Digital Signal Processing: A User's Guide

Collection Editor:
Douglas L. Jones

Digital Signal Processing: A User's Guide

Collection Editor:
Douglas L. Jones

Authors:

Richard Baraniuk
Benjamin Fite
Anders Gjendemsjø
Michael Haag
Don Johnson
Douglas L. Jones
Stephen Kruzick
Robert Nowak
Ricardo Radaelli-Sanchez
Justin Romberg
Clayton Scott
Ivan Selesnick
Melissa Selik

Online:
< http://cnx.org/content/col10372/1.2/ >

C O N N E X I O N S

Rice University, Houston, Texas

Collection structure revised: August 29, 2006
PDF generated: July 29, 2010
For copyright and attribution information for the modules contained in this collection, see p. 312.

Table of Contents

Preface for Digital Signal Processing: A User's Guide[1]

Digital signal processing (DSP) has matured in the past few decades from an obscure research discipline to a large body of practical methods with very broad application. Both practicing engineers and students specializing in signal processing need a clear exposition of the ideas and methods comprising the core signal processing "toolkit" so widely used today.

This text reflects my belief that the skilled practitioner must understand the key ideas underlying the algorithms to select, apply, debug, extend, and innovate most effectively; only with real insight can the engineer make novel use of these methods in the seemingly infinite range of new problems and applications. It also reflects my belief that the needs of the typical student and the practicing engineer have converged in recent years; as the discipline of signal processing has matured, these core topics have become less a subject of active research and more a set of tools applied in the course of other research. The modern student thus has less need for exhaustive coverage of the research literature and detailed derivations and proofs as preparation for their own research on these topics, but greater need for intuition and practical guidance in their most effective use. The majority of students eventually become practicing engineers themselves and benefit from the best preparation for their future careers.

This text both explains the principles of classical signal processing methods and describes how they are used in engineering practice. It is thus much more than a recipe book; it describes the ideas behind the algorithms, gives analyses when they enhance that understanding, and includes derivations that the practitioner may need to extend when applying these methods to new situations. Analyses or derivations that are only of research interest or that do not increase intuitive understanding are left to the references. It is also much more than a theory book; it contains more description of common applications, discussion of actual implementation issues, comments on what really works in the real world, and practical "know-how" than found in the typical academic textbook. The choice of material emphasizes those methods that have found widespread practical use; techniques that have been the subject of intense research but which are rarely used in practice (for example, RLS adaptive filter algorithms) often receive only limited coverage.

The text assumes a familiarity with basic signal processing concepts such as ideal sampling theory, continuous and discrete Fourier transforms, convolution and filtering. It evolved from a set of notes for a second signal processing course, ECE 451: Digital Signal Processing II, in Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign, aimed at second-semester seniors or first-semester graduate students in signal processing. Over the years, it has been enhanced substantially to include descriptions of common applications, sometimes hard-won knowledge about what actually works and what doesn't, useful tricks, important extensions known to experienced engineers but rarely discussed in academic texts, and other relevant "know-how" to aid the real-world user. This is necessarily an ongoing process, and I continue to expand and refine this component as my own practical knowledge and experience grows. The topics are the core signal processing methods that are used in the majority of signal processing applications; discrete Fourier analysis and FFTs, digital filter design, adaptive filtering, multirate signal processing, and efficient algorithm implementation and finite-precision issues. While many of these topics are covered at an intro-

[1] This content is available online at <http://cnx.org/content/m13782/1.1/>.

ductory level in a first course, this text aspires to cover all of the methods, both basic and advanced, in these areas which see widespread use in practice. I have also attempted to make the individual modules and sections somewhat self-sufficient, so that those who seek specific information on a single topic can quickly find what they need. Hopefully these aspirations will eventually be achieved; in the meantime, I welcome your comments, corrections, and feedback so that I can continue to improve this text.

As of August 2006, the majority of modules are unedited transcriptions of handwritten notes and may contain typographical errors and insufficient descriptive text for documents unaccompanied by an oral lecture; I hope to have all of the modules in at least presentable shape by the end of the year.

Publication of this text in Connexions would have been impossible without the help of many people. A huge thanks to the various permanent and temporary staff at Connexions is due, in particular to those who converted the text and equations from my original handwritten notes into CNXML and MathML. My former and current faculty colleagues at the University of Illinois who have taught the second DSP course over the years have had a substantial influence on the evolution of the content, as have the students who have inspired this work and given me feedback. I am very grateful to my teachers, mentors, colleagues, collaborators, and fellow engineers who have taught me the art and practice of signal processing; this work is dedicated to you.

Chapter 1

Background, Review, and Reference

1.1 Discrete-Time Signals and Systems[1]

Mathematically, analog signals are functions having as their independent variables continuous quantities, such as space and time. Discrete-time signals are functions defined on the integers; they are sequences. As with analog signals, we seek ways of decomposing discrete-time signals into simpler components. Because this approach leading to a better understanding of signal structure, we can exploit that structure to represent information (create ways of representing information with signals) and to extract information (retrieve the information thus represented). For symbolic-valued signals, the approach is different: We develop a common representation of all symbolic-valued signals so that we can embody the information they contain in a unified way. From an information representation perspective, the most important issue becomes, for both real-valued and symbolic-valued signals, efficiency: what is the most parsimonious and compact way to represent information so that it can be extracted later.

1.1.1 Real- and Complex-valued Signals

A discrete-time signal is represented symbolically as $s(n)$, where $n = \{\ldots, -1, 0, 1, \ldots\}$.

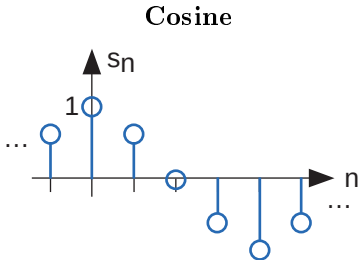

Figure 1.1: The discrete-time cosine signal is plotted as a stem plot. Can you find the formula for this signal?

We usually draw discrete-time signals as stem plots to emphasize the fact they are functions defined only on the integers. We can delay a discrete-time signal by an integer just as with analog ones. A signal delayed by m samples has the expression $s(n - m)$.

[1] This content is available online at <http://cnx.org/content/m10342/2.15/>.

1.1.2 Complex Exponentials

The most important signal is, of course, the **complex exponential sequence**.

$$s(n) = e^{j2\pi fn} \tag{1.1}$$

Note that the frequency variable f is dimensionless and that adding an integer to the frequency of the discrete-time complex exponential has no effect on the signal's value.

$$\begin{aligned} e^{j2\pi(f+m)n} &= e^{j2\pi fn}e^{j2\pi mn} \\ &= e^{j2\pi fn} \end{aligned} \tag{1.2}$$

This derivation follows because the complex exponential evