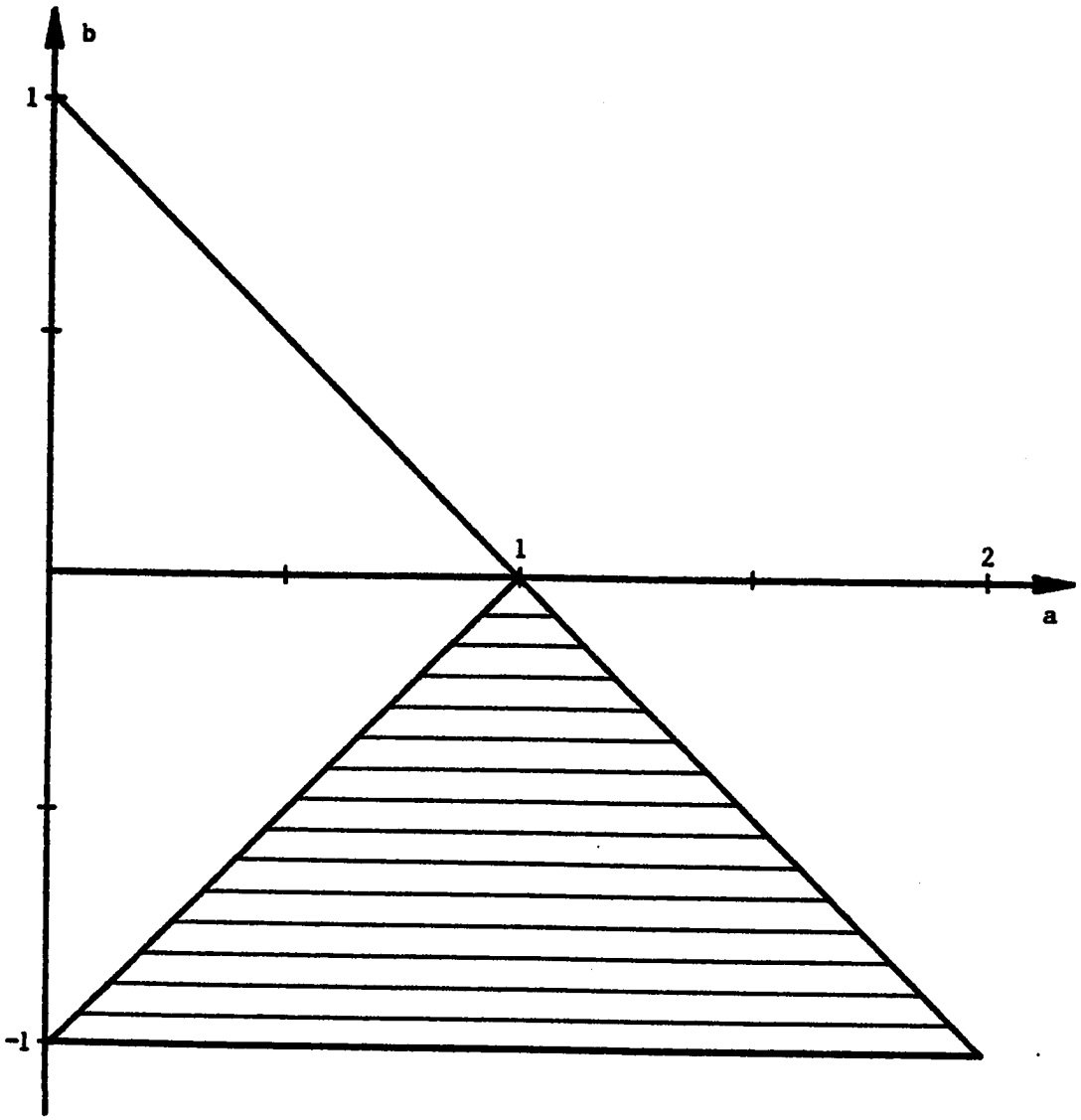


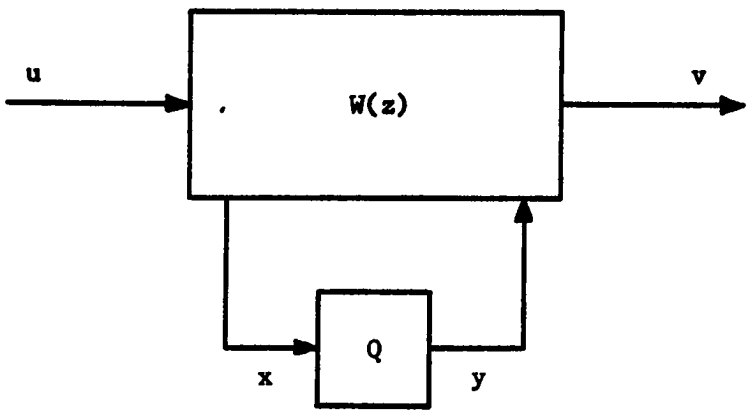
Figure 4.5. Region where a direct form filter with one truncation quantizer and two's complement overflow is g.a.s. by the constructive algorithm

absent for the filter with truncation quantization and two's complement overflow as Ebert, Mazo and Taylor for a single two's complement overflow nonlinearity. This conclusion is not surprising since the extreme matrices used by the constructive algorithm for the overflow nonlinearity only are the same as those used for the overflow and truncation nonlinearity considered together.

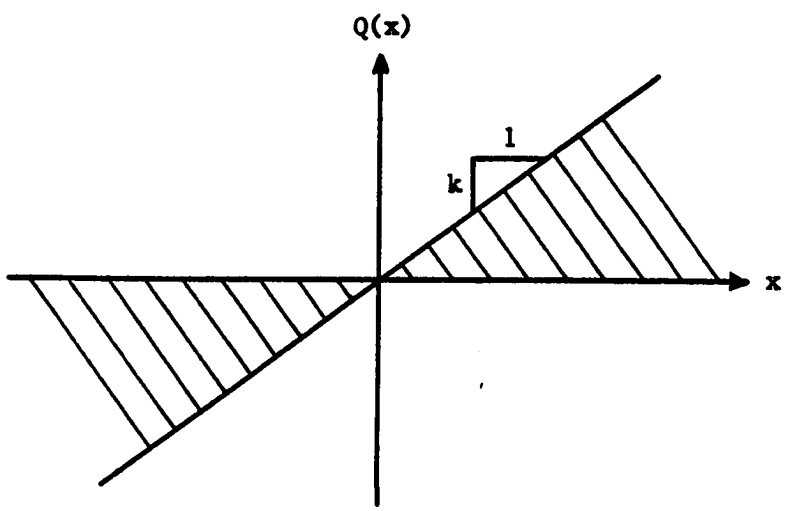
b. Roundoff quantizer For the direct form digital filter with one roundoff quantizer, Claasen et al. [19] have derived a sufficient condition for the absence of zero-input limit cycles. To develop sufficient conditions for the absence of limit cycles in the filter structure shown in Figure 3.6 with just the quantization nonlinearity, consider a nonlinear discrete system with one nonlinear element, Q, depicted in Figure 4.6(a). In considering zero-input limit cycles, the linear part of the system, W, is described by the transfer function, W(z) where $X(z) = W(z)Y(z)$. For Q, we assume that

$$\begin{aligned} &Q(0) = 0 \\ &0 \leq \frac{Q(x)}{x} \leq k , \quad x \neq 0 \\ &[Q(x + h) - Q(x)]h \geq 0, \text{ for all } x \text{ and } h \\ &Q(-x) = - Q(x) . \end{aligned} \tag{4.7}$$

These assumptions imply that the nonlinear characteristic lies in the sector shown in Figure 4.6(b) and is a nondecreasing, odd and symmetric function of x. The result of Claasen et al. [19] is now stated without proof.



a) Nonlinear discrete system



b) Sector in which Q must lie

Figure 4.6. Nonlinear discrete system considered in Theorem 4.2.

Theorem 4.2 Let the discrete system be modeled as depicted in Figure 4.6(a), containing a linear part described by the transfer function W(z), which must be finite for $|z| = 1$, and a nonlinearity satisfying (4.7). Limit cycles of length N are absent from the discrete system if there exist $\alpha_p, \beta_p > 0$ such that for $\ell = 0, 1, \cdots, \lfloor \frac{N}{2} \rfloor$,

$$\mathrm{Re}\left[W(z_\ell) \left[1 + \sum_{p=1}^{N-1} \{\alpha_p(1 - z_\ell^p) + \beta_p(1 + z_\ell^p)\} \right] \right] - \frac{1}{k} < 0 \tag{4.8}$$

where $z_\ell = e^{j(\frac{2\pi}{N})\ell}$ and $\lfloor r \rfloor$ denotes the integer part of r.

Claasen et al. [19] implement this criterion by transforming it into a linear programming problem and applying existing linear programming algorithms. The region in the parameter plane where no limit cycles exist is approximated by taking a large value of N (e.g., N = 70). For roundoff quantization (k = 2), the region in the parameter plane where no limit cycles exist by Theorem 4.2 is indicated as the unhatched region in Figure 4.7. This region is symmetric about the b axis. This criterion can also be applied to the case of one magnitude truncation quantizer, but the region where no limit cycles exist is smaller than the region obtained by Theorem 4.1.

The extreme matrices determined in (3.16) were used to apply the constructive algorithm to the stability analysis of the direct form digital filter with one roundoff quantizer. The region of global asymptotic stability in the parameter plane obtained by the

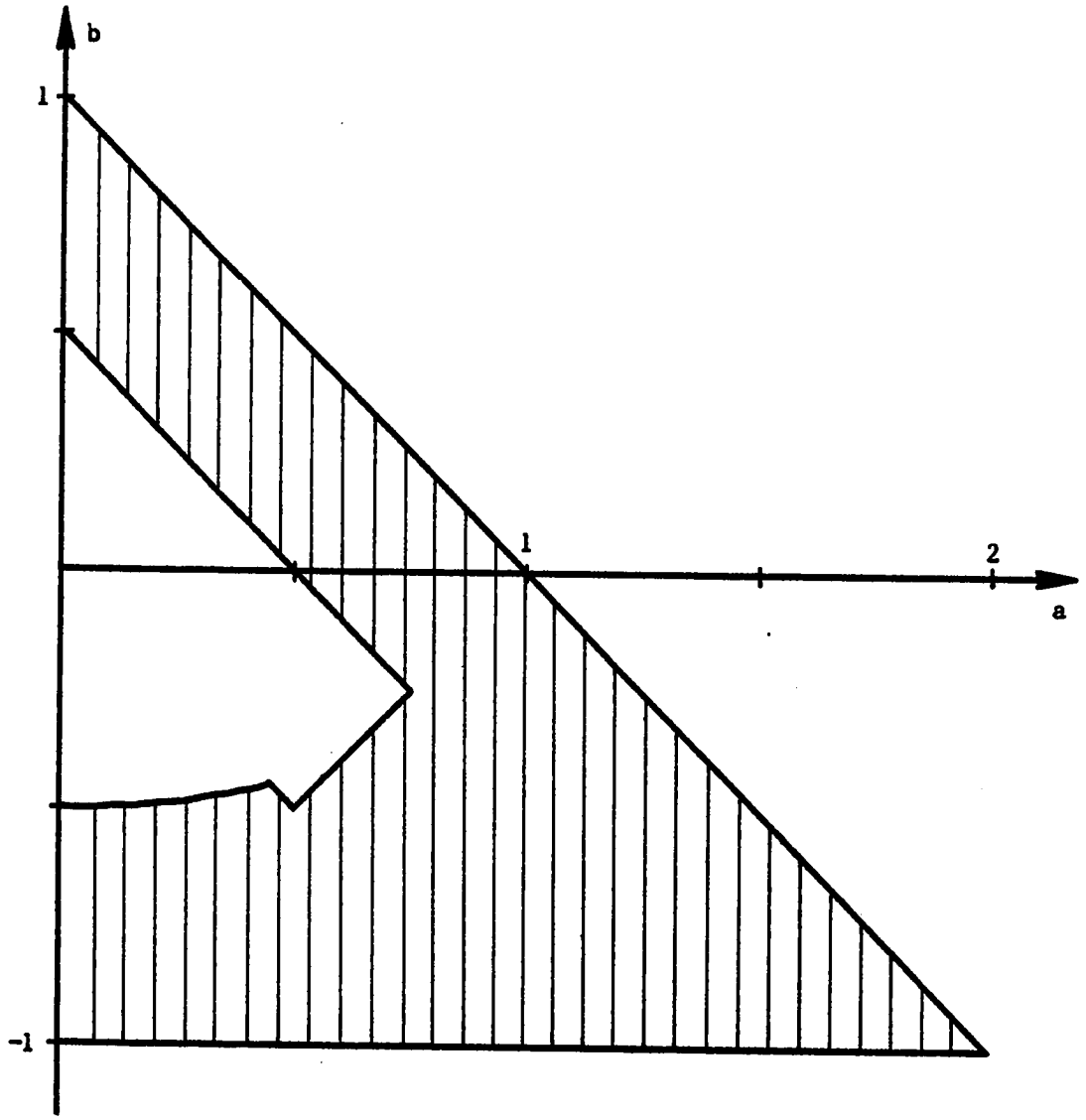


Figure 4.7. Region where a direct form filter with one roundoff quantizer and no overflow is free of limit cycles by Theorem 4.2

constructive algorithm for this filter with a roundoff quantizer and saturation, zeroing or no overflow is shown in Figure 4.8. For this case, the region where the filter is globally asymptotically stable is slightly larger than the region obtained by applying Theorem 4.2. Again, only half of the regions are shown since they are symmetric about the b axis. The horizontally hatched area indicates the region where at least one of the extreme matrices has an eigenvalue with magnitude greater than one. Limit cycles have been found by others in all of the horizontally hatched region [7]. Vertical hatching indicates the remaining region in which the qualitative analysis of the filter is uncertain.

For roundoff quantization and triangular overflow, the region in the parameter plane where the filter is globally asymptotically stable by the constructive algorithm is indicated in Figure 4.9. The corresponding region when two's complement overflow is used is shown in Figure 4.10. Again, horizontal hatching indicates the region where at least one of the extreme matrices has an eigenvalue with magnitude greater than one. Vertical hatching indicates the rest of the uncertain region where we can draw no conclusion about the stability of the system. These results appear to be new.

2. Two quantizers

An absolute stability criterion by Jury and Lee [20] can be used to give a sufficient condition for the global asymptotic stability of the second order direct form filter with two truncation quantizers or

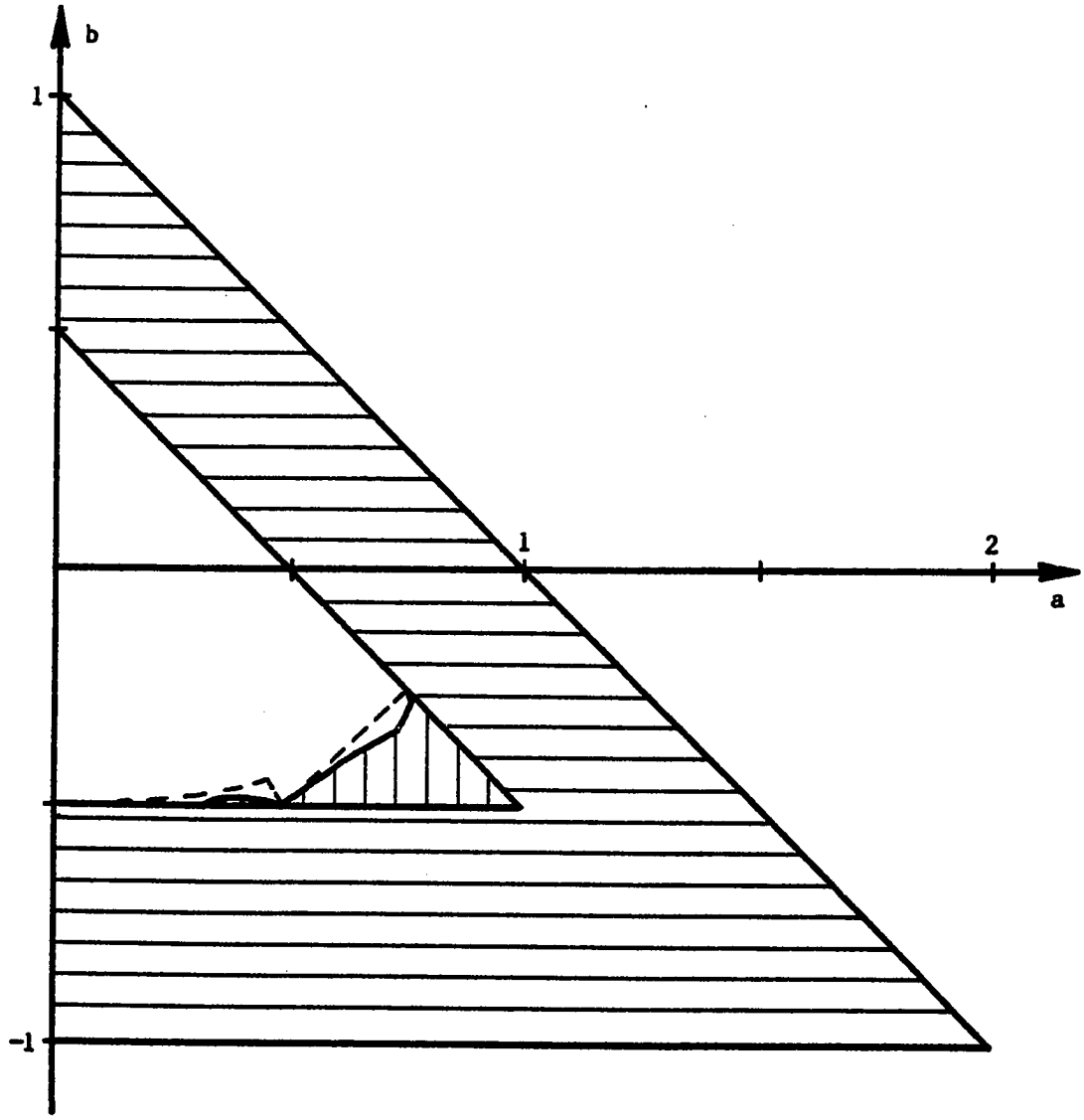


Figure 4.8. Region where a direct form filter with one roundoff quantizer and saturation, zeroing or no overflow is g.a.s. by the constructive algorithm

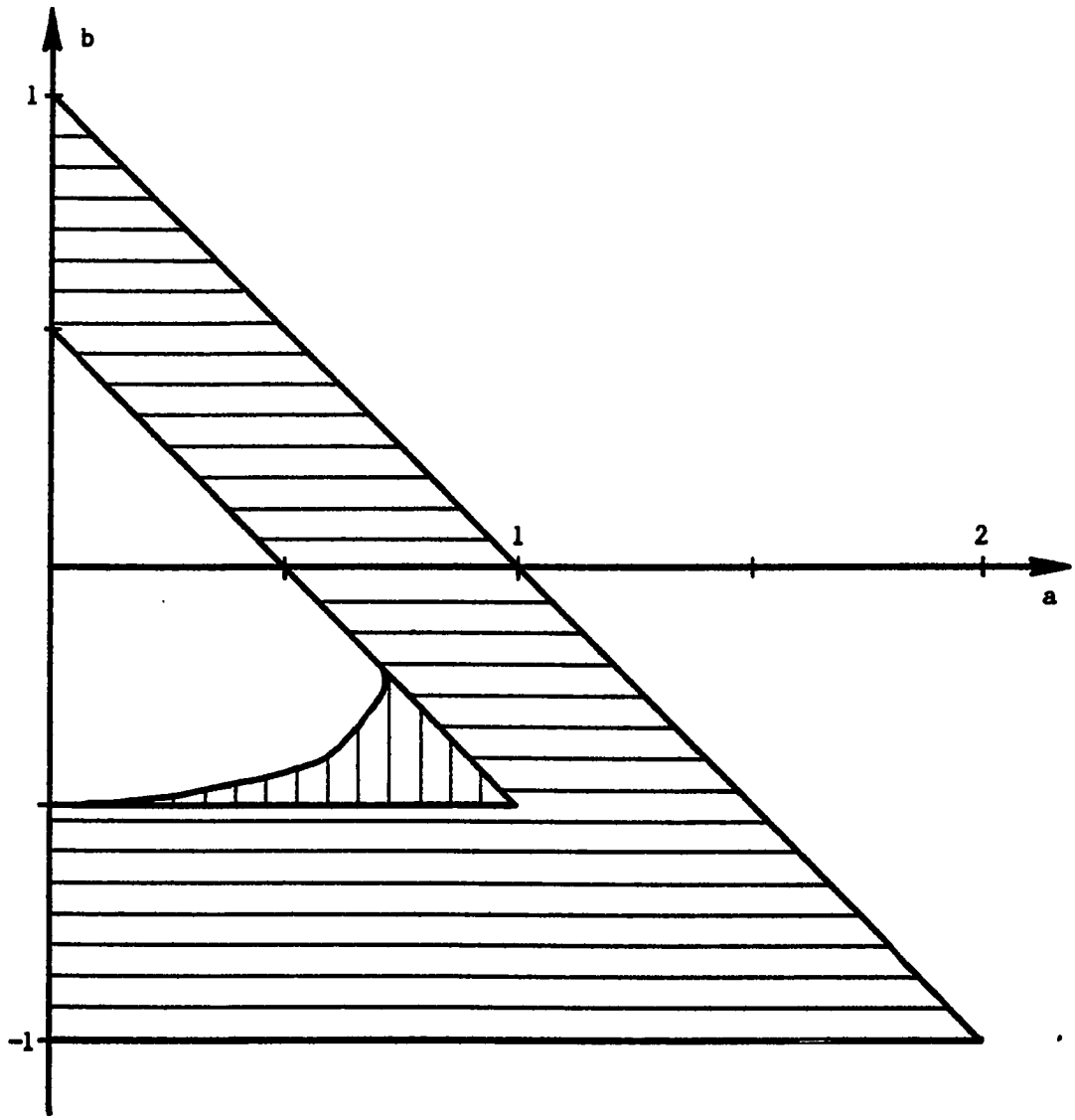


Figure 4.9. Region where a direct form filter with one roundoff quantizer and triangular overflow is g.a.s. by the constructive algorithm

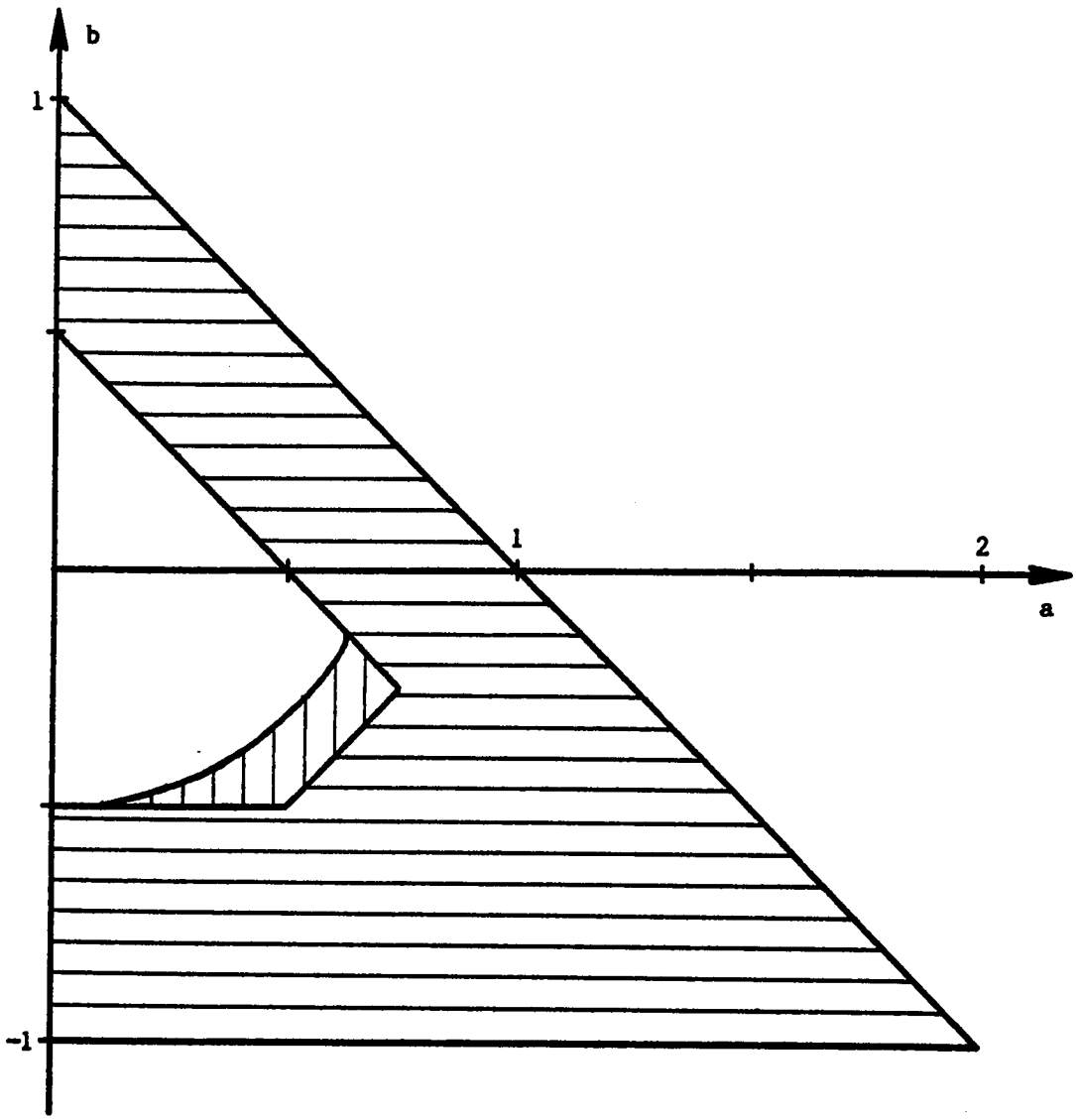


Figure 4.10. Region where a direct form filter with one roundoff quantizer and two's complement overflow is g.a.s. by the constructive algorithm

two roundoff quantizers and no overflow. A nonlinear system is called absolutely stable if the equilibrium x = 0 is globally asymptotically stable for all nonlinearities satisfying Equation (4.9).

A discrete system with several nonlinearities is represented by the system shown in Figure 4.11. The m nonlinear elements are represented by the vector valued function $\underline{f}(\underline{\sigma})$ where $f_i(\sigma_i)$ is the output of the i^{th} nonlinear element. The input of this element is the i^{th} component of the vector $\underline{\sigma}$. The inputs and outputs of the nonlinear elements are interconnected by linear filters with transfer functions $g_{ij}(z)$, which are assumed to be controllable and observable [21]. The functions $g_{ij}(z)$ are the elements of the m x m transfer matrix G(z). We assume that each element $g_{ij}(z)$ has all of its poles within the unit circle except possibly one pole at z = 1. The linear filter with the transfer function $g_{ij}(z)$ connects the output of the j^{th} nonlinear element and the input of the i^{th} nonlinear element. We assume that nonlinearities $f_i(\sigma_i)$ satisfy the following conditions:

$$
\begin{aligned}
&\text{i)} \quad f_i(0) = 0, \qquad i = 1,2, \cdots, m \\
&\text{ii)} \quad 0 < \frac{f_i(\sigma_i)}{\sigma_i} < k_{ii}, \text{ for all } \sigma_i \neq 0 \\
&\text{iii)} \quad \underline{\sigma}(k) \to \underline{0} \text{ implies } \underline{y}(k) \to \underline{0} \\
&\text{iv)} \quad -\infty < \frac{df_i(\sigma_i)}{d\sigma_i} < \infty
\end{aligned}
\tag{4.9}
$$

where k_{ii} is the i^{th} diagonal element of the m x m matrix K. We now

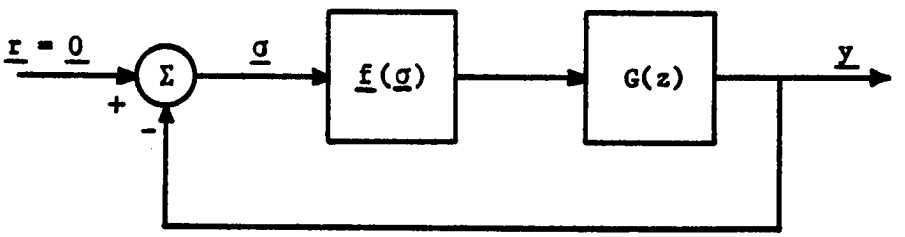


Figure 4.11. A general discrete system with many nonlinearities

state the absolute stability criterion of Jury and Lee [20] without proof.

Theorem 4.3 The system of Figure 4.11 satisfying the above conditions on G(z) with nonlinearities described by (4.9) is absolutely stable if

$$H(z) = 2K^{-1} + G(z) + G^{*}(z) > 0, \text{ for all } |z|=1 \quad (4.10)$$

where $G^{*}(z)$ denotes the conjugate transpose of G(z) and ">" means that the matrix is positive definite.

A sufficient condition that guarantees the absence of limit cycles in a direct form filter with two quantizers that is equivalent to Theorem 4.3 is found in Claasen et al. [19].

For the second order direct form digital filter with two quantization nonlinearities as shown in Figure 3.5, the matrix G(z) may be written as

$$G(z) = \begin{bmatrix} -az^{-1} & -az^{-1} \\ -bz^{-2} & -bz^{-2} \end{bmatrix}. \quad (4.11)$$

The matrix H(z), given by

$$H(z) = \begin{bmatrix} \frac{2}{k_{11}} - az - az^{-1} & -az^{-1} - bz^{2} \\ -az - bz^{-2} & \frac{2}{k_{22}} - bz^{2} - bz^{-2} \end{bmatrix} \quad (4.12)$$

must be positive definite for $|z| = 1$. For magnitude truncation

quantizers, $k_{11} = k_{22} = 1$. The corresponding region in the parameter plane where the filter is globally asymptotically stable is shown as the unhatched region in Figure 4.12. Only half of the region is shown, since it is symmetric about the b axis. For two roundoff quantizers, $k_{11} = k_{22} = 2$. The region where the filter is globally asymptotically stable for this case is presented in Figure 4.13.

We note that Theorem 4.3 cannot be readily applied to triangular or two's complement overflow nonlinearities, since nonlinearities described by (4.9) are constrained to lie entirely in the first and third quadrants.

To apply the constructive stability algorithm to this filter structure, we use the extreme matrices determined in Equation (3.25). The regions in the parameter plane where the digital filter is globally asymptotically stable by the constructive algorithm for all cases are shown as the unhatched regions in Figures 4.14 to 4.19. Horizontal hatching indicates the region where at least one extreme matrix has one eigenvalue with a magnitude greater than one. Vertical hatching indicates the rest of the region where we can draw no conclusion about the stability of the system.

As can be seen from Figures 4.14 and 4.17, the constructive algorithm obtains a less conservative result than the application of Theorem 4.3. Others have shown that limit cycles exist in this filter with truncation quantization and no overflow for all of the

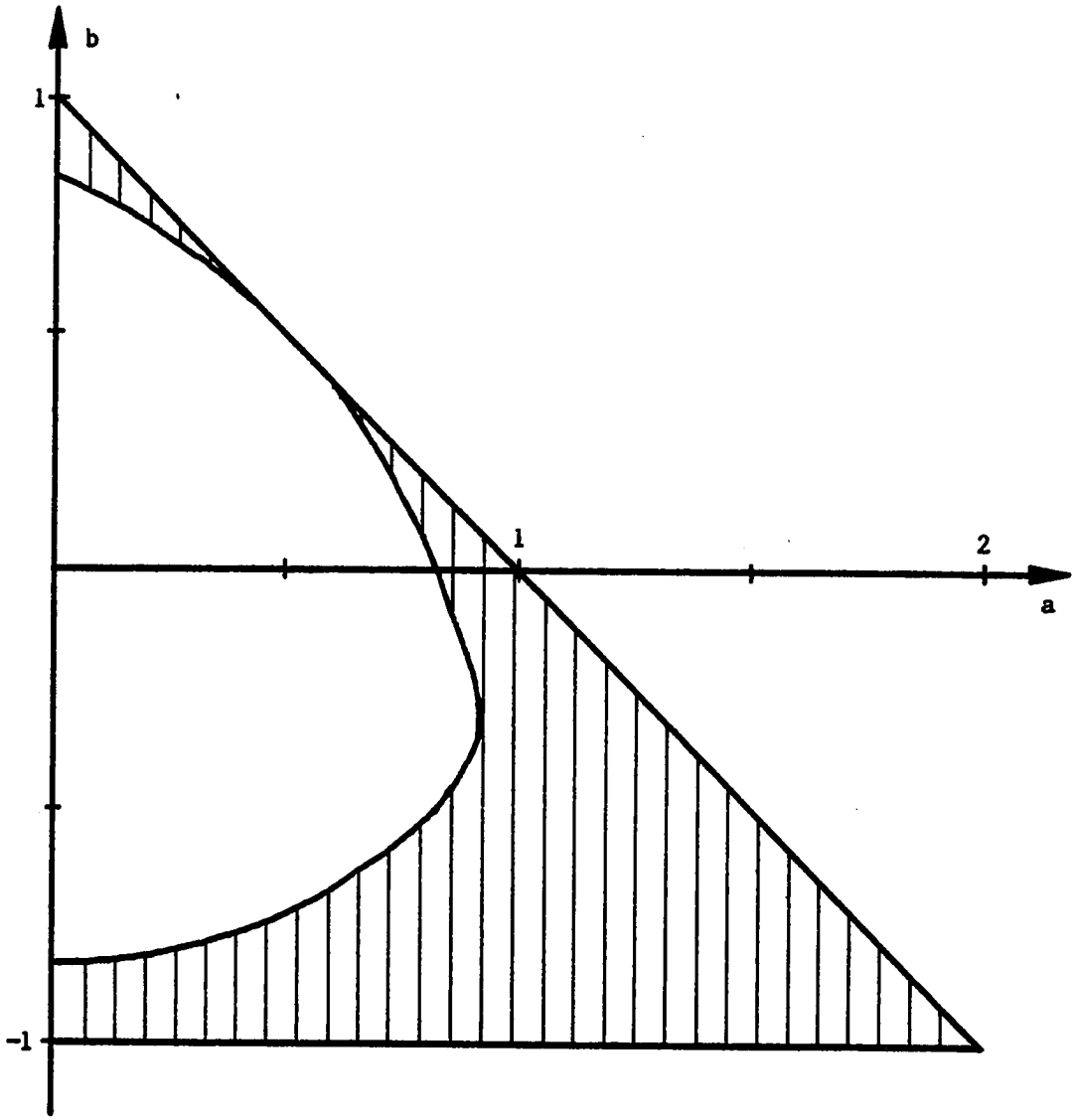


Figure 4.12. Region where a direct form filter with two truncation quantizers and no overflow is g.a.s. by Theorem 4.3

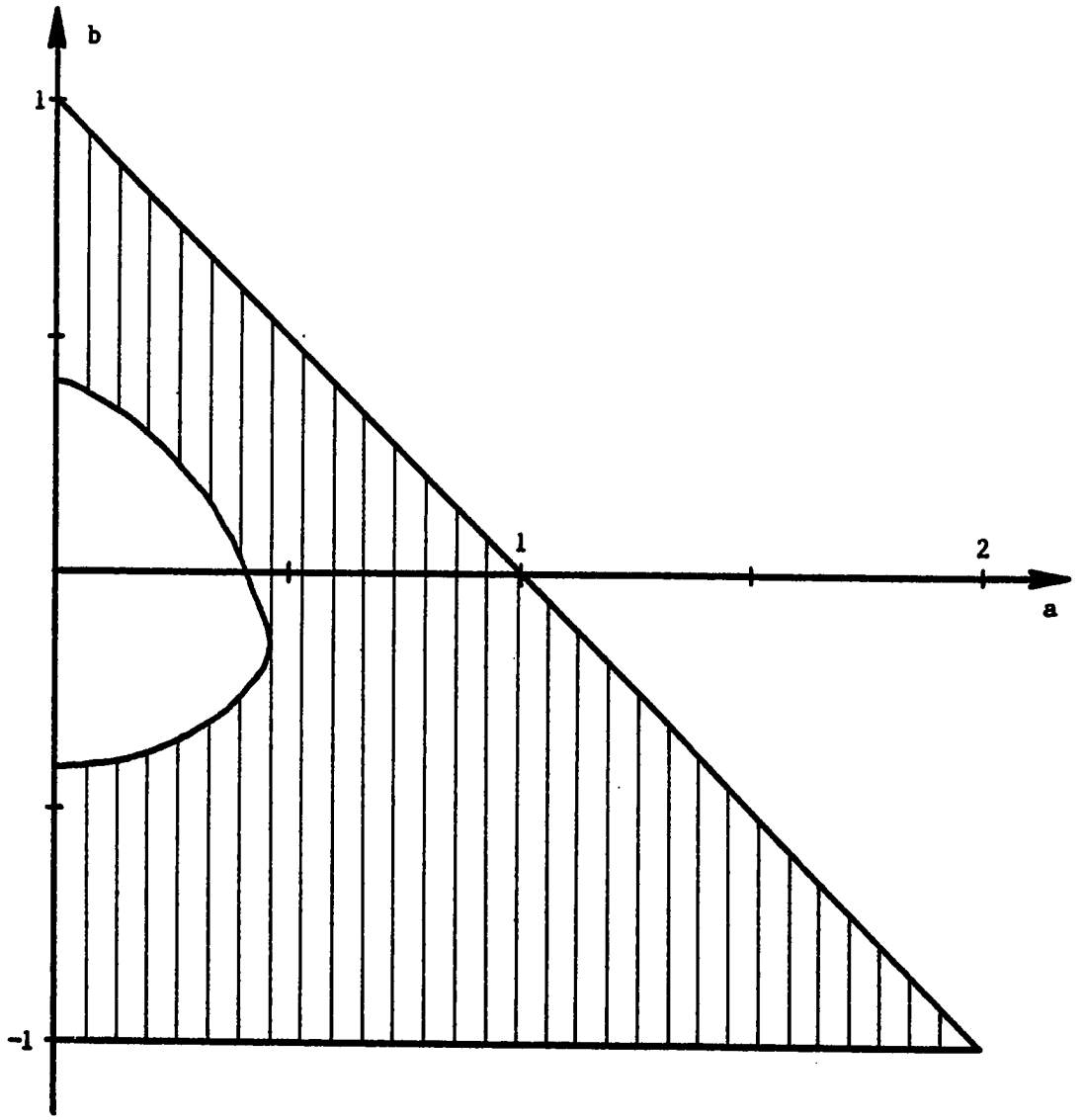


Figure 4.13. Region where a direct form filter with two roundoff quantizers and no overflow is g.a.s. by Theorem 4.3

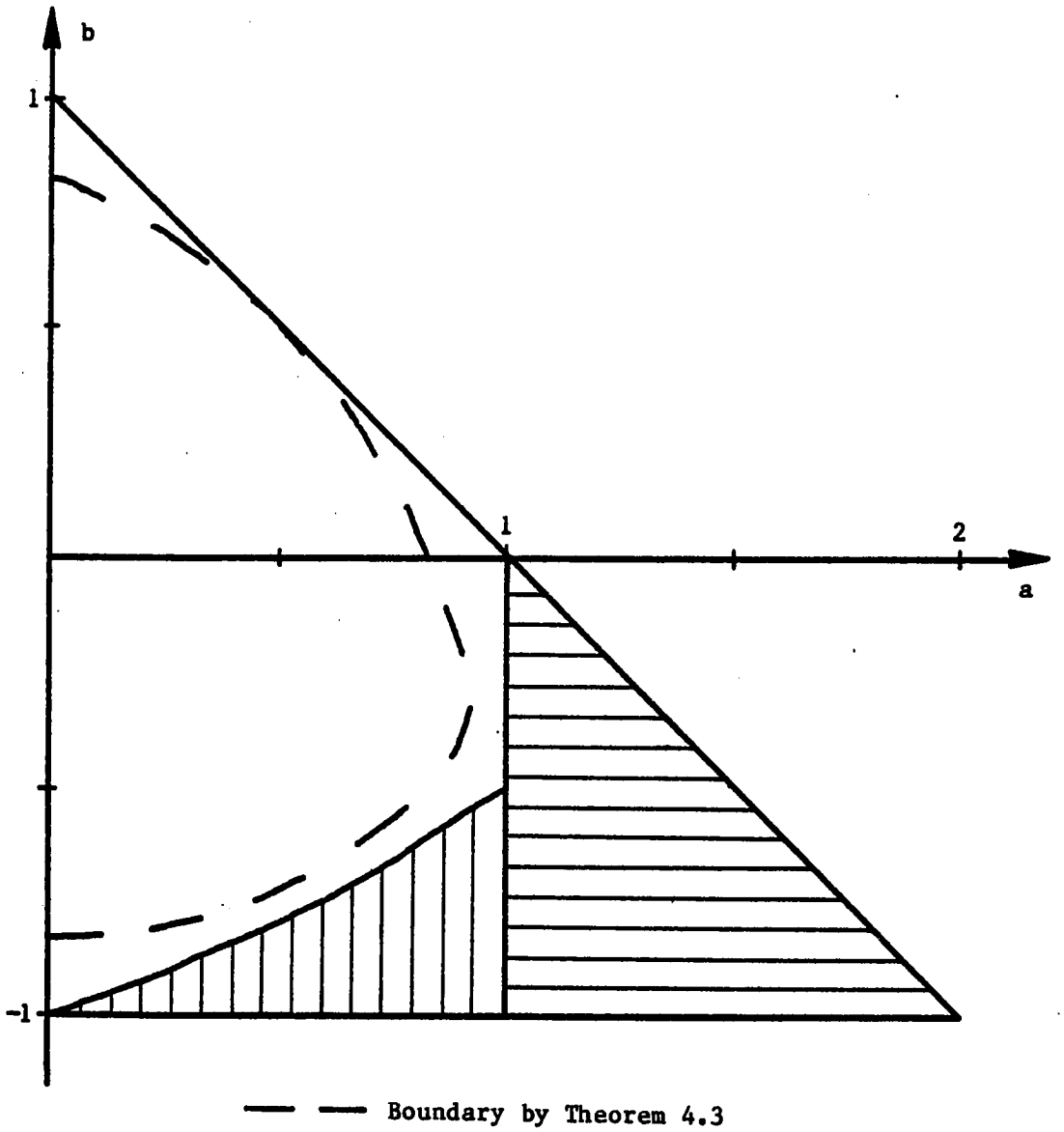


Figure 4.14. Region where a direct form filter with two truncation quantizers and saturation, zeroing or no overflow is g.a.s. by the constructive algorithm

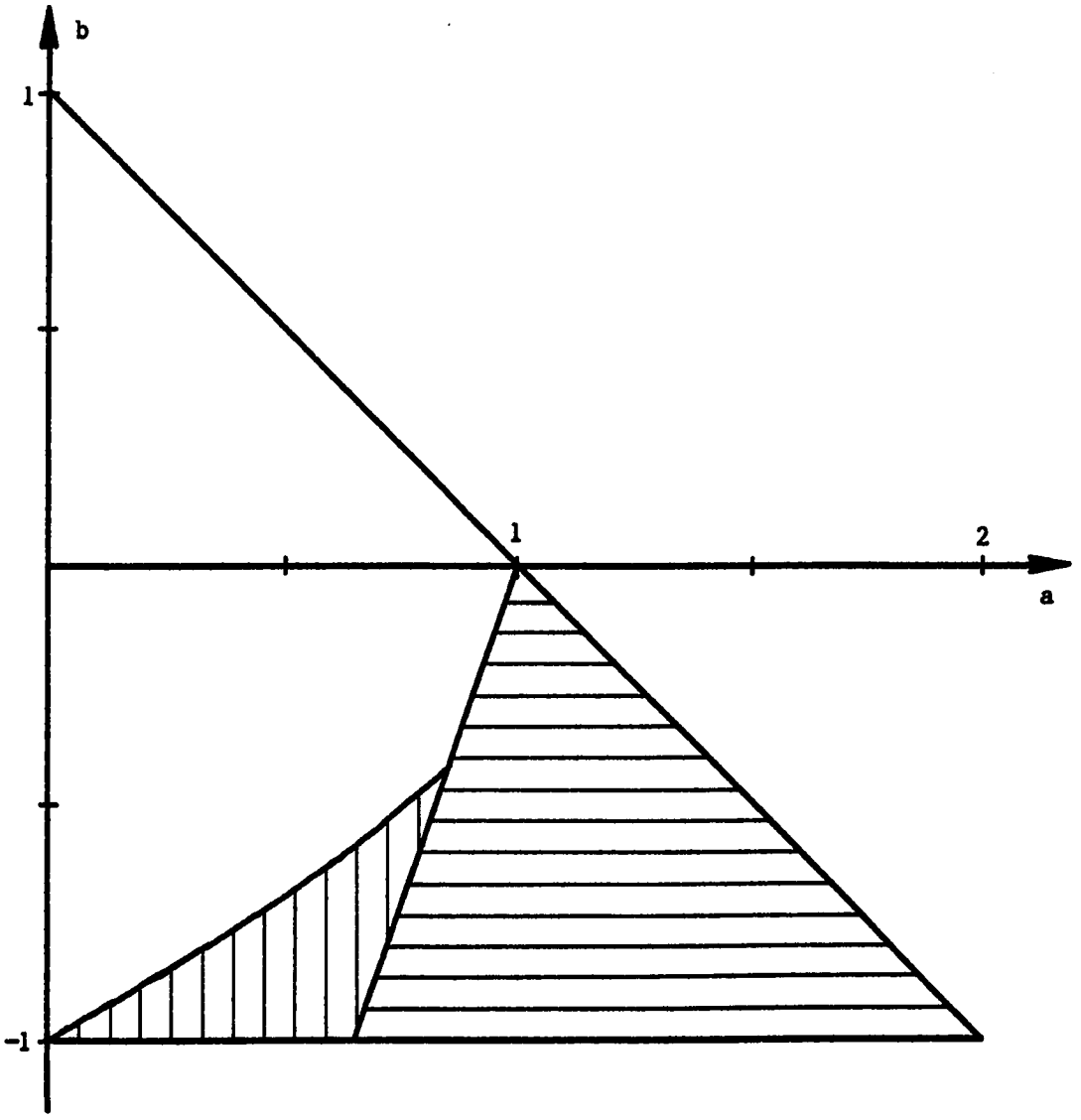


Figure 4.15. Region where a direct form filter with two truncation quantizers and triangular overflow is g.a.s. by the constructive algorithm

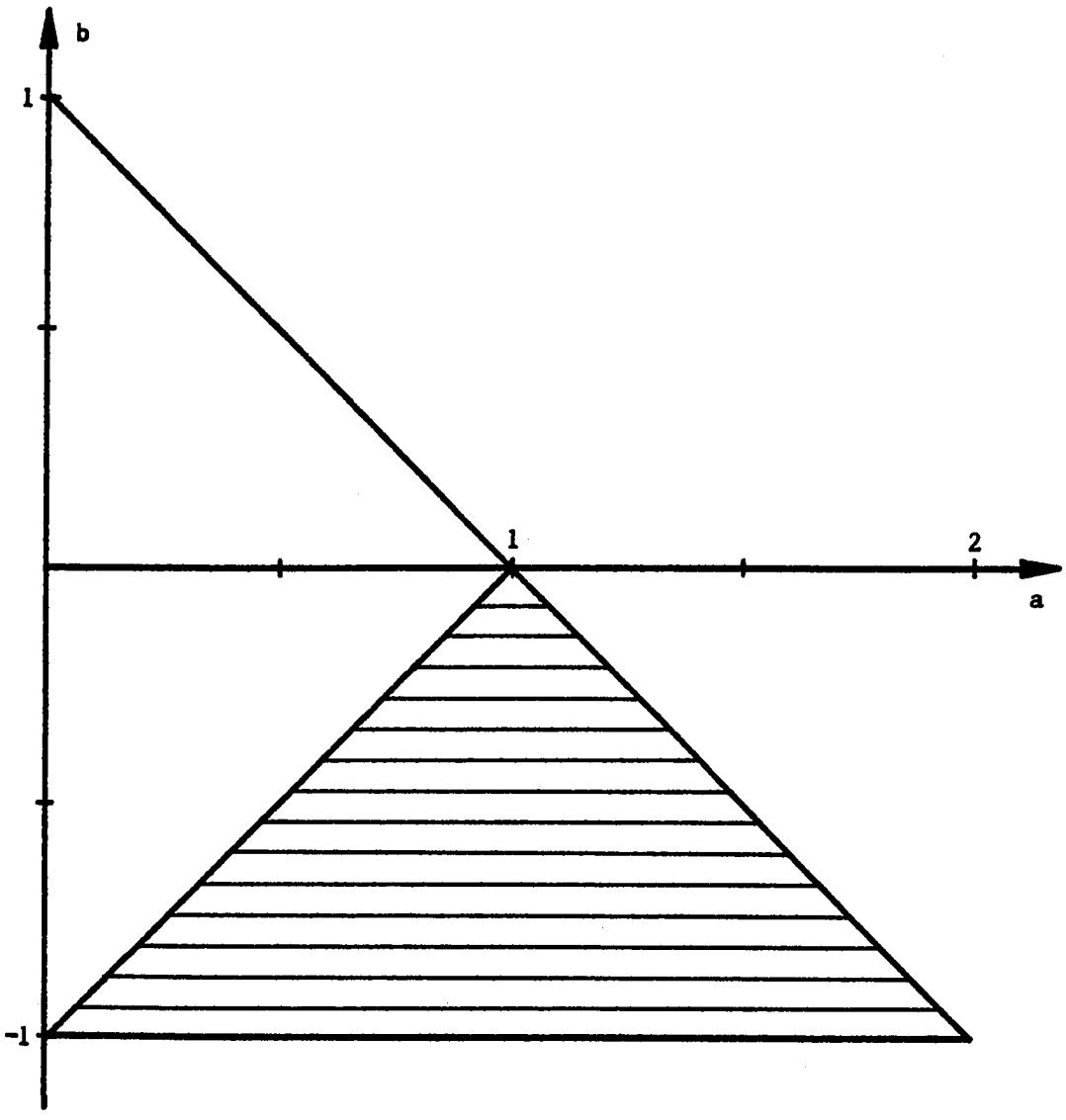


Figure 4.16. Region where a direct form filter with two truncation quantizers and two's complement overflow is g.a.s. by the constructive algorithm

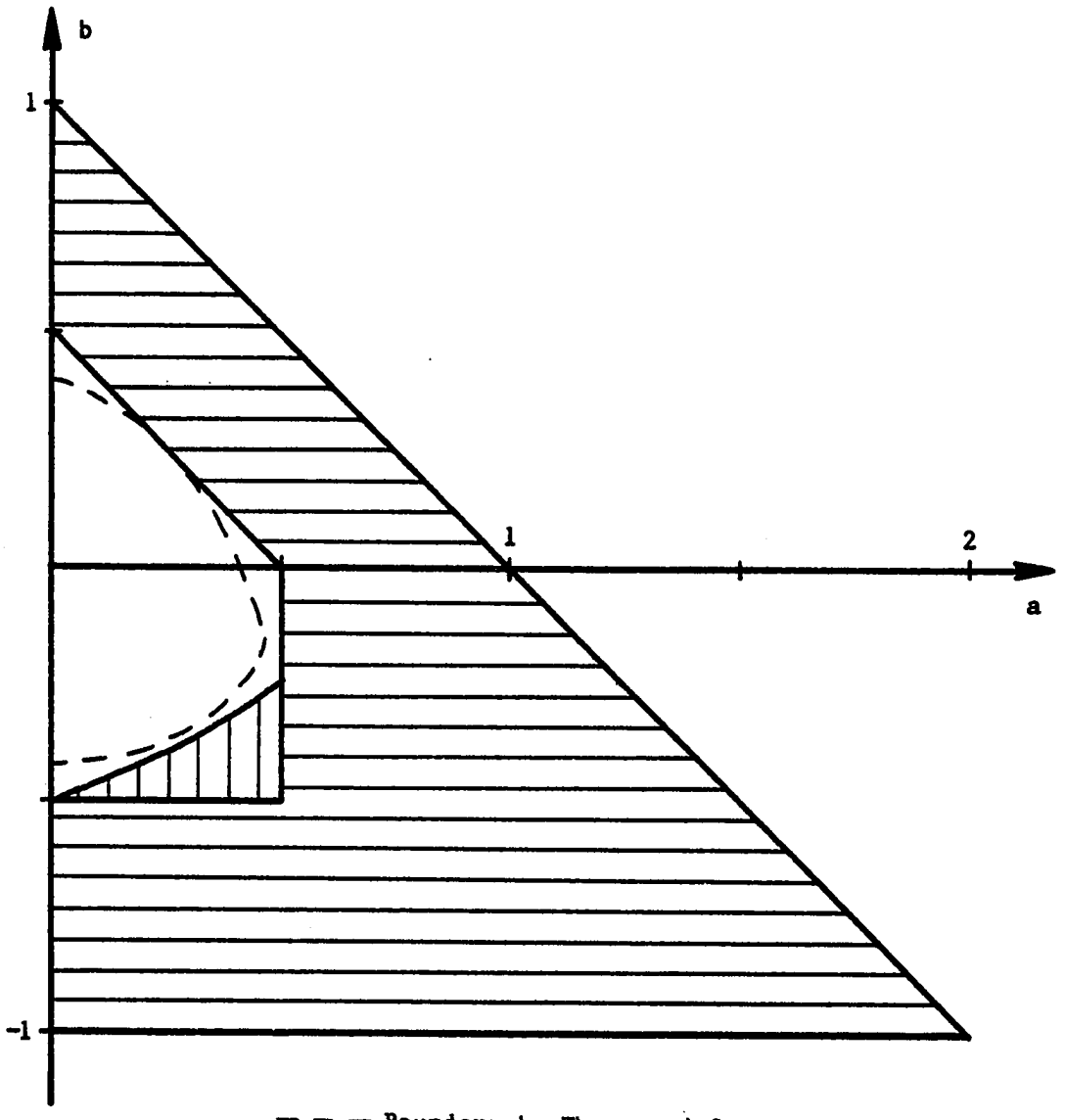


Figure 4.17. Region where a direct form filter with two roundoff quantizers and saturation, zeroing or no overflow is g.a.s. by the constructive algorithm

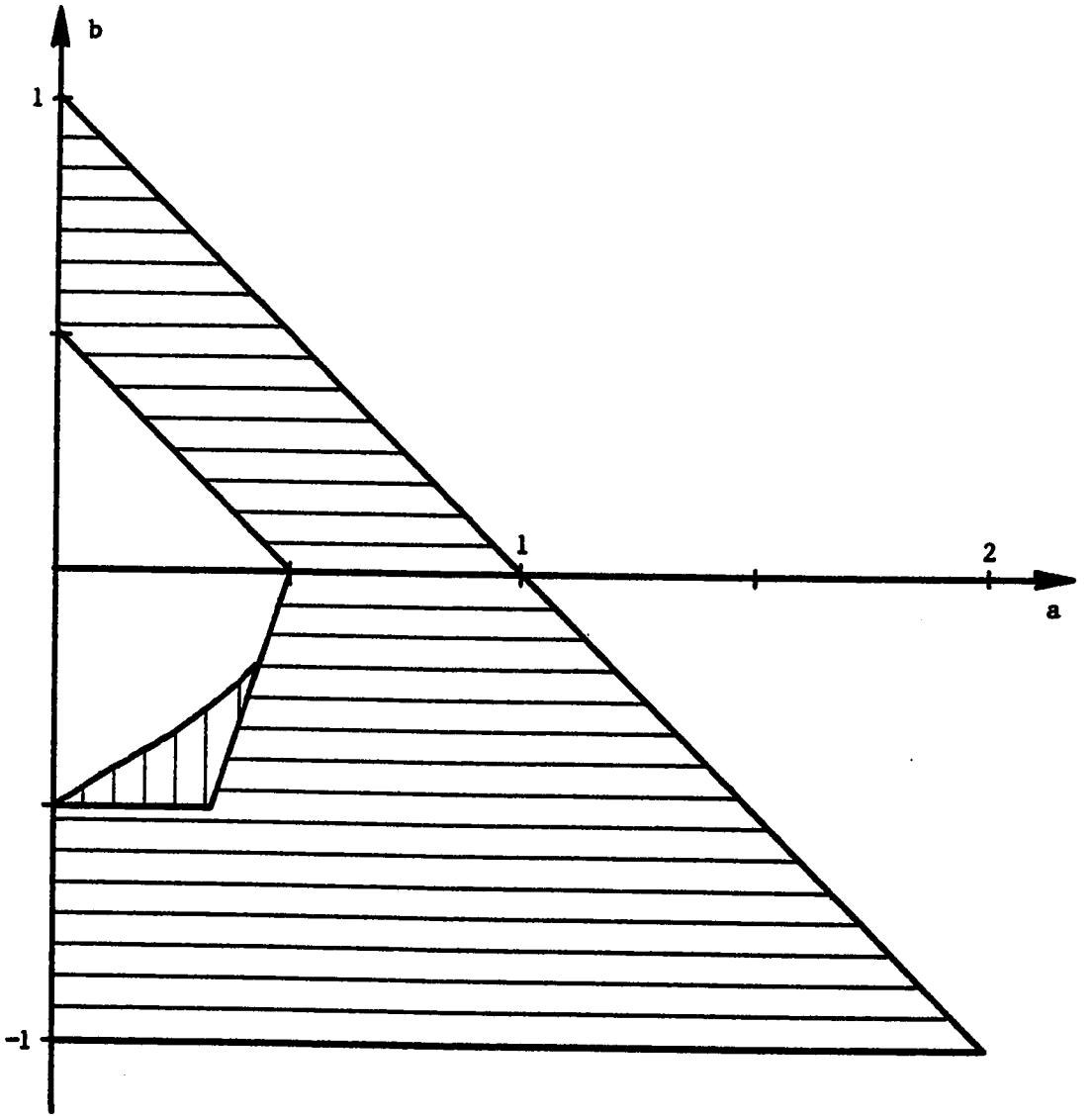


Figure 4.18. Region where a direct form filter with two roundoff quantizers and triangular overflow is g.a.s. by the constructive algorithm

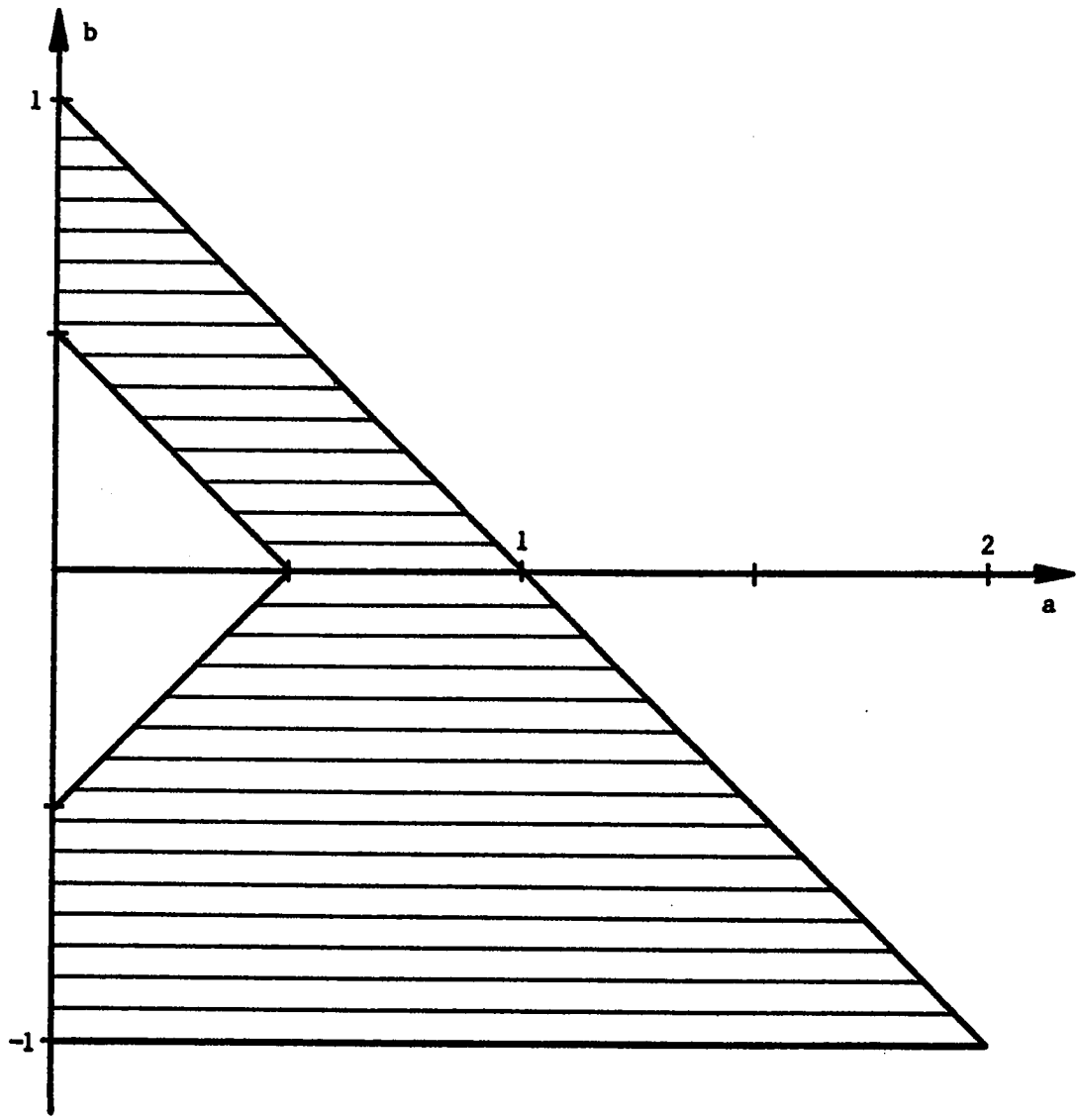


Figure 4.19. Region where a direct form filter with two roundoff quantizers and two's complement overflow is g.a.s. by the constructive algorithm

horizontally hatched region of Figure 4.14 [7]. For this filter with roundoff quantization and no overflow, others have found that limit cycles exist in most of the horizontally hatched region of Figure 4.17 [7]. All of the results obtained for the overflow nonlinearities seem to be new. It is interesting to note that the region in the parameter plane where the filter is globally asymptotically stable is the same for two's complement overflow with either one or two truncation quantizers.

B. Coupled Form Digital Filter

For the coupled form digital filter, we consider the fixed-point filter implemented with two or four quantizers. For both structures considered, we review existing results on the stability of the filter and then compare these results with the stability results obtained by the constructive algorithm.

1. Two quantizers

For the coupled form digital filter of Figure 3.9, previous results indicate that this structure is free of overflow and quantization limit cycles when truncation is used in the quantizer. Barnes and Fam [22] show that the coupled form is free of limit cycles due to overflow nonlinearities. They consider autonomous nonlinear systems of the type

$$x(k + 1) = f[Ax(k)] \qquad (4.13)$$

where $f(\cdot)$ is a bounded nonlinear function defined on R^n. Specifically, they assume the existence of a real number $\mu > 0$, such that for every $x \in R^n$

$$|f(x)|_2 \leq \mu |x|_2 \tag{4.14}$$

where $|\cdot|_2$ denotes the Euclidean vector norm on R^n. Let $\|A\|_2$ denote the matrix norm of A induced by the Euclidean norm. They show that if

$$\mu \|A\|_2 < 1 \tag{4.15}$$

then no autonomous limit cycles will exist in the system described by (4.13). The coupled form filter of Figure 3.9 without the quantizers fulfills condition (4.15) and thus no limit cycles exist. Jackson [23] extends these results to also include the quantization nonlinearity by noting that the truncation quantization nonlinearity also fulfills the condition (4.15).

For roundoff quantizers, Barnes and Shinnaka [24] prove that quantization limit cycles will not be supported by the coupled form for parameters located within the unit square depicted in Figure 4.20. They consider the second order linear filter of Figure 3.7 described by the state equations,

$$\begin{aligned} x_1(k+1) &= ax_1(k) - bx_2(k) \\ x_2(k+1) &= bx_1(k) + ax_2(k) \ . \end{aligned} \tag{4.16}$$

Letting f(•) denote roundoff quantization, the autonomous system with two roundoff quantizers is defined as

$$\begin{aligned} x_1(k+1) &= f[ax_1(k) - bx_2(k)] \\ x_2(k+1) &= f[bx_1(k) + ax_2(k)] \ . \end{aligned} \tag{4.17}$$

Their assertion and its proof are given here because we will extend it to the case when overflow nonlinearities are present.

Assertion 4.4 [24] For the system given by Equation (4.17), if the point (a,b) is within the unit square of Figure 4.20, then

$$|x(k+1)|_2 < |x(k)|_2 \ . \tag{4.18}$$

Proof: We consider the construction of embedded squares in Figure 4.21. If x(k) is on the boundary of square 1 in Figure 4.21, then x(k + 1) will be within or on the boundary of square 2. Furthermore, if x(k) is on a midpoint of a side of square 1, then x(k + 1) will be within or on the boundary of square 3. Thus, the desired result follows.

Since the norm is decreasing monotonically, they conclude that no limit cycles exist in the filter when (a,b) is within the unit square.

When overflow is considered with roundoff quantization, then the coupled form filter will not support limit cycles when the poles are within the unit square of Figure 4.20. This conclusion follows immediately from Assertion 4.4. The proof of this assertion is the

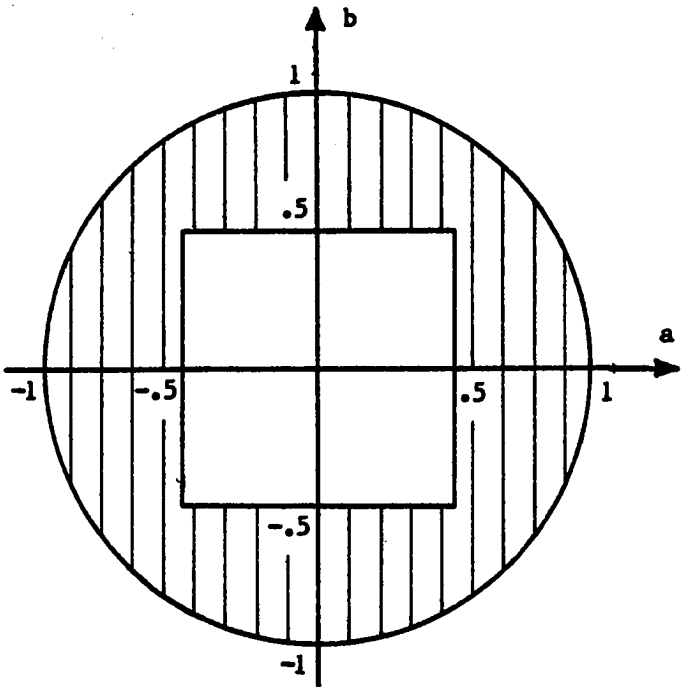


Figure 4.20. Region where a coupled form filter with two or four roundoff quantizers and any overflow is free of limit cycles by [24]

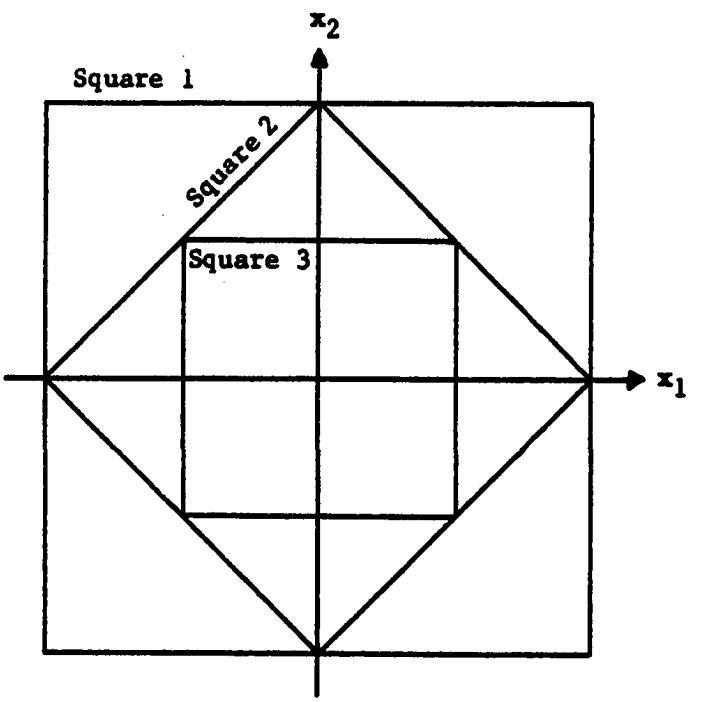


Figure 4.21. Imbedded squares in state space of coupled form filter

same if the operator f(•) in Equation (4.17) represents a roundoff and overflow, since

$$|P(x)| \leq |x| \tag{4.19}$$

where P(•) represents any of the overflow nonlinearities in Figure 3.2.

To apply the constructive algorithm to the coupled form digital filter with two quantizers, we use the extreme matrices determined in Equation (3.33). When truncation quantizers are used with any type of overflow, the constructive algorithm shows that this filter is globally asymptotically stable everywhere that the linear filter is globally asymptotically stable. This result is identical to existing results. For roundoff quantizers with any type of overflow, the constructive algorithm shows that this filter is globally asymptotically stable when the parameters a and b satisfy

$$a^2 + b^2 < 0.25 \ . \tag{4.20}$$

This region is a circle of radius 0.5 centered at the origin of the a-b parameter plane. However, this region is smaller than the region in the parameter plane where Barnes and Shinnaka [24] show that no limit cycles exist. It is not too surprising that our results are more conservative since we show that the filter is globally asymptotically stable for a <u>class</u> of nonlinearities whereas Barnes and Shinnaka consider some <u>specific</u> nonlinearities.

2. Four quantizers

For the coupled form digital filter with quantizers after each multiplication in Figure 3.8, the only known results deal with the absence of limit cycles in this structure without overflow. When truncation quantization is used without overflow, Jackson and Judell [25] have given a region where no limit cycles exist. They state that no limit cycles exist if the parameters of the coupled form structure of Figure 3.8 satisfy

$$a^2 + |ab| + b^2 < 1 \ . \tag{4.21}$$

However, no proof of their assertion is given in [25]. This region in the parameter plane where no limit cycles exist is shown as the unhatched region in Figure 4.22. The region is symmetric about the a and b axes. For roundoff quantizers, Barnes and Shinnaka [24] prove that limit cycles due to quantization will not exist if the parameters of the coupled form filter are within the unit square shown in Figure 4.20. They consider the linear filter of Figure 3.7 described by Equation (4.16). Letting f(•) denote roundoff, the autonomous system with four roundoff quantizers is defined as

$$\begin{aligned} x_1(k+1) &= f[ax_1(k)] + f[-bx_2(k)] \\ x_2(k+1) &= f[bx_1(k)] + f[ax_2(k)] \ . \end{aligned} \tag{4.22}$$

If the point (a,b) is within the unit square of Figure 4.20, then

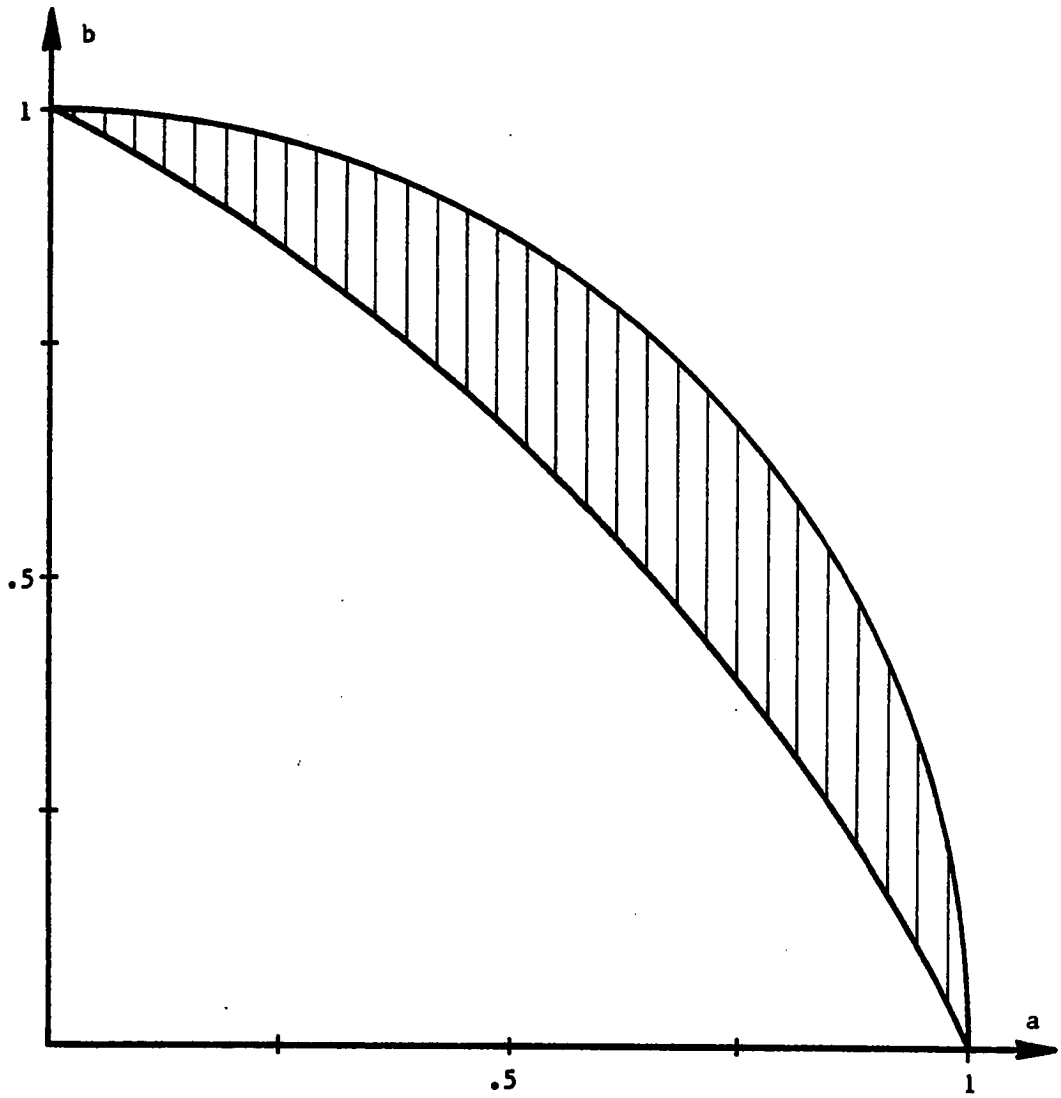


Figure 4.22. Region where a coupled form filter with four truncation quantizers and no overflow is free of limit cycles by Equation 4.21

$$|x(k+1)|_2 < |x(k)|_2 \qquad (4.23)$$

and thus no quantization limit cycles exist. The interested reader is referred to [24] for details of the proof. The extension of their proof to overflow nonlinearities does not seem obvious at this time, even though their proof could be extended in the coupled form filter with two roundoff quantizers.

To apply the constructive algorithm to this filter structure, we use the extreme matrices determined in Equation (3.41). The regions in the parameter plane where the digital filter is globally asymptotically stable by the constructive algorithm for all cases are shown as the unhatched regions in Figures 4.23 to 4.28. Horizontal hatching indicates the region where at least one extreme matrix has an eigenvalue with a magnitude greater than one. Vertical hatching indicates the rest of the region where we can draw no conclusion about the stability of the filter. Only the first quadrants of these regions are shown since they are symmetric about both the a and b axes.

As indicated in Figure 4.23, the constructive algorithm shows a region where limit cycles are absent that is larger than the region where Jackson and Judell [25] indicate the absence of limit cycles for four truncation quantizers and no overflow nonlinearities. However, with four roundoff quantizers, the constructive algorithm shows a region where no limit cycles exist that is smaller than the region where Barnes and Shinnaka [24] prove the absence of limit

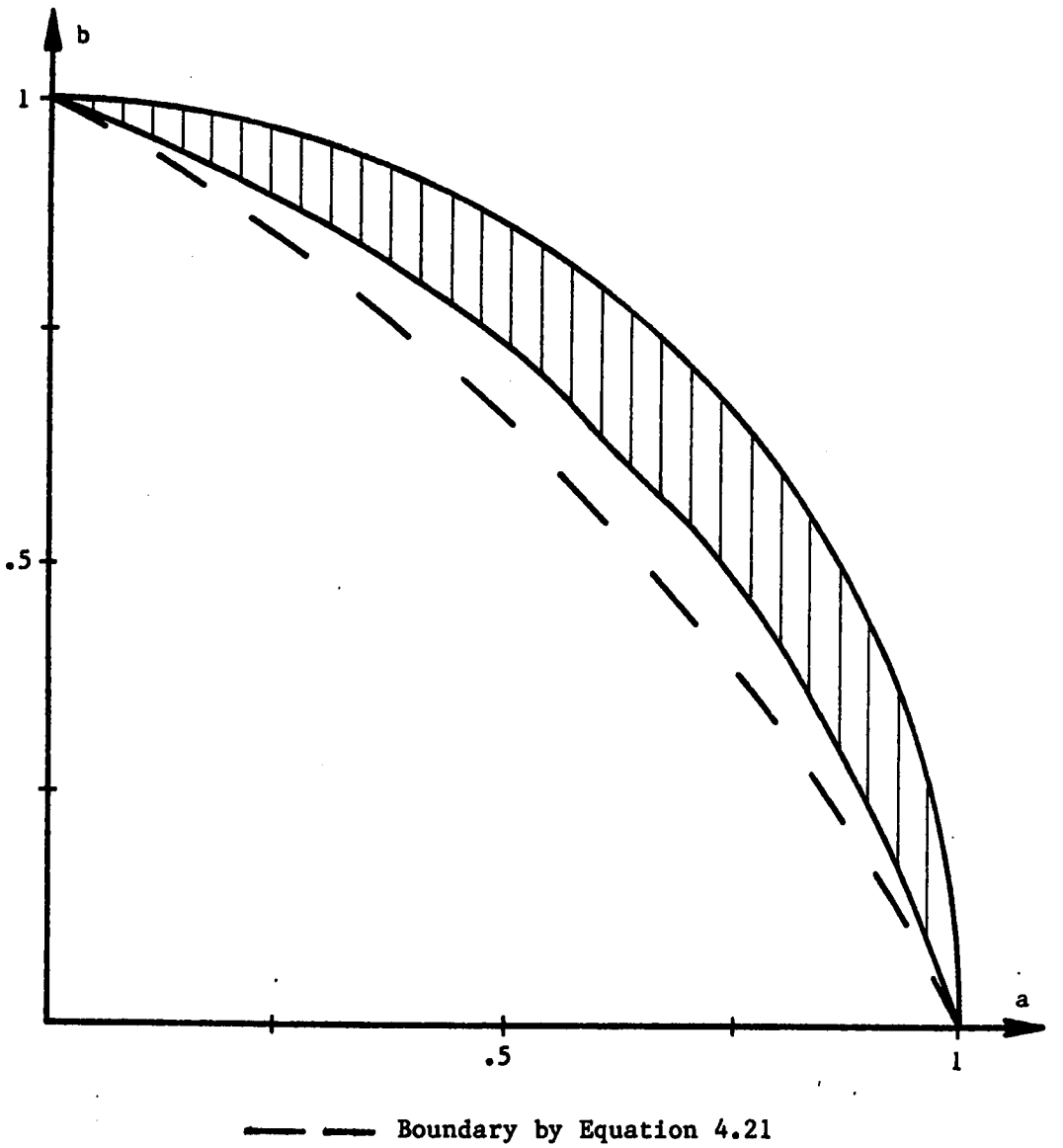


Figure 4.23. Region where a coupled form filter with four truncation quantizers and saturation, zeroing or no overflow is g.a.s. by the constructive algorithm

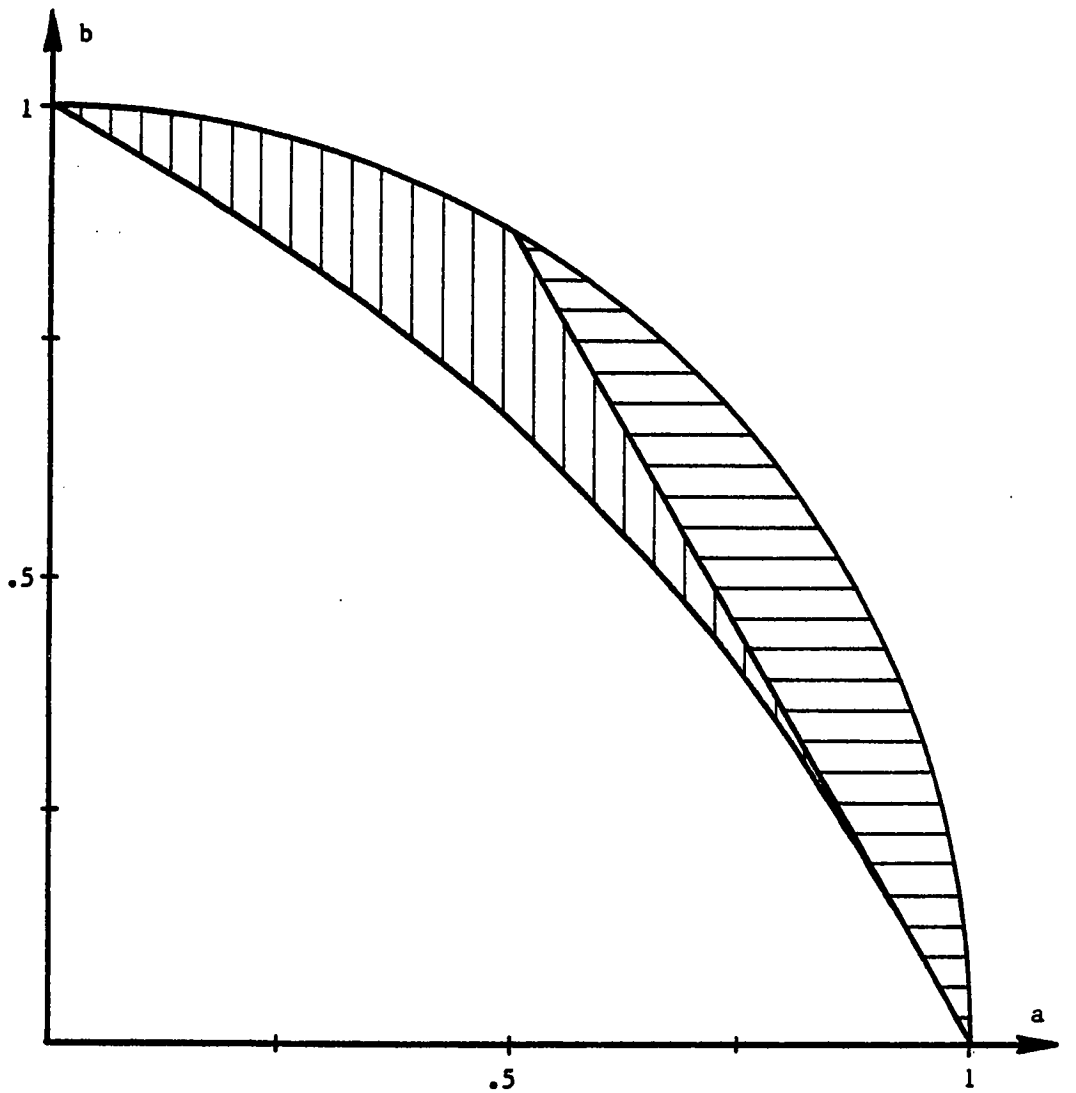


Figure 4.24. Region where a coupled form filter with four truncation quantizers and triangular overflow is g.a.s. by the constructive algorithm

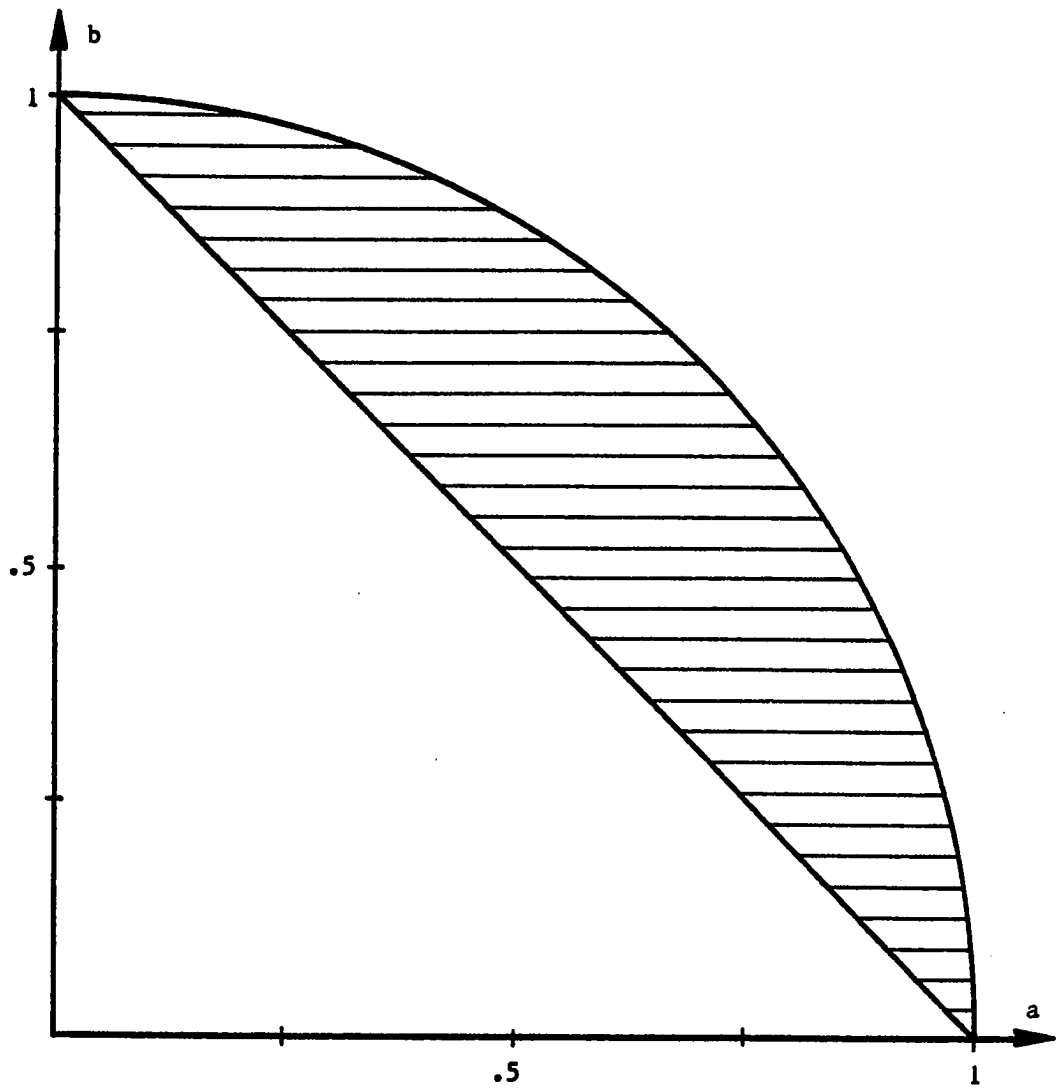


Figure 4.25. Region where a coupled form filter with four truncation quantizers and two's complement overflow is g.a.s. by the constructive algorithm

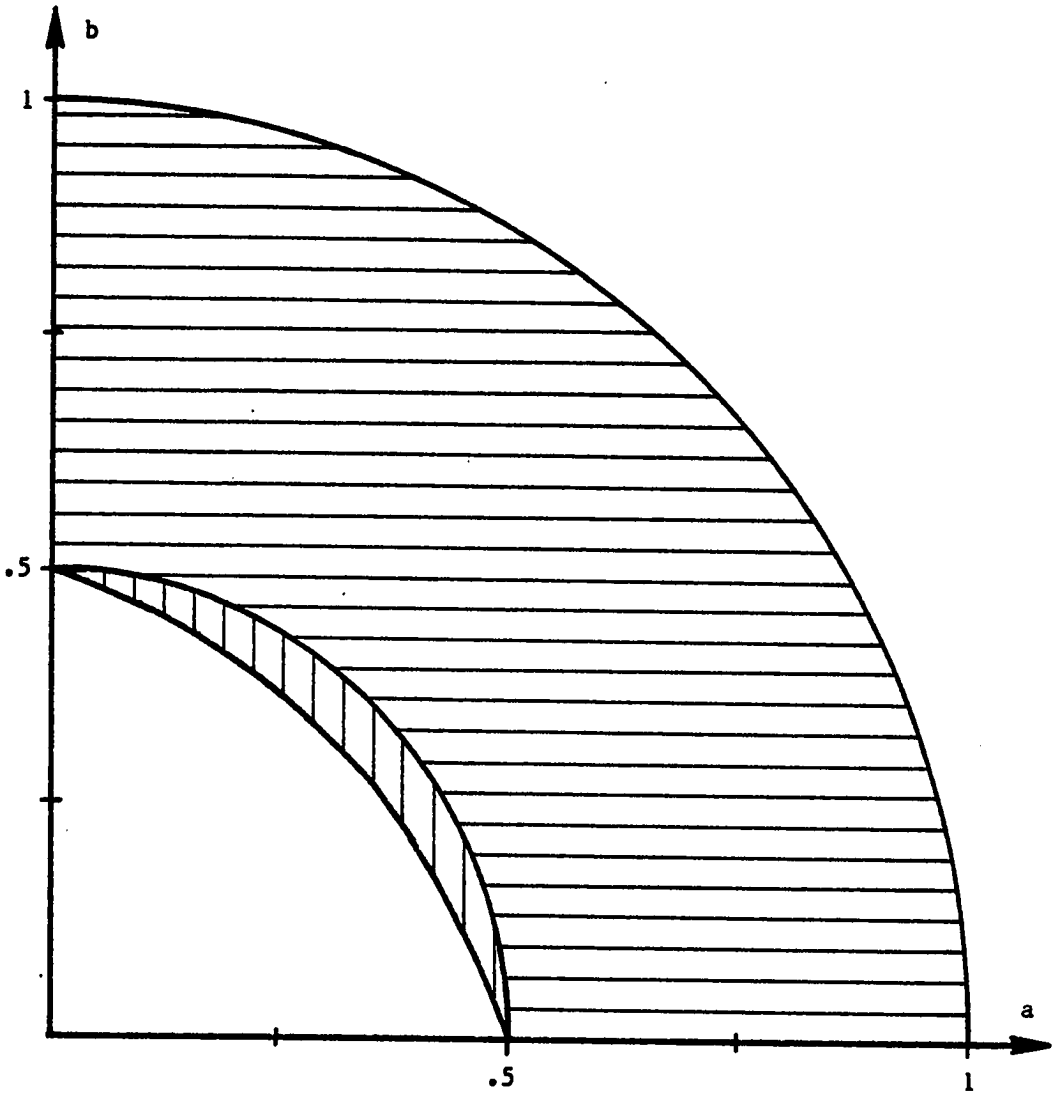


Figure 4.26. Region where a coupled form filter with four roundoff quantizers and saturation, zeroing or no overflow is g.a.s. by the constructive algorithm

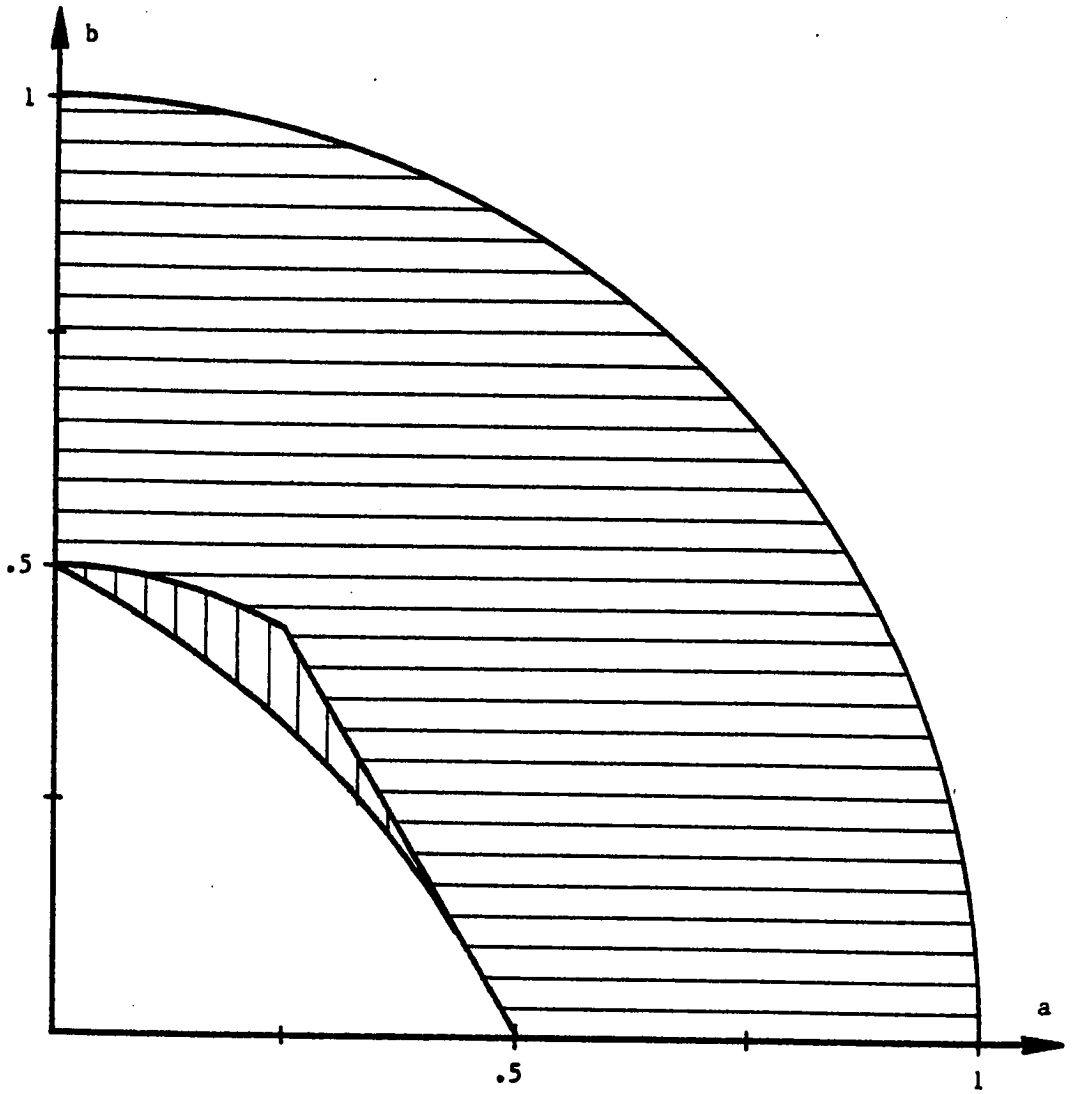


Figure 4.27. Region where a coupled form filter with four roundoff quantizers and triangular overflow is g.a.s. by the constructive algorithm

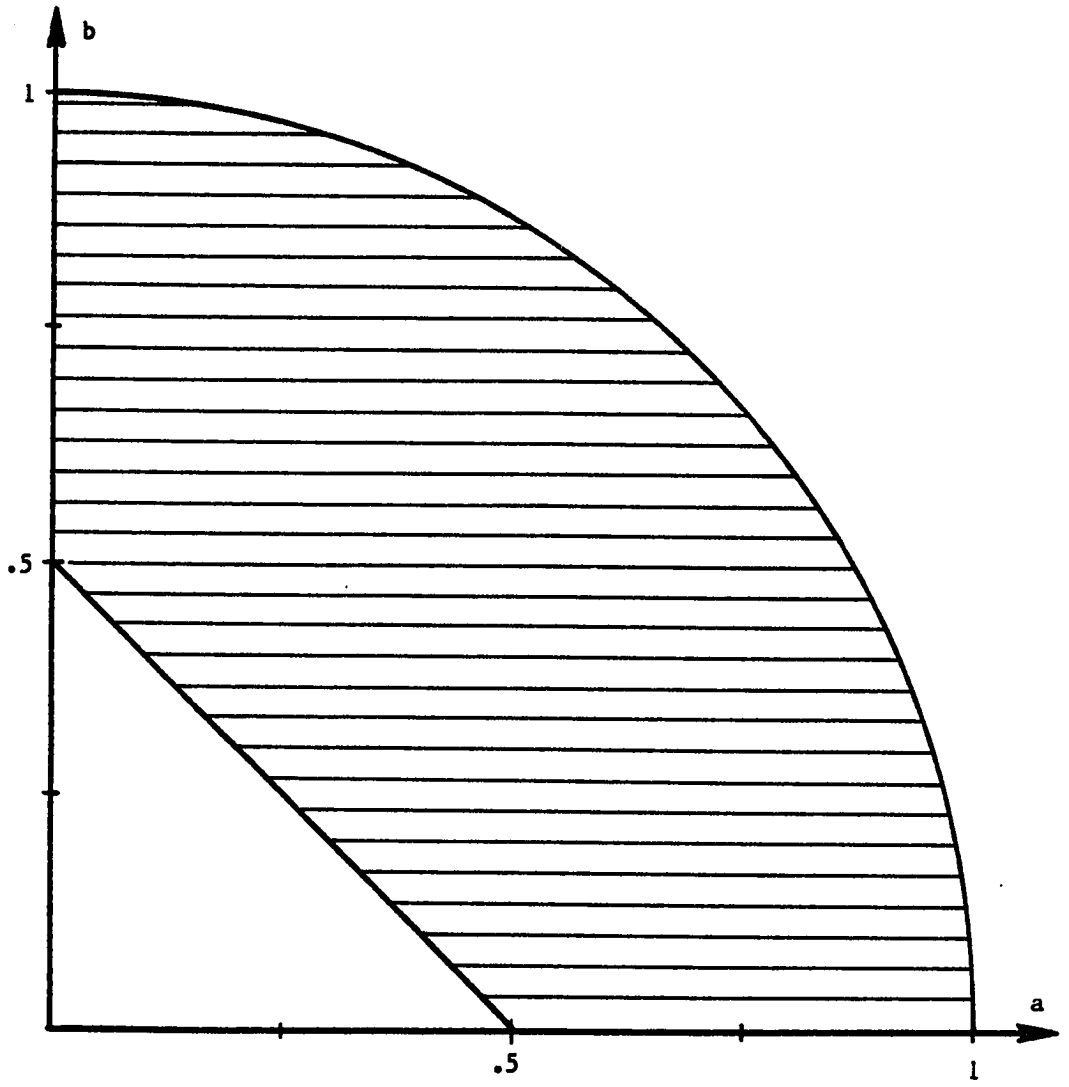


Figure 4.28. Region where a coupled form filter with four roundoff quantizers and two's complement overflow is g.a.s. by the constructive algorithm

cycles (Figure 4.20). However, our result by the constructive algorithm shows the region where the filter is globally asymptotically stable for a class of nonlinearities whereas only a specific nonlinearity, i.e., roundoff quantization, is considered in [24]. All of the results obtained by the constructive algorithm for saturation and two's complement overflow with roundoff or truncation quantization seem to be new results.

C. Wave Digital Filter

For the specific wave digital filter presented in Chapter III, we consider the filter implemented with two or three quantizers. The only known stability results apply to this wave digital filter implemented with two truncation quantizers. However, we can apply the Jury-Lee absolute stability criterion (Theorem 4.3) to some of the other cases to obtain stability results. These stability results are compared with the stability results obtained by the constructive algorithm.

1. Two quantizers

a. Truncation quantizers The only known stability analysis for wave digital filters has been done by Fettweis and Meerkötter [26], [27]. They use the concept of a stored pseudoenergy to show the complete stability of wave digital filters. (The precise definition of complete stability is given below.) The stored

pseudoenergy function (called a pseudopower function in [26]), plays the role of a Lyapunov function in their proof. The nonlinear arithmetic operations, i.e., overflow and quantization, are assumed to be applied to the signals b_i ($i = 3, 4, \cdots, n$) in Figure 3.10. These signals correspond to the states of the specific example that we consider in Figure 3.13.

Before we state the stability result of Fettweis and Meerkötter, we require their definition of complete stability. Consider a general wave digital filter such as the one of Figure 3.10. Arbitrary initial conditions are established in the filter at a certain initial time, t_o. The inputs to the filter are zero for all time greater than t_o, i.e., $a_1(t_o + k) = a_2(t_o + k) = 0$ for all $k > 0$. The wave digital filter is said to be <u>completely stable</u> if the signals $b_i(t_o + k)$, $i = 1, \cdots, n$ become permanently zero for a $k > 0$. Clearly, complete stability implies that the filter is free from any limit cycles.

The stability result of Fettweis and Meerkötter is stated without proof. The interested reader is referred to [26] and [27] for details.

<u>Theorem 4.5</u> The wave digital filter of Figure 3.10 is completely stable if:

1) the linear n-port network is pseudopassive [12],
2) the linear n-port network is free from any limit cycles under zero input conditions and

3) the nonlinearities $f_i(\cdot)$ at b_i ($i = 3, 4, \cdots, n$) satisfy

$$\begin{aligned} &|f_i(b_i)| \leq b_i \\ &|f_i(b_i)| = |b_i| \text{ implies } f_i(b_i) = b_i \ . \end{aligned} \tag{4.24}$$

This theorem applies to the specific wave digital filter we consider, since all linear wave digital filters derived from LC networks are pseudopassive and globally asymptotically stable [12]. The nonlinearities that will satisfy condition 3) of Theorem 4.5 are truncation quantizers with any of the overflow characteristics that we consider. Thus, Theorem 4.5 shows that the specific wave digital filter that we consider is free of limit cycles for any parameter values when truncation quantization with any overflow is applied at the states of the filter.

To apply the constructive algorithm to the wave digital filter structure with two quantizers (Figure 3.14), we use the extreme matrices determined by Equation (3.56). The constructive algorithm shows that the filter is globally asymptotically stable for truncation quantization with any overflow for parameters a and b satisfying

$$\begin{aligned} &0 < a \leq 100 \\ &0 < b \leq 100 \ . \end{aligned} \tag{4.25}$$

We did not try larger values of the parameters a and b because the run time of the computer program increased significantly as a and b

are increased in value. However, this region does cover any reasonable values of these parameters, since the larger values of a and b mean that the sampling frequency of the filter is fairly high compared to the cutoff frequency of the filter. Our result shows that the filter is free of limit cycles for any of the parameters in the region defined by (4.25) and thus agrees with existing results.

b. Roundoff quantizers There do not seem to be any existing results for the stability of wave digital filters when roundoff quantization is used at the states, however, the absolute stability criterion of Jury and Lee (Theorem 4.3) can be applied to this case. Applying Theorem 4.3 to the wave digital filter with two quantizers as shown in Figure 3.14, the matrix G(z) may be written as

$$G(z) = \begin{bmatrix} -c_{11} z^{-1} & -c_{12} z^{-1} \\ -c_{21} z^{-1} & -c_{22} z^{-1} \end{bmatrix} \qquad (4.26)$$

where c_{11}, c_{12}, c_{21} and c_{22} are determined by Equations (3.43) and (3.50). The matrix H(z), given by

$$H(z) = \begin{bmatrix} \frac{2}{k_{11}} - c_{11}(z + z^{-1}) & -c_{12} z^{-1} - c_{21} z \\ -c_{21} z^{-1} - c_{12} z & \frac{2}{k_{22}} - c_{22}(z + z^{-1}) \end{bmatrix} \qquad (4.27)$$

must be positive definite for $|z| = 1$. For two roundoff quantizers, $k_{11} = k_{22} = 2$. The region in the parameter plane where the filter is

globally asymptotically stable is presented as the unhatched region in Figure 4.29.

To apply the constructive algorithm to the wave digital filter structure with two quantizers (Figure 3.14), we use the extreme matrices determined by Equation (3.56). The regions in the parameter plane where the digital filter is globally asymptotically stable by the constructive algorithm for all cases are shown in Figures 4.30 and 4.32. Horizontal hatching indicates the region where at least one extreme matrix has an eigenvalue with a magnitude greater than one. Although only a portion of the first quadrant is shown, this horizontally hatched region extends to at least $a = b = 100$, which is the most extensive region we examined. Vertical hatching indicates the rest of the region where we can make no conclusion about the stability of the filter.

As can be seen from Figure 4.30, the constructive algorithm yields a less conservative result than the application of Theorem 4.3. All of the results obtained for the roundoff quantization in conjunction with overflow seem to be new results.

2. Three quantizers

To apply the constructive stability algorithm to the wave digital filter structure with three quantizers (Figure 3.15), we used the extreme matrices defined in Equation (3.66). For truncation or roundoff quantization, the constructive algorithm failed to find any

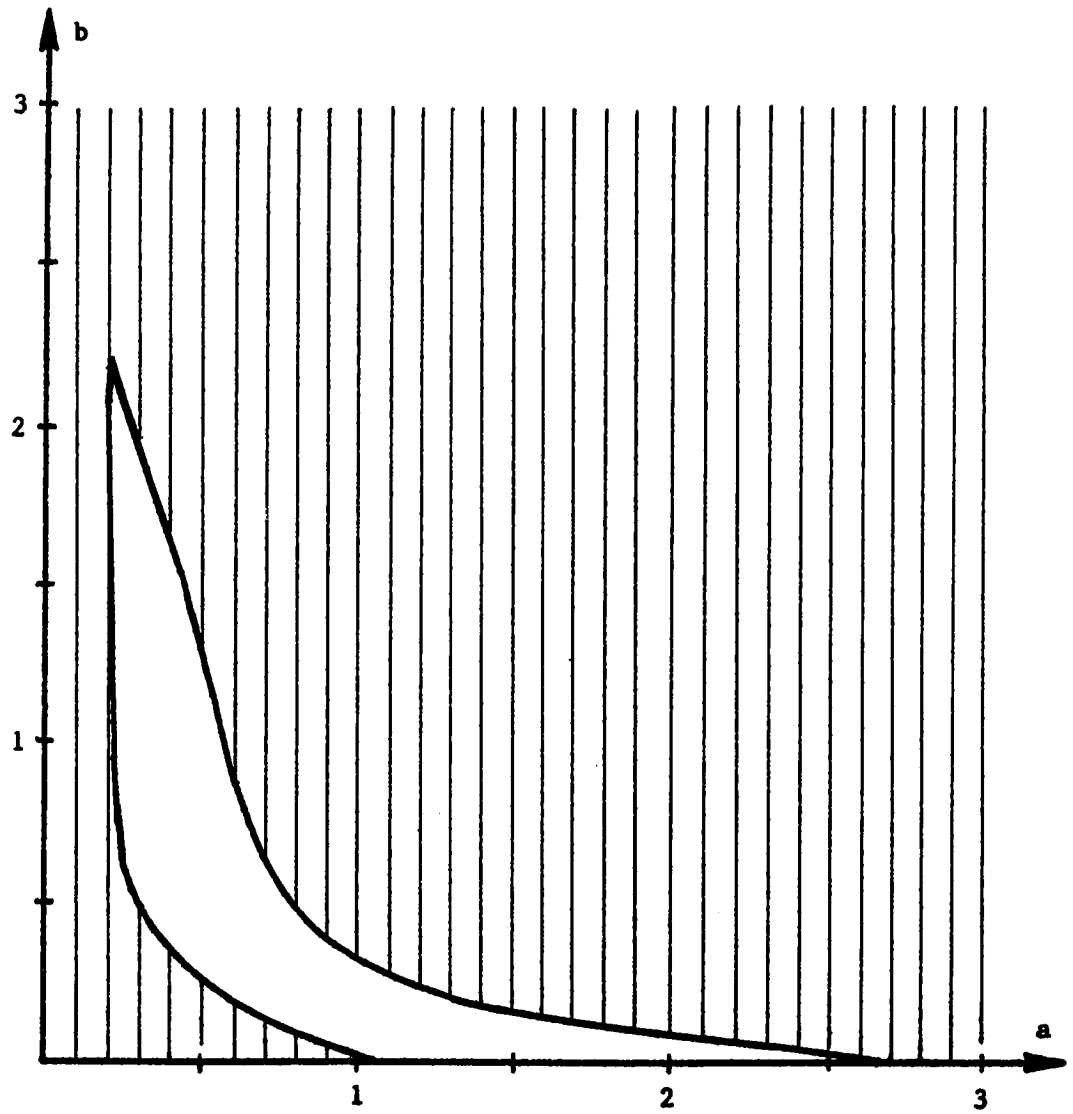


Figure 4.29. Region where the specific wave filter with two roundoff quantizers and no overflow is g.a.s. by Theorem 4.3

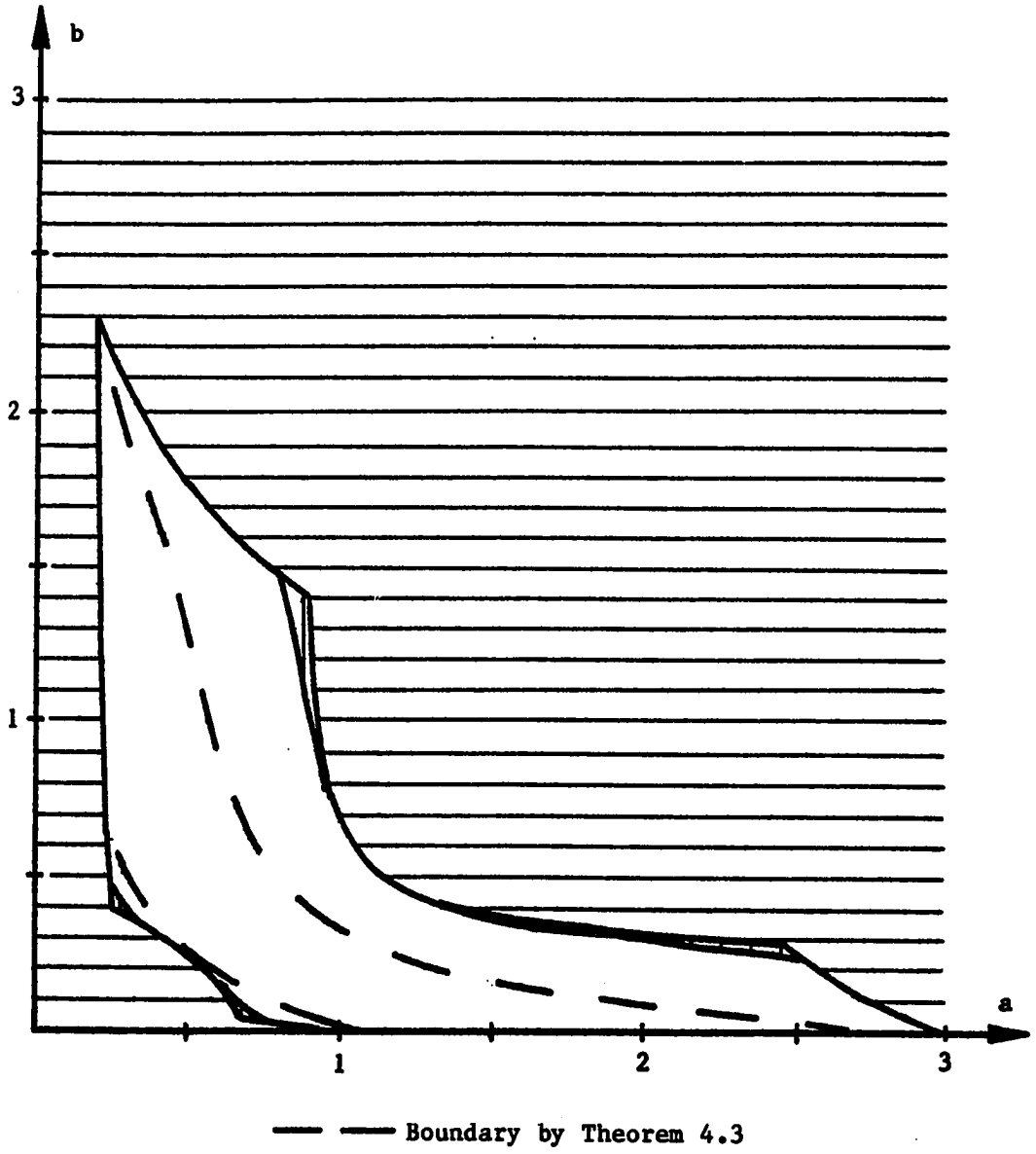


Figure 4.30. Region where the specific wave filter with two roundoff quantizers and saturation, zeroing or no overflow is g.a.s. by the constructive algorithm

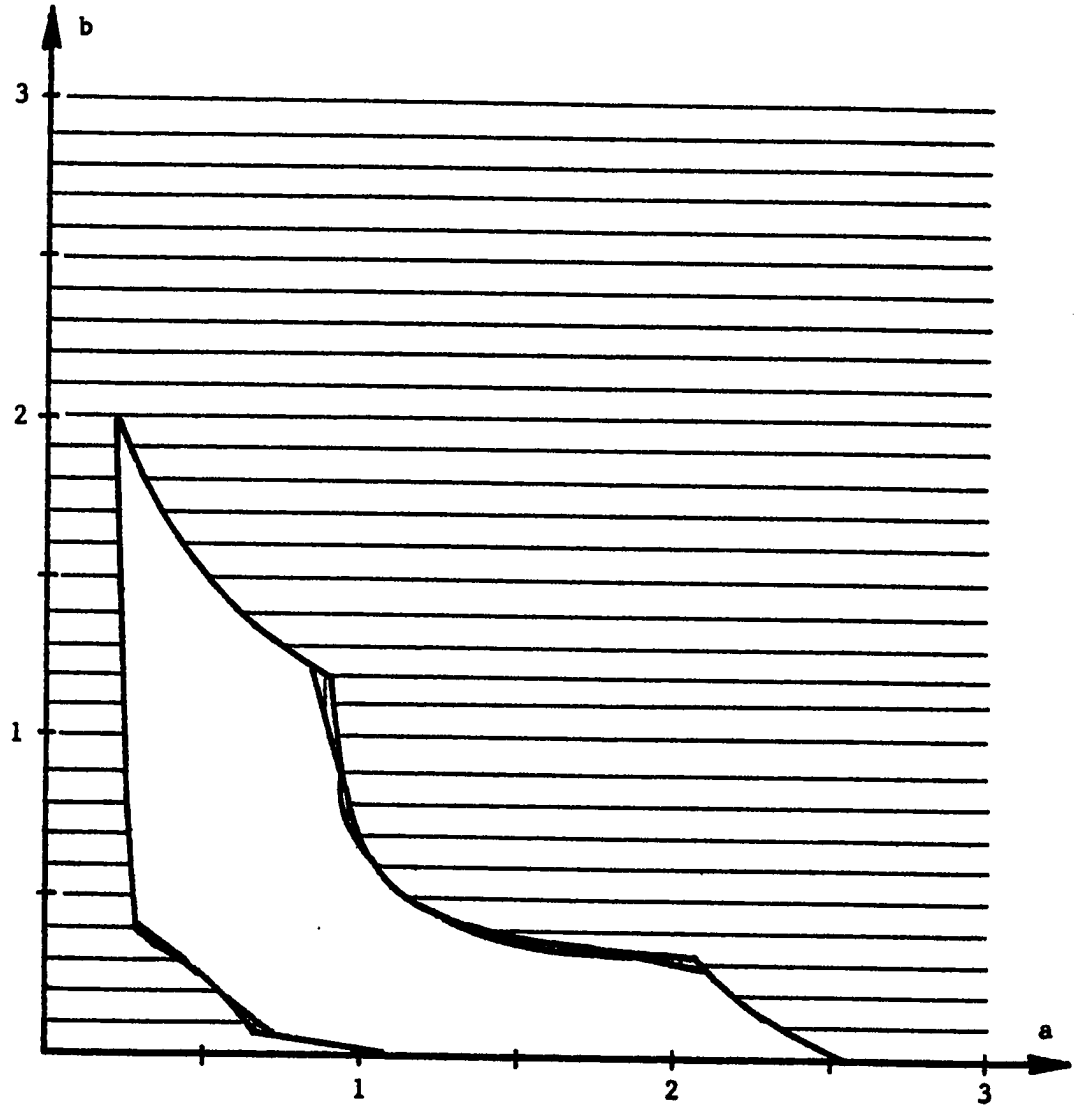


Figure 4.31. Region where the specific wave filter with two roundoff quantizers and triangular overflow is g.a.s. by the constructive algorithm

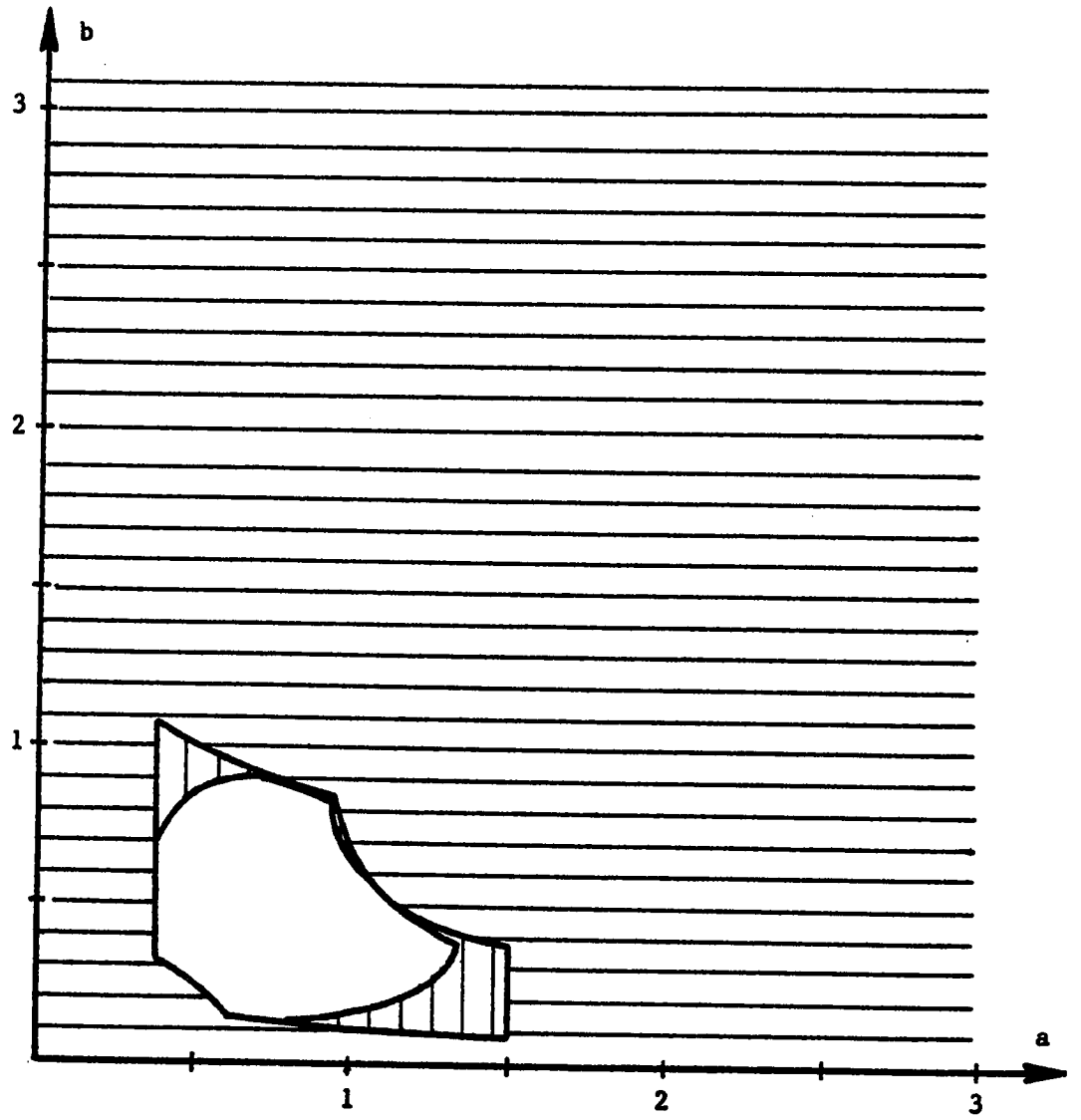


Figure 4.32. Region where the specific wave filter with two roundoff quantizers and two's complement overflow is g.a.s. by the constructive algorithm

region in the parameter plane where the filter is globally asymptotically stable. Although no details are given, application of the Jury-Lee criterion (Theorem 4.3) also failed to find any region in the parameter plane for $a > 0$ and $b > 0$ where the filter is globally asymptotically stable.

D. Lattice Digital Filter

For the second order lattice digital filter, we consider the filter implemented with two or three quantizers. The only known stability results apply to the lattice filter with two truncation quantizers. However, application of the Jury-Lee critierion (Theorem 4.3) does obtain some stability results that we use for comparison. These existing results are then compared with the stability results obtained by the constructive algorithm.

1. Two quantizers

a. Truncation quantizers Gray [28] uses energy analogies to show the absence of limit cycles in nonlinear lattice digital filters. The approach is similar to that of Fettweis and Meerkötter [26]. Gray shows that the nonlinear lattice digital filter will be free of limit cycles whenever the linear filter is globally asymptotically stable if truncation quantization and overflow nonlinearities are applied at each section output (i.e., at A_m and B_{m+1} of Figure 3.15(b)). This result applies to any of the overflow

characteristics that we are considering (Figure 3.2). When applied to a second order lattice filter, this result shows the absence of limit cycles even if truncation quantization and overflow are applied only at the states of Figure 3.17.

To apply the constructive algorithm to the stability analysis of the lattice structure with two quantizers, we use the extreme matrices determined by Equation (3.73). With truncation quantization and any overflow nonlinearity, the constructive algorithm shows that this nonlinear lattice filter is globally asymptotically stable in the region of the k_1-k_2 parameter plane determined by

$$\begin{aligned} |k_1| &< 1 \\ |k_2| &< 1 \ . \end{aligned} \tag{4.28}$$

This result agrees with the result of Gray [28].

b. Roundoff quantizers There do not appear to be any stability results for the second order lattice digital filter implemented with two roundoff quantizers (Figure 3.17). Although no details are given here, application of the Jury-Lee criterion yields no region in the parameter plane where this filter is globally asymptotically stable. To apply the constructive algorithm to the lattice structure with two roundoff quantizers, we use the extreme matrices determined by Equation (3.73). The regions in the k_1-k_2 parameter plane where this filter is globally asymptotically stable by the constructive algorithm are shown as the unhatched

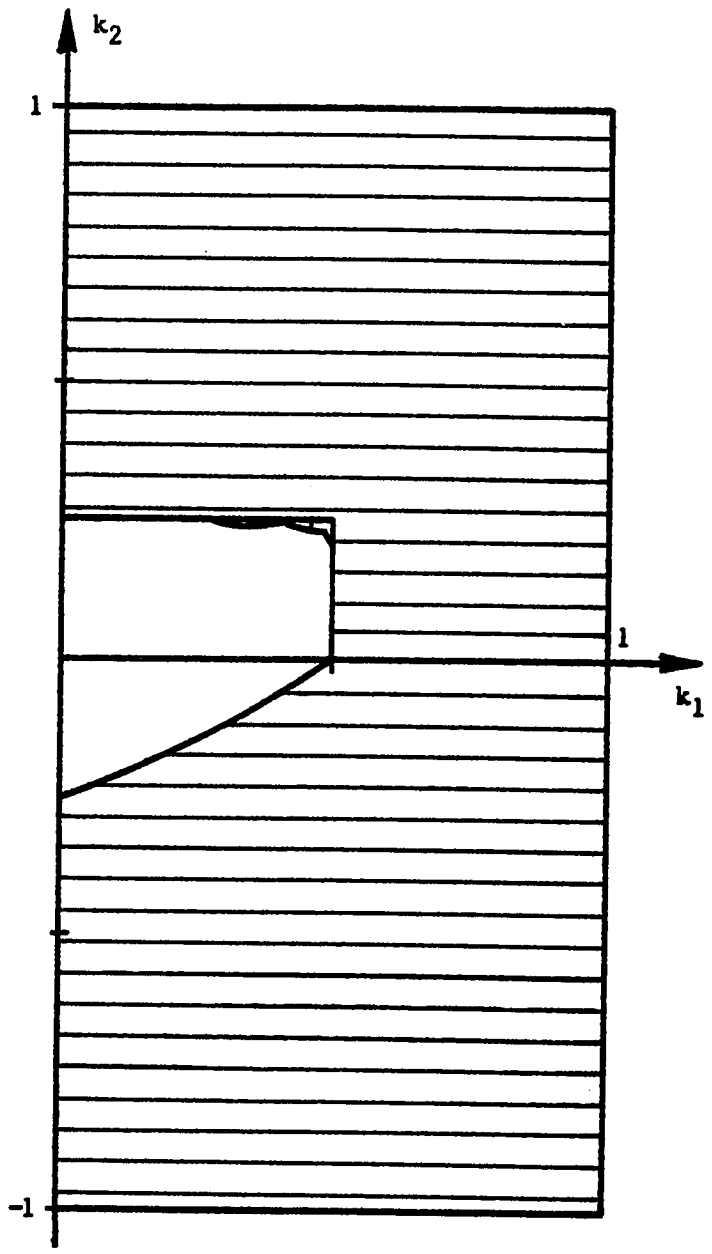


Figure 4.33. Region where the lattice filter with two roundoff quantizers and saturation, zeroing or no overflow is g.a.s. by the constructive algorithm

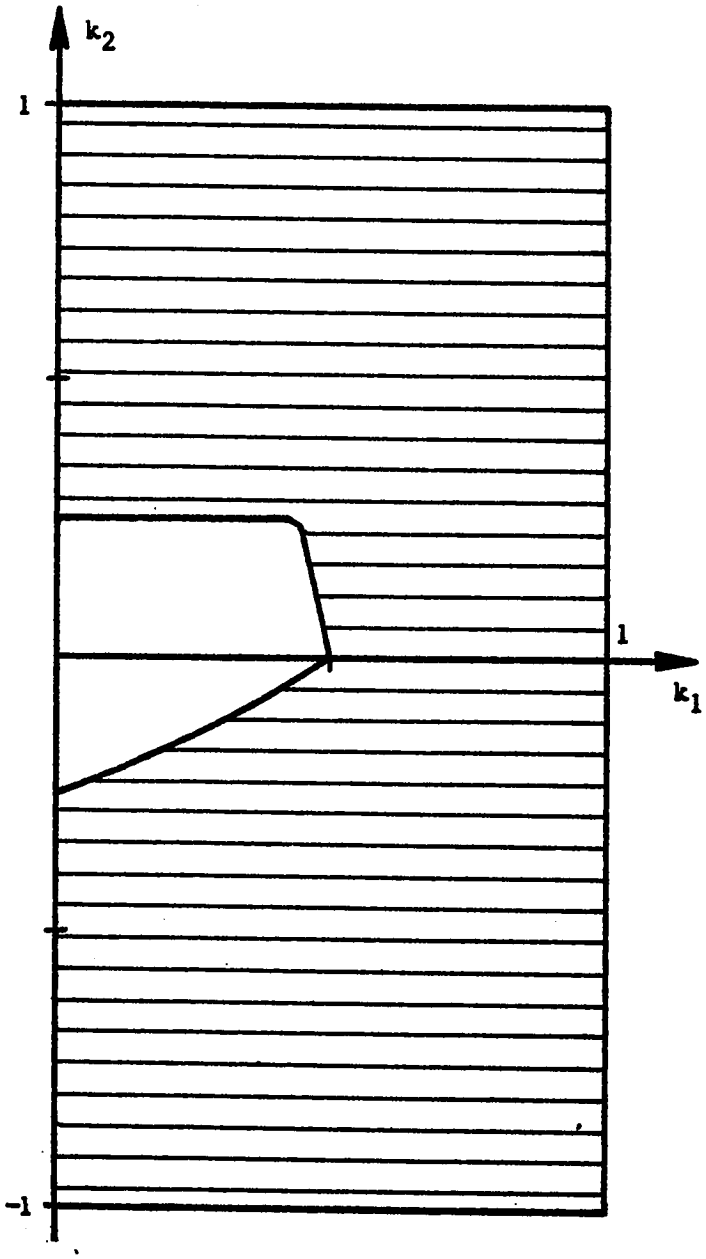


Figure 4.34. Region where the lattice filter with two roundoff quantizers and triangular overflow is g.a.s. by the constructive algorithm

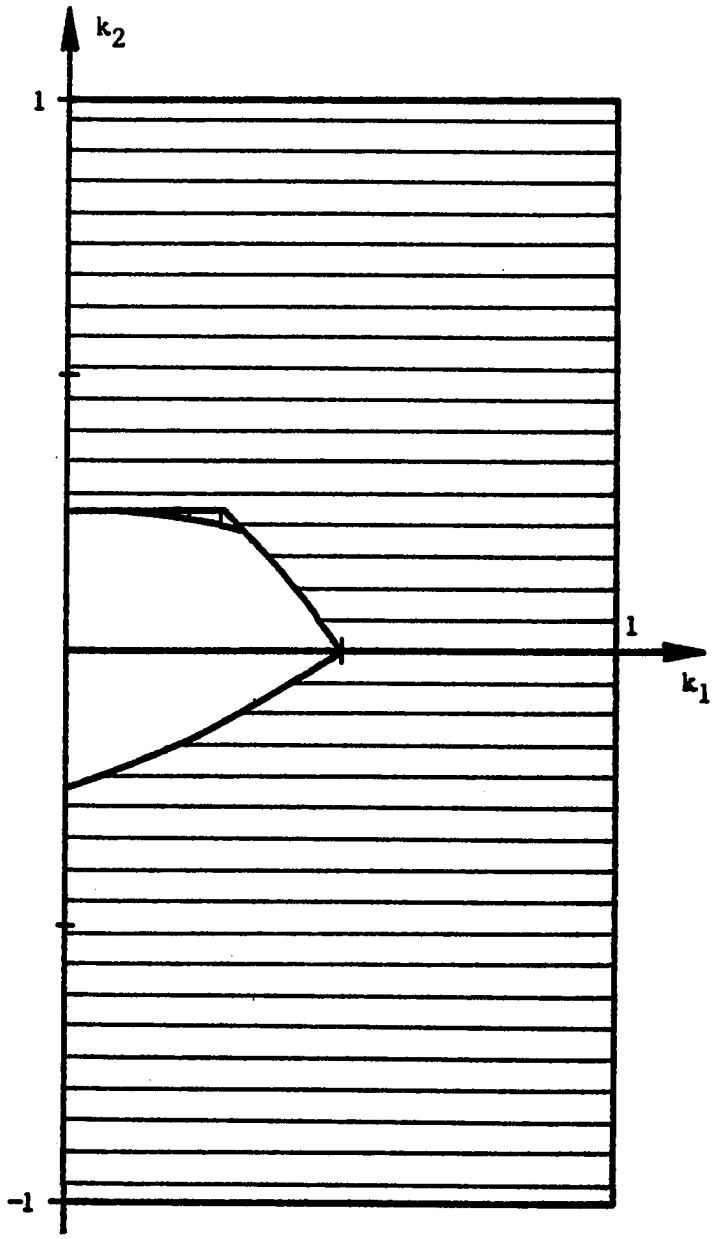


Figure 4.35. Region where the lattice filter with two roundoff quantizers and two's complement overflow is g.a.s. by the constructive algorithm

regions in Figures 4.33 to 4.35. The horizontally hatched region indicates the region where at least one extreme matrix has an eigenvalue with a magnitude that is greater than one. Vertical hatching indicates the rest of the region where the constructive algorithm does not show global asymptotic stability. These regions are symmetric about the k_2 axis. These results appear to be new.

2. Three quantizers

There do not seem to be any existing stability results for the lattice digital filter with three quantizers (Figure 3.18). However, the absolute stability criterion of Jury and Lee (Theorem 4.3) can be applied to this structure without the overflow nonlinearities. Applying Theorem 4.3 to the lattice digital filter with three quantizers as shown in Figure 3.18, the matrix G(z) may be written as

$$G(z) = \begin{bmatrix} 0 & k_1 & k_1 \\ 0 & k_1 z^{-1} & k_1 z^{-1} \\ -k_2 z^{-1} & k_2 z^{-2} & k_2 z^{-2} \end{bmatrix}. \qquad (4.29)$$

The matrix H(z), given by

$$H(z) = \begin{bmatrix} \frac{2}{k_{11}} & k_1 & k_1 - k_2 z \\ k_1 & \frac{2}{k_{22}} + k_1(z + z^{-1}) & k_1 z^{-1} + k_2 z^2 \\ k_1 - k_2 z^{-1} & k_1 z + k_2 z^{-2} & \frac{2}{k_{33}} + k_2(z^2 + z^{-2}) \end{bmatrix} \qquad (4.30)$$

must be positive definite for $|z| = 1$. For three truncation

quantizers, $k_{11} = k_{22} = k_{33} = 1$. The region in the parameter plane where the filter is globally asymptotically stable is shown as the unhatched region in Figure 4.36. For three roundoff quantizers, $k_{11} = k_{22} = k_{33} = 2$. The region where the filter is globally asymptotically stable for this case is presented as the unhatched region in Figure 4.37. These regions are symmetric about the k_2 axis.

To apply the constructive algorithm to the lattice structure with three quantizers (Figure 3.18), we use the extreme matrices determined in Equation (3.81). The regions in the parameter plane where the filter is globally asymptotically stable for all cases are presented in Figures 4.38 to 4.45. Only half of these regions are shown since they are symmetric about the k_2 axis. The horizontally hatched regions are those regions where at least one extreme matrix has an eigenvalue whose magnitude is greater than one. Vertical hatching indicates the rest of the region where the constructive algorithm can draw no conclusion as to the stability of the filter.

As can be seen in Figures 4.38 and 4.42, the constructive algorithm obtains a less conservative result than the application of the Jury-Lee criterion. All of the stability results obtained by the constructive algorithm for the overflow nonlinearities seem to be new results.

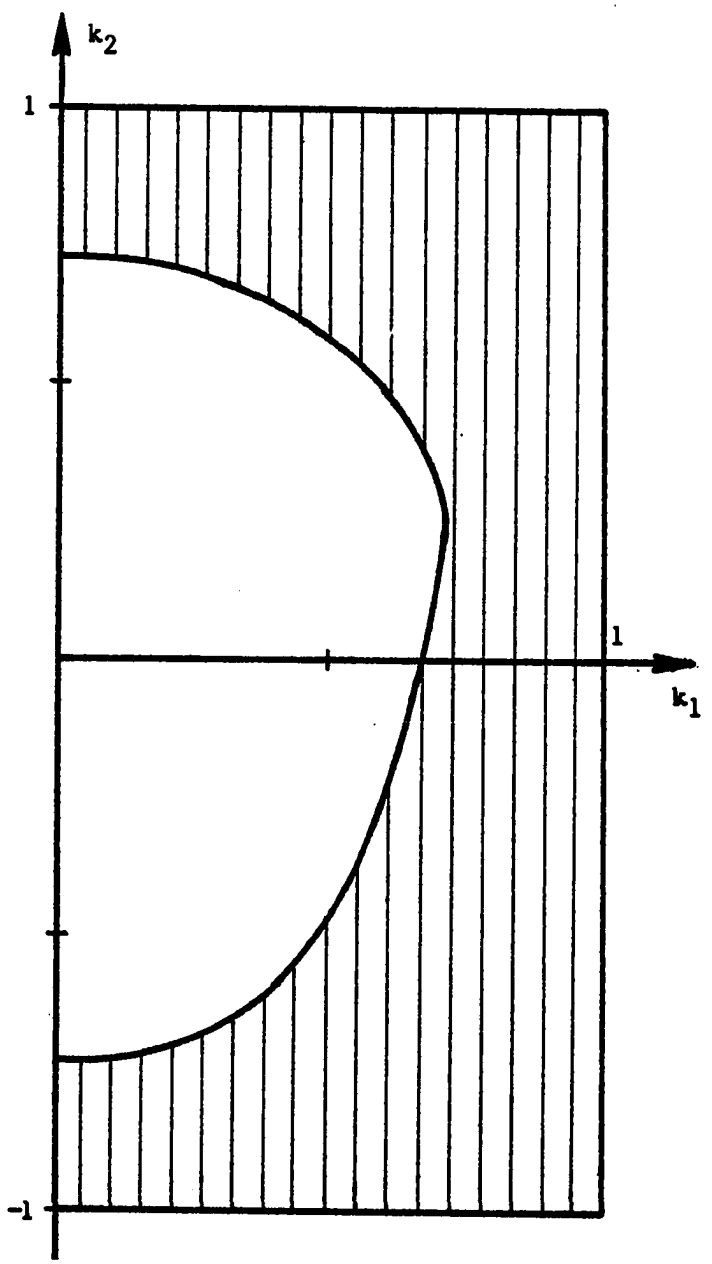


Figure 4.36. Region where the lattice filter with three truncation quantizers and no overflow is g.a.s. by Theorem 4.3

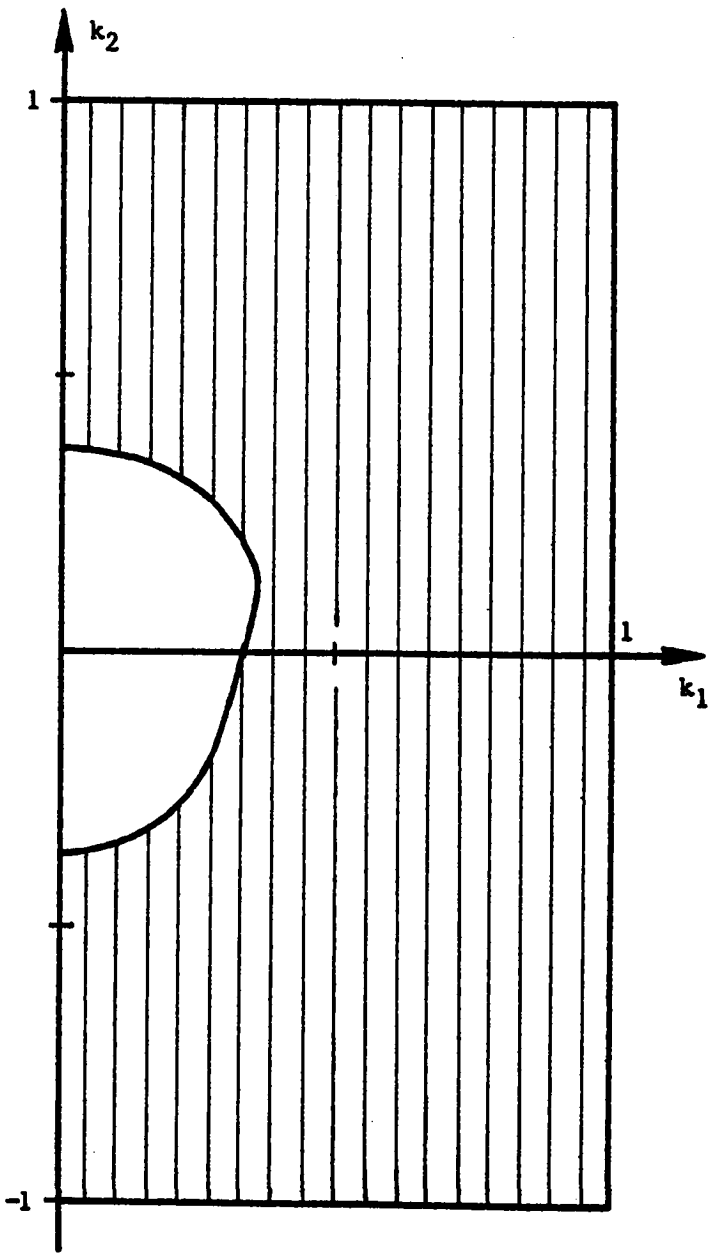


Figure 4.37. Region where the lattice filter with three roundoff quantizers and no overflow is g.a.s. by Theorem 4.3

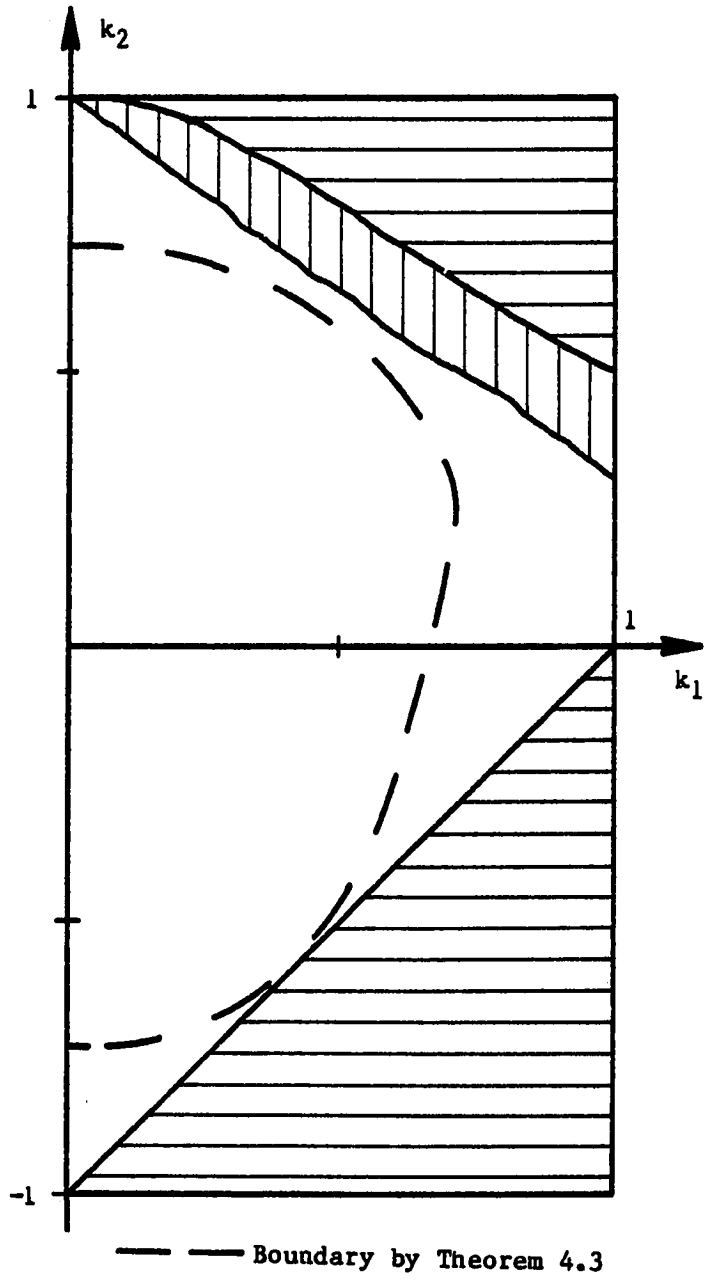


Figure 4.38. Region where the lattice filter with three truncation quantizers and no overflow is g.a.s. by the constructive algorithm

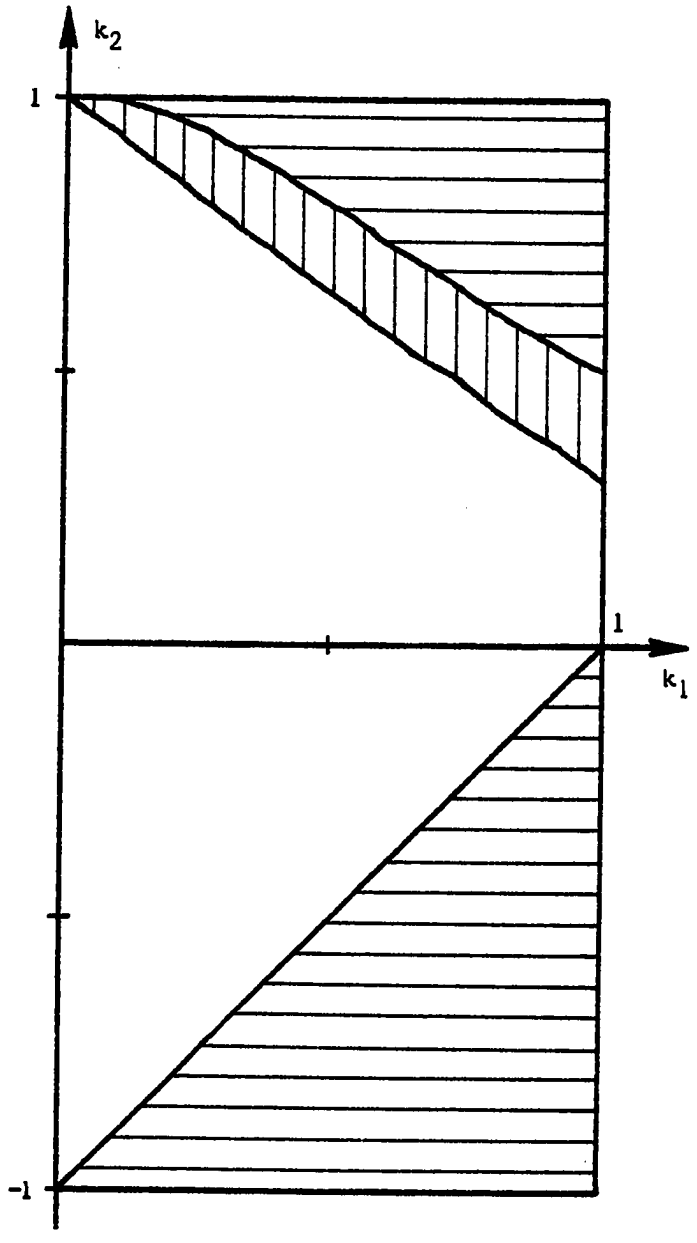


Figure 4.39. Region where the lattice filter with three truncation quantizers and saturation or zeroing overflow is g.a.s. by the constructive algorithm

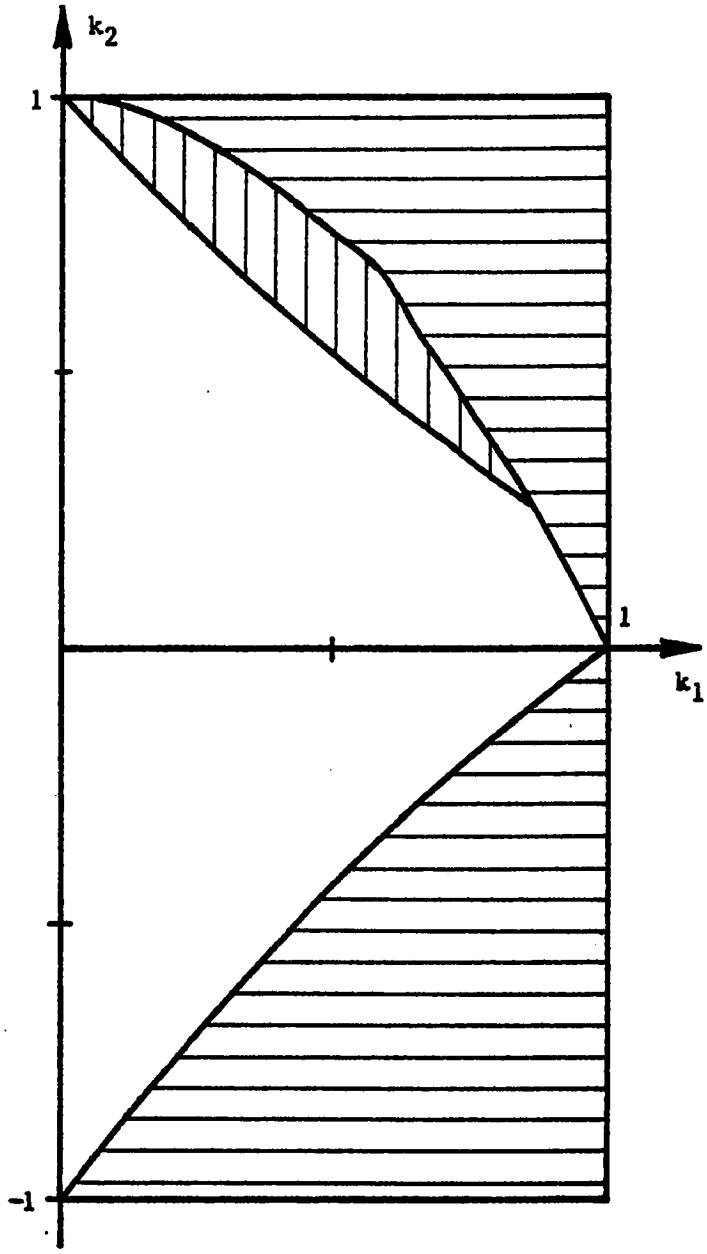


Figure 4.40. Region where the lattice filter with three truncation quantizers and triangular overflow is g.a.s. by the constructive algorithm

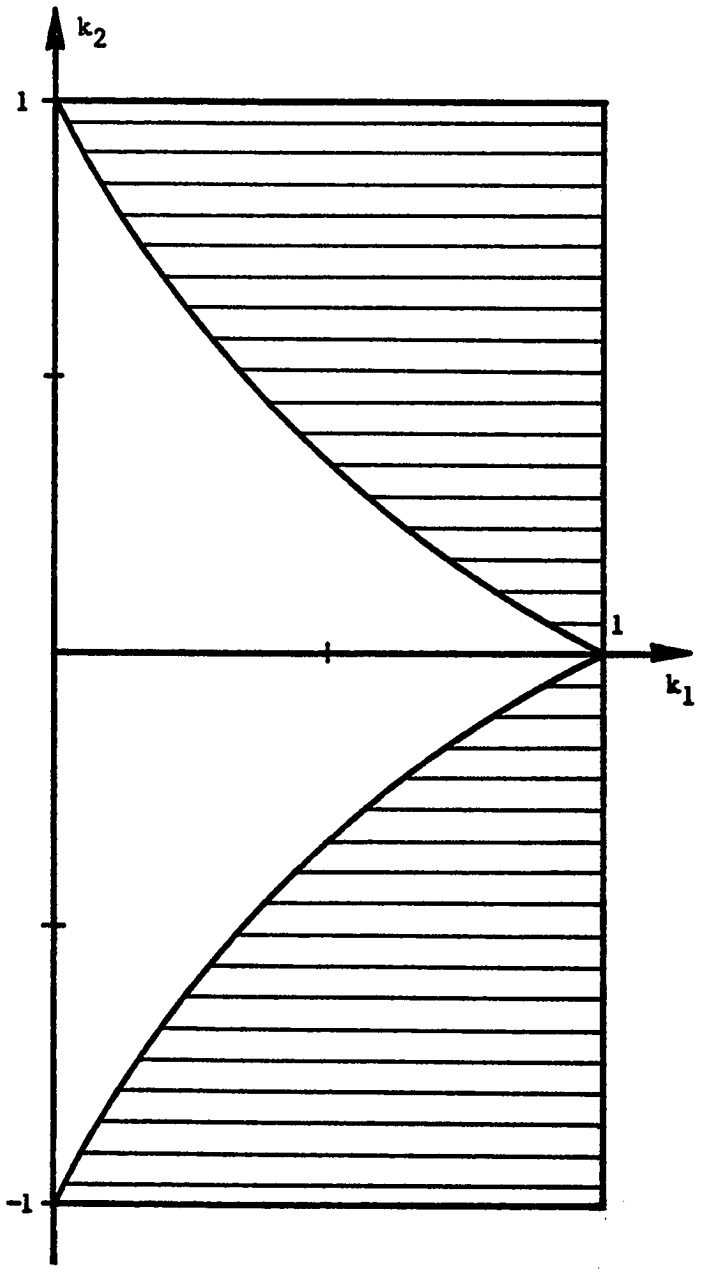


Figure 4.41. Region where the lattice filter with three truncation quantizers and two's complement overflow is g.a.s. by the constructive algorithm

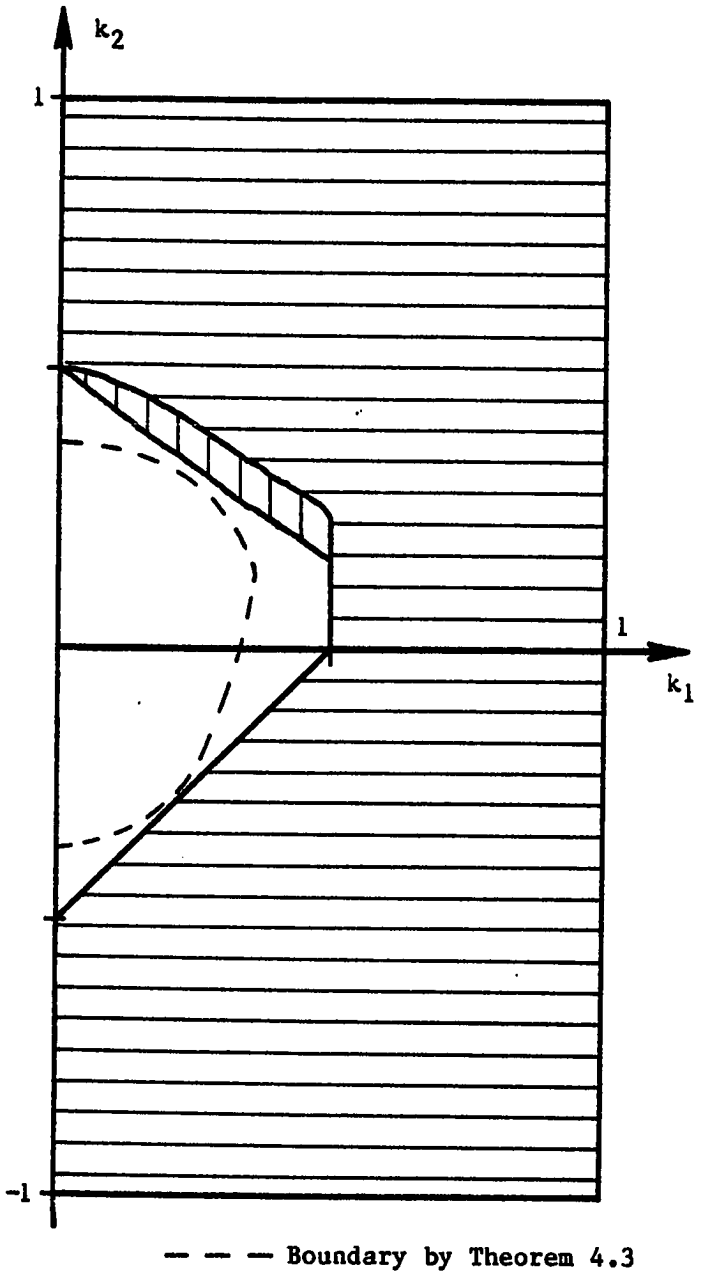


Figure 4.42. Region where the lattice filter with three roundoff quantizers and no overflow is g.a.s. by the constructive algorithm

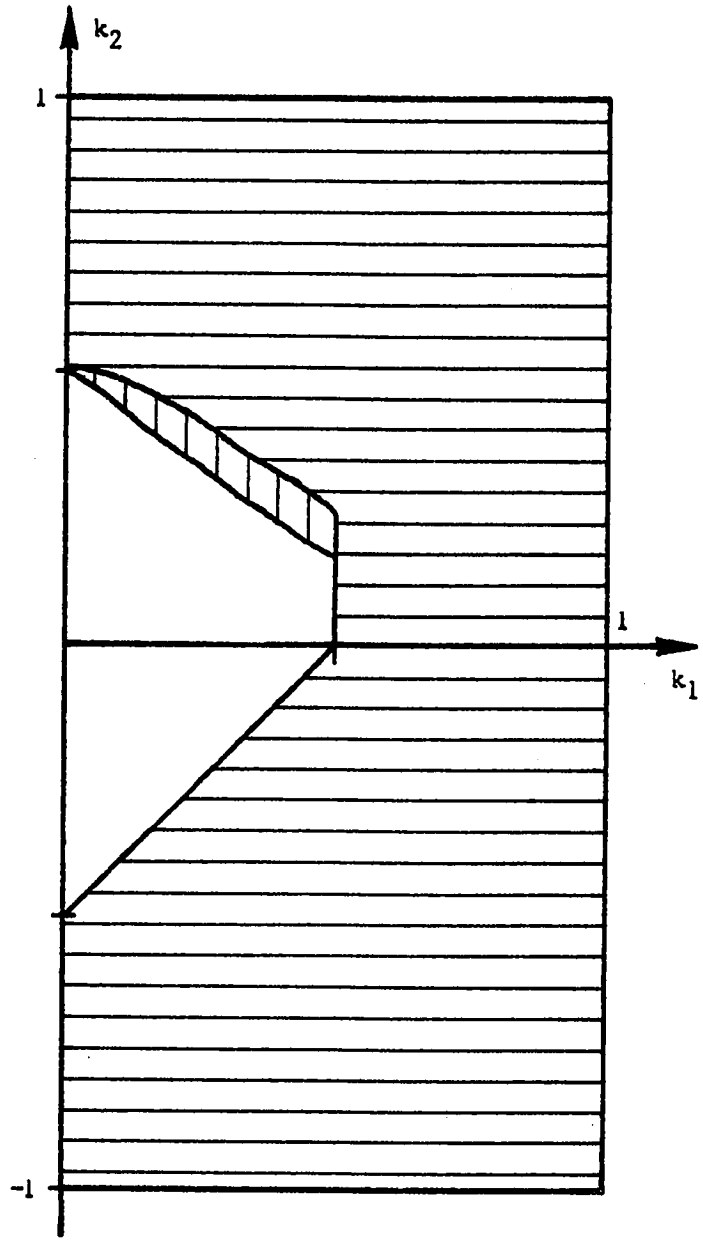


Figure 4.43. Region where the lattice filter with three roundoff quantizers and saturation or zeroing overflow is g.a.s. by the constructive algorithm

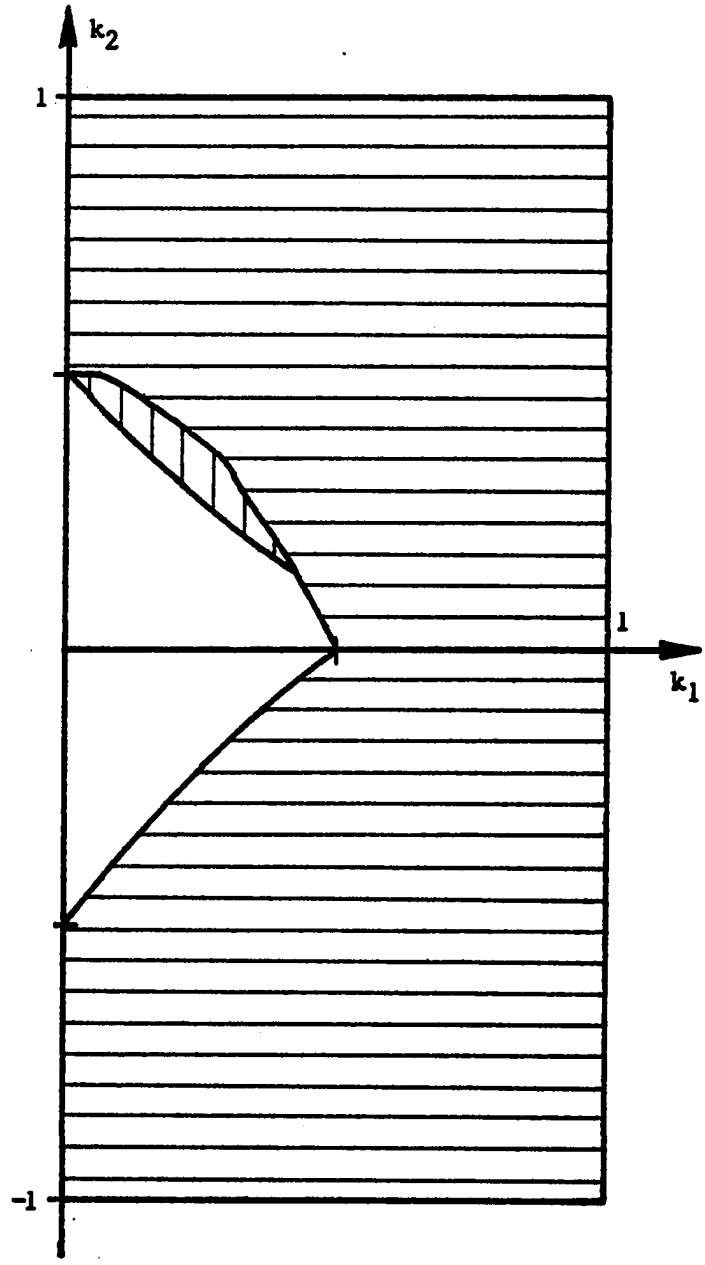


Figure 4.44. Region where the lattice filter with three roundoff quantizers and triangular overflow is g.a.s. by the constructive algorithm

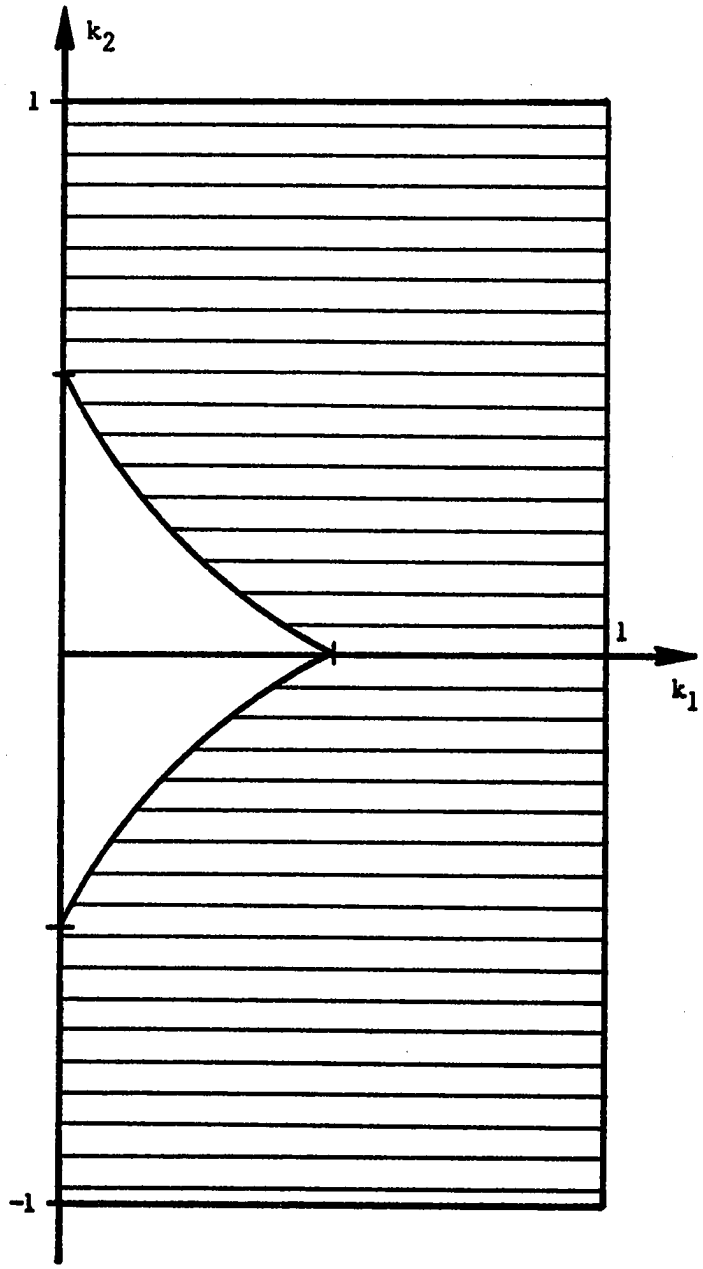


Figure 4.45. Region where the lattice filter with three roundoff quantizers and two's complement overflow is g.a.s. by the constructive algorithm

V. CONCLUSION

The fixed-point digital filter structures that we have considered demonstrate that the Brayton-Tong constructive algorithm is a powerful tool in the stability analysis of fixed-point digital filters. We have obtained new results for many of the structures which we have analyzed. We have also improved upon many existing stability results. Our results are more conservative only for a few cases. In these cases, the existing results consider a specific nonlinearity, whereas the constructive algorithm obtains a stability result that applies to a class of nonlinearities.

Following is a summary of the stability results obtained by the constructive algorithm for the various digital filter structures that we studied. For comparison, the references for the existing results are also listed.

New stability results:

1) Direct form, one quantizer with overflow
2) Direct form, two quantizers with overflow
3) Coupled form, four quantizers with overflow
4) Wave filter, two roundoff quantizers with overflow
5) Lattice filter, two roundoff quantizers with or without overflow
6) Lattice filter, three quantizers with overflow

Improvement upon existing results:

1) Direct form, one roundoff quantizer without overflow [19]
2) Direct form, two quantizers without overflow [20]
3) Coupled form, four truncation quantizers without overflow [25]
4) Wave filter, two roundoff quantizers without overflow [20]
5) Lattice filter, three quantizers without overflow [20]

Same as existing results:

1) Direct form, one truncation quantizer without overflow [16]
2) Coupled form, two truncation quantizers with or without overflow [22]
3) Wave filter, two truncation quantizers with or without overflow [26]
4) Wave filter, three quantizers without overflow [20]
5) Lattice filter, two truncation quantizers with or without overflow [28]

More conservative than existing results:

1) Direct form, saturation or triangular overflow only [18]
2) Coupled form, two roundoff quantizers with or without overflow [24]

3) Coupled form, four roundoff quantizers without overflow [24]

The Brayton-Tong constructive stability algorithm is powerful since it can be applied to a wide range of fixed-point digital filter structures. While existing methods of stability analysis are generally different for each particular structure, the constructive algorithm allows us to use one method to study the stability of nonlinear digital filter structures. This method can be cumbersome for complicated structures, but it is straightforward. We feel that this method should become a tool to be used in the evaluation of any proposed new filter structure.

We have only examined those digital filter structures that have received much attention in the literature. There are many other digital filter structures that are slight modifications of the structures that we have considered (e.g., modified coupled form and one multiplier lattice) that could also be studied.

We have only considered second order digital filters in this dissertation. The constructive method can be applied to higher order filters either directly or by considering the higher order filter as an interconnection of lower order structures (see, e.g., [29], [30]). The method could also be extended to the stability analysis of a feedback control system using a digital compensator.

VI. REFERENCES

1. Brayton, R. K. and Tong, C. H. "Stability of Dynamical Systems: A Constructive Approach." IEEE Transactions Circuits and Systems CAS-26 (1979): 224-234.

2. Brayton, R. K. and Tong, C. H. "Constructive Stability and Asymptotic Stability of Dynamical Systems." IEEE Transactions Circuits and Systems CAS-27 (1980): 1121-1130.

3. Dunford, N. and Schwartz, J. T. Linear Operators - Part I: General Theory. New York: Interscience, 1958.

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

5. Oppenheim, A. V. and Schafer, R. W. Digital Signal Processing. Englewood Cliffs, N. J.: Prentice-Hall, 1975.

6. Eckhardt, B. and Winkelnkemper, W. "Implementation of a Second Order Digital Filter Section with Stable Overflow Behaviour." Nachrichtentechnische Zeitschrift 26 (1973): 282-284.

7. Claasen, T. A. C., Mecklenbräuker , W. F. G., and Peek, J. B. H. "Effects of Quantization and Overflow in Recursive Digital Filters." IEEE Transactions Acoustics, Speech, and Signal Processing ASSP-24 (1976): 517-529.

8. Jury, E. I. "A Simplified Stability Criterion for Linear Discrete Systems." Proceedings of the IRE 50 (1962): 1493-1500.

9. Rader, C. M. and Gold, B. "Effects of Parameter Quantization on the Poles of a Digital Filter." Proceedings of the IEEE 55 (1967): 688-689.

10. Fettweis, A. "Digital Filter Structures Related to Classical Filter Networks." Archiv fürElektronik und Übertragungstechnik 25 (1971): 79-89.

11. Antoniou, A. Digital Filters: Analysis and Design. New York: McGraw-Hill, 1979.

12. Fettweis, A. "Pseudopassivity, Sensitivity, and Stability of Wave Digital Filters." IEEE Transactions Circuit Theory CT-19 (1972): 668-673.

13. Itakura F. and Saito S. "Digital Filtering Techniques for Speech Analysis and Synthesis." Proceedings of the Seventh International Congress on Acoustics 3 (1971): 261-264.

14. Friedlander, B. "Lattice Filters for Adaptive Processing." Proceedings of the IEEE 70 (1982): 829-867.

15. Gray, A. H. and Markel, J. D. "Digital Lattice and Ladder Filter Synthesis." IEEE Transactions Audio and Electroacoustics AU-21 (1973): 491-500.

16. Claasen, T. A. C. M. "Survey of Stability Concepts of Digital Filters." Telecommunication Theory, Electrical Engineering, Royal Institute of Technology (Stockholm, Sweden) Technical Report No. 108, 1976.

17. Claasen, T. A. C. M. and Kristiansson, L. O. G. "Necessary and Sufficient Conditions for the Absence of Overflow Phenomena in a Second-Order Recursive Digital Filter." IEEE Transactions Acoustics, Speech, and Signal Processing ASSP-23 (1975): 509-515.

18. Ebert, P. M., Mazo, J. E., and Taylor, M. G. "Overflow Oscillations in Digital Filters." Bell System Technical Journal 48 (1969): 2999-3020.

19. Claasen, T., Mecklenbräuker, W. F. G., and Peek J. B. H. "Frequency Domain Criteria for the Absence of Zero-Input Limit Cycles in Nonlinear Discrete-Time Systems, with Applications to Digital Filters." IEEE Transactions Circuits and Systems CAS-22 (1975): 232-239.

20. Jury, E. I. and Lee, B. W. "The Absolute Stability of Systems With Many Nonlinearities." Automation and Remote Control 26 (1965): 943-961.

21. Franklin, G. F. and Powell, J. D. Digital Control of Dynamical Systems. Reading, Mass.: Addison-Wesley, 1980.

22. Barnes, C. W. and Fam A. T. "Minimum Norm Recursive Digital Filters that Are Free of Overflow Limit Cycles." IEEE Transactions Circuits and Systems CAS-24 (1977): 569-574.

23. Jackson, L. B. "Limit Cycles in State-Space Structures for Digital Filters." IEEE Transactions Circuits and Systems CAS-26 (1979): 67-68.

24. Barnes, C. W. and Shinnaka, S. "Stability Domains for Second-Order Recursive Digital Filters in Normal Form with 'Matrix Power' Feedback." IEEE Transactions Circuits and Systems CAS-27 (1980): 841-843.

25. Jackson, L. B. and Judell, N. K. H. "Addendum to 'Limit Cycles in State-Space Structures for Digital Filters'." IEEE Transactions Circuits and Systems CAS-27 (1980): 320.

26. Fettweis, A. and Meerkötter, K. "Suppression of Parasitic Oscillations in Wave Digital Filters." IEEE Transactions Circuits and Systems CAS-22 (1975): 239-246.

27. Fettweis, A. and Meerkötter, K. "Correction to 'Suppression of Parasitic Oscillations in Wave Digital Filters'." IEEE Transactions Circuits and Systems CAS-22 (1975): 575.

28. Gray, A. H. "Passive Cascaded Lattice Digital Filters." IEEE Transactaions Circuits and Systems CAS-27 (1980): 337-344.

29. Michel, A. N., Miller, R. K., and Nam, B. H. "Stability Analysis of Interconnected Systems Using Computer Generated Lyapunov Functions." IEEE Transactions Circuits and Systems CAS-29 (1982): 431-440.

30. Michel, A. N. and Miller, R. K. Qualitative Analysis of Large Scale Dynamical Systems. New York: Academic Press, 1977.

VII. ACKNOWLEDGEMENTS

The author wishes to gratefully acknowledge his advisor, Dr. Anthony N. Michel for suggesting the dissertation topic and for providing valuable assistance throughout the development of this work. Acknowledgements are also extended to Dr. R. Grover Brown, Dr. Thomas F. Piatkowski, Dr. Harry W. Hale and Dr. Richard K. Miller for serving on his committee. Thanks go to Roberta North for typing this dissertation.

This work was financially supported by a National Science Foundation graduate fellowship and by an Engineering Research Institute graduate research assistantship. This work was also supported in part by the National Science Foundation under Grant ECS-8100690. All graduate work was done during an educational leave of absence from Fisher Controls Company, Marshalltown, Iowa.

The author also wishes to thank his parents, Mr. and Mrs. Donald H. Erickson, and his wife's parents, Dr. and Mrs. Raymond L. Venable, for their encouragement during his graduate work. The author also appreciates his wife, Fran, for her assistance, patience, encouragement and companionship as he worked on this dissertation topic. The author is also grateful for the diversion provided by his children, Esther and David.

Last, but not least, thanks be to God for His infinite mercy and grace through Jesus Christ.

VIII. VITA

Kelvin Todd Erickson was born to Donald H. and Julia Bolz Erickson on August 9, 1957, in Ridgeway, Pennsylvania. Educated in the public schools of Fort Wayne, Indiana and Jackson, Tennessee, he received his diploma from Jackson Central-Merry High School in 1975. He attended the University of Missouri-Rolla, receiving the Bachelor of Science in Electrical Enginering in May 1978 and the Master of Science in Electrical Engineering in December 1979.

He has been enrolled in the Graduate College of Iowa State University since August 1981. A National Science Foundation Graduate Fellowship was held by him for the periods August 1978 to July 1979 and August 1981 to July 1983.

On May 8, 1977 he was married to the former Francilda Ann Venable. Their union has been blessed with two children, Esther Anne, born on August 14, 1979 and David Andrew, born on November 14, 1981.

IX. APPENDIX A: DESCRIPTION OF COMPUTER PROGRAMS

In this appendix we describe the computer programs that we used in our investigation of digital filter stability by the constructive algorithm of Brayton and Tong. In Section A, we give a short description of the subroutine that implements the constructive algorithm. In Section B, we describe a program, BGRID, that uses this constructive algorithm to find the region in the parameter plane where a second order filter is globally asymptotically stable. BGRID finds this region by checking individual points in the parameter plane. In Section C, we present a description of a program, BORDR, that finds the boundary of this region where a digital filter is globally asymptotically stable.

In a normal sequence, the program BGRID is used to get a general idea of the form of the region where the filter is globally asymptotically stable. Then, the program BORDR is used to list the points along the boundary of this region. These points are then used to draw the boundaries of the region. This is the sequence that we followed in producing the figures in Chapter IV that depict the regions where the various digital filters are globally asymptotically stable.

All of these programs are written in FORTRAN IV for use on an HP 1000 Model 40 minicomputer with the RTE-IVB operating system in the Electrical Engineering Department at Iowa State University. Consequently, there are some differences in these programs from the

standard FORTRAN IV programming language. These differences have been kept to a minimum and they are noted in each of the respective program descriptions.

Source listings for all of these computer programs are in Appendix B. For each of the subroutines in the listings, there is a short description of the function of the subroutine along with a description of the input variables and output variables. In addition, for each main program and major subroutine, there is also a section labelled "PSEUDO-CODE" that contains a structured English-like description of the operation of the subroutine or program.

Each program and subroutine listed in Appendix B has been tested. The operation of each program or subroutine was independently tested by invoking the program or subroutine with appropriate test inputs. These test inputs were chosen to exercise all of the branches of the routine and to check the possible values of the variables. The results of each routine were then compared with the expected results. In addition, using program BGRID, the operation of the constructive algorithm was checked against the example in the Section VI of Brayton and Tong [1]. We obtained the same region in the parameter plane where this set of matrices was stable as reported in [1].

A. Constructive Algorithm Subroutine

The Brayton-Tong constructive algorithm is implemented as a subroutine, BRAYT, so that it can be used by more than one computer

program. A complete description of the constructive algorithm is given in [1] and [2]. Our implementation of the constructive algorithm follows the algorithm presented in [2]. Since the major part of the algorithm handles the formation of the new vertex set $E(W_k)$ from the vertex set $E(W_{k-1})$, we describe this part of the algorithm. In the following description, M_i, $i = 1, \cdots, m$, is an element of the set of matrices $\underline{A}$. The set V is initially $E(W_o)$ and is set to $E(W_k) \cap E(W_{k-m})$ for $k>m$, where k is the number of the convex hull. Steps 1) - 17) are the major part of the algorithm. More detail about the algorithm is found in Section A of Appendix B.

Formation of the new vertex set $E(W_k)$:

```
1)  Set E(W_k) = E(W_k-1)
2)  Set done-flag to "no"
3)  Do for each vertex in E(W_k-1) while done is "no"
4)     If vertex ∉ V or k ≤ m
5)        Set NEWPT = vertex
6)        Do while new points are being added to E(W_k)
7)           NEWPT = M_i(NEWPT)
8)           If NEWPT ∉ κ[W_k]
9)              Add NEWPT to E(W_k), eliminating all
                 vertices contained within κ[W_k]
10)             If E(W_o) ∩ E(W_k) = φ
11)                Set done-flag to "yes"
12)                Set status to "unstable"
           End do while
```

End of processing one vertex in $E(W_{k-1})$

13) If k > m and done is "no"

14) If $E(W_k) = E(W_{k-m})$

15) Set done-flag to "yes"

16) Set status to "stable"

17) Else set $V = E(W_k) \cap E(W_{k-m})$

The constructive algorithm as we have implemented it is not exactly the same as the form of the algorithm given in [2] but the differences are relatively minor and do not change the outcome of the algorithm. The first difference is that we do not use the improved instability criterion given in [2]. Also, whenever a new vertex is added to a convex hull, we eliminate all of those vertices that are now contained within the new convex hull. This second difference greatly simplifies the algorithm description, since much of the algorithm description in [2] determines if a given vertex from the previous convex hull needs to be eliminated from the current convex hull.

The only known differences from standard FORTRAN IV are contained in the first two lines of the source listing in Section A of Appendix B. The first line is the HP-1000 compiler directive to use four-word double precision numbers and to produce a listing of the source program. The second line places the common data area DAT in the extended memory area (EMA) since the computer that we used has a limited amount of main memory.

The routines that implement the constructive algorithm can be

modified for use with higher order systems. In addition to obvious changes in array sizes, the subroutines AINIT, ADDVT, CKVRT, SANGL and EIGEN will need modification, since they only apply to second order systems.

B. Program to Find the Region of Stability for a Digital Filter

The computer program BGRID, described in this section, uses the constructive algorithm to find the region in the parameter plane where the equilibrium point $x = 0$ of a second order digital filter is stable or globally asymptotically stable. BGRID finds this region of stability by checking the stability of the filter at individual points within a given two-dimensional region. The user specifies these points where the filter stability is checked as a grid of points within a desired region. This region may be a rectangle, a point or a default region dependent on the particular digital filter. The default region is the region in the parameter plane in which we are most interested in checking the stability of the filter. The default region is a portion of the region where the linear digital filter is globally asymptotically stable and is chosen because of symmetry considerations. For example, the default region for a direct form filter is the right half of the triangular region in Figure 3.4 since the regions where the nonlinear filter is globally asymptotically stable are symmetric about the b-axis.

BGRID starts by querying the user for the data needed to run the program. The user is first asked for the type of quantization and

overflow nonlinearities within the digital filter. Then the user is asked for the type of region where the filter stability is checked. If the user selects the default region (dependent on filter structure) then the grid increment is also entered. The grid increment is the change in the x and/or y coordinate from one point to another in the region. The grid of points is uniform. The user may also select the reflection of the default region which is the default region reflected through the y axis. If the user wants to check the stability of the filter in a rectangular region, the user enters the horizontal and vertical limits of the rectangle as well as the horizontal and vertical grid increment. If the user desires to check the stability of the filter at a single point in the parameter plane, then the coordinates of that point are entered. In this case, the user may also request a "trace" of the constructive algorithm. Regardless of the type of region where the filter stability is checked, the user is asked for the value of ρ. The value of ρ must be greater than or equal to one. If ρ is greater than one, then BGRID will find the region where the filter is globally asymptotically stable.

For each of the points in the region, the set of extreme matrices is generated and then used by the constructive algorithm to determine the stability "status" of the set of extreme matrices for the filter at that point. The constructive algorithm indicates that the set of extreme matrices is stable, unstable or indeterminant. The set of extreme matrices is indeterminant if the constructive

algorithm terminated abnormally due to a condition other than the stable or unstable stopping criteria of the constructive algorithm. The conditions that cause the constructive algorithm to terminate abnormally are given in the source listing for the constructive algorithm subroutines. The program then prints the stability "status" of the set of matrices at each point in the grid. The values of this stability "status" and their corresponding indication are:

0	-	set is stable
1	-	set is unstable
2,3,or 5	-	stability of set is indeterminate
4	-	at least one matrix in the set of extreme matrices has one eigenvalue with a magnitude greater than one (set is unstable).

We can conclude that the digital filter is globally asymptotically stable at that point only when the set of extreme matrices is globally asymptotically stable. If the set of extreme matrices is not stable, then we can draw no conclusions about the stability of the filter.

BGRID uses a group of subroutines that are not listed with the program. This group of subroutines is unique to each digital filter structure. Included in this group are the subroutines that identify the program output for a particular filter structure (GETLB and

DESCR), define the default region (DFAUL and GTEND) and generate the set of extreme matrices (GTMAT). For a particular filter structure, these subroutines are compiled separately and then are linked with the main program, BGRID, and the constructive algorithm subroutine BRAYT, to form a program that finds the region of stability for that particular filter. The group of subroutines for each of the digital filter structures that we studied is listed in Section D of Appendix B. If one wishes to study another digital filter structure, then these subroutines will need to be written for that particular structure.

Example outputs of BGRID are given in Figures 9.1 and 9.2. The digital filter structure is a direct form with one quantizer (Figure 3.6). In this case, the program BGRID was formed by linking the unique subroutines associated with the direct form structure with one quantizer (Appendix B, Subsection D1) and the constructive algorithm subroutine (Appendix B, Section A) with the BGRID program (Appendix B, Section B). The points where the constructive algorithm finds the set of extreme matrices stable are points where the filter is globally asymptotically stable since ρ is chosen to be 1.0000001. For both figures, the user selected a roundoff quantizer with triangular overflow. In Figure 9.1, the user specified the default region with a grid increment of 0.1. The default region for the direct form filter is the right half of the triangular region of Figure 3.4. In Figure 9.2, the user specified a rectangular region

```
  BRAYTON-TONG STABILITY   BRAYA
CANONICAL FORM DIGITAL FILTER WITH ONE QUANTIZER

 ROUNDOFF QUANTIZER
 TRIANGULAR OVERFLOW

 DEFAULT REGION.   GRID INCREMENT IS  .100000

 RHO IS  1.0000001


 1.000  4
  .900  44
  .800  444
  .700  4444
  .600  44444
  .500  444444
  .400  0444444
  .300  00444444
  .200  000444444
  .100  0000444444
  .000  00000444444
 -.100  000000444444
 -.200  0000000444444
 -.300  000000001444444
 -.400  0000001111444444
 -.500  44444444444444444
 -.600  444444444444444444
 -.700  4444444444444444444
 -.800  44444444444444444444
 -.900  444444444444444444444
-1.000  4444444444444444444444
```

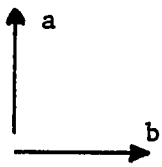


Figure 9.1. Example output of BGRID: default region

```
  BRAYTON-TONG STABILITY   BRAYA
CANONICAL FORM DIGITAL FILTER WITH ONE QUANTIZER

 ROUNDOFF QUANTIZER
 TRIANGULAR OVERFLOW

 HOR START, END, AND INC IS       .5000     1.0001      .0200
 VER START, END, AND INC IS      -.2000     -.5001     -.0200

 RHO IS  1.0000001


 -.200  00000000004444444444444444
 -.220  00000000000444444444444444
 -.240  00000000000144444444444444
 -.260  00000000000114444444444444
 -.280  00000000001111444444444444
 -.300  00000000001111144444444444
 -.320  00000000011111114444444444
 -.340  00000000111111111444444444
 -.360  00000001111111111144444444
 -.380  00000011111111111114444444
 -.400  00000111111111111111444444
 -.420  00111111111111111111144444
 -.440  11111111111111111111114444
 -.460  11111111111111111111111444
 -.480  11111111111111111111111144
 -.500  44444444444444444444444444
```

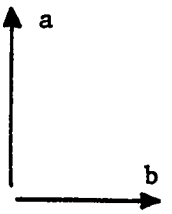


Figure 9.2. Example output of BGRID: rectangular region

in which to check the global asymptotic stability of the filter. The horizontal limits are 0.5 and 1.0 with a horizontal increment of 0.02. The vertical limits are -0.2 and -0.5 with a vertical increment of 0.02. We do not give example outputs of BGRID when the filter stability is checked at a single point or when a trace of the constructive algorithm is requested.

BGRID can be run interactively where the user answers questions from a terminal, or it can be run in a batch environment where the responses to the questions are taken from a data file. In addition, BGRID can build the response data file to be used in a batch environment. However, this feature is dependent on the computer on which BGRID is run. If BGRID is to be run on a different computer, the part of the program that sets up the input and output device numbers will need to be modified. The first three lines of the program source listing (Appendix B, Section B) are also deviations from the standard FORTRAN IV language. The first line is the HP-1000 Fortran compiler directive. The second line places the common area DAT into the extended memory area (EMA). This line is required by the constructive algorithm subroutine BRAYT. The third line identifies the name of the program.

The run times of BGRID depended mainly on the number of extreme matrices and the number of points in the grid. All of the run times given here are for the default region for that filter with a grid increment of 0.1 units. Some example average run times are:

Direct form, one quantizer (2 matrices) - 25 sec

Direct form, two quantizers (4 matrices) - 40 sec

Coupled form, two quantizers (4 matrices) - 12 sec

Coupled form, four quantizers (16 matrices) - 60 sec

C. Program to Find the Boundary of a Region

The computer program, BORDR, described in this section, finds points on the boundary of the region in the parameter plane where a digital filter is globally asymptotically stable or the boundary of the region in the parameter plane where all of the extreme matrices of the filter have eigenvalues on or inside the unit circle. BORDR only works for two-dimensional regions. The output of BORDR is a list of points along the boundary of the desired region. Consequently, these points are used to draw the boundaries of the region.

The constructive algorithm only gives an indication of the stability of a system. There is no indication of how close a particular system is to the boundary between the stable region and the region where the stability of the system is uncertain by the constructive algorithm. Therefore, BORDR uses a binary search technique to find the boundary between the two regions to a desired accuracy.

BORDR uses two axes to find the boundary of a two-dimensional region. Along the search axis, a binary search is used between an

inner search point and an outer search point in order to find the boundary point to a desired accuracy. The inner search point is inside the region and the outer search point is outside the region. The search axis is parallel to either the x-axis or the y-axis. The other axis used by BORDR, the increment axis, is perpendicular to the search axis. In the direction of the increment axis, the next boundary point is predicted using a given search increment, the boundary gradient and the previous boundary point. Using the predicted boundary point, the inner and outer search points are predicted. The next boundary point is then found by searching between the inner and outer search points. If both predicted search points happen to be outside or inside the region, then the search axis is changed to the previous increment axis and the program tries to find the boundary point along this axis. If BORDR cannot find the next boundary point along the new axis, the program stops. BORDR terminates normally if it returns to the initial boundary point or if it receives some indication that the end of the desired boundary has been reached. In order that the search algorithm starts properly, an initial search axis and search points must be specified. The initial inner search point must be inside the region and the initial outer search point must be outside the region.

To illustrate the operation of the two-dimensional search algorithm of BORDR, suppose that the boundary of the region is the triangle in Figure 9.3. An initial search axis could be the y-

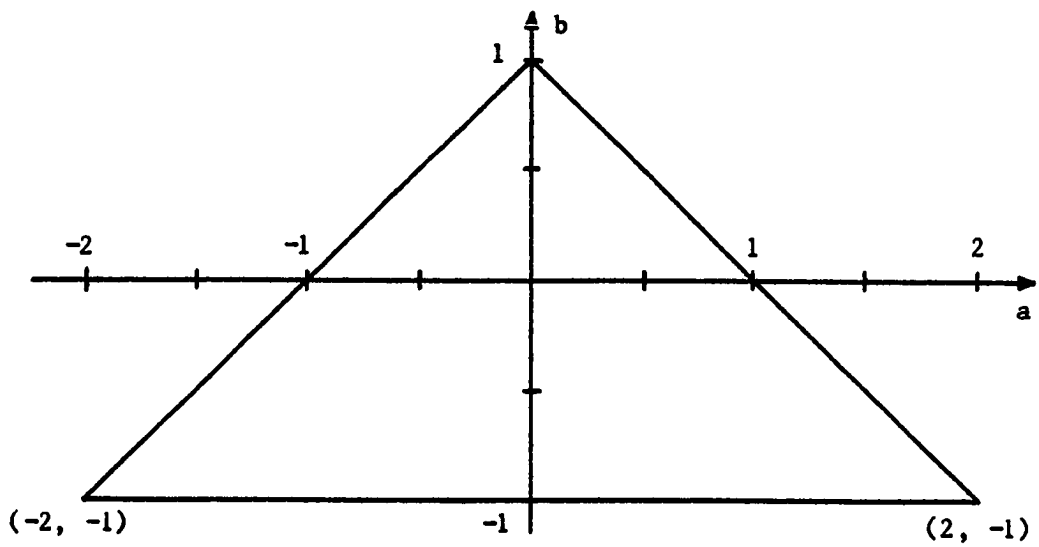


Figure 9.3. **Triangular region used to illustrate the operation of BORDR**

axis. The initial inner search point could be (0.0, 0.5) and the initial outer search point could be (0.0, 1.5). Let the search increment be 0.1. The initial boundary point is determined to be (0.0, 1.0). Since there is no previous boundary point, BORDR guesses the next boundary point to be (0.1, 1.0). The algorithm estimates the inner search point to be (0.1, 0.8) and the outer search point to be (0.1, 1.2). The next boundary point is found to be (0.1, 0.9). BORDR proceeds to "walk" down the boundary until it comes to a corner. At the corner, the search axis is changed and BORDR proceeds to "walk" along the boundary again.

BORDR also uses a group of subroutines that is not listed with the program. This group of subroutines is unique to each digital filter structure. Included in this group are the subroutines that print the first three lines of the output (LINIT), determine if the end of the desired boundary has been reached (DNCHK) and generate the set of extreme matrices (GTMAT). For a particular filter structure, these subroutines are compiled separately and then are linked with the main program and the constructive algorithm subroutines to form a program that finds the boundary of the desired region for that particular filter. The group of subroutines for each of the digital filter structures that we studied is listed in Section D of Appendix B. If one wishes to study another digital filter structure, then these subroutines will need to be written for that particular structure.

An example output of this program is given in Figure 9.4. In this case, the program BORDR was formed by linking the unique subroutines associated with the direct form structure with one quantizer (Appendix B, Subsection D1) and the constructive algorithm subroutine (Appendix B, Section A) with the BORDR program (Appendix B, Section C). The first line identifies the boundary data by describing the filter structure and by giving the values of the program input parameters. The first parameter is the type of quantizer, the second is the type of overflow nonlinearity and the third is the type of boundary. The specific values of these parameters are explained in the program source listing (Appendix B, Section C). For the example in Figure 9.4, the boundary points are for the region where a direct form digital filter with one roundoff quantizer and two's complement overflow (Figure 3.6) is globally asymptotically stable. The second line in Figure 9.4 indicates the minimum and maximum values of the boundary points. The first two values are the minimum and maximum horizontal values, respectively. The last two values are the minimum and maximum vertical values, respectively. The third line of the program output is an indication of the symmetry of the boundary so that all of the boundary points do not have to be listed. Possible values of the symmetry indication and their meanings are:

```
DIRECT FORM, ONE QUANTIZER     1    2    1    0
   -1.0000    1.0000    -.5000     .5000
 1
     .50098   0.00000
     .52098   -.02000
     .54098   -.04000
     .56098   -.06000
     .58098   -.08000
     .60098   -.10000
     .62098   -.12000
     .64098   -.14000
     .66098   -.16000
            .
            .
            .
            .
     .19598   -.50000
     .17598   -.50000
     .15598   -.50000
     .13598   -.50000
     .11598   -.50000
     .09598   -.50000
     .07598   -.50000
     .05598   -.50000
     .03598   -.50000
     .01598   -.50000
    0.00000   0.00000
```

Figure 9.4. Example output of BORDR

0 - no symmetry

1 - symmetric about y-axis

2 - symmetric about x-axis

3 - symmetric about x-axis and y-axis.

For the example in Figure 9.4, the boundary is symmetric about the y-axis. Succeeding lines contain the x- and y-coordinates, respectively of the boundary points. Not all of the boundary points are shown in Figure 9.4. A boundary point of (0,0) signifies the end of the boundary points. These points were used to draw Figure 9.5. The boundary in Figure 9.5 is a part of Figure 4.10.

The input parameters that are needed by BORDR are passed into the program by an HP-1000 operating system utility. These parameters are the output device number, the type of quantization nonlinearity, the type of overflow nonlinearity and the type of boundary. The method of setting these input parameters will need to be modified if BORDR is run on a different computer. The first three lines of the program source listing (Appendix B, Section C) are also deviations from the standard FORTRAN IV language. The first line is the HP-1000 Fortran compiler directive. The second line places the common area DAT into the extended memory area (EMA). This line is required by the constructive algorithm subroutine BRAYT. The third line identifies the name of the program.

The run times of BORDR depended mainly on the number of extreme matrices. All of the run times given here are for an increment

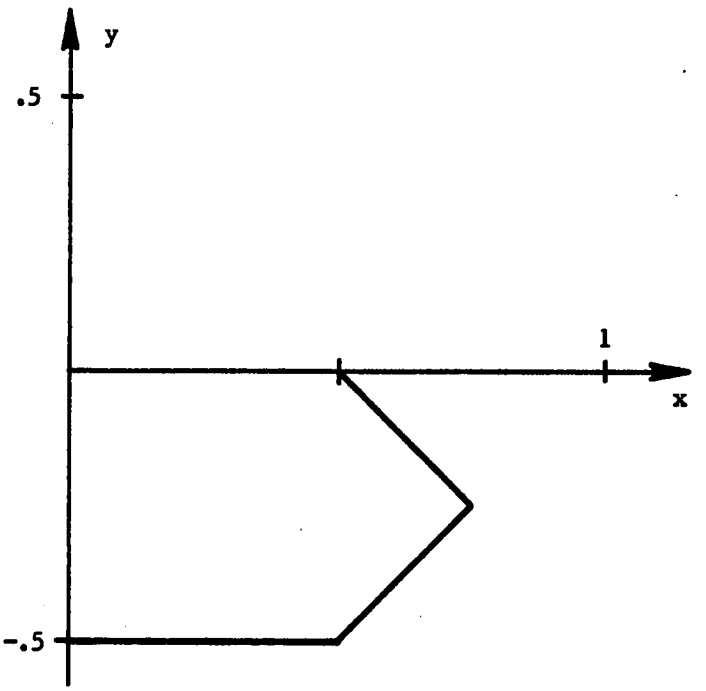


Figure 9.5. Boundary drawn from example output of BORDR

between points of 0.02 units and a search accuracy of 0.005 units.

Some example average times are:

Direct form, one quantizer (2 matrices)	- 8 min
Direct form, two quantizers (4 matrices)	- 150 min
Lattice filter, three quantizers (16 matrices)	- 300 min

X. APPENDIX B: LISTING OF COMPUTER PROGRAMS

A. Constructive Algorithm Subroutine: BRAYT

Directory

```
FTN4,Y,L
$EMA(DAT)
**********************************************************************
**********************************************************************
**                                                                  **
**                                                                  **
**        BRAYTON - TONG CONSTRUCTIVE ALGORITHM                     **
**                                                                  **
**                                                                  **
**                                                                  **
**    THIS FILE CONTAINS THE BRAYTON-TONG ALGORITHM                 **
**  SUBROUTINES THAT ARE USED TO FIND THE REGIONS OF                **
**  STABILITY FOR SECOND ORDER DIGITAL FILTERS.                     **
**                                                                  **
**********************************************************************
**********************************************************************
**
**
**    FILE &BSUP
**
**
**********************************************************************
**                                                                  **
**                          BRAYT                                   **
**                                                                  **
**********************************************************************
**
**
**      This Subroutine implements the Brayton - Tong
**    algorithm to find if a given set of extreme matrices
**    is stable
**
**
**    INPUTS:  EMAT   - Difference eqn extreme matrices
**             EDIM   - Dimension of system
**             NUMEXT - Number of extreme matrices
**             WZERO  - Initial vertex set
**             NZERO  - No. of vertices in initial vertex set
**             TRACE  - Set to nonzero value if want a trace
**                        of results
**
**    OUTPUTS: WSTAR  - Convex hull W* if stable system,
**                        otherwise it is the last convex
**                         hull generated
**             NSTAR  - Number of vertices in W*
**             SFLAG  - Stable/Unstable status of set of
**                        extreme matrices
```

```
**                        0 - stable
**                        1 - unstable
**                        2 - indeterminate (algorithm
**                             stopped because too many
**                             times through loop)
**                        3 - indeterminate (algorithm
**                             stopped because too many
**                             vertices in hull)
**                        4 - one extreme matrix has eigen-
**                             value outside unit circle
**                        5 - indeterminate (algorithm
**                             stopped because too many
**                             times in accelerated procedure
**              MXVRT  - Maximum number of vertices in any
**                        hull during course of algorithm.
**
**    PSEUDO-CODE:
**
**    If eigenvalues of extreme matrices are not outside
**       unit circle
**      Initialize previous-vertex-sets to initial-vertex-
**         set
**      Initialize next-vertex-set to initial-vertex-set
**      Initialize intersection-set to initial-vertex-set
**      Initialize vertex-angle array
**      Initialize array of 'angles'
**      Set K to 1
**      Set IEXT to 1
**      Set DONE to 'no'
**      Set stable-flag to 'stable'
**
**      Do while not done
**        Do for each vertex in W(k-1) while stable-flag is
**           stable
**          NEWPT = vertex of W(k-1)
**          If K <= number-of-extreme matrices or NEWPT is
**              not in intersection-set
**             Set New-point-flag to 'yes'
**             Set number-of-iterations to zero
**             Do while new points are being added and set
**                is stable
**               Increment number-of-iterations
**               If number-of-iterations > 10
**                 Do alternate accelerated procedure
**               Else
**                 Generate new vertex point, adding it to
**                    hull if possible.
**        End of processing one vertex in W(k-1)
```

```
**
**
**        If stable-flag is not 'stable'
**          Set DONE to 'yes'
**        Else
**          If K > number-of-extreme-matrices
**             If W(k) = W(k-m)    (m is num-of-ext-matrices)
**               Set DONE to 'yes'
**               Set stable-flag to 'stable'
**             Else
**               Set intersection-set to intersection of
**                  W(k) and W(k-m)
**
**          Adjust array of previous convex hulls
**
**          IEXT = IEXT + 1 mod m
**          K = K + 1
**
**          If too many convex hulls generated
**            Set DONE to 'yes'
**            Set stable-flag to 'indeterminate'
**
**      End of do while
**
**
**    LOCAL VARIABLES:
**
**      WSTAR  - Contains next convex hull as it is formed.
**      ANG    - Array of angles for WSTAR vertices, used to
**                 speed up adding a new vertex to set
**      WPREV  - Previous convex hulls.  In EMA storage
**                 WPREV(m,n,1) contains W(k-1)
**                 WPREV(m,n,2) contains W(k-2)  etc
**      PREV   - Contains W(k-m) so that EMA accesses do not
**                 have to be done by subordinate routines
**                 'EQVRT' and 'FRMIN'
**      NPREV  - Array containing no. of vertices in WPREV
**      V      - Intersection set. Contains the intersection
**                 of W(k) and W(k-m).  Any vertices in this
**                 set will not give any more new points for
**                 W(k), k > m
**      NV     - Number of vertices in V
**
**      K      - Convex hull number
**      NUMEXT - Number of extreme matrices
**      IEXT   - Current extreme matrix being used to
**                  generate W(k)
**      DONE   - Signals when it is time to exit loop
```

```
**      ITER    - Number of iterations through loop that gen-
**                erates new points for vertex set.  If a
**                certain threshold is reached, the
**                accelerated procedure is used.
**
**      J       - Number of vertices in W(k-1)
**      M,N     - Matrix index variables
**      MAXI    - Maximum number of iterations allowed
**                through big loop.
**
**
      SUBROUTINE BRAYT(EMAT, EDIM, NUMEXT, WZERO, NZERO,
     &    WSTAR, NSATR, SFLAG, MXVRT, TRACE)
**
**
      DOUBLE PRECISION WPREV
      COMMON/DAT/ WPREV(2,256,16)
**
**
      DOUBLE PRECISION EMAT(3,3,16), WZERO(2,10),
     &                 PREV(2,256), V(2,256), NEWPT(3),
     &                 WSTAR(2,256), ANG(256)
      INTEGER EDIM, NZERO, NSTAR, NPREV(16), TRACE
      INTEGER SFLAG, I, NUMEXT, IEXT, ITER, DONE
      INTEGER NV, K, J, M, N, II, IJ, IK, MAXI, MXVRT
**
**    The following are function subprograms
      INTEGER EQVRT, INVRT, CKVRT, VRTEL
**
**
**    Calculate maximum number of iterations allowed
      MAXI = NUMEXT * 80
**
**
**    Tracing function
      IF (TRACE.EQ.0) GOTO 110
      WRITE(6,22)
      CALL PVRT(WZERO, NZERO, EDIM)
      DO 105 M = 1, NUMEXT
      WRITE(6,2) M
      DO 102 I=1,EDIM
102   WRITE(6,6) (EMAT(I,J,M),J=1,EDIM)
105   CONTINUE
**
**    Check to make sure eigenvalues are not outside unit
**       circle
110   CALL EIGEN(EMAT, EDIM, NUMEXT, SFLAG, TRACE)
      IF (SFLAG.NE.0) GOTO 510
```

```
**
**
**     Init WPREV, WSTAR, and V to WZERO
       DO 120 J=1,NZERO
       DO 120 I=1,EDIM
       DO 115 M=1,NUMEXT
115    WPREV(I,J,M) = WZERO(I,J)
       WSTAR(I,J) = WZERO(I,J)
120    V(I,J) = WZERO(I,J)
       DO 125 M=1, NUMEXT
125    NPREV(M) = NZERO
       NSTAR = NZERO
       NV = NZERO
**
**     Other initialization
       CALL AINIT(WSTAR, NSTAR, EDIM, ANG)
       K = 1
       IEXT = 1
       DONE = 0
       SFLAG = 0
**
**     Do while not done
130    IF (DONE.NE.0) GOTO 500
**
**       Tracing function
         IF (TRACE.EQ.0) GOTO 135
         WRITE(6,4) K, IEXT
         WRITE(6,8)
         CALL PVRT(V, NV, EDIM)
135      CONTINUE
**
**       Do for each vertex in W(k-1) while stable-flag is
**         "stable"
         J = 1
140      IF ((J.GT.NPREV(1)).OR.(SFLAG.NE.0)) GOTO 300
**
**         Set NEWPT to point J of W(k-1)
           DO 150 M = 1, EDIM
150          NEWPT(M) = WPREV(M,J,1)
**
**         If K <= NUMEXT or point J of W(k-1) is not in V
           IF ((K.GT.NUMEXT).AND.
     &         (VRTEL(NEWPT,EDIM,V,NV).EQ.1))  GOTO 210
**
**           Initialization before next loop
             NEWFLG = 1
             ITER = 0
**
```

```
            IF (TRACE.NE.0) WRITE(6,18) (NEWPT(M), M=1,EDIM)
**
**          Do while new points are being added and system
**           stable
160         IF ((NEWFLG.EQ.0).OR.(SFLAG.NE.0)) GOTO 210
              ITER = ITER + 1
**            Check if need to do accelerated procedure
              IF (ITER.LE.10) GOTO 170
                CALL ACCEL(EMAT(1,1,IEXT), EDIM, WSTAR,
     &            NSTAR, ANG, WZERO, NZERO, NEWPT, TRACE,
     &            MXVRT, SFLAG)
                NEWFLG = 0
                GOTO 200
**
**            Regular procedure
170           CALL GENPT(EMAT(1,1,IEXT), EDIM, WSTAR, NSTAR,
     &          ANG, WZERO, NZERO, NEWPT, TRACE, MXVRT,
     &          SFLAG, NEWFLG)
**
200           GOTO 160
**
**
**        Increment to next point of W(k-1)
210       J = J + 1
**
**        Tracing function
***       IF (TRACE.EQ.0) GOTO 220
***       WRITE(6,24)
***       CALL PVRT(WSTAR, NSTAR, EDIM)
**
220       GOTO 140
**
**      End of processing vertices in W(k-1)
300     CONTINUE
**
**      If stable-flag <> "stable" then set DONE so that we
**       exit
        IF (SFLAG.EQ.0) GOTO 310
          DONE = 1
          GOTO 400
**
**      Else if K > number-of-extreme-matrices, check for
**         stability and get intersection-set
310       IF (K.LE.NUMEXT) GOTO 340
            M = NUMEXT
**
**          Set PREV array to k-m hull
```

```
            DO 320 IJ = 1, NPREV(M)
            DO 320 II = 1, EDIM
320         PREV(II,IJ) = WPREV(II, IJ, M)
**
**          If current convex hull is same as k-m hull,
**            exit stable
            IF (EQVRT(PREV(1,1),NPREV(M),WSTAR,NSTAR,EDIM)
     &           .NE.1)  GOTO 330
              DONE = 1
              SFLAG = 0
              GOTO 340
**
**          Form intersection-set
330           CALL FRMIN(WSTAR, NSTAR, PREV(1,1), NPREV(M),
     &          EDIM, V, NV)
**
**
340       CONTINUE
**
**        Adjust WPREV array backwards
          IF (NUMEXT.LT.2) GOTO 380
            DO 370 IK = NUMEXT ,2, -1
            NPREV(IK) = NPREV(IK-1)
            DO 360 IJ = 1, NPREV(IK)
            DO 360 II = 1, EDIM
360         WPREV(II,IJ,IK) = WPREV(II,IJ,IK-1)
370         CONTINUE
**
**        Put WSTAR in WPREV( , ,1)
380       NPREV(1) = NSTAR
          DO 390 IJ = 1, NPREV(1)
          DO 390 II = 1, EDIM
390       WPREV(II,IJ,1) = WSTAR(II,IJ)
**
**
**
**        Increment hull number and extreme matrix number
          IEXT = IEXT+1
          IF (IEXT.GT.NUMEXT) IEXT = 1
          K = K + 1
**
**        Check for slowly converging solution
          IF (K.LE.MAXI) GOTO 400
            DONE = 1
            SFLAG = 2
**
**
**    End of do while
```

```
400   CONTINUE
**
**    Tracing function
      IF ((TRACE.EQ.0).OR.(DONE.NE.0)) GOTO 295
      M = K-1
      WRITE(6,10) M
      CALL PVRT(WSTAR, NSTAR, EDIM)
C
295   GOTO 130
**
**
**
**    Exit
500   CONTINUE
      IF (TRACE.EQ.0) GOTO 510
      WRITE(6,12)
      CALL PVRT(WSTAR, NSTAR, EDIM)
510   IF (TRACE.NE.0) WRITE(6,14) SFLAG
      RETURN
**
**
2     FORMAT('0EXTREME MATRIX ',I2)
4     FORMAT(///' ******  START OF ITERATION ',I4,' *******'/,
     &       ' PROCESSING WITH EXTREME MATRIX ',I2)
6     FORMAT(' ',4F10.4)
8     FORMAT('0INTERSECTION SET V IS')
10    FORMAT('0FINAL HULL FOR ITERATION ',I2)
12    FORMAT('0 FINAL CONVEX HULL IS ')
14    FORMAT('0 STABILITY STATUS IS ',I2,//)
18    FORMAT('0#######  NOW PROCESSING VERTEX ',4F10.4)
22    FORMAT('0INITIAL CONVEX HULL')
24    FORMAT('0TEMPORARY HULL IS NOW')
      END
**
**
```

```
**********************************************************************
**                                                                  **
**                            GENPT                                 **
**                                                                  **
**********************************************************************
**
**
**      Generates new vertex point, given the extreme
**    matrix and current vertex. If it is outside the
**    current hull, it is added and stability is checked.
**
**    INPUTS:  EMAT   - Extreme matrix
**             EDIM   - Dimension of system
**             WSTAR  - Convex hull
**             NSTAR  - Number in hull
**             ANG    - Array of angles for WSTAR
**             WZERO  - Initial vertex set
**             NZERO  - Number of vertices in initial vertex
**                      set
**             NEWPT  - Current vertex
**             TRACE  - Set to nonzero value if want a trace
**                      of results
**
**    OUTPUTS: WSTAR  -
**             NSTAR  -
**             NEWPT  -
**             MXVRT  - Maximum number of vertices in any
**                      hull during course of algorithm.
**             SFLAG  - Stable/Unstable status
**             NEWFLG - Tells if new point added to hull
**                        0 - not added
**                        1 - added to hull
**
**    PSEUDO-CODE:
**
**     NEWPT = Extreme-matrix * NEWPT
**     If NEWPT is outside current hull
**       Set new-point-added-flag to 'yes'
**       Add new point to next-vertex-set
**       If not overflow
**         If next-vertex-set and initial-vertex-set do
**            not intersect,
**           Set stable-flag to 'unstable'
**     Else
**       Set new-point-added-flag to 'no'
**
**
```

```
      SUBROUTINE GENPT(EMAT, EDIM, WSTAR, NSTAR, ANG, WZERO,
     &    NZERO, NEWPT, TRACE, MXVRT, SFLAG, NEWFLG)
**
**
      DOUBLE PRECISION EMAT(3,3), WZERO(2,10), WSTAR(2,256)
      DOUBLE PRECISION NEWPT(6), ANG(256)
      INTEGER EDIM, NZERO, NSTAR, TRACE
      INTEGER SFLAG, MXVRT, NEWFLG
**
**    Function subprograms
      INTEGER CKVRT, INVRT
**
**
      CALL MMULT(EMAT, NEWPT, EDIM, EDIM, 1, NEWPT)
      IF (TRACE.NE.0) WRITE(6,20) (NEWPT(M),M=1,EDIM)
**
**    If not on or in convex hull, put it in
171   IF (CKVRT(NEWPT, WSTAR, NSTAR, EDIM).EQ.1) GOTO 190
        NEWFLG = 1
        IF ((NSTAR+1).GT.MXVRT) MXVRT = NSTAR + 1
        CALL ADDVT(NEWPT, WSTAR, NSTAR, ANG, EDIM, TRACE,
     &             SFLAG)
**      If not overflow, check for instability
        IF (SFLAG.NE.0) GOTO 189
          IF (INVRT(WZERO, NZERO, WSTAR, NSTAR, EDIM).EQ.0)
     &        SFLAG = 1
189     GOTO 200
**
**    Else in convex hull
190     NEWFLG = 0
        IF (TRACE.NE.0) WRITE(6,16)
200   RETURN
**
**
**
16    FORMAT('  POINT NOT ADDED ')
20    FORMAT(' NEW POSSIBLE POINT IS ',4F10.4)
      END
**
**
```

```
**********************************************************************
**                                                                  **
**                            VRTEL                                 **
**                                                                  **
**********************************************************************
**
**      Checks to see if a given point is in the vertex set
**
**    INPUTS:  POINT  - Coordinates of point to check
**             IDIM   - Dimension of points
**             SET    - Vertex set
**             NSET   - Number of vertices in set
**
**    OUTPUT:  VRTEL  - Set to 1 if point is found in vertex
**                        set, otherwise it is 0.
**
**    PSEUDO-CODE:
**
**      Set VRTEL to "not found"
**      Do for each vertex in set while not found
**        If vertex is close to POINT
**          Set VRTEL to "found"
**      End
**
**
      INTEGER FUNCTION VRTEL(POINT, IDIM, SET, NSET)
**
**
      DOUBLE PRECISION SET(2,256), POINT(3), ERR
      INTEGER IDIM, NSET, I
**    Function subroutine
      DOUBLE PRECISION PDIFF
**
**
      ERR = 1.D-8
**
      VRTEL = 0
      I = 1
110   IF ((I.GT.NSET).OR.(VRTEL.EQ.1)) GOTO 120
        IF(PDIFF(SET(1,I), POINT, IDIM).LE.ERR) VRTEL = 1
        I = I + 1
        GOTO 110
120   CONTINUE
**
**
      RETURN
      END
```

```
**********************************************************************
**                                                                  **
**                            EQVRT                                 **
**                                                                  **
**********************************************************************
**
**
**      Checks to see if the two convex hull vertex sets are
**    equivalent ("close").
**
**    INPUTS:  SETA   - Convex set A
**             NUMA   - Number of vertices in set A
**             SETB   - Convex set B
**             NUMB   - Number of vertices in set B
**             IDIM   - Dimension of points in a vertex
**
**    OUTPUTS: EQVRT  - Result of check
**                        0 - not equivalent
**                        1 - equivalent
**
**
**    PSEUDO-CODE:
**
**        If number of vertices in each set are not equal
**          Set EQVRT to not-equivalent
**        Else
**          Set EQVRT to equivalent
**          Do for each vertex while still equivalent
**            If magnitude of vector difference between
**               respective vertices is larger than allow-
**               able error then
**              Set EQVRT to not-equivalent
**          End
**
**
      INTEGER FUNCTION EQVRT(SETA, NUMA, SETB, NUMB, IDIM)
**
**
      DOUBLE PRECISION SETA(2,256), SETB(2,256), ERR
      INTEGER NUMA, NUMB, IDIM, I
**    This is a function subprogram
      DOUBLE PRECISION PDIFF
**
**
      ERR = 1.D-8
**
**    If number of vertices not equal, exit
```

```
      IF (NUMA.EQ.NUMB) GOTO 110
         EQVRT = 0
         GOTO 130
**
**    Else check each vertex while sets are still equivalent
110      EQVRT = 1
         I = 1
120      IF((EQVRT.NE.1).OR.(I.GT.NUMA))  GOTO 130
            IF (PDIFF(SETA(1,I), SETB(1,I), IDIM).GT.ERR)
     &         EQVRT = 0
            I = I + 1
         GOTO 120
**
**
130   CONTINUE
**
      RETURN
      END
**
**
```

```
**********************************************************************
**                                                                  **
**                            INVRT                                 **
**                                                                  **
**********************************************************************
**
**
**      Checks to see if the two convex hull vertex sets
**    intersect.
**
**    INPUTS:  SETA   - Convex set A
**             NUMA   - Number of vertices in set A
**             SETB   - Convex set B
**             NUMB   - Number of vertices in set B
**             IDIM   - Dimension of points in a vertex
**
**    OUTPUTS: INVRT  - Result of check
**                        0 - do not intersect
**                        1 - intersect
**
**
**
**    PSEUDO-CODE:
**
**        Set INVRT to no-intersection
**        Do for each vertex in set A while no intersection
**          For each vertex in set B
**            If magnitude of vector difference between
**               vertices is smaller than allowable error
**              Set INVRT to intersection
**          End
**        End
**
      INTEGER FUNCTION INVRT(SETA, NUMA, SETB, NUMB, IDIM)
**
**
      DOUBLE PRECISION SETA(2,256), SETB(2,256), ERR
      INTEGER NUMA, NUMB, IDIM, I,J
**    This is a function subprogram
      DOUBLE PRECISION PDIFF
**
**
      ERR = 1.D-8
**
**
**    Check each vertex while no intersection
      INVRT = 0
      I = 1
```

```
100   IF((INVRT.NE.0).OR.(I.GT.NUMA))  GOTO 200
**
**
         DO 110 J = 1,NUMB
            IF (PDIFF(SETA(1,I), SETB(1,J), IDIM).LE.ERR)
     &           INVRT = 1
110      CONTINUE
**
         I = I + 1
      GOTO 100
**
**
200   CONTINUE
**
      RETURN
      END
**
**
```

```
**********************************************************************
**                                                                  **
**                          FRMIN                                   **
**                                                                  **
**********************************************************************
**
**
**      Gets the intersection of the two convex hull vertex
**    sets.
**
**    INPUTS:  SETA    - Convex set A
**             NUMA    - Number of vertices in set A
**             SETB    - Convex set B
**             NUMB    - Number of vertices in set B
**             IDIM    - Dimension of points in a vertex
**
**    OUTPUTS: SETZ    - Intersection of the two sets
**             NUMZ    - Number of vertices in SETZ
**
**
**    PSEUDO-CODE:
**
**        Set Number-in-SETZ to zero
**        Do for each vertex in set A
**          Set intersection-flag to "no"
**          For each vertex in set B while no intersection
**            If magnitude of vector difference between
**               vertices is smaller than allowable error
**              Increment number-in-SETZ
**              Place vertex in SETZ
**              Set intersection-flag to "yes"
**          End
**        End
**
**
      SUBROUTINE FRMIN(SETA, NUMA, SETB, NUMB, IDIM, SETZ,
     &                 NUMZ)
**
**
      DOUBLE PRECISION SETA(2,256), SETB(2,256),SETZ(2,256),
     &                 ERR
      INTEGER NUMA, NUMB, NUMZ, IDIM, I,J, INFLG
**    This is a function subprogram
      DOUBLE PRECISION PDIFF
**
**
      ERR = 1.D-8
**
```

```
      NUMZ = 0
**
**    Check each vertex in set A
      DO 200 I = 1, NUMA
**
**       Check each vertex in set B while no intersection
         INFLG = 0
         J = 1
110      IF((J.GT.NUMB).OR.(INFLG.NE.0)) GOTO 150
            IF (PDIFF(SETA(1,I), SETB(1,J), IDIM).GT.ERR)
     &         GOTO 140
           NUMZ = NUMZ + 1
           DO 130 K = 1, IDIM
130        SETZ(K,NUMZ) = SETA(K,I)
           INFLG = 1
140      CONTINUE
         J = J + 1
         GOTO 110
**
150      CONTINUE
200   CONTINUE
**
      RETURN
      END
**
**
```

```
**********************************************************************
**                                                                  **
**                            PDIFF                                 **
**                                                                  **
**********************************************************************
**
**
**    Finds magnitude of difference between two vertices.
**      PDIFF = SQUARE ROOT((A1-B1)**2 + (A2-B2)**2 + . . )
**
**
**    INPUTS:  A      - Point A
**             B      - Point B
**             NUM    - Dimension of point
**
**    OUTPUTS: PDIFF - Magnitude of difference between
**                      points.
**
**
      DOUBLE PRECISION FUNCTION PDIFF(A,B,NUM)
**
**
      DOUBLE PRECISION A(10), B(10), SUM
      INTEGER NUM, I
**
**
      SUM = 0.
      DO 100 I = 1,NUM
100   SUM = SUM + (A(I)-B(I))**2
**
      PDIFF = DSQRT(SUM)
**
      RETURN
      END
**
**
```

```
**********************************************************************
**                                                                  **
**                            AINIT                                 **
**                                                                  **
**********************************************************************
**
**       Initialize vertex angle array
**       ONLY WORKS FOR SECOND ORDER SYSTEMS
**
**     INPUTS:  HULL   - Convex hull
**              NHULL  - Number in hull
**              IDIM   - Dimension of points
**
**     OUTPUTS: ANGLE  - Array of angles of vertices in hull
**
**
      SUBROUTINE AINIT(HULL, NHULL, IDIM, ANGLE)
**
**
      DOUBLE PRECISION HULL(2,256), ANGLE(256)
      INTEGER NHULL, IDIM, I
**
**
      DO 100 I = 1, NHULL
100   ANGLE(I) = DATN2(HULL(2,I), HULL(1,I))
**
      RETURN
      END
**
**
```

```
**********************************************************************
**                                                                  **
**                            MPOWR                                 **
**                                                                  **
**********************************************************************
**
**      Raise matrix to a positive power.
**
**    INPUTS:  A      - Matrix
**             IDIM   - Number of rows and cols in A
**             IPWR   - Matrix is raised to this number
**
**    OUTPUTS: RSLT   - Result
**
**
**
      SUBROUTINE MPOWR(A, IDIM, IPWR, RSLT)
**
      DOUBLE PRECISION A(3,3), RSLT(3,3), TEMP(3,3)
      INTEGER IDIM, IPWR, I, J
**
**    Set TEMP to input matrix
      DO 100 I = 1, IDIM
      DO 100 J = 1, IDIM
100   TEMP(I,J) = A(I,J)
**
**    If power is not 1, then do multiplications
      IF (IPWR.LE.1) GOTO 140
      DO 120 I = 1, IPWR-1
120   CALL MMULT(A, TEMP, IDIM, IDIM, IDIM, TEMP)
**
140   CONTINUE
**
**    Now form result
      DO 150 I = 1, IDIM
      DO 150 J = 1, IDIM
150   RSLT(I,J) = TEMP(I,J)
**
**
      RETURN
      END
**
**
```

```
**********************************************************************
**                                                                  **
**                          MMULT                                   **
**                                                                  **
**********************************************************************
**
**
**    Multiplies two matrices together:  C = A * B
**
**    INPUTS:  A,B    - Matrices to be multiplied
**                        A: ROWA  rows
**                           COLA  columns
**                        B: COLA  rows
**                           COLB  columns
**
**    OUTPUTS: C      - Resultant matrix
**                        ROWA rows
**                        COLB columns
**
      SUBROUTINE MMULT(A, B, ROWA, COLA, COLB, C)
**
**
      DOUBLE PRECISION A(3,3), B(3,3), C(3,3), RSLT(3,3),SUM
      INTEGER I, J, K, ROWA, COLA, COLB
**
**
      DO 160 I=1, ROWA
         DO 140 J=1, COLB
            SUM = 0.
               DO 120 K=1,COLA
                  SUM = SUM + A(I,K)*B(K,J)
120            CONTINUE
            RSLT(I,J) = SUM
140      CONTINUE
160   CONTINUE
**
      DO 180 I = 1, ROWA
      DO 180 J = 1, COLB
180      C(I,J) = RSLT(I,J)
**
      RETURN
      END
**
**
```

```
**********************************************************************
**                                                                  **
**                            ADDVT                                 **
**                                                                  **
**********************************************************************
**
**
**      Add a vertex to convex-hull
**      ONLY WORKS FOR SECOND ORDER SYSTEMS
**
**    INPUTS:  POINT  - Vertex to add to hull
**             HULL   - vertex set
**             NHULL  - Number of vertices in vertex set
**             ANGLE  - Array of angles of points
**             IDIM   - Dimension of points in convex hull
**             TRACE  - Nonzero if results are to be traced
**
**    OUTPUTS: HULL   - New vertex set
**             NHULL  - Number of vertices in new vertex set
**             OVRFLW - Overflow flag
**                        0 - no overflows occured
**                        3 - too many vertices in set
**
**
**    PSEUDO-CODE:
**
**    If too many vertices
**      Set overflow indication
**    Else
**      Clear overflow indication
**      Calculate angle of new-point
**      Do for all vertices in HULL while place for new
**          point not found
**        If angle of new-point < angle of vertex
**          Set found indication to "yes"
**      End
**
**      If place found
**        Do from last vertex back to place at which to
**           insert new-point
**          Next vertex = Current vertex
**          Next angle  = Current angle
**        End
**      Else
**        Set place to put point at end of array
**
**      Put new-point in place
**      Put angle of new-point in place
```

```
**      Increment number in hull
**
**      Eliminate any vertices that do not form a
**         convex hull
**
**
**
**
**
      SUBROUTINE ADDVT(POINT, HULL, NHULL, ANGLE, IDIM,
     &                 TRACE, OVRFLW)
**
**
      DOUBLE PRECISION HULL(2,256), ANGLE(256)
      DOUBLE PRECISION POINT(4), PTANG
      INTEGER NHULL,IDIM, OVRFLW, I, J, TRACE, IVERT, FOUND
**
**
**    Tracing function
      IF (TRACE.NE.0) WRITE(6,2)
2     FORMAT('  POINT ADDED TO HULL')
**
**    Check for overflow
100   IF (NHULL.NE.256) GOTO 110
        OVRFLW = 3
        GOTO 200
**
**
110     CONTINUE
        OVRFLW = 0
**
**      Calculate angle for new point
        PTANG = DATN2(POINT(2), POINT(1))
**
**      Do for each vertex in hull while place not found
        FOUND = 0
        I = 1
120     IF ((I.GT.NHULL).OR.(FOUND.NE.0)) GOTO 130
          IF (PTANG.LT.ANGLE(I)) FOUND = 1
          I = I + 1
          GOTO 120
**
**      Place to put new point is one less than last
**        index "I", if found
130     IF (FOUND.EQ.0) GOTO 155
        IVERT = I - 1
**
```

```
**      Move vertices and angles to make room for new point
        DO 150 J = NHULL, IVERT, -1
          DO 140 I = 1, IDIM
140         HULL(I, J+1) = HULL(I, J)
150       ANGLE(J+1) = ANGLE(J)
        GOTO 160
**
**      If not found, set place to end of array
155     IVERT = I
**
**      Put new point in its place
160     DO 170 I = 1, IDIM
170     HULL(I,IVERT) = POINT(I)
        ANGLE(IVERT) = PTANG
**
**      Increment number in hull
        NHULL = NHULL + 1
**
**
**      Eliminate vertices that do not form a convex hull
        CALL ELIMN(HULL, NHULL, ANGLE, IDIM, TRACE)
200   CONTINUE
      RETURN
      END
**
**
```

```
**********************************************************************
**                                                                  **
**                            CKVRT                                 **
**                                                                  **
**********************************************************************
**
**
**
**      Check point to see if it is within or on the given
**    convex hull.
**      ONLY WORKS FOR SECOND ORDER SYSTEMS
**
**    INPUTS:  POINT  - Coordinates of point
**             WHULL  - Vertices of convex hull
**             NHULL  - Number of vertices in hull
**             IDIM   - Dimension of points
**
**    OUTPUTS: CKVRT  - Result of check
**                        0 - Not in convex hull
**                        1 - In convex hull
**
**
**    PSEUDO-CODE:
**
**      If point is in third or fourth quadrant, reflect
**         into first or second quadrant
**
**      Do for each vertex in convex hull while point still
**         between hull boundary and origin.
**        Check if point is between this segment of hull and
**           origin.
**           (If first vertex, a reflection of the last
**           vertex must be used.  If last vertex, a reflec-
**           tion of the first vertex must be used)
**      End
**
**
**
      INTEGER FUNCTION CKVRT(POINT, WHULL, NHULL, IDIM)
**
**
      DOUBLE PRECISION POINT(4), WHULL(2,256), TEMP(4)
      INTEGER NHULL, IDIM, I, J
**      This is a function subprogram
      INTEGER SANGL
**
**
**
```

```
**    Put point in first or second quadrant
      IF (POINT(2).GE.0.) GOTO 99
        DO 98 I = 1, IDIM
98      POINT(I) = -POINT(I)
99    CONTINUE
**
**    Initialize loop variables
      CKVRT = 1
      I = 0
**
**    Do while I <= NHULL and CKVRT = In-hull
105   IF((I.GT.NHULL).OR.(CKVRT.NE.1)) GOTO 180
**
**        Check for first vertex
          IF (I.NE.0) GOTO 120
            DO 110 J = 1, IDIM
110           TEMP(J) = -WHULL(J,NHULL)
            CKVRT = SANGL(TEMP, WHULL(1,1), POINT, IDIM)
            GOTO 150
**
**        Check if last vertex
120         IF (I.NE.NHULL) GOTO 140
              DO 130 J = 1, IDIM
130             TEMP(J) = -WHULL(J,1)
              CKVRT = SANGL(WHULL(1,I), TEMP, POINT,
     &                      IDIM)
              GOTO 150
**
**          Check for any other vertex
140           CKVRT = SANGL(WHULL(1,I), WHULL(1,I+1),
     &                      POINT, IDIM)
**
150       I = I + 1
          GOTO 105
**
180   CONTINUE
      RETURN
      END
**
**
```

```
**********************************************************************
**                                                                  **
**                             SANGL                                **
**                                                                  **
**********************************************************************
**
**
**      This routine is used to find if the point C is to
**    the left of the vector determined by AB, or if the
**    point is on this vector.  If the point is coincident
**    with A or B then it is assumed to lie on the vector.
**    If the sign of the angle between the vectors deter-
**    mined by AB and AC is positive, then the point is to
**    the left of the vector. A positive angle is measured
**    counterclockwise.  If the angle is near zero or +-180
**    then the point is assumed to be colinear with the
**    vector through A,B.
**
**      ROUTINE ONLY WORKS FOR SECOND ORDER SYSTEMS.
**
**
**    INPUTS:  A,B,C  - Cordinates of the points A, B, and C
**             IDIM   - Dimension of the points
**
**    OUTPUTS: SANGL  - Indication of point C
**                        1 - C is on the line or to the
**                            left of the vector determined
**                            by points A and B.
**                        0 - C is not close or to the left
**                            of the vector.
**
**    PSEUDO-CODE:
**
**      If point C is close to A or B then
**        SANGL = 1
**      Else
**        ANGLE1 = angle of vector AB with x-axis
**        ANGLE2 = angle of vector AC with x-axis
**        Angle from AB to AC = ANGLE2 - ANGLE1
**
**        If Angle >  PI then Angle = Angle - 2*PI
**        If Angle < -PI then Angle = Angle + 2*PI
**
**        If (Angle > 0 and Angle < PI) or (|Angle| < Error)
**          or (|Angle-PI| < Error) or (|Angle+PI| < Error)
**          then SANGL = 1
**        Else SANGL = 0
**
```

```
**
      INTEGER FUNCTION SANGL(A, B, C, IDIM)
**
**
      DOUBLE PRECISION A(4), B(4), C(4)
      DOUBLE PRECISION ANGLE1, ANGLE2, ANGLE, ERR, PI
      INTEGER IDIM
**    This is a function subroutine
      DOUBLE PRECISION PDIFF
**
**
      ERR = 1.D-8
      PI = 3.14159265359D0
**
**    If point C is close to A or B, set flag and return
      IF((PDIFF(A,C,IDIM).GT.ERR).AND.
     &   (PDIFF(B,C,IDIM).GT.ERR))   GOTO 100
         SANGL = 1
         GOTO 150
**
**    Else calculate angles
100     ANGLE1 = DATN2(B(2)-A(2), B(1)-A(1))
        ANGLE2 = DATN2(C(2)-A(2), C(1)-A(1))
        ANGLE = ANGLE2 - ANGLE1
**
**      Make sure angle in range from -PI to +PI
        IF (ANGLE.GT.PI)   ANGLE = ANGLE - 2*PI
        IF (ANGLE.LT.-PI)  ANGLE = ANGLE + 2*PI
**
**      and now, for the BIG test.....
        SANGL = 0
        IF( ((ANGLE.GT.0.).AND.(ANGLE.LT.PI)).OR.
     &    (DABS(ANGLE).LT.ERR).OR.(DABS(ANGLE-PI).LT.ERR).OR.
     &    (DABS(ANGLE+PI).LT.ERR) )   SANGL = 1
**
150   CONTINUE
      RETURN
      END
**
**
```

```
**********************************************************************
**                                                                  **
**                            ELIMN                                 **
**                                                                  **
**********************************************************************
**
**
**      Eliminates any vertices in set that do not form a
**    convex hull, i.e. if the vertex is contained in the
**    convex hull of the rest of the vertices.  Practically
**    speaking we only need to check if that vertex is
**    contained in the hull if a line were drawn from the
**    previous vertex to the next vertex
**      The vertices must be sorted in order upon entry to
**    this routine.
**
**    INPUTS:  VSET   - Vertex set
**             NVERT  - Number of vertices in set
**             IDIM   - Dimension of points
**             ANGLE  - Array of vertex angles
**             TRACE  - Nonzero if results are to be traced
**
**    OUTPUTS: VSET   - Vertex set that forms a convex hull
**             NVERT  - Number of vertices in convex hull
**
**    PSEUDO-CODE:
**
**      Do while a vertex is eliminated
**        Do for each vertex in the set while none
**           eliminated
**          If first vertex then
**            Set TEMP vertex to reflection of last vertex
**            Check if first vertex is inside hull
**              determined by TEMP and second vertex
**          Else
**            If last vertex then
**              Set TEMP vertex to reflection of first
**                 vertex.
**              Check if last vertex is inside the hull
**                 determined by previous vertex and TEMP.
**            Else
**              Check if vertex is inside hull determined
**                 by previous and next vertex
**
**          If vertex not to be eliminated
**            Increment vertex set index
**        End
**
```

```
**        If a vertex is to be eliminated
**          If number of vertices is less than 2 then
**            Set eliminate-flag to "no" so that this
**               subroutine will be terminated.
**          Else
**            Decrement number in set
**            Starting at vertex to be eliminated
**              Move next vertex to the current position in
**                 set.
**              Move next angle to the current position in
**                 the array
**
**
**    LOCAL VARIABLES:
**
**    EFLAG  - Indicates if a vertex is to be eliminated
**                0 - not eliminated
**                1 - eliminated
**    TEMP   - Coordinates of vertex if it is before first
**               vertex in set or after last vertex.
**    I,J,K  - Iteration variables
**
**
      SUBROUTINE ELIMN(VSET, NVERT, ANGLE, IDIM, TRACE)
**
**
      DOUBLE PRECISION VSET(2,256), ANGLE(256), TEMP(4)
      INTEGER NVERT, IDIM, I, J, K, EFLAG, TRACE
**
**    This is a function subroutine
      INTEGER SANGL
**
**
**    Set flag to indicate 'eliminated'
      EFLAG = 1
**
**    Start of big do while loop
100   IF (EFLAG.NE.1) GOTO 200
        EFLAG = 0
        I = 1
**
**      Do check for each vertex in set while none
**         eliminated
105     IF((I.GT.NVERT).OR.(EFLAG.NE.0)) GOTO 180
**

**        Check if first vertex
          IF (I.NE.1) GOTO 120
```

```
            DO 110 J = 1, IDIM
110           TEMP(J) = -VSET(J,NVERT)
            EFLAG = SANGL(TEMP, VSET(1,2), VSET(1,1), IDIM)
            GOTO 150
**
**        Check if last vertex
120         IF (I.NE.NVERT) GOTO 140
              DO 130 J = 1, IDIM
130             TEMP(J) = -VSET(J,1)
              EFLAG = SANGL(VSET(1,I-1), TEMP, VSET(1,I),
     &                      IDIM)
              GOTO 150
**
**          Check for any other vertex
140           EFLAG = SANGL(VSET(1,I-1), VSET(1,I+1),
     &                      VSET(1,I), IDIM)
**
**        Increment index count if nothing to be eliminated
150       IF (EFLAG.EQ.0) I = I + 1
**
**        Back to top of loop
          GOTO 105
**
**      Eliminate vertex if it is to be eliminated
180     IF (EFLAG.EQ.0) GOTO 195
          IF (NVERT.GE.2) GOTO 183
            EFLAG = 0
            GOTO 195
**
**          Trace eliminated vertex
183         IF (TRACE.EQ.0) GOTO 185
              WRITE(6,2) I,(VSET(J,I),J=1,IDIM)
2             FORMAT('0ELIMINATE ',I2,4F10.4,/)
**
**          Eliminate vertex
185         NVERT = NVERT - 1
            DO 193 J = I, NVERT
              DO 190 K = 1, IDIM
190             VSET(K,J) = VSET(K,J+1)
193           ANGLE(J) = ANGLE(J+1)
**
195     CONTINUE
**      Back to top of loop
        GOTO 100
**
200   CONTINUE
210   RETURN
      END
```

```
**********************************************************************
**                                                                  **
**                          EIGEN                                   **
**                                                                  **
**********************************************************************
**
**      Checks that the eigenvalues of the extreme matrices
**    are all on or within the unit circle.
**
**      ONLY WORKS FOR SECOND ORDER SYSTEMS !
**
**    INPUTS:  AMAT   - Matrices to check
**             IDIM   - Dimension of system
**             NMAT   - Number of matrices
**             TRACE  - Set to nonzero value if want trace
**                       of results
**
**    OUTPUTS: EFLAG  - Set to 4 if one eigenvalue outside
**                       unit circle
**                      Set to 0 otherwise
**
**    PSEUDO-CODE:
**
**      Set EFLAG to "stable"
**
**      Do for each matrix while EFLAG is "stable"
**        Calculate coefficients of polynomial
**        Calculate discriminate
**        If roots imaginary
**          Largest-root = magnitude of root
**        Else
**          Largest-root = maximum of absolute value of
**             both roots
**
**        If Largest-root > 1 then EFLAG = "unstable"
**      End
**
**
**
      SUBROUTINE EIGEN(AMAT, IDIM, NMAT, EFLAG, TRACE)
**
**
      DOUBLE PRECISION AMAT(3,3,2), A, B, C, DIS, LROOT,
     &                 RONE, RTWO
      INTEGER NMAT, IDIM, EFLAG, I, TRACE
**
**
```

```
**    Set EFLAG to "stable"
      EFLAG = 0
**
**    Check each matrix while still stable
      I = 1
110   IF ((I.GT.NMAT).OR.(EFLAG.NE.0)) GOTO 170
**
**      Calculate polynomial coefficients
        A = 1.D0
        B = -AMAT(1,1,I)-AMAT(2,2,I)
        C = AMAT(1,1,I)*AMAT(2,2,I)-AMAT(1,2,I)*AMAT(2,1,I)
**
**      Calculate discriminate
        DIS = B*B - 4.D0*A*C
        IF (DIS.GE.0.D0) GOTO 130
**
**        Imaginary roots
          LROOT = (DSQRT(B*B-DIS))/(2.D0*DABS(A))
          GOTO 150
**
**        Real roots
130       RONE = (-B + DSQRT(DIS))/(2.D0*A)
          RTWO = (-B - DSQRT(DIS))/(2.D0*A)
          LROOT = DMAX1(DABS(RONE), DABS(RTWO))
**
**      Check eigenvalues
150     IF (LROOT.GT.1.D0) EFLAG = 4
**
        I = I + 1
        GOTO 110
**
**    Tracing function
170   IF (TRACE.EQ.0) GOTO 180
        I = I - 1
        IF (EFLAG.NE.0) WRITE(6,2) I
        IF (EFLAG.EQ.0) WRITE(6,4)
180   RETURN
**
**
2     FORMAT('0EXT MATRIX ',I2,' HAS EIGENVALUE OUTSIDE',
     &       ' UNIT CIRCLE.')
4     FORMAT('0NO EXT MATRIX HAS EIGENVALUE OUTSIDE',
     &       ' UNIT CIRCLE.')
      END
**
**
```

```
**********************************************************************
**                                                                  **
**                            PVRT                                  **
**                                                                  **
**********************************************************************
**
**
**      Prints the vertex set.
**
**    INPUTS:  VERT   - Set of vertices
**             NVERT  - Number of vertices
**             IDIM   - Dimension of points
**
**
**
      SUBROUTINE PVRT(VERT, NVERT, IDIM)
**
**
      DOUBLE PRECISION VERT(2,256)
      INTEGER NVERT, IDIM, I, J
**
**
***   WRITE(6,2)
      DO 100 J = 1, NVERT
100   WRITE(6,4) J, (VERT(I,J), I=1,IDIM)
      RETURN
**
2     FORMAT('  ')
4     FORMAT(1X,I5,6F10.5)
      END
**
**
```

```
**********************************************************************
**                                                                  **
**                            ACCEL                                 **
**                                                                  **
**********************************************************************
**
**
**      This subroutine implements the accelerated procedure
**    for the Brayton-Tong algorithm when the eigenvalues of
**    the extreme matrix are close to 1.  It finds the first
**    power of the extreme matrix that maps the given
**    vertex inside the convex hull.
**      Starting from 2, the power of the extreme matrix is
**    doubled until a point lands inside the hull. A binary
**    search between the last power used and the current
**    power finds the first power of the extreme matrix
**    that causes the point to land inside the hull.  The
**    The point that is generated by the previous extreme
**    matrix is then added to the hull.
**      The procedure is terminated if instability is
**    detected, an overflow in the number of vertices occurs
**    or the power of the extreme matrix reaches some
**    maximum.
**
**
**
**    INPUTS:  EMAT   - Difference equation extreme matrix
**             EDIM   - Dimension of system
**             WSTAR  - Convex hull
**             NSTAR  - Number in hull
**             ANG    - Array of angles for WSTAR
**             WZERO  - Initial vertex set
**             NZERO  - Number of vertices in initial vertex
**                        set
**             POINT  - Current vertex
**             TRACE  - Set to nonzero value if want a trace
**                        of results
**
**    OUTPUTS: WSTAR  -
**             NSTAR  -
**             MXVRT  - Maximum number of vertices in any
**                        hull during course of algorithm.
**             SFLAG  - Stable/Unstable status
**
**    PSEUDO-CODE:
**
**    Set Current-matrix to extreme-matrix
**    Set current-power to one
```

```
**
**    Do while new point is outside hull and hull is stable
**      Set last-point to new-point
**      Set new-point to vertex
**      Current-matrix = current-matrix * current-matrix
**      Current-power = current-power * 2
**
**      Generate new vertex point, adding it to hull if
**         possible.
**
**      If hull is stable and current-power > 1024
**        Set stable-flag to 'indeterminate by accelerated'
**    End (of do while)
**
**    If stable
**      Lower = 0
**      Upper = current-power / 2
**      If upper = 1 then set upper to 2
**      Do while (upper - lower) > 1 and hull is stable
**        Midpoint = (lower + upper) /2
**        Current-matrix =  (extreme-matrix ** midpoint)
**        Generate new vertex point, adding it to hull if
**           possible
**        If added to hull (outside hull)
**          Lower = midpoint
**        Else
**          Upper = midpoint
**      End
**
**    End
**
**
**    LOCAL VARIABLES:
**      CMAT    - Current power of extreme matrix
**                  (current-matrix)
**      CPWR    - Power of ext matrix being used
**                  (current-power)
**      IPWR    - Power of ext matrix used to get current
**                   point from vertex passed in.
**      NEWPT   - New vertex point
**      NEWFLG  - New-point-outside-flag   1=yes  0=no
**      LPNT    - Last vertex generated that fell outside the
**                   hull
**
**      LOW     - Lower index on binary search
**      UPP     - Upper index on binary search
**      MID     - Middle index on binary search
**
```

```
**
**
      SUBROUTINE ACCEL(EMAT, EDIM, WSTAR, NSTAR, ANG, WZERO,
     &    NZERO, POINT, TRACE, MXVRT, SFLAG)
**
**
      DOUBLE PRECISION EMAT(3,3), WZERO(2,10), WSTAR(2,256),
     &                 POINT(6), ANG(256)
      INTEGER EDIM, NZERO, NSTAR, TRACE, SFLAG, MXVRT
**
      DOUBLE PRECISION CMAT(3,3), NEWPT(3), LPNT(3)
      INTEGER CPWR, NEWFLG, LOW, UPP, MID, I, J, IPWR
**
**
      IF (TRACE.NE.0) WRITE(6,10)
**
**    Set Current-matrix to extreme-matrix
      DO 110 I = 1, EDIM
      DO 110 J = 1, EDIM
110   CMAT(I,J) = EMAT(I,J)
**
**    Set current-power to one
      CPWR = 1
**
**
      DO 115 I = 1, EDIM
115   NEWPT(I) = POINT(I)
**
**    Do while new point is outside hull and hull is stable
      SFLAG = 0
      NEWFLG = 1
130   IF ((NEWFLG.EQ.0).OR.(SFLAG.NE.0)) GOTO 200
**
**      Figure out IPWR
        IPWR = CPWR
        IF (CPWR.EQ.1) IPWR = 0
**
**      Set last-point
        DO 150 I = 1, EDIM
150     LPNT(I) = NEWPT(I)
**
**      Set new-point to vertex
        DO 155 I = 1, EDIM
155     NEWPT(I) = POINT(I)
**
**      Current-matrix = current-matrix * current-matrix
160     CALL MMULT(CMAT, CMAT, EDIM, EDIM, EDIM, CMAT)
**
```

```
**      Current-power = current-power * 2
        CPWR = CPWR * 2
**
**      Tracing function
        IF (TRACE.NE.0) WRITE(6,12) CPWR
**
**      Generate new vertex point, adding it to hull if
**        possible
        CALL GENPT(CMAT, EDIM, WSTAR, NSTAR, ANG, WZERO,
     &           NZERO, NEWPT, TRACE, MXVRT, SFLAG, NEWFLG)
**
**      If hull is stable and current-power > 1024
**        Set stable-flag to 'indeterminate by accelerated'
        IF ((SFLAG.EQ.0).AND.(CPWR.GT.1024)) SFLAG = 5
        GOTO 130
**
**    End (of do while)
200   CONTINUE
**
**
**    If stable
      IF (SFLAG.NE.0) GOTO 270
**
**      Lower = 0
**      Upper = current-power / 2
        LOW = 0
        UPP = CPWR / 2
**      If upper = 1 then set upper to 2
        IF (UPP.EQ.1) UPP = 2
**
**      Do while (upper - lower) > 1 and hull is stable
220     IF (((UPP-LOW).LE.1).OR.(SFLAG.NE.0)) GOTO 260
**
**        Midpoint = (lower + upper) /2
          MID = (LOW + UPP) / 2
          IF (TRACE.NE.0) WRITE(6,16) MID
**
**        Current-matrix = (extreme-matrix ** midpoint)
          CALL MPOWR(EMAT, EDIM, MID, CMAT)
**
**        Generate new vertex point, adding it to hull if
**          possible
          DO 230 I = 1, EDIM
230       NEWPT(I) = LPNT(I)
**
**
          CALL GENPT(CMAT, EDIM, WSTAR, NSTAR, ANG, WZERO,
     &           NZERO, NEWPT, TRACE, MXVRT, SFLAG, NEWFLG)
```

```
**
**          If added to hull (outside hull)
**            Lower = midpoint
            IF (NEWFLG.EQ.0) GOTO 240
              LOW = MID
              GOTO 250
**
**          Else
**            Upper = midpoint
240           UPP = MID
250         CONTINUE
            GOTO 220
**
**
**      End
260     CONTINUE
270     CONTINUE
**
**      Tracing function
        IPWR = IPWR + LOW
        IF ((TRACE.NE.0).AND.(SFLAG.EQ.0)) WRITE(6,14) IPWR
**
**
        RETURN
**
**
10      FORMAT('0  ++++++ACCELERATED PROCEDURE STARTED++++++')
12      FORMAT('    MATRIX POWER IS ',I4)
14      FORMAT('    POWER OF POINT LAST ADDED IS ',I4)
16      FORMAT('    BINARY SEARCH: MID = ',I4)
        END
```

B. Program to Find the Region of Stability for a Digital Filter: BGRID

Directory

```
FTN4,Y,L
$EMA(DAT)
      PROGRAM BGRID (4, 100)
**
**
*******************************************************************************
*******************************************************************************
**                                                                           **
**                                                                           **
**    THIS PROGRAM USES THE BRAYTON-TONG CONSTRUCTIVE                        **
**  ALGORITHM TO FIND THE REGION IN THE PARAMETER                            **
**  PLANE WHERE THE EQUILIBRIUM X = 0 OF A SECOND                            **
**  ORDER DIGITAL FILTER IS STABLE OR G. A. S.                               **
**                                                                           **
*******************************************************************************
*******************************************************************************
**
**
**
**    FILE "BGRID
**
**
**
**      This program is invoked by the command:
**           RU,BGRID,LU1,LU2,LU3
**    where LU1 is the lu # where questions and prompts will
**              be printed
**          LU2 is the lu # where responses to questions are
**              expected. An error message will be printed
**              if this is zero.
**          LU3 is the lu # where responses are echoed (to
**               build a batch run response file)
**    If no LUs are specified, the defaults are:
**          LU1 = 1 (terminal)
**          LU2 = 1 (terminal) if LU1 is also 1
**                0 (bit bucket) otherwise
**          LU3 = 0 (bit bucket)
**
**
**
**
**    PSEUDO-CODE
**
**    Set up LU numbers with parameters from program
**       run string.
**    Get data used to run program
**    If not building response file for batch
**      Set gains of nonlinearities
```

```
**      Print data entered by user
**
**      Get initial convex hull
**      Set A2 to initial value
**      Do while  A2 > A2-stop-value
**        Set A1 to initial value
**        Get A1-stop-value
**        Do while A1 <= A1-stop-value
**          Get set of extreme matrices
**          If set exists,
**            Do Brayton-Tong algorithm
**          Put result in result-array
**          Increment A1
**        End
**        Print result-array for this A2 value
**        Increment A2
**      End
**
**
**
       DOUBLE PRECISION EMAT(3,3,16), WZERO(2,10),
      &                 WSTAR(2,256), A1, A2, GAIN, GAINL,
      &                 RHO, A1INIT, A1INC, A1STOP, A2INIT,
      &                 A2INC, A2STOP, NAME
       INTEGER IBUF(5), PLU, ANSLU, ECHOLU, ROUND, RTYPE,
      &        NUMEXT, EDIM, NZERO, NSTAR, SFLAG, TRACE, YES,
      &        RSLT(101), IRSLT, I,
      &        OVRFLW, MXVRT
**
**
**     These are function subroutines
       DOUBLE PRECISION GETLB, GTEND
**
**     Initial data
       DATA YES/2HY /
**
**
**
**
**********************************************************************
**                                                                  **
**   THIS PORTION IS A DEVIATION FROM STANDARD FORTRAN              **
**                                                                  **
**                                                                  **
**     Get parameters entered when program is invoked               **
       CALL RMPAR(IBUF)
**                                                                  **
```

```
**    First one is LU where questions will be printed     **
      PLU = IBUF(1)
**                                                        **
**    Second is LU of answers.  Also must handle case if  **
**      program is started with no parameters specified.  **
      ANSLU = IBUF(2)
      IF ((ANSLU.EQ.0).AND.(PLU.EQ.1)) ANSLU = 1
**                                                        **
**    Third one is LU of echos if building batch run file **
      ECHOLU = IBUF(3)
**                                                        **
**    Print error and abort if no answer LU specified     **
      IF (ANSLU.NE.0) GOTO 99
      WRITE(1, 50)
      WRITE(6, 50)
      STOP
**                                                        **
************************************************************************
**
**
**    Get program name
99    NAME = GETLB(0)
**
**    Write title
      IF (PLU.NE.0) WRITE(PLU, 1) NAME
**
**    Get data about run
100   CALL QUEST(PLU, ANSLU, ECHOLU, ROUND, A1INIT, A1STOP,
     &           A1INC, A2INIT, A2STOP, A2INC, RTYPE, TRACE,
     &           RHO, OVRFLW)
**
**
**    If only building response file for batch, don't run
**      program
      IF (ECHOLU.NE.0) GOTO 300
**
**    Set upper gain according to type of quantizer
      GAIN = 1.D0
      IF (ROUND.EQ.1) GAIN = 2.D0
**
**    Set lower gain according to type of overflow
      GAINL = 0.D0
      IF (OVRFLW.EQ.1) GAINL = -1.D0/3.D0
      IF (OVRFLW.EQ.2) GAINL = -1.D0
**
**
**
```

```
**    Do printing of entered info
      WRITE(6,30) NAME
      CALL DESCR(6)
      IF (ROUND.EQ.1) WRITE(6,32)
      IF (ROUND.EQ.0) WRITE(6,34)
      IF (OVRFLW.EQ.0) WRITE(6,45)
      IF (OVRFLW.EQ.1) WRITE(6,46)
      IF (OVRFLW.EQ.2) WRITE(6,48)
**
      IF (RTYPE.NE.1) GOTO 120
      WRITE(6,42) A1INC
      GOTO 140
**
120   IF (RTYPE.NE.2) GOTO 130
      WRITE(6,36) A1INIT, A1STOP, A1INC
      WRITE(6,38) A2INIT, A2STOP, A2INC
      GOTO 140
**
130   WRITE(6,44) A1INIT, A2INIT
**
140   WRITE(6,40) RHO
**
      EDIM  = 2
      MXVRT = 2
**
**    Get initial hull
      CALL WINIT(EDIM, WZERO, NZERO)
**
**
**    MAIN PART OF PROGRAM
**
      A2 = A2INIT
**
**    Do while value of  A2 > stop value
220   IF (A2.LT.A2STOP) GOTO 250
        IF (PLU.NE.0) WRITE(PLU,10) A2
**
**      Set initial values
        A1 = A1INIT
        IRSLT = 1
**
**      Set A1STOP
        IF (RTYPE.NE.1) GOTO 229
        A1STOP = GTEND(A2)
        IF (A1INC.LT.0.D0) A1STOP = -A1STOP
229     CONTINUE
**
```

```
**      Do while value of A1 <= stop value for positive hor
**        increment or value of A1 >= stop value for neg-
**        ative hor increment
230     IF (((A1.GT.A1STOP).AND.(A1INC.GE.0.D0)).OR.
     &      ((A1.LT.A1STOP).AND.(A1INC.LT.0.D0))) GOTO 240
**
**        Get extreme matrix
          CALL GTMAT(A1, A2, GAIN, RHO, EMAT, NUMEXT, SFLAG,
     &               GAINL)
          IF (SFLAG.NE.0) GOTO 235
**
**        Do Brayton-Tong algorithm
          CALL BRAYT(EMAT, EDIM, NUMEXT, WZERO, NZERO,
     &               WSTAR, NSTAR, SFLAG, MXVRT, TRACE)
**
**        Put result in result array
235       RSLT(IRSLT) = SFLAG
          IRSLT = IRSLT + 1
**        Increment A1
          A1 = A1 + A1INC
          GOTO 230
**
**
240     CONTINUE
**
**      Get index of last result
        IRSLT = IRSLT - 1
**      Print result for this value of A2
        WRITE(6,2) A2, (RSLT(I), I=1,IRSLT)
**      Increment A2
        A2 = A2 + A2INC
        GOTO 220
**
**
250   CONTINUE
**
**
**    Print max number of vertices in any hull
***   WRITE(6, 52) MXVRT
**
      IF (PLU.NE.0) WRITE(PLU,7) 1799
**
**    Ask if want to run again
300   IF (PLU.NE.0) WRITE(PLU,23)
      READ(ANSLU,3) I
      IF (ECHOLU.NE.0) WRITE(ECHOLU, 3) I
      IF (I.EQ.YES) GOTO 100
**
```

```
      STOP
**
**
1     FORMAT('BRAYTON-TONG STABILITY FOR DIGITAL FILTER  ',
     &        A8)
2     FORMAT(' ',F6.3,2X,101I1)
3     FORMAT(A1)
7     FORMAT(A2,//)
10    FORMAT(1X,'A2 IS ',F6.3)
23    FORMAT('DO YOU WANT TO RUN PROGRAM AGAIN ?')
30    FORMAT('1  BRAYTON-TONG STABILITY   ',A8)
32    FORMAT('0 ROUNDOFF QUANTIZER')
34    FORMAT('0 MAGNITUDE TRUNCATION QUANTIZER')
36    FORMAT('0 HOR START, END, AND INC IS ',3F10.4)
38    FORMAT('  VER START, END, AND INC IS ',3F10.4)
40    FORMAT('0 RHO IS ',F10.7,///)
42    FORMAT('0 DEFAULT REGION.   GRID INCREMENT IS ',F8.6)
44    FORMAT('0 SINGLE POINT AT ',2F10.4)
45    FORMAT('  SATURATION OR ZEROING OVERFLOW')
46    FORMAT('  TRIANGULAR OVERFLOW')
48    FORMAT("  TWO'S COMPLEMENT OVERFLOW")
50    FORMAT('  ERROR IN BGRID: NO RESPONSE LU OR FILE.',/,
     &       '  PROGRAM BGRID ABORTED.')
52    FORMAT('0MAX NUMBER OF VERTICES IN ANY HULL IS ',I5)
      END
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                            QUEST                                 **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Gets the responses needed to run the Brayton-Tong
**    algorithm for digital filters regardless of the filter
**    structure to be analyzed.
**      The responses are echoed to the device ECHOLU so
**    that a response file for batch can be formatted.
**
**    INPUTS:  PLU    - LU # of the device to ask questions
**             ANSLU  - LU # of device to get responses from
**             ECHOLU - LU # of device to echo responses if
**                       building a batch run file
**
**    OUTPUTS: ROUND  - Type of quantization nonlinearity
**                        0 - truncation quantizer
**                        1 - roundoff quantizer
**             HSTART - Start of horizontal range to check
**                       stability
**             HSTOP  - End of horizontal range to check
**                       stability
**             HINC   - Horizontal grid increment
**             VSTART - Start of vertical range to check
**                       stabilty
**             VSTOP  - End of vertical range to check
**                       stabilty
**             VINC   - Vertical grid increment
**             RTYPE  - Type of grid desired
**                        1 - default region, HSTOP may
**                            depend on the vertical
**                            coordinate
**                        2 - rectangle region, HSTOP fixed
**                        3 - single point
**             TRACE  - Flag set to 1 if operator wants a
**                       trace of the algorithm for one
**                       point
**             RHO    - Value which will multiply all of the
**                       extreme matrices to get a measure
**                       of the global asymptotic stability
**                       of the system
**             OVRFLW - Type of overflow nonlinearity
**                        0 - saturation or zeroing
```

```
**                        1 - triangular
**                        2 - two's complement
**
**
**    PSEUDO-CODE:
**
**      Set trace-flag to "no"
**      Set grid-type-flag to "no run"
**
**      Do while grid-type-flag is "no run"
**        Ask for roundoff or truncation quantizers
**        If roundoff, set ROUND-flag to "round"
**        If truncation, set ROUND-flag to "truncation"
**
**        Set Overflow-flag to "saturation"
**        Ask for type of overflow
**        Set Overflow-flag appropriately
**
**        If operator wants to check the default region
**          Set grid-type-flag to "default"
**          Get grid increment from operator
**          Get default values
**          If operator wants to check reflection of default
**             region
**            Negate horizontal-increment
**        Else
**          If operator wants to check a region
**            Set grid-type-flag to "region"
**            Prompt operator for horizontal axis range &
**               increment
**            Prompt operator for vertical axis range & inc
**            Make sure signs on grid increments are correct
**          Else
**            If operator wants to check a single point
**              Set grid-type-flag to "single-point"
**              Get coordinates of point
**              If operator wants a trace
**                Set trace-flag to "yes"
**      End
**
**      Get value of rho
**
**
      SUBROUTINE QUEST(PLU, ANSLU, ECHOLU, ROUND, HSTART,
     &                 HSTOP, HINC, VSTART, VSTOP, VINC,
     &                 RTYPE, TRACE, RHO, OVRFLW)
**
**
```

```
      DOUBLE PRECISION HSTART, HSTOP, HINC, VSTART, VSTOP,
     &                 VINC, RHO
      INTEGER PLU, ANSLU, ECHOLU, ROUND, RTYPE, TRACE,
     &        I, R, T, YES, OVRFLW, S, C
**
      DATA R/2HR /, T/2HT /, YES/2HY /, S/2HS /, C/2HC /
**
**
**    Clear flags
      TRACE = 0
      RTYPE = 0
**
100   IF (RTYPE.NE.0) GOTO 200
**
**    Ask for roundoff or truncation
110   IF (PLU.NE.0) WRITE(PLU, 10)
      READ(ANSLU,3) I
      IF ((I.NE.R).AND.(I.NE.T)) GOTO 110
      IF (I.EQ.R) ROUND = 1
      IF (I.EQ.T) ROUND = 0
      IF (ECHOLU.NE.0) WRITE(ECHOLU, 3) I
**
      OVRFLW = 0
**
**      Get type of overflow
        IF (PLU.NE.0) WRITE(PLU, 42)
        READ(ANSLU,3) I
        IF (ECHOLU.NE.0) WRITE(ECHOLU, 3) I
        IF (I.EQ.S) OVRFLW = 0
        IF (I.EQ.T) OVRFLW = 1
        IF (I.EQ.C) OVRFLW = 2
**
**    Ask if want to use default region
120   IF (PLU.NE.0) WRITE(PLU, 12)
      READ(ANSLU, 3) I
      IF (ECHOLU.NE.0) WRITE(ECHOLU, 3) I
      IF (I.NE.YES) GOTO 130
        RTYPE = 1
**
**      Get axis increment
        IF (PLU.NE.0) WRITE(PLU, 14)
        READ(ANSLU, *) HINC
        IF (ECHOLU.NE.0) WRITE(ECHOLU, 5) HINC
**
**      Get default values
        CALL DFAUL(HSTART, VSTART, VSTOP)
        HINC = DABS(HINC)
        VINC = DSIGN(HINC, VSTOP-VSTART)
```

```
**
**      Ask if want reflection of default region
        IF (PLU.NE.0) WRITE(PLU,15)
        READ(ANSLU, 3) I
        IF (ECHOLU.NE.0) WRITE(ECHOLU, 3) I
        IF (I.EQ.YES) HINC = - HINC
        GOTO 190
**
**      Ask if want to check a specified region
130     IF (PLU.NE.0) WRITE(PLU, 16)
        READ(ANSLU, 3) I
        IF (ECHOLU.NE.0) WRITE(ECHOLU, 3) I
        IF (I.NE.YES) GOTO 160
          RTYPE = 2
**
**        Get horizontal axis range and increment
140       IF (PLU.NE.0) WRITE(PLU, 18)
          READ(ANSLU,*) HSTART, HSTOP
          IF (HSTOP.LE.HSTART) GOTO 140
          IF (ECHOLU.NE.0) WRITE(ECHOLU, 5) HSTART, HSTOP
          IF (PLU.NE.0) WRITE(PLU, 20)
          READ(ANSLU,*) HINC
          IF (ECHOLU.NE.0) WRITE(ECHOLU, 5) HINC
**
**        Get vertical axis range and increment
150       IF (PLU.NE.0) WRITE(PLU, 22)
          READ(ANSLU,*) VSTART, VSTOP
          IF (VSTOP.GE.VSTART) GOTO 150
          IF (ECHOLU.NE.0) WRITE(ECHOLU, 5) VSTART, VSTOP
          IF (PLU.NE.0) WRITE(PLU,24)
          READ(ANSLU,*) VINC
          IF (ECHOLU.NE.0) WRITE(ECHOLU, 5) VINC
**
**        Make sure correct sign on HINC
          HINC = DSIGN(HINC, HSTOP-HSTART)
**
**        Make sure correct sign on VINC
          VINC = DSIGN(VINC, VSTOP-VSTART)
          GOTO 190
**
**        Ask if want to check just one point
160       IF (PLU.NE.0) WRITE(PLU, 26)
          READ(ANSLU,3) I
          IF (ECHOLU.NE.0) WRITE(ECHOLU, 3) I
          IF (I.NE.YES) GOTO 190
          RTYPE = 3
**
```

```
**          Get coordinates of point
            IF (PLU.NE.0) WRITE(PLU, 28)
            READ(ANSLU,*) HSTART, VSTART
            IF (ECHOLU.NE.0) WRITE(ECHOLU, 5) HSTART, VSTART
            HINC = .01
            HSTOP = HSTART + .005
            VINC = -.01
            VSTOP = VSTART - .005
**
**          Ask if want to trace algorithm
            IF (PLU.NE.0) WRITE(PLU, 30)
            READ(ANSLU,3) I
            IF (ECHOLU.NE.0) WRITE(ECHOLU, 3) I
            IF (I.EQ.YES) TRACE = 1
**
**
190     GOTO 100
**
**      Get value of rho
200     RHO = 1.
        IF (PLU.NE.0) WRITE(PLU, 32)
        READ(ANSLU,*) RHO
        IF (ECHOLU.NE.0) WRITE(ECHOLU, 5) RHO
        IF (RHO.LT.1.) RHO = 1.
**
**
        RETURN
**
**
3       FORMAT(A1)
5       FORMAT(F12.7,1X,F12.7)
10      FORMAT('ROUNDOFF OR TRUNCATION ?  (R or T)')
12      FORMAT('DO YOU WANT TO USE DEFAULT REGION ?')
14      FORMAT('ENTER GRID INCREMENT')
15      FORMAT('DO YOU WANT TO CHECK THE REFLECTION',
     &         ' OF THE DEFAULT REGION ?')
16      FORMAT('DO WANT TO CHECK A SPECIFIC REGION ?')
18      FORMAT('ENTER HORIZONTAL AXIS RANGE')
20      FORMAT('ENTER HORIZONTAL AXIS INCREMENT')
22      FORMAT('ENTER VERTICAL AXIS RANGE')
24      FORMAT('ENTER VERTICAL AXIS INCREMENT')
26      FORMAT('DO YOU WANT TO CHECK JUST ONE POINT ?')
28      FORMAT('ENTER COORDINATES OF POINT')
30      FORMAT('DO YOU WANT A TRACE ?')
32      FORMAT('ENTER RHO')
42      FORMAT("SAT., TRI., OR TWO'S COMPL. ?  (S, T or C)")
        END
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          WINIT                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**       Returns the initial vertex set
**
**     INPUTS:  EDIM    - Dimension of system
**
**     OUTPUTS: WZERO   - Initial vertex set
**              NZERO   - Number of vertices in WZERO
**
**
      SUBROUTINE WINIT(EDIM, WZERO, NZERO)
**
**
      DOUBLE PRECISION WZERO(2,10)
      DOUBLE PRECISION ZDATA(2,10)
      INTEGER EDIM, NZERO, I, J
      DATA ZDATA/1.D0,  0.D0,
     &           0.D0,  1.D0/
**
**
      NZERO = 2
      DO 210 I=1,EDIM
      DO 210 J=1,NZERO
210   WZERO(I,J) = ZDATA(I,J)
      RETURN
      END
```

C. Program to Find the Boundary of a Region: BORDR

Directory

```
FTN4,L,Y
$EMA(DAT)
      PROGRAM BORDR(4, 100)
**
**
*******************************************************************************
*******************************************************************************
**                                                                           **
**       THIS PROGRAM SEARCHES FOR A BORDER OF A 2-D                         **
**     CLOSED REGION.                                                        **
**                                                                           **
*******************************************************************************
*******************************************************************************
**
**
**     FILE: "BORDR
**
**
**     PSEUDO-CODE:
**
**     Set search accuracy
**     Initialize search points and search-axis
**
**     Find initial boundary
**     If not found
**       Set error to "yes" and print message
**     Else
**       Set initial boundary point
**       Set previous boundary point so that gradient
**          will be determined correctly
**
**       Set first-time flag to "yes"
**
**     Do while not done and no error
**       Print boundary coordinates
**       Calculate gradient
**       Save previous inner and boundary(outer) point
**       Calculate next search points
**
**       Find boundary between two search points
**
**       If status = done then DONE = "yes"
**       If both search points outside
**         Try modified calculation of predicted inner and
**            outer search points.  The inner point is
**            predicted closer to boundary.  This is done
**            for sharp corners, to allow us to "walk"
**            to corner as far as possible.
```

```
**       Find boundary between two search points
**
**     If both search points outside or inside
**       Change search-axis
**       Calculate next search points
**       Find boundary between two search points
**       If boundary not found
**         Set error to "yes" and print message
**       Else
**         Set previous boundary point so that gradient
**          will be determined correctly
**
**     Else if search points reversed (inner one is
**         outside and outer one is inside)
**       Set error to "yes" and print message
**
**     If first-time flag is "yes"
**       Set first-time flag to "no"
**     Else
**       If boundary point close to initial boundary
**         Set DONE to "yes"
**
**   End (of do while)
**
**   Print point (0,0) to signal end of points
**
**
**   VARIABLES
**
**   SAXIS  - Search-axis.  Search for boundary along a
**            line parallel to this axis.
**               1  +y axis      3  -y axis
**               2  +x axis      4  -x axis
**   SACC   - Desired search accuracy
**   IPOINT - Inside search point
**   OPOINT - Outside search point
**   BPOINT - Boundary point
**   BSTAT  - Status of boundary search routine
**              0 - boundary found
**              1 - both search points outside
**              2 - both search points inside
**              3 - search points reversed(inside one was
**                   outside and outside one was inside)
**              4 - done
**
**   DELX   - Change in x from previous boundary point
**   DELY   - Change in y from previous boundary point
**
```

```
**    BPREV  - Previous boundary point
**    BINIT  - Initial boundary point
**
**    DONE   - Done flag for big do while loop  0=not done
**    ERR    - Error flag   0=no error  1=error
**
**    OUTLU  - Output logical unit # where boundary
**               points are printed
**    PRTLU  - Error mesages are printed here
**
**    PARMS  - Array of parameters from the run string.
**             The first one is logical unit # of file
**             in which to place the boundary data. The
**             other parameters are passed to BFUNC and
**             LINIT.
**       PARMS(2) - Quantization nonlinearity
**                  0 - truncation
**                  1 - roundoff
**       PARMS(3) - Overflow nonlinearity
**                  0 - zeroing or saturation
**                  1 - triangular
**                  2 - two's complement
**       PARMS(4) - Type of constructive algorithm boundary
**                  0 - stability boundary
**                  1 - finiteness criteria boundary
**       PARMS(5) - Unspecified
**
**
      INTEGER SAXIS, BSTAT, DONE, ERR, PARMS(5), I,
     &        OUTLU, PRTLU, FIRST, TRACE
      DOUBLE PRECISION IPOINT(2), OPOINT(2), BPOINT(2),
     &          BPREV(2), DELX, DELY, BINIT(2), IPREV(2),
     &          SACC, SINC, IINIT(2), OINIT(2)
**
**
*************************************************************************
**                                                                     **
**   THIS PORTION IS A DEVIATION FROM STANDARD FORTRAN                 **
**                                                                     **
**    Get parameters passed in from run string                         **
      CALL RMPAR(PARMS)
**                                                                     **
**    Set output logical unit numbers                                  **
      OUTLU = PARMS(1)
      PRTLU = 6
**                                                                     **
*************************************************************************
**
```

```
**
**    Set trace flag
      TRACE = 0
**
**    Set search accuracy
      SACC = .005
      SINC = .020
**
**    Set initial search points and search axis
      CALL LINIT(OUTLU, PARMS(2), IINIT, OINIT, SAXIS)
**
**    Initialize search points
      IPOINT(1) = IINIT(1)
      IPOINT(2) = IINIT(2)
      OPOINT(1) = OINIT(1)
      OPOINT(2) = OINIT(2)
**
**    Find initial boundary
      CALL BOUND(IPOINT, OPOINT, SACC, PARMS(2), TRACE,
     &           BPOINT, BSTAT)
**
**    If boundary not found, set ERR and print message
      IF (BSTAT.EQ.0) GOTO 130
        ERR = 1
        WRITE(PRTLU, 25) (IINIT(I), I=1,2),
     &                   (OINIT(I), I=1,2)
        GOTO 150
**
**    Else
**      Set initial boundary point
130     BINIT(1) = BPOINT(1)
        BINIT(2) = BPOINT(2)
**
**      Set previous boundary so that gradient is
**         determined correctly
        BPREV(1) = BPOINT(1)
        BPREV(2) = BPOINT(2)
        IF (SAXIS.EQ.1) BPREV(1) = BPOINT(1) - SINC
        IF (SAXIS.EQ.2) BPREV(2) = BPOINT(2) - SINC
        IF (SAXIS.EQ.3) BPREV(1) = BPOINT(1) + SINC
        IF (SAXIS.EQ.4) BPREV(2) = BPOINT(2) + SINC
**
**
        DONE = 0
        ERR  = 0
        FIRST = 0
**
**    Do while not done and no error
```

```
150   IF ((DONE.NE.0).OR.(ERR.NE.0)) GOTO 400
**
**      Print boundary coordinates
        WRITE(OUTLU, 5) BPOINT(1), BPOINT(2)
        IF (TRACE.NE.0) WRITE(6,5) BPOINT(1), BPOINT(2)
**
**      Calculate gradient
        DELX = BPOINT(1) - BPREV(1)
        DELY = BPOINT(2) - BPREV(2)
        IF (TRACE.NE.0) WRITE(6,30) DELX, DELY
   30   FORMAT(' GRADIENT:', 2F10.4)
**
**      Save previous inner point and boundary (outer) point
        IPREV(1) = IPOINT(1)
        IPREV(2) = IPOINT(2)
        BPREV(1) = BPOINT(1)
        BPREV(2) = BPOINT(2)
**
**      Calculate next search points
        CALL CALPT(BPOINT, DELX, DELY, SINC, SACC, SAXIS, 0,
     &             IPREV, IPOINT, OPOINT)
**
**
**      Find boundary between two search points
        CALL BOUND(IPOINT, OPOINT, SACC, PARMS(2), TRACE,
     &             BPOINT, BSTAT)
**
**      If status = done then DONE = "yes"
        IF (BSTAT.NE.4) GOTO 160
          DONE = 1
          GOTO 300
**
**      Else if both search points outside
160     IF (BSTAT.NE.1) GOTO 165
**
**        Try modified calculation of predicted inner and
**         outer points.  The inner point is not predicted
**         so far away.  This is done for sharp corners, to
**         allow us "walk" to corner as far as possible.
          CALL CALPT(BPREV, DELX, DELY, SINC, SACC, SAXIS,
     &               1, IPREV, IPOINT, OPOINT)
**
**        Now try to find boundary
          CALL BOUND(IPOINT, OPOINT, SACC, PARMS(2), TRACE,
     &               BPOINT, BSTAT)
**
**        Now if both search points both inside or outside,
**         try to turn corner.
```

```
165       IF ((BSTAT.NE.1).AND.(BSTAT.NE.2)) GOTO 195
**
**
**        NOW WE ARE TRYING TO TURN CORNER
**
**
**          Change search-axis
**
**          If both outside, rotate clockwise
            IF (BSTAT.EQ.1) SAXIS = SAXIS + 1
**
**          If both inside, rotate counterclockwise
            IF (BSTAT.EQ.2) SAXIS = SAXIS + 3
**
            IF (SAXIS.GT.4) SAXIS = SAXIS - 4
            IF (TRACE.NE.0) WRITE(6,35) SAXIS
   35       FORMAT(' SEARCH AXIS IS NOW ',I2)
**
**          Calculate next search points
**          IF (BSTAT.EQ.1) GOTO 175
**          OPOINT(1) = BPREV(1)
**          OPOINT(2) = BPREV(2)
**          IPOINT(1) = OPOINT(1)
**          IPOINT(2) = OPOINT(2)
**          GOTO 178
**
175         IPOINT(1) = IPREV(1)
            IPOINT(2) = IPREV(2)
            OPOINT(1) = IPOINT(1)
            OPOINT(2) = IPOINT(2)
178         IF (SAXIS.EQ.1) OPOINT(2) = IPOINT(2)+4.D0*SINC
            IF (SAXIS.EQ.2) OPOINT(1) = IPOINT(1)+4.D0*SINC
            IF (SAXIS.EQ.3) OPOINT(2) = IPOINT(2)-4.D0*SINC
            IF (SAXIS.EQ.4) OPOINT(1) = IPOINT(1)-4.D0*SINC
**
**
**          Find boundary between two search points
            CALL BOUND(IPOINT, OPOINT, SACC, PARMS(2),
     &                 TRACE, BPOINT, BSTAT)
**
**        If boundary not found
**          Set error to "yes" and print message
          IF (BSTAT.EQ.0) GOTO 190
            ERR = 1
            WRITE(PRTLU, 10) (IPREV(I), I=1,2),
     &                       (BPREV(I), I=1,2)
            GOTO 200
**        Else set previous boundary so that gradient is
```

```
**           not determined by previous points
190         BPREV(1) = BPOINT(1)
            BPREV(2) = BPOINT(2)
            IF (SAXIS.EQ.1) BPREV(1) = BPOINT(1) - SINC
            IF (SAXIS.EQ.2) BPREV(2) = BPOINT(2) - SINC
            IF (SAXIS.EQ.3) BPREV(1) = BPOINT(1) + SINC
            IF (SAXIS.EQ.4) BPREV(2) = BPOINT(2) + SINC
            GOTO 200
**
**
**      Else
**        If search points reversed
**          Set error to "yes" and print message
195       IF (BSTAT.NE.3) GOTO 200
            ERR = 1
            WRITE(PRTLU, 20) (IPOINT(I),I=1,2),
     &                       (OPOINT(I),I=1,2)
**
200     CONTINUE
**
**      If first time, set flag to indicate not first time
        IF (FIRST.NE.0) GOTO 220
          FIRST = 1
          GOTO 240
**
**      Else
**      If boundary point close to initial boundary
**        Set DONE to "yes"
**        Print initial boundary coordinates
220     IF(((BPOINT(1)-BINIT(1))**2
     &    + (BPOINT(2)-BINIT(2))**2).GT.(1.4*(SINC**2)))
     &      GOTO 240
          DONE = 1
          WRITE(OUTLU, 5) BINIT(1), BINIT(2)
240     CONTINUE
**
**
300   GOTO 150
**    End (of do while)
**
**
**    Print point (0,0) to signal end of points
400   CONTINUE
      WRITE(OUTLU, 5) 0.D0, 0.D0
      STOP
**
**
**
```

```
5     FORMAT(1X, 2F10.5)
10    FORMAT('  COULD NOT TURN CORNER    ', 2(2F10.4,4X))
20    FORMAT('  SEARCH POINTS REVERSED   ', 2(2F10.4,4X))
25    FORMAT('  INITIAL SEARCH POINTS INCORRECT   ',
     &       2(2F10.4,4X))
      END
```

```
**********************************************************************
**                                                                  **
**                          CALPT                                   **
**                                                                  **
**********************************************************************
**
**
**      This routine calculates the next search points to
**    use, based on the predicted next boundary point.
**
**    INPUTS:  BPOINT - Last boundary point
**             DELX   - Change in x from prev boundary pt
**             DELY   - Change in y from prev boundary pt
**             SINC   - Search increment
**             SACC   - Search accuracy  NOT USED
**             SAXIS  - Search axis
**             MFLAG  - Modified calculation flag
**                        0 - inner search point predicted
**                            same manner as outer search pt
**                        1 - inner search point predicted
**                            same as last inner search point
**             IPREV  - Last inner search point
**
**    OUTPUTS: IPOINT - Inner search point to use to find
**                        next boundary point.
**             OPOINT - Outer search point to use to find
**                        next boundary point.
**
**
      SUBROUTINE CALPT(BPOINT, DELX, DELY, SINC, SACC,
     &           SAXIS, MFLAG, IPREV, IPOINT, OPOINT)
**
**
      INTEGER SAXIS, MFLAG
      DOUBLE PRECISION BPOINT(2), DELX, DELY, SINC,
     &                 IPOINT(2), OPOINT(2), IPREV(2),
     &                 PRED, IBND, OBND, SACC
**
**
**    Set how far away from the predicted boundary the
**      inner and outer points are put
      OBND = 2.0D0 * SINC
      IBND = 2.0D0 * SINC
**
      IF (SAXIS.GT.4) SAXIS = 1
**
**    Do case SAXIS
      GOTO (100, 200, 300, 400) SAXIS
```

```
**
**
**********************************
**       +Y search axis     ***
**********************************
100     IPOINT(1) = BPOINT(1) + SINC
        OPOINT(1) = IPOINT(1)
**
**      Predicted value
        PRED = BPOINT(2) + SINC*DELY/DABS(DELX)
**
        OPOINT(2) = PRED + OBND
        IPOINT(2) = PRED - IBND
        IF (MFLAG.NE.0) IPOINT(2) = IPREV(2)
        GOTO 500
**
**
**********************************
**      +X search axis      ***
**********************************
200     IPOINT(2) = BPOINT(2) - SINC
        OPOINT(2) = IPOINT(2)
**
**      Predicted value
        PRED = BPOINT(1) + SINC*DELX/DABS(DELY)
**
        OPOINT(1) = PRED + OBND
        IPOINT(1) = PRED - IBND
        IF (MFLAG.NE.0) IPOINT(1) = IPREV(1)
        GOTO 500
**
**
**********************************
**       -Y search axis     ***
**********************************
300     IPOINT(1) = BPOINT(1) - SINC
        OPOINT(1) = IPOINT(1)
**
**      Predicted value
        PRED = BPOINT(2) + SINC*DELY/DABS(DELX)
**
        OPOINT(2) = PRED - OBND
        IPOINT(2) = PRED + IBND
        IF (MFLAG.NE.0) IPOINT(2) = IPREV(2)
        GOTO 500
**
**
**
```

```
      **
      ************************************
      **     -X search axis      ***
      ************************************
      400     IPOINT(2) = BPOINT(2) + SINC
              OPOINT(2) = IPOINT(2)
      **
      **      Predicted value
              PRED = BPOINT(1) + SINC*DELX/DABS(DELY)
      **
              OPOINT(1) = PRED - OBND
              IPOINT(1) = PRED + IBND
              IF (MFLAG.NE.0) IPOINT(1) = IPREV(1)
              GOTO 500
      **
      **
      500   CONTINUE
            RETURN
            END
```

```
**********************************************************************
**                                                                  **
**                            BOUND                                 **
**                                                                  **
**********************************************************************
**
**
**      Finds the boundary between the two given search
**    points to a given accuracy.  This subroutine calls an
**    integer function subprogram BFUNC that returns a 0 if
**    point is inside the region, a 1 if it is outside the
**    region or on the boundary and a 2 if at end of
**    boundary.
**
**    INPUTS: IPT     - Inside search point
**            OPT     - Outside search point
**            SACC    - Search accuracy
**            PARMS   - Array of parameters from the run
**                        string to be passed to BFUNC
**            TRACE   - Set to nonzero value if want to trace
**
**    OUTPUTS:BPOINT  - Boundary point
**            BSTAT   - Status of this routine
**                       0 - boundary found
**                       1 - both search points outside
**                       2 - both search points inside
**                       3 - search points reversed(inside
**                            one was outside and outside
**                            one was inside)
**                       4 - done
**            IPT,OPT- Last values of these points
**
**
**    PSEUDO-CODE
**
**    Set axis-flag according to search axis
**    Get initial status of search points
**
**    If inner-status="inside" and outer-status="outside"
**      Do while difference between inner and outer points
**       is greater than accuracy
**        Calculate midpoint
**        Get status of midpoint
**        If outside, outer point = midpoint
**        Else inner point = midpoint
**      End
**      Boundary-status = "found"
**
```

```
**    Else
**      If status of either search points is "at end of
**       boundary" then Boundary-status = "done"
**      Else
**        If both search points inside
**          Boundary-status = "both inside"
**        Else
**          If both search points outside
**            Boundary-status = "both outside"
**          Else
**            Boundary-status = "reversed"
**
**
      SUBROUTINE BOUND(IPT,    OPT,    SACC, PARMS, TRACE,
     &                 BPOINT, BSTAT)
**
**
      INTEGER PARMS, BSTAT, ISTAT, OSTAT, MSTAT, AXIS, TRACE
      DOUBLE PRECISION IPOINT(2), OPOINT(2), SACC,BPOINT(2),
     &                 IPT(2), OPT(2), MPT(2)
**
**    This is a function subprogram
      INTEGER BFUNC
**
**
**    Set axis-flag according to search axis
      AXIS = 1
      IF (DABS(IPT(1)-OPT(1)).LE.1.D-7) AXIS = 2
**
**    Get initial status of search points
      ISTAT = BFUNC(IPT(1), IPT(2), PARMS)
      OSTAT = BFUNC(OPT(1), OPT(2), PARMS)
      IF (TRACE.NE.0) WRITE(6, 5) IPT(1),IPT(2),ISTAT,
     &                            OPT(1),OPT(2),OSTAT
    5 FORMAT(' SEARCH POINTS:', 2(2F10.4,I5))
**
**    If inner-status="inside" and outer-status="outside"
      IF ((ISTAT.NE.0).OR.(OSTAT.NE.1)) GOTO 200
**
**      Do while difference between inner and outer points
**       is greater than accuracy
150   IF (DABS(IPT(AXIS)-OPT(AXIS)).LE.SACC) GOTO 180
**
**        Calculate midpoint
          MPT(1) = IPT(1)
          MPT(2) = IPT(2)
          MPT(AXIS) = (IPT(AXIS) + OPT(AXIS)) / 2.D0
**
```

```
**        Get status of midpoint
          MSTAT = BFUNC(MPT(1), MPT(2), PARMS)
***       WRITE(6, 10) MPT(1), MPT(2), MSTAT
***10     FORMAT(' MIDPOINT:', 2F10.4, I5)
**
**        If outside, outer point = midpoint
          IF (MSTAT.EQ.0) GOTO 160
            OPT(1) = MPT(1)
            OPT(2) = MPT(2)
            GOTO 170
**
**        Else inner point = midpoint
160         IPT(1) = MPT(1)
            IPT(2) = MPT(2)
**
170       GOTO 150
**      End
180   CONTINUE
**
**      Boundary-status = "found"
        BSTAT = 0
**
**      Boundary-point = last outer-point
        BPOINT(1) = OPT(1)
        BPOINT(2) = OPT(2)
        GOTO 250
**
**
**    Else
**      If status of either search points is "at end of
**       boundary" then Boundary-status = "done"
200     IF ((ISTAT.NE.2).AND.(OSTAT.NE.2)) GOTO 210
          BSTAT = 4
          GOTO 250
**
**      Else
**        If both search points inside
**          Boundary-status = "both inside"
210       IF ((ISTAT.NE.0).OR.(OSTAT.NE.0)) GOTO 220
            BSTAT = 2
            GOTO 250
**
**        Else
**          If both search points outside
**            Boundary-status = "both outside"
220         IF ((ISTAT.NE.1).OR.(OSTAT.NE.1)) GOTO 230
              BSTAT = 1
              GOTO 250
```

```
**
**          Else
**            Boundary-status = "reversed"
230           BSTAT = 3
**
**
**
250   CONTINUE
      RETURN
      END
```

```
**********************************************************************
**                                                                  **
**                            BFUNC                                 **
**                                                                  **
**********************************************************************
**
**      This function subprogram is the interface between
**    the border-determining program and all of the
**    Brayton-Tong subroutines.
**
**
**    INPUTS:  X, Y    - Coordinates of point
**             PARMS   - Parameters passed in by run string
**                       PARMS(1)- type of quantizer
**                                 0  truncation
**                                 1  roundoff
**                       PARMS(2)- type of overflow
**                                 0  zeroing or saturation
**                                 1  triangular
**                                 2  two's complement
**                       PARMS(3)- type of boundary
**                                 0  stability
**                                 1  finiteness criteria
**
**    OUTPUTS: BFUNC   - Status of point
**                       0 - inside region
**                       1 - outside region
**                       2 - at end of boundary
**
**
      INTEGER FUNCTION BFUNC(X, Y, PARMS)
**
**
      DOUBLE PRECISION X, Y, EMAT(3,3,16), WSTAR(2,256),
     &                 WZERO(2,10), A1, A2, GAIN, GAINL, RHO
      INTEGER PARMS(4), ROUND, NUMEXT, EDIM, NZERO, NSTAR,
     &        SFLAG, OVRFLW, MXVRT
**
**    This is a function subprogram
      INTEGER DNCHK
**
**
      A1 = X
      A2 = Y
**
**    Check if done
      BFUNC = DNCHK(A1, A2)
**
```

```
**    If not done, check status of point
      IF (BFUNC.EQ.2) GOTO 300
**
**      Set quantizer, overflow gain limits
        ROUND = PARMS(1)
        OVRFLW= PARMS(2)
**
        GAIN = 1.D0
        IF (ROUND.EQ.1) GAIN = 2.D0
**
        GAINL = 0.D0
        IF (OVRFLW.EQ.1) GAINL = -1.D0/3.D0
        IF (OVRFLW.EQ.2) GAINL = -1.D0
**
**      Set  RHO to check for asymptotic stability
        RHO = 1.0000001D0
**
**      Other initialization
        EDIM = 2
        MXVRT = 2
**
**      Get initial convex hull
        CALL WINIT(EDIM, WZERO, NZERO)
**
**      Get set of extreme matrices
        CALL GTMAT(A1, A2, GAIN, RHO, EMAT, NUMEXT,
     &             SFLAG, GAINL)
**
**      If checking stability boundary, do constructive
**        algorithm
        IF (PARMS(3).NE.0) GOTO 130
          CALL BRAYT(EMAT, EDIM, NUMEXT, WZERO, NZERO,
     &               WSTAR, NSTAR, SFLAG, MXVRT, 0)
          GOTO 140
**
**      Else check eigenvalues of extreme matrices
130       CALL EIGEN(EMAT, EDIM, NUMEXT, SFLAG, 0)
140     CONTINUE
**
      BFUNC = 0
**
**    If SFLAG <> 0 then set BFUNC to 1
      IF (SFLAG.NE.0) BFUNC = 1
300   CONTINUE
      RETURN
      END
```

```
**********************************************************************
**                                                                  **
**                          WINIT                                   **
**                                                                  **
**********************************************************************
**
**      Returns the initial vertex set
**
**    INPUTS:  EDIM   - Dimension of system
**    OUTPUTS: WZERO  - Initial vertex set
**             NZERO  - Number of vertices in WZERO
**
**
      SUBROUTINE WINIT(EDIM, WZERO, NZERO)
**
**
      DOUBLE PRECISION WZERO(2,10)
      DOUBLE PRECISION ZDATA(2,10)
      INTEGER EDIM, NZERO, I, J
      DATA ZDATA/1.D0,  0.D0,
     &           0.D0,  1.D0/
**
**
      NZERO = 2
      DO 210 I=1,EDIM
      DO 210 J=1,NZERO
210   WZERO(I,J) = ZDATA(I,J)
      RETURN
      END
```

D. Subroutines that are Unique to Each Digital Filter Structure

1. Direct form with one quantizer

Directory

```
FTN4,Y,L
**
**
*************************************************************************
*************************************************************************
**                                                                     **
**                                                                     **
**       DIRECT FORM  --  ONE QUANTIZER                                **
**                                                                     **
**                                                                     **
**                                                                     **
**    THIS FILE CONTAINS THE ROUTINES FOR USING THE                    **
**  BRAYTON-TONG ALGORITHM PROGRAMS TO FIND THE                        **
**  STABILITY REGIONS FOR A SECOND ORDER DIRECT FORM                   **
**  DIGITAL FILTER WITH ONE QUANTIZER.                                 **
**                                                                     **
*************************************************************************
*************************************************************************
**
**
**    FILE "BRAYA
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          GETLB                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**    Returns the name of this program version
**
**
      DOUBLE PRECISION FUNCTION GETLB(I)
**
**
      DOUBLE PRECISION LABEL
      DATA LABEL/8HBRAYA   /
**
      GETLB = LABEL
      RETURN
      END
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                            DESCR                                 **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**       Prints any additional description of program version
**
**     INPUT:  LU - Device number of printer
**
      SUBROUTINE DESCR(LU)
**
**
      WRITE(LU, 1)
      RETURN
**
1     FORMAT(' CANONICAL FORM DIGITAL FILTER WITH ONE',
     &       ' QUANTIZER ')
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          DFAUL                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Supplies the default values for the triangle in the
**    parameter plane.
**
**    OUTPUT: HSTART - Start of hor range to check stability
**            VSTART - Start of ver range to check stability
**            VSTOP  - End   of  "    "    "   "       "
**
**
      SUBROUTINE DFAUL(HSTART, VSTART, VSTOP)
**
**
      DOUBLE PRECISION HSTART, VSTART, VSTOP
**
**
      HSTART =  0.D0
      VSTART =  1.D0
      VSTOP  = -1.D0
      RETURN
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          GTEND                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Returns the horizontal stop value for the default
**    region, given the vertical coordinate.
**
      DOUBLE PRECISION FUNCTION GTEND(VERT)
**
**
      DOUBLE PRECISION VERT
**
      GTEND = 1.D0 - VERT + 1.D-12
      RETURN
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                            GTMAT                                 **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Generates the set of extreme matrices
**
**    INPUTS:  A1,A2   - Coordinates of point for which set
**                       is generated
**             GAIN    - Upper gain of quantizer nonlinearity
**             GAINL   - Lower gain of overflow nonlinearity
**             RHO     - All of the extreme matrices will
**                       be multiplied by this value to get
**                       the asymptotic stability of the
**                       system.
**
**    OUTPUTS: EMAT    - Set of extreme matrices
**             NUMEXT  - Number of extreme matrices
**             SFLAG   - Always set to 0 since the filter can
**                       realize any parameters.
**
**
      SUBROUTINE GTMAT(A1, A2, GAIN, RHO, EMAT, NUMEXT,
     &                 SFLAG, GAINL)
**
**
      DOUBLE PRECISION A1, A2, GAIN, RHO, EMAT(3,3,4)
      DOUBLE PRECISION GAINL
      DOUBLE PRECISION SLOPE(4,2), G1, G2, G3, G4
      INTEGER NUMEXT, SFLAG, I, J, K
**
**
**    Clear the flag
      SFLAG = 0
**
**    Set the number of extreme matrices
      NUMEXT = 2
**
**    Set up the slope limits on the nonlinearity
      DO 105 I = 1,4
      SLOPE(I,1) = GAIN
105   SLOPE(I,2) = GAINL
**
**
```

```
**    Set up extreme matrices
      DO 110 I = 1, NUMEXT
**
      K = I - 1
      G1 = SLOPE(1, MOD(K,2) + 1)
**
      EMAT(1,1,I) =  G1 * A1
      EMAT(1,2,I) =  G1 * A2
      EMAT(2,1,I) =  1.D0
      EMAT(2,2,I) =  0.D0
110   CONTINUE
**
**
      DO 170 K = 1, NUMEXT
      DO 170 J = 1, 2
      DO 170 I = 1, 2
170   EMAT(I,J,K) = RHO * EMAT(I,J,K)
**
**
      RETURN
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          LINIT                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Sets up initialization for the boundary search
**    program.
**
**    INPUTS:  LU     - Logical unit number of output file
**             PARMS  - Parameters passed in by run string
**                       PARMS(1)- type of quantizer
**                                 0  truncation
**                                 1  roundoff
**                       PARMS(2)- type of overflow
**                                 0  none, zeroing, satur
**                                 1  triangular
**                                 2  two's complement
**                       PARMS(3)- type of boundary
**                                 0  stability
**                                 1  eigenvalues < 1
**
**    OUTPUTS: IINIT  - Initial inside search point
**             OINIT  - Initial outside search point
**             SAXIS  - Initial search axis
**
**
      SUBROUTINE LINIT(LU, PARMS, IINIT, OINIT, SAXIS)
**
**
      INTEGER LU, PARMS(4), SAXIS, I
      DOUBLE PRECISION IINIT(2), OINIT(2)
**
**
**    Print description
      WRITE(LU, 5)  (PARMS(I), I=1,4)
**
**    Print data range (depending on quantizer)
      IF (PARMS(1).EQ.0) WRITE(LU, 10)  0.0, 2.0, -1., 1.
      IF (PARMS(1).EQ.1) WRITE(LU, 10)  0.0, 1.0, -.5,  .5
**
**    Print symmetry ( 1 = symmetric about y axis)
      WRITE(LU, 15) 1
**
```

```
**    Set initial search points and search axis
      IINIT(1) = 0.00D0
      OINIT(1) = 1.50D0
      IINIT(2) = 0.00D0
      OINIT(2) = IINIT(2)
      SAXIS = 2
**
      RETURN
**
**
5     FORMAT('DIRECT FORM, ONE QUANTIZER ',4I5)
10    FORMAT(4F10.4)
15    FORMAT(I2)
      END
**
**
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                            DNCHK                                 **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**
**      Checks if we are done with border searching
**    program.  Since these plots are all symmetric
**    about Y axis, we are done when X < 0.
**
**
**    INPUTS: X, Y  - Coordinates of point
**
**    OUTPUT: DNCHK - '0' if not done
**                    '2' if done
**
**
      INTEGER FUNCTION DNCHK(X, Y)
**
**
      DOUBLE PRECISION X, Y
**
**
      DNCHK = 0
      IF (X.LT.0.D0) DNCHK = 2
      RETURN
      END
```

2. Direct form with two quantizers

Directory

```
FTN4,Y,L
**
**
**********************************************************************
**********************************************************************
**                                                                  **
**                                                                  **
**        DIRECT FORM  --  TWO QUANTIZERS                           **
**                                                                  **
**                                                                  **
**                                                                  **
**     THIS FILE CONTAINS THE ROUTINES FOR USING THE                **
**   BRAYTON-TONG ALGORITHM PROGRAMS TO FIND THE                    **
**   STABILITY REGIONS FOR A SECOND ORDER DIRECT FORM               **
**   DIGITAL FILTER WITH TWO QUANTIZERS.                            **
**                                                                  **
**********************************************************************
**********************************************************************
**
**
**     FILE "BRAYB
**
**
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          GETLB                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**    Returns the name of this program version
**
**
      DOUBLE PRECISION FUNCTION GETLB(I)
**
**
      DOUBLE PRECISION LABEL
      DATA LABEL/8HBRAYB   /
**
      GETLB = LABEL
      RETURN
      END
**
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          DESCR                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Prints any additional description of program version
**
**    INPUT:  LU - Device number of printer
**
      SUBROUTINE DESCR(LU)
**
**
      WRITE(LU, 1)
      RETURN
**
1     FORMAT(' CANONICAL FORM DIGITAL FILTER WITH TWO',
     &       ' QUANTIZERS')
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          DFAUL                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Supplies the default values for the triangle in the
**    parameter plane.
**
**    OUTPUT: HSTART - Start of hor range to check stability
**            VSTART - Start of ver range to check stability
**            VSTOP  - End   of  "    "    "   "       "
**
**
      SUBROUTINE DFAUL(HSTART, VSTART, VSTOP)
**
**
      DOUBLE PRECISION HSTART, VSTART, VSTOP
**
**
      HSTART =  0.D0
      VSTART =  1.D0
      VSTOP  = -1.D0
      RETURN
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          GTEND                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Returns the horizontal stop value for the default
**    region, given the vertical coordinate.
**
      DOUBLE PRECISION FUNCTION GTEND(VERT)
**
**
      DOUBLE PRECISION VERT
**
      GTEND = 1.D0 - VERT + 1.D-12
      RETURN
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          GTMAT                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Generates the set of extreme matrices
**
**    INPUTS:  A1,A2   - Coordinates of point for which set
**                       is generated
**             GAIN    - Upper gain of quantizer nonlinearity
**             GAINL   - Lower gain of overflow nonlinearity
**             RHO     - All of the extreme matrices will
**                       be multiplied by this value to get
**                       the asymptotic stability of the
**                       system.
**
**    OUTPUTS: EMAT    - Set of extreme matrices
**             NUMEXT  - Number of extreme matrices
**             SFLAG   - Always set to 0 since the filter can
**                       realize any parameters.
**
**
      SUBROUTINE GTMAT(A1, A2, GAIN, RHO, EMAT, NUMEXT,
     &                 SFLAG, GAINL)
**
**
      DOUBLE PRECISION A1, A2, GAIN, RHO, EMAT(3,3,4)
      DOUBLE PRECISION GAINL
      DOUBLE PRECISION SLOPE(4,2), G1, G2, G3, G4
      INTEGER NUMEXT, SFLAG, I, J, K
**
**
**    Clear the flag
      SFLAG = 0
**
**    Set the number of extreme matrices
      NUMEXT = 4
**
**    Set up the slope limits on the nonlinearity
      DO 105 I = 1,4
      SLOPE(I,1) = GAIN
105   SLOPE(I,2) = GAINL * GAIN
**
**
```

```
**      Set up extreme matrices
        DO 110 I = 1, NUMEXT
**
        K = I - 1
        G1 = SLOPE(1, MOD(K,2) + 1)
        G2 = SLOPE(2, MOD(K/2,2) + 1)
**
        EMAT(1,1,I) =  G1 * A1
        EMAT(1,2,I) =  G2 * A2
        EMAT(2,1,I) =  1.D0
        EMAT(2,2,I) =  0.D0
110     CONTINUE
**
**
        DO 170 K = 1, NUMEXT
        DO 170 J = 1, 2
        DO 170 I = 1, 2
170     EMAT(I,J,K) = RHO * EMAT(I,J,K)
**
**
        RETURN
        END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          LINIT                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Sets up initialization for the boundary search
**    program.
**
**    INPUTS:  LU     - Logical unit number of output file
**             PARMS  - Parameters passed in by run string
**                      PARMS(1)- type of quantizer
**                                0  truncation
**                                1  roundoff
**                      PARMS(2)- type of overflow
**                                0  none, zeroing, satur
**                                1  triangular
**                                2  two's complement
**                      PARMS(3)- type of boundary
**                                0  stability
**                                1  eigenvalues < 1
**
**    OUTPUTS: IINIT  - Initial inside search point
**             OINIT  - Initial outside search point
**             SAXIS  - Initial search axis
**
**
      SUBROUTINE LINIT(LU, PARMS, IINIT, OINIT, SAXIS)
**
**
      INTEGER LU, PARMS(4), SAXIS, I
      DOUBLE PRECISION IINIT(2), OINIT(2)
**
**
**    Print description
      WRITE(LU, 5)  (PARMS(I), I=1,4)
**
**    Print data range (depending on quantizer)
      IF (PARMS(1).EQ.0) WRITE(LU, 10)  0.0, 1.0, -1., 1.
      IF (PARMS(1).EQ.1) WRITE(LU, 10)  0.0,  .5, -.5,  .5
**
**    Print symmetry ( 1 = symmetric about y axis)
      WRITE(LU, 15) 1
**
```

```
**    Set initial search points and search axis
      IINIT(1) = 0.00D0
      OINIT(1) = 1.50D0
      IINIT(2) = 0.00D0
      OINIT(2) = IINIT(2)
      SAXIS = 2
**
      RETURN
**
**
5     FORMAT('DIRECT FORM, TWO QUANTIZERS ',4I5)
10    FORMAT(4F10.4)
15    FORMAT(I2)
      END
**
**
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                            DNCHK                                 **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**
**      Checks if we are done with border searching
**    program.  Since these plots are all symmetric
**    about Y axis, we are done when X < 0.
**
**
**    INPUTS: X, Y  - Coordinates of point
**
**    OUTPUT: DNCHK - '0' if not done
**                    '2' if done
**
**
      INTEGER FUNCTION DNCHK(X, Y)
**
**
      DOUBLE PRECISION X, Y
**
**
      DNCHK = 0
      IF (X.LT.-.04D0) DNCHK = 2
      RETURN
      END
```

3. Coupled form with two quantizers

Directory

```
FTN4,Y,L
**
**
*************************************************************************
*************************************************************************
**                                                                     **
**                                                                     **
**       COUPLED FORM  --  TWO QUANTIZERS                              **
**                                                                     **
**                                                                     **
**                                                                     **
**    THIS FILE CONTAINS THE ROUTINES FOR USING THE                    **
**  BRAYTON-TONG ALGORITHM PROGRAMS TO FIND THE                        **
**  STABILITY REGIONS FOR A SECOND ORDER COUPLED FORM                  **
**  DIGITAL FILTER WITH TWO QUANTIZERS.                                **
**                                                                     **
*************************************************************************
*************************************************************************
**
**
**    FILE "BRAYJ
**
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          GETLB                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**    Returns the name of this program version
**
**
      DOUBLE PRECISION FUNCTION GETLB(I)
**
**
      DOUBLE PRECISION LABEL
      DATA LABEL/8HBRAYJ   /
**
      GETLB = LABEL
      RETURN
      END
**
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          DESCR                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**       Prints any additional description of program version
**
**     INPUT:  LU - Device number of printer
**
       SUBROUTINE DESCR(LU)
**
**
       WRITE(LU, 1)
       RETURN
**
1      FORMAT('  GOLD-RADER COUPLED FORM FILTER WITH TWO',
     &         ' QUANTIZERS.')
       END
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          DFAUL                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**      Supplies the default values for the first quadrant
**    of the unit circle.
**
**    OUTPUT: HSTART - Start of hor range to check stability
**            VSTART - Start of ver range to check stability
**            VSTOP  - End   of  "    "    "   "       "
**
**
      SUBROUTINE DFAUL(HSTART, VSTART, VSTOP)
**
**
      DOUBLE PRECISION HSTART, VSTART, VSTOP
**
**
      HSTART =  0.D0
      VSTART =  1.D0
      VSTOP  =  0.D0
      RETURN
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          GTEND                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**       Returns the horizontal stop value for the default
**     region, given the vertical coordinate.
**
      DOUBLE PRECISION FUNCTION GTEND(VERT)
**
**
      DOUBLE PRECISION VERT
**
      GTEND = DSQRT(1.D0 - VERT*VERT) + 1.D-12
      RETURN
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                            GTMAT                                 **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Generates the set of extreme matrices
**
**    INPUTS:  A1,A2   - Coordinates of point for which set
**                       is generated
**             GAIN    - Upper gain of quantizer nonlinearity
**             GAINL   - Lower gain of overflow nonlinearity
**             RHO     - Value which will multiply all of the
**                       extreme matrices to get a measure
**                       of the asymptotic stability of the
**                       system.
**
**    OUTPUTS: EMAT    - Set of extreme matrices
**             NUMEXT  - Number of extreme matrices
**             SFLAG   - Always set to 0 since filter can
**                       realize any conjugate poles.
**
**
**
      SUBROUTINE GTMAT(A1, A2, GAIN, RHO, EMAT, NUMEXT,
     &                 SFLAG, GAINL)
**
**
      DOUBLE PRECISION A1, A2, GAIN, RHO, EMAT(3,3,16)
      DOUBLE PRECISION GAINL
      DOUBLE PRECISION SLOPE(4,2), G1, G2, G3, G4
      INTEGER NUMEXT, SFLAG, I, J, K
**
**
**
**    Clear flag
      SFLAG = 0
**
**    Set the number of extreme matrices
      NUMEXT = 4
**
**    Set up the slope limits on the nonlinearity
      DO 105 I = 1,4
      SLOPE(I,1) = GAIN
105   SLOPE(I,2) = GAINL
```

```
**
**
**    Set up extreme matrices
      DO 110 I = 1, NUMEXT
**
**
      K = I - 1
      G1 = SLOPE(1, MOD(K,2) + 1)
      G2 = SLOPE(2, MOD(K/2,2) + 1)
**
**
      EMAT(1,1,I) =  G1 * A1
      EMAT(1,2,I) = -G1 * A2
      EMAT(2,1,I) =  G2 * A2
      EMAT(2,2,I) =  G2 * A1
110   CONTINUE
**
**
      DO 170 K = 1, NUMEXT
      DO 170 J = 1, 2
      DO 170 I = 1, 2
170   EMAT(I,J,K) = RHO * EMAT(I,J,K)
**
**
      RETURN
      END
```

4. Coupled form with four quantizers

Directory

```
FTN4,Y,L
**
**
**********************************************************************
**********************************************************************
**                                                                  **
**                                                                  **
**        COUPLED FORM  --  FOUR QUANTIZERS                         **
**                                                                  **
**                                                                  **
**                                                                  **
**                                                                  **
**     THIS FILE CONTAINS THE ROUTINES FOR USING THE                **
**   BRAYTON-TONG ALGORITHM PROGRAMS TO FIND THE                    **
**   STABILITY REGIONS FOR A SECOND ORDER COUPLED FORM              **
**   DIGITAL FILTER WITH FOUR QUANTIZERS.                           **
**                                                                  **
**********************************************************************
**********************************************************************
**
**
**     FILE "BRAYI
**
**
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          GETLB                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**    Returns the name of this program version
**
**
      DOUBLE PRECISION FUNCTION GETLB(I)
**
**
      DOUBLE PRECISION LABEL
      DATA LABEL/8HBRAYI   /
**
      GETLB = LABEL
      RETURN
      END
**
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          DESCR                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Prints any additional description of program version
**
**    INPUT:  LU - Device number of printer
**
      SUBROUTINE DESCR(LU)
**
**
      WRITE(LU, 1)
      RETURN
**
1     FORMAT('  GOLD-RADER COUPLED FORM FILTER WITH FOUR',
     &       '  QUANTIZERS.')
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          DFAUL                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Supplies the default values for the first quadrant
**    of the unit circle.
**
**    OUTPUT: HSTART - Start of hor range to check stability
**            VSTART - Start of ver range to check stability
**            VSTOP  - End   of  "    "    "   "       "
**
**
      SUBROUTINE DFAUL(HSTART, VSTART, VSTOP)
**
**
      DOUBLE PRECISION HSTART, VSTART, VSTOP
**
**
      HSTART =  0.D0
      VSTART =  1.D0
      VSTOP  =  0.D0
      RETURN
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          GTEND                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**       Returns the horizontal stop value for the default
**     region, given the vertical coordinate.
**
      DOUBLE PRECISION FUNCTION GTEND(VERT)
**
**
      DOUBLE PRECISION VERT
**
      GTEND = DSQRT(1.D0 - VERT*VERT) + 1.D-12
      RETURN
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                            GTMAT                                 **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Generates the set of extreme matrices
**
**    INPUTS:  A1,A2   - Coordinates of point for which set
**                       is generated
**             GAIN    - Upper gain of quantizer nonlinearity
**             GAINL   - Lower gain of overflow nonlinearity
**             RHO     - Value which will multiply all of the
**                       extreme matrices to get a measure
**                       of the asymptotic stability of the
**                       system.
**
**    OUTPUTS: EMAT    - Set of extreme matrices
**             NUMEXT  - Number of extreme matrices
**             SFLAG   - Always set to 0 since the filter can
**                       realize any conjugate poles.
**
      SUBROUTINE GTMAT(A1, A2, GAIN, RHO, EMAT, NUMEXT,
     &                 SFLAG, GAINL)
**
**
      DOUBLE PRECISION A1, A2, GAIN, RHO, EMAT(3,3,16)
      DOUBLE PRECISION GAINL
      DOUBLE PRECISION SLOPE(4,2), G1, G2, G3, G4
      INTEGER NUMEXT, SFLAG, I, J, K
**
**
**    Clear flag
      SFLAG = 0
**
**    Set the number of extreme matrices
      NUMEXT = 16
**
**    Set up the slope limits on the nonlinearity
      DO 105 I = 1,4
      SLOPE(I,1) = GAIN
105   SLOPE(I,2) = GAINL * GAIN
**
**
```

```
**    Set up extreme matrices
      DO 110 I = 1, NUMEXT
**
      K = I - 1
      G1 = SLOPE(1, MOD(K,2) + 1)
      G2 = SLOPE(2, MOD(K/2,2) + 1)
      G3 = SLOPE(3, MOD(K/4,2) + 1)
      G4 = SLOPE(4, MOD(K/8,2) + 1)
**
      EMAT(1,1,I) =  G1 * A1
      EMAT(1,2,I) = -G2 * A2
      EMAT(2,1,I) =  G3 * A2
      EMAT(2,2,I) =  G4 * A1
110   CONTINUE
**
**
      DO 170 K = 1, NUMEXT
      DO 170 J = 1, 2
      DO 170 I = 1, 2
170   EMAT(I,J,K) = RHO * EMAT(I,J,K)
**
**
      RETURN
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                        LINIT                                     **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Sets up initialization for the boundary search
**    program.
**
**    INPUTS:  LU     - Logical unit number of output file
**             PARMS  - Parameters passed in by run string
**                        PARMS(1)- type of quantizer
**                                  0  truncation
**                                  1  roundoff
**                        PARMS(2)- type of overflow
**                                  0  none, zeroing, satur
**                                  1  triangular
**                                  2  two's complement
**                        PARMS(3)- type of boundary
**                                  0  stability
**                                  1  eigenvalues < 1
**
**    OUTPUTS: IINIT  - Initial inside search point
**             OINIT  - Initial outside search point
**             SAXIS  - Initial search axis
**
**
      SUBROUTINE LINIT(LU, PARMS, IINIT, OINIT, SAXIS)
      INTEGER LU, PARMS(4), SAXIS, I
      DOUBLE PRECISION IINIT(2), OINIT(2)
**
**
**    Print description
      WRITE(LU, 5)  (PARMS(I), I=1,4)
**
**    Print data range (depending on quantizer)
      IF (PARMS(1).EQ.0) WRITE(LU, 10)  0.0, 1.0,  0., 1.
      IF (PARMS(1).EQ.1) WRITE(LU, 10)  0.0, 0.5,  0., 0.5
**
**    Print symmetry ( 3 = symmetric about both axes)
      WRITE(LU, 15) 3
**
**    Set initial search points and search axis
      IINIT(1) = 0.00D0
      OINIT(1) = IINIT(1)
```

```
      IINIT(2) = 0.00D0
      OINIT(2) = 1.25D0
      SAXIS = 1
**
      RETURN
**
**
5     FORMAT('COUPLED FORM, FOUR QUANTIZERS ',4I5)
10    FORMAT(4F10.4)
15    FORMAT(I2)
      END
**
**
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          DNCHK                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**
**      Checks if we are done with border searching
**    program.  We are done if Y < 0.
**
**
**
**    INPUTS: X, Y  - Coordinates of point
**
**    OUTPUT: DNCHK - '0' if not done
**                    '2' if done
**
**
      INTEGER FUNCTION DNCHK(X, Y)
**
**
      DOUBLE PRECISION X, Y
**
**
      DNCHK = 0
      IF (Y.LT.-.04D0) DNCHK = 2
      RETURN
      END
```

5. Wave filter with two quantizers

Directory

```
FTN4,Y,L
**
**
**********************************************************************
**********************************************************************
**                                                                  **
**                                                                  **
**       WAVE FILTER  --  TWO QUANTIZERS                            **
**                                                                  **
**                                                                  **
**                                                                  **
**    THIS FILE CONTAINS THE ROUTINES FOR USING THE                 **
**  BRAYTON-TONG ALGORITHM PROGRAMS TO FIND THE                     **
**  STABILITY REGIONS FOR A SECOND ORDER WAVE DIGITAL               **
**  FILTER WITH TWO QUANTIZERS.                                     **
**                                                                  **
**********************************************************************
**********************************************************************
**
**
**    FILE "BRAYQ
**
**
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          GETLB                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**    Returns the name of this program version
**
**
      DOUBLE PRECISION FUNCTION GETLB(I)
**
**
      DOUBLE PRECISION LABEL
      DATA LABEL/8HBRAYQ   /
**
      GETLB = LABEL
      RETURN
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          DESCR                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**      Prints any additional description of program version
**
**    INPUT:  LU - Device number of printer
**
      SUBROUTINE DESCR(LU)
**
**
      WRITE(LU, 1)
      RETURN
**
1     FORMAT('  WAVE DIGITAL FILTER WITH TWO',
     &       ' QUANTIZERS.')
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          DFAUL                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Supplies the default values for the region in the
**    parameter plane.
**
**    OUTPUT: HSTART - Start of hor range to check stability
**            VSTART - Start of ver range to check stability
**            VSTOP  - End   of  "    "    "   "       "
**
**
      SUBROUTINE DFAUL(HSTART, VSTART, VSTOP)
**
**
      DOUBLE PRECISION HSTART, VSTART, VSTOP
**
**
      HSTART =  0.D0
      VSTART = 10.D0
      VSTOP  =  0.D0
      RETURN
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          GTEND                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Returns the horizontal stop value for the default
**    region, given the vertical coordinate.
**
      DOUBLE PRECISION FUNCTION GTEND(VERT)
**
**
      DOUBLE PRECISION VERT
**
      GTEND =10.D0 + 1.D-12
      RETURN
      END
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                            GTMAT                                 **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Generates the set of extreme matrices
**
**    INPUTS:  A1,A2   - Coordinates of point for which set
**                       is generated
**             GAIN    - Upper gain of quantizer nonlinearity
**             GAINL   - Lower gain of overflow nonlinearity
**             RHO     - Value which will multiply all of the
**                       extreme matrices to get a measure
**                       of the asymptotic stability of the
**                       system.
**
**    OUTPUTS: EMAT    - Set of extreme matrices
**             NUMEXT  - Number of extreme matrices
**             SFLAG   - Always set to 0 since filter can
**                       realize any parameters.
**
**
      SUBROUTINE GTMAT(A1, A2, GAIN, RHO, EMAT, NUMEXT,
     &                 SFLAG, GAINL)
**
**
      DOUBLE PRECISION A1, A2, GAIN, RHO, EMAT(3,3,16)
      DOUBLE PRECISION E, G, H
      DOUBLE PRECISION GAINL
      DOUBLE PRECISION SLOPE(4,2), G1, G2, G3, G4
      INTEGER NUMEXT, SFLAG, I, J, K
**
**
**
**    Clear flag
      SFLAG = 0
**
**
      E = (-2.D0*A1) / (1.D0 + 2.D0*A1)
      G = -1.D0 / (1.D0 + A1 + A2 + 2.D0*A1*A2)
      H = G * (1.D0 + A1)
**
**    Set the number of extreme matrices
      NUMEXT = 4
```

```
**
**    Set up the slope limits on the nonlinearity
      DO 105 I = 1,4
      SLOPE(I,1) = GAIN
105   SLOPE(I,2) = GAINL
**
**
**    Set up extreme matrices
      DO 110 I = 1, NUMEXT
**
      K = I - 1
      G1 = SLOPE(1, MOD(K,2) + 1)
      G2 = SLOPE(2, MOD(K/2,2) + 1)
**
      EMAT(1,1,I) =  G1 * (-E*G - 2.D0*E - 1.D0)
      EMAT(1,2,I) =  G1 * G
      EMAT(2,1,I) =  G2 * (-E*G - 2.D0*E - E*H)
      EMAT(2,2,I) =  G2 * (G + H + 1.D0)
110   CONTINUE
**
**
      DO 170 K = 1, NUMEXT
      DO 170 J = 1, 2
      DO 170 I = 1, 2
170   EMAT(I,J,K) = RHO * EMAT(I,J,K)
**
**
200   RETURN
      END
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          LINIT                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**      Sets up initialization for the boundary search
**    program.
**
**    CAUTION :  USE BOUNDARY SEARCH PROGRAM FOR ROUNDOFF
**               QUANTIZERS ONLY ! ! !
**
**
**    INPUTS:  LU     - Logical unit number of output file
**             PARMS  - Parameters passed in by run string
**                        PARMS(1)- type of quantizer
**                                  0  truncation
**                                  1  roundoff
**                        PARMS(2)- type of overflow
**                                  0  none, zeroing, satur
**                                  1  triangular
**                                  2  two's complement
**                        PARMS(3)- type of boundary
**                                  0  stability
**                                  1  eigenvalues < 1
**
**    OUTPUTS: IINIT  - Initial inside search point
**             OINIT  - Initial outside search point
**             SAXIS  - Initial search axis
**
**
      SUBROUTINE LINIT(LU, PARMS, IINIT, OINIT, SAXIS)
**
**
      INTEGER LU, PARMS(4), SAXIS, I
      DOUBLE PRECISION IINIT(2), OINIT(2)
**
**
**    Print description
      WRITE(LU, 5)  (PARMS(I), I=1,4)
**
**    Print data range (depending on quantizer)
      WRITE(LU, 10)  0.0, 3.0, 0.0, 3.
**
**    Print symmetry ( 0 = none)
      WRITE(LU, 15) 0
```

```
**
**    Set initial search points and search axis
      IINIT(1) = 1.00D0
      OINIT(1) = IINIT(1)
      IINIT(2) = 0.30D0
      OINIT(2) = 0.00D0
      SAXIS = 3
**
      RETURN
**
**
5     FORMAT('WAVE FILTER, TWO QUANTIZERS ',4I5)
10    FORMAT(4F10.4)
15    FORMAT(I2)
      END
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          DNCHK                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Checks if we are done with border searching
**    program.  We will assume we are done if we
**    are reasonably close to right edge. For
**    two's complement, let border-searching
**    program get back to first boundary point.
**
**    INPUTS: X, Y  - Coordinates of point
**    OUTPUT: DNCHK - '0' if not done
**                    '2' if done
**
      INTEGER FUNCTION DNCHK(X, Y)
**
**
      DOUBLE PRECISION X, Y
**
**
      DNCHK = 0
      IF ((Y.LT.0.D0).AND.(X.GT.2.D0)) DNCHK = 2
      RETURN
      END
```

6. Wave filter with three quantizers

Directory

```
FTN4,Y,L
**
**
*************************************************************************
*************************************************************************
**                                                                     **
**                                                                     **
**       WAVE FILTER  --  THREE QUANTIZERS                             **
**                                                                     **
**                                                                     **
**                                                                     **
**    THIS FILE CONTAINS THE ROUTINES FOR USING THE                    **
**  BRAYTON-TONG ALGORITHM PROGRAMS TO FIND THE                        **
**  STABILITY REGIONS FOR A SECOND ORDER WAVE DIGITAL                  **
**  FILTER WITH THREE QUANTIZERS.                                      **
**                                                                     **
*************************************************************************
*************************************************************************
**
**
**    FILE "BRAYR
**
**
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          GETLB                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**    Returns the name of this program version
**
**
      DOUBLE PRECISION FUNCTION GETLB(I)
**
**
      DOUBLE PRECISION LABEL
      DATA LABEL/8HBRAYR   /
**
      GETLB = LABEL
      RETURN
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                            DESCR                                 **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**       Prints any additional description of program version
**
**     INPUT:  LU - Device number of printer
**
      SUBROUTINE DESCR(LU)
**
**
      WRITE(LU, 1)
      RETURN
**
1     FORMAT('  WAVE DIGITAL FILTER WITH THREE',
     &       ' QUANTIZERS.')
      END
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          DFAUL                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Supplies the default values for the region in the
**    parameter plane.
**
**    OUTPUT: HSTART - Start of hor range to check stability
**            VSTART - Start of ver range to check stability
**            VSTOP  - End   of  "    "    "   "       "
**
**
      SUBROUTINE DFAUL(HSTART, VSTART, VSTOP)
**
**
      DOUBLE PRECISION HSTART, VSTART, VSTOP
**
**
      HSTART =  0.D0
      VSTART = 10.D0
      VSTOP  =  0.D0
      RETURN
      END
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          GTEND                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Returns the horizontal stop value for the default
**    region, given the vertical coordinate.
**
      DOUBLE PRECISION FUNCTION GTEND(VERT)
**
**
      DOUBLE PRECISION VERT
**
      GTEND =10.D0 + 1.D-12
      RETURN
      END
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                            GTMAT                                 **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Generates the set of extreme matrices
**
**    INPUTS:  A1,A2   - Coordinates of point for which set
**                       is generated
**             GAIN    - Upper gain of quantizer nonlinearity
**             GAINL   - Lower gain of overflow nonlinearity
**             RHO     - Value which will multiply all of the
**                       extreme matrices to get a measure
**                       of the asymptotic stability of the
**                       system.
**
**    OUTPUTS: EMAT    - Set of extreme matrices
**             NUMEXT  - Number of extreme matrices
**             SFLAG   - Always set to 0 since filter can
**                       realize any parameters.
**
**
      SUBROUTINE GTMAT(A1, A2, GAIN, RHO, EMAT, NUMEXT,
     &                 SFLAG, GAINL)
**
**
      DOUBLE PRECISION A1, A2, GAIN, RHO, EMAT(3,3,16)
      DOUBLE PRECISION GAINL
      DOUBLE PRECISION SLOPE(8,2), G1, G2, G3, G4, G5, X(3)
      DOUBLE PRECISION E, G, H
      INTEGER NUMEXT, SFLAG, I, J, K
**
**
**    Clear flag
      SFLAG = 0
**
**
      E = (-2.D0*A1) / (1.D0 + 2.D0*A1)
      G = -1.D0 / (1.D0 + A1 + A2 + 2.D0*A1*A2)
      H = G * (1.D0 + A1)
**
**    Set the number of extreme matrices
      NUMEXT = 16
**
```

```
**
**    Set up the slope limits on the nonlinearities
      DO 105 I = 1,3
      SLOPE(I,1) = GAIN
105   SLOPE(I,2) = GAINL
**
      DO 110 I = 4,5
      SLOPE(I,1) = GAIN*GAIN
110   SLOPE(I,2) = GAINL*GAINL
**
**    Initialize for finding min and max values for
**      slopes 6 - 8
      G1 = SLOPE(1,1)
      G2 = SLOPE(2,1)
      G3 = SLOPE(3,1)
      G4 = SLOPE(4,1)
      G5 = SLOPE(5,1)
      X(1) = -1.D0 - 2.D0*E*G1 - E*G*G4
      X(2) = -2.D0*E*G1 - E*G*G4 - E*H*G5
      X(3) = 1.D0 + G*G2 + H*G3
      SLOPE(6,1) = X(1)
      SLOPE(6,2) = X(1)
      SLOPE(7,1) = X(2)
      SLOPE(7,2) = X(2)
      SLOPE(8,1) = X(3)
      SLOPE(8,2) = X(3)
**
**    Find min and max values for slopes 6 - 8
      DO 120 I = 1, 32
      K = I - 1
**
      G1 = SLOPE(1, MOD(K,2) + 1)
      G2 = SLOPE(2, MOD(K/2,2) + 1)
      G3 = SLOPE(3, MOD(K/4,2) + 1)
      G4 = SLOPE(4, MOD(K/8,2) + 1)
      G5 = SLOPE(5, MOD(K/16,2) + 1)
**
      X(1) = -1.D0 - 2.D0*E*G1 - E*G*G4
      X(2) = -2.D0*E*G1 - E*G*G4 - E*H*G5
      X(3) = 1.D0 + G*G2 + H*G3
**
      SLOPE(6,1) = DMAX1(SLOPE(6,1), X(1))
      SLOPE(6,2) = DMIN1(SLOPE(6,2), X(1))
      SLOPE(7,1) = DMAX1(SLOPE(7,1), X(2))
      SLOPE(7,2) = DMIN1(SLOPE(7,2), X(2))
      SLOPE(8,1) = DMAX1(SLOPE(8,1), X(3))
      SLOPE(8,2) = DMIN1(SLOPE(8,2), X(3))
***
```

```
***   WRITE(6,4) X(3), SLOPE(8,1), SLOPE(8,2)
***4  FORMAT(1X,3F12.5)
**
120   CONTINUE
**
**    Set up extreme matrices
      DO 130 I = 1, NUMEXT
**
      K = I - 1
      G1 = SLOPE(6, MOD(K,2) + 1)
      G2 = SLOPE(2, MOD(K/2,2) + 1)
      G3 = SLOPE(7, MOD(K/4,2) + 1)
      G4 = SLOPE(8, MOD(K/8,2) + 1)
**
      EMAT(1,1,I) =  G1
      EMAT(1,2,I) =  G2*G
      EMAT(2,1,I) =  G3
      EMAT(2,2,I) =  G4
130   CONTINUE
**
**
      DO 170 K = 1, NUMEXT
      DO 170 J = 1, 2
      DO 170 I = 1, 2
170   EMAT(I,J,K) = RHO * EMAT(I,J,K)
**
**
200   RETURN
      END
```

7. Lattice filter with two quantizers

Directory

```
FTN4,Y,L
**
**
**********************************************************************
**********************************************************************
**                                                                  **
**                                                                  **
**       LATTICE FILTER  --  TWO QUANTIZERS (AT STATES)             **
**                                                                  **
**                                                                  **
**                                                                  **
**    THIS FILE CONTAINS THE ROUTINES FOR USING THE                 **
**  BRAYTON-TONG ALGORITHM PROGRAMS TO FIND THE                     **
**  STABILITY REGIONS FOR A SECOND ORDER LATTICE                    **
**  DIGITAL FILTER WITH TWO QUANTIZERS AT STATES.                   **
**                                                                  **
**********************************************************************
**********************************************************************
**
**
**    FILE "BRAYO
**
**
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          GETLB                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**    Returns the name of this program version
**
**
      DOUBLE PRECISION FUNCTION GETLB(I)
**
**
      DOUBLE PRECISION LABEL
      DATA LABEL/8HBRAYO   /
**
      GETLB = LABEL
      RETURN
      END
**
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          DESCR                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Prints any additional description of program version
**
**    INPUT:  LU - Device number of printer
**
      SUBROUTINE DESCR(LU)
**
**
      WRITE(LU, 1)
      RETURN
**
1     FORMAT('  GRAY-MARKEL LATTICE FORM FILTER WITH TWO',
     &       ' QUANTIZERS.',/,
     &       '  QUANTIZATION APPLIED AT STATES')
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          DFAUL                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Supplies the default values for the square in the
**    parameter plane.
**
**    OUTPUT: HSTART - Start of hor range to check stability
**            VSTART - Start of ver range to check stability
**            VSTOP  - End   of  "    "    "   "       "
**
**
      SUBROUTINE DFAUL(HSTART, VSTART, VSTOP)
**
**
      DOUBLE PRECISION HSTART, VSTART, VSTOP
**
**
      HSTART =  0.D0
      VSTART =  1.D0
      VSTOP  = -1.D0
      RETURN
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          GTEND                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Returns the horizontal stop value for the default
**    region, given the vertical coordinate.
**
      DOUBLE PRECISION FUNCTION GTEND(VERT)
**
**
      DOUBLE PRECISION VERT
**
      GTEND = 1.D0 + 1.D-12
      RETURN
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                            GTMAT                                 **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Generates the set of extreme matrices
**
**    INPUTS:  B1,B2   - Coordinates of point for which set
**                       is generated
**             GAIN    - Upper gain of quantizer nonlinearity
**             GAINL   - Lower gain of overflow nonlinearity
**             RHO     - Value which will multiply all of the
**                       extreme matrices to get a measure
**                       of the asymptotic stability of the
**                       system.
**
**    OUTPUTS: EMAT    - Set of extreme matrices
**             NUMEXT  - Number of extreme matrices
**             SFLAG   - Always set to 0 since the filter can
**                       realize any parameters
**
**
      SUBROUTINE GTMAT(B1, B2, GAIN, RHO, EMAT, NUMEXT,
     &                 SFLAG, GAINL)
**
**
      DOUBLE PRECISION B1, B2, GAIN, RHO, EMAT(3,3,16)
      DOUBLE PRECISION GAINL
      DOUBLE PRECISION SLOPE(4,2), G1, G2, G3, G4
      INTEGER NUMEXT, SFLAG, I, J, K
**
**
**    Clear the flag
      SFLAG = 0
**
**    Set the number of extreme matrices
      NUMEXT = 4
**
**    Set up the slope limits on the nonlinearity
      DO 105 I = 1,4
      SLOPE(I,1) = GAIN
105   SLOPE(I,2) = GAINL
**
**
```

```
**    Set up extreme matrices
      DO 110 I = 1, NUMEXT
**
      K = I - 1
      G1 = SLOPE(1, MOD(K,2) + 1)
      G2 = SLOPE(2, MOD(K/2,2) + 1)
**
      EMAT(1,1,I) = -G1 * B1
      EMAT(1,2,I) = -G1 * B2
      EMAT(2,1,I) =  G2 * (1.D0 - B1*B1)
      EMAT(2,2,I) = -G2 * B1 * B2
110   CONTINUE
**
**
      DO 170 K = 1, NUMEXT
      DO 170 J = 1, 2
      DO 170 I = 1, 2
170   EMAT(I,J,K) = RHO * EMAT(I,J,K)
**
**
      RETURN
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          LINIT                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Sets up initialization for the boundary search
**    program.
**
**    INPUTS:  LU     - Logical unit number of output file
**             PARMS  - Parameters passed in by run string
**                        PARMS(1)- type of quantizer
**                                  0  truncation
**                                  1  roundoff
**                        PARMS(2)- type of overflow
**                                  0  none, zeroing, satur
**                                  1  triangular
**                                  2  two's complement
**                        PARMS(3)- type of boundary
**                                  0  stability
**                                  1  eigenvalues < 1
**
**    OUTPUTS: IINIT  - Initial inside search point
**             OINIT  - Initial outside search point
**             SAXIS  - Initial search axis
**
**
      SUBROUTINE LINIT(LU, PARMS, IINIT, OINIT, SAXIS)
**
**
      INTEGER LU, PARMS(4), SAXIS, I
      DOUBLE PRECISION IINIT(2), OINIT(2)
**
**
**    Print description
      WRITE(LU, 5)  (PARMS(I), I=1,4)
**
**    Print data range (depending on quantizer)
      IF (PARMS(1).EQ.0) WRITE(LU, 10)  0.0, 1.0, -1., 1.
      IF (PARMS(1).EQ.1) WRITE(LU, 10)  0.0,  .5, -.5,  .5
**
**    Print symmetry ( 1 = symmetric about y axis)
      WRITE(LU, 15) 1
**
**
```

```
**    Set initial search points and search axis
      IINIT(1) = 0.00D0
      OINIT(1) = IINIT(1)
      IINIT(2) = 0.00D0
      OINIT(2) = 1.25D0
      SAXIS = 1
**
**
      RETURN
**
**
5     FORMAT('LATTICE FILTER, TWO QUANTIZERS ',4I5)
10    FORMAT(4F10.4)
15    FORMAT(I2)
      END
**
**
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          DNCHK                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Checks if we are done with border searching
**    program.  Since these plots are all symmetric
**    about Y axis, we are done when X < 0.
**
**
**    INPUTS: X, Y  - Coordinates of point
**
**    OUTPUT: DNCHK - '0' if not done
**                    '2' if done
**
**
      INTEGER FUNCTION DNCHK(X, Y)
**
**
      DOUBLE PRECISION X, Y
**
**
      DNCHK = 0
      IF (X.LT.-.04D0) DNCHK = 2
      RETURN
      END
```

8. Lattice filter with three quantizers and no overflow

Directory

```
FTN4,Y,L
**
**
**********************************************************************
**********************************************************************
**                                                                  **
**                                                                  **
**       LATTICE FILTER  --  THREE QUANTIZERS                       **
**                             (NO OVERFLOW)                        **
**                                                                  **
**                                                                  **
**                                                                  **
**                                                                  **
**    THIS FILE CONTAINS THE ROUTINES FOR USING THE                 **
**  BRAYTON-TONG ALGORITHM PROGRAMS TO FIND THE                     **
**  STABILITY REGIONS FOR A SECOND ORDER LATTICE                    **
**  DIGITAL FILTER WITH THREE QUANTIZERS AND NO                     **
**  OVERFLOW.                                                       **
**                                                                  **
**********************************************************************
**********************************************************************
**
**
**    FILE &BRAYL
**
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                                                                  **
**                          GETLB                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**    Returns the name of this program version
**
**
      DOUBLE PRECISION FUNCTION GETLB(I)
**
**
      DOUBLE PRECISION LABEL
      DATA LABEL/8HBRAYL   /
**
      GETLB = LABEL
      RETURN
      END
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                            DESCR                                 **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Prints any additional description of program version
**
**    INPUT:  LU - Device number of printer
**
      SUBROUTINE DESCR(LU)
**
**
      WRITE(LU, 1)
      RETURN
**
1     FORMAT('  GRAY-MARKEL LATTICE FORM FILTER WITH THREE',
     &       '  QUANTIZERS.',/,'  NO OVERFLOW.')
      END
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          DFAUL                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Supplies the default values for the square in the
**    parameter plane.
**
**    OUTPUT: HSTART - Start of hor range to check stability
**            VSTART - Start of ver range to check stability
**            VSTOP  - End   of  "    "    "   "       "
**
**
      SUBROUTINE DFAUL(HSTART, VSTART, VSTOP)
**
**
      DOUBLE PRECISION HSTART, VSTART, VSTOP
**
**
      HSTART =  0.D0
      VSTART =  1.D0
      VSTOP  = -1.D0
      RETURN
      END
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          GTEND                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Returns the horizontal stop value for the default
**    region, given the vertical coordinate.
**
      DOUBLE PRECISION FUNCTION GTEND(VERT)
**
**
      DOUBLE PRECISION VERT
**
      GTEND = 1.D0 + 1.D-12
      RETURN
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                            GTMAT                                 **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Generates the set of extreme matrices
**
**    INPUTS:  B1,B2   - Coordinates of point for which set
**                         is generated
**             GAIN    - Upper gain of quantizer nonlinearity
**             GAINL   - Ignored
**             RHO     - Value which will multiply all of the
**                         extreme matrices to get a measure
**                         of the asymptotic stability of the
**                         system.
**
**    OUTPUTS: EMAT    - Set of extreme matrices
**             NUMEXT  - Number of extreme matrices
**             SFLAG   - Always set to 0 since the filter can
**                         realize any parameters
**
**
      SUBROUTINE GTMAT(B1, B2, GAIN, RHO, EMAT, NUMEXT,
     &                 SFLAG, GAINL)
**
**
      DOUBLE PRECISION B1, B2, GAIN, RHO, EMAT(3,3,16)
      DOUBLE PRECISION GAINL
      DOUBLE PRECISION SLOPE(6,2), G1, G2, G3, G4, G5, X
      INTEGER NUMEXT, SFLAG, I, J, K
**
**
**    Clear the flag
      SFLAG = 0
**
**    Set the number of extreme matrices
      NUMEXT = 16
**
**    Set up the slope limits on the nonlinearities
      DO 125 I = 1,2
      SLOPE(I,1) = GAIN
125   SLOPE(I,2) = 0.D0
      SLOPE(3,1) = GAIN*GAIN
      SLOPE(3,2) = 0.D0
```

```
      SLOPE(4,1) = 1.D0
      SLOPE(4,2) = 1.D0 - B1*B1*GAIN*GAIN
**
**
**    Set up extreme matrices
      DO 130 I = 1, NUMEXT
**
      K = I - 1
      G1 = SLOPE(1, MOD(K,2) + 1)
      G2 = SLOPE(2, MOD(K/2,2) + 1)
      G3 = SLOPE(3, MOD(K/4,2) + 1)
      G4 = SLOPE(4, MOD(K/8,2) + 1)
**
      EMAT(1,1,I) = -G1 * B1
      EMAT(1,2,I) = -G2 * B2
      EMAT(2,1,I) =  G4
      EMAT(2,2,I) = -G3 * B1 * B2
130   CONTINUE
**
**
      DO 170 K = 1, NUMEXT
      DO 170 J = 1, 2
      DO 170 I = 1, 2
170   EMAT(I,J,K) = RHO * EMAT(I,J,K)
**
**
      RETURN
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          LINIT                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Sets up initialization for the boundary search
**    program.
**
**    INPUTS:  LU     - Logical unit number of output file
**             PARMS  - Parameters passed in by run string
**                       PARMS(1)- type of quantizer
**                                 0  truncation
**                                 1  roundoff
**                       PARMS(2)- type of overflow
**                                 0  none, zeroing, satur
**                                 1  triangular
**                                 2  two's complement
**                       PARMS(3)- type of boundary
**                                 0  stability
**                                 1  eigenvalues < 1
**
**    OUTPUTS: IINIT  - Initial inside search point
**             OINIT  - Initial outside search point
**             SAXIS  - Initial search axis
**
**
      SUBROUTINE LINIT(LU, PARMS, IINIT, OINIT, SAXIS)
**
**
      INTEGER LU, PARMS(4), SAXIS, I
      DOUBLE PRECISION IINIT(2), OINIT(2)
**
**
**    Print description
      WRITE(LU, 5)  (PARMS(I), I=1,4)
**
**    Print data range (depending on quantizer)
      IF (PARMS(1).EQ.0) WRITE(LU, 10)  0.0, 1.0, -1., 1.
      IF (PARMS(1).EQ.1) WRITE(LU, 10)  0.0,  .5, -.5,  .5
**
**    Print symmetry ( 1 = symmetric about y axis)
      WRITE(LU, 15) 1
**
```

```
**    Set initial search points and search axis
      IINIT(1) = 0.00D0
      OINIT(1) = IINIT(1)
      IINIT(2) = 0.00D0
      OINIT(2) = 1.25D0
      SAXIS = 1
**
      RETURN
**
**
5     FORMAT('LATTICE FILTER, THREE QUANTIZERS ',4I5)
10    FORMAT(4F10.4)
15    FORMAT(I2)
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          DNCHK                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**
**      Checks if we are done with border searching
**    program.  Since these plots are all symmetric
**    about Y axis, we are done when X < 0.
**
**    INPUTS: X, Y  - Coordinates of point
**
**    OUTPUT: DNCHK - '0' if not done
**                    '2' if done
**
**
      INTEGER FUNCTION DNCHK(X, Y)
**
**
      DOUBLE PRECISION X, Y
**
**
      DNCHK = 0
      IF (X.LT.-.04D0) DNCHK = 2
      RETURN
      END
```

9. Lattice filter with three quantizers and overflow

Directory

```
FTN4,Y,L
**
**
**********************************************************************
**********************************************************************
**                                                                  **
**                                                                  **
**        LATTICE FILTER  --  THREE QUANTIZERS                      **
**                                                                  **
**                                                                  **
**                                                                  **
**     THIS FILE CONTAINS THE ROUTINES FOR USING THE                **
**   BRAYTON-TONG ALGORITHM PROGRAMS TO FIND THE                    **
**   STABILITY REGIONS FOR A SECOND ORDER LATTICE                   **
**   DIGITAL FILTER WITH THREE QUANTIZERS.                          **
**                                                                  **
**********************************************************************
**********************************************************************
**
**
**     FILE "BRAYM
**
**
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          GETLB                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**    Returns the name of this program version
**
**
      DOUBLE PRECISION FUNCTION GETLB(I)
**
**
      DOUBLE PRECISION LABEL
      DATA LABEL/8HBRAYM   /
**
      GETLB = LABEL
      RETURN
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                            DESCR                                 **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Prints any additional description of program version
**
**    INPUT:  LU - Device number of printer
**
      SUBROUTINE DESCR(LU)
**
      WRITE(LU, 1)
      RETURN
**
1     FORMAT('  GRAY-MARKEL LATTICE FORM FILTER WITH THREE',
     &       '  QUANTIZERS.',/)
      END
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          DFAUL                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Supplies the default values for the square in the
**    parameter plane.
**
**    OUTPUT: HSTART - Start of hor range to check stability
**            VSTART - Start of ver range to check stability
**            VSTOP  - End   of  "    "    "   "       "
**
**
      SUBROUTINE DFAUL(HSTART, VSTART, VSTOP)
**
      DOUBLE PRECISION HSTART, VSTART, VSTOP
**
**
      HSTART =  0.D0
      VSTART =  1.D0
      VSTOP  = -1.D0
      RETURN
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                              GTEND                               **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Returns the horizontal stop value for the default
**    region, given the vertical coordinate.
**
      DOUBLE PRECISION FUNCTION GTEND(VERT)
**
      DOUBLE PRECISION VERT
**
      GTEND = 1.D0 + 1.D-12
      RETURN
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                            GTMAT                                 **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Generates the set of extreme matrices
**
**    INPUTS:  B1,B2   - Coordinates of point for which set
**                       is generated
**             GAIN    - Upper gain of quantizer nonlinearity
**             GAINL   - Lower gain of overflow nonlinearity
**             RHO     - Value which will multiply all of the
**                       extreme matrices to get a measure
**                       of the asymptotic stability of the
**                       system.
**
**    OUTPUTS: EMAT    - Set of extreme matrices
**             NUMEXT  - Number of extreme matrices
**             SFLAG   - Always set to 0 since the filter can
**                       realize any parameters
**
**
      SUBROUTINE GTMAT(B1, B2, GAIN, RHO, EMAT, NUMEXT,
     &                 SFLAG, GAINL)
**
**
      DOUBLE PRECISION B1, B2, GAIN, RHO, EMAT(3,3,16)
      DOUBLE PRECISION GAINL
      DOUBLE PRECISION SLOPE(6,2), G1, G2, G3, G4, G5, X
      INTEGER NUMEXT, SFLAG, I, J, K
**
**    Clear the flag
      SFLAG = 0
**
**    Set the number of extreme matrices
      NUMEXT = 16
**
**    Set up the slope limits on the nonlinearities
**      to be used to get min and max values of slope 6
      DO 105 I = 1,3
      SLOPE(I,1) = GAIN
105   SLOPE(I,2) = 0.D0
**
**
```

```
      DO 110 I = 4,5
      SLOPE(I,1) = 1.D0
110   SLOPE(I,2) = GAINL
**
**    Initialize for finding min and max values for
**      slope  6
      G1 = SLOPE(1,1)
      G3 = SLOPE(3,1)
      G4 = SLOPE(4,1)
      G5 = SLOPE(5,1)
      X  = G5*( 1.D0 - B1*B1*G1*G3*G4 )
      SLOPE(6,1) = X
      SLOPE(6,2) = X
**
**    Find min and max values for slope 6
      DO 120 I = 1, 16
      K = I - 1
**
      G1 = SLOPE(1, MOD(K,2) + 1)
      G3 = SLOPE(3, MOD(K/2,2) + 1)
      G4 = SLOPE(4, MOD(K/4,2) + 1)
      G5 = SLOPE(5, MOD(K/8,2) + 1)
**
      X  = G5*( 1.D0 - B1*B1*G1*G3*G4 )
**
      SLOPE(6,1) = DMAX1(SLOPE(6,1), X)
      SLOPE(6,2) = DMIN1(SLOPE(6,2), X)
***
***   WRITE(6,4) X, SLOPE(6,1), SLOPE(6,2)
***4  FORMAT(1X,3F12.5)
**
120   CONTINUE
**
**    Set up the slope limits on the other nonlinearities
      DO 125 I = 1,2
      SLOPE(I,1) = GAIN
125   SLOPE(I,2) = GAINL*GAIN
      SLOPE(3,1) = GAIN*GAIN
      SLOPE(3,2) = GAIN*GAIN*GAINL
**
**    Set up extreme matrices
      DO 130 I = 1, NUMEXT
**
      K = I - 1
      G1 = SLOPE(1, MOD(K,2) + 1)
      G2 = SLOPE(2, MOD(K/2,2) + 1)
      G3 = SLOPE(3, MOD(K/4,2) + 1)
      G6 = SLOPE(6, MOD(K/8,2) + 1)
```

```
**
      EMAT(1,1,I) = -G1 * B1
      EMAT(1,2,I) = -G2 * B2
      EMAT(2,1,I) =  G6
      EMAT(2,2,I) = -G3 * B1 * B2
130   CONTINUE
**
**
      DO 170 K = 1, NUMEXT
      DO 170 J = 1, 2
      DO 170 I = 1, 2
170   EMAT(I,J,K) = RHO * EMAT(I,J,K)
**
**
      RETURN
      END
**
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          LINIT                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**
**      Sets up initialization for the boundary search
**    program.
**
**    INPUTS:  LU     - Logical unit number of output file
**             PARMS  - Parameters passed in by run string
**                        PARMS(1)- type of quantizer
**                                  0  truncation
**                                  1  roundoff
**                        PARMS(2)- type of overflow
**                                  0  none, zeroing, satur
**                                  1  triangular
**                                  2  two's complement
**                        PARMS(3)- type of boundary
**                                  0  stability
**                                  1  eigenvalues < 1
**
**    OUTPUTS: IINIT  - Initial inside search point
**             OINIT  - Initial outside search point
**             SAXIS  - Initial search axis
**
**
      SUBROUTINE LINIT(LU, PARMS, IINIT, OINIT, SAXIS)
**
      INTEGER LU, PARMS(4), SAXIS, I
      DOUBLE PRECISION IINIT(2), OINIT(2)
**
**
**    Print description
      WRITE(LU, 5)  (PARMS(I), I=1,4)
**
**    Print data range (depending on quantizer)
      IF (PARMS(1).EQ.0) WRITE(LU, 10)  0.0, 1.0, -1., 1.
      IF (PARMS(1).EQ.1) WRITE(LU, 10)  0.0,  .5, -.5,  .5
**
**    Print symmetry ( 1 = symmetric about y axis)
      WRITE(LU, 15) 1
**
**    Set initial search points and search axis
```

```
      IINIT(1) = 0.00D0
      OINIT(1) = IINIT(1)
      IINIT(2) = 0.00D0
      OINIT(2) = 1.25D0
      SAXIS = 1
**
      RETURN
**
**
5     FORMAT('LATTICE FILTER, THREE QUANTIZERS ',4I5)
10    FORMAT(4F10.4)
15    FORMAT(I2)
      END
**
**
```

```
**********************************************************************
**                                                                  **
**                                                                  **
**                          DNCHK                                   **
**                                                                  **
**                                                                  **
**********************************************************************
**
**      Checks if we are done with border searching
**    program.  Since these plots are all symmetric
**    about Y axis, we are done when X < 0.
**
**    INPUTS: X, Y  - Coordinates of point
**    OUTPUT: DNCHK - '0' if not done
**                    '2' if done
**
**
      INTEGER FUNCTION DNCHK(X, Y)
**
      DOUBLE PRECISION X, Y
**
**
      DNCHK = 0
      IF (X.LT.-.04D0) DNCHK = 2
      RETURN
      END
```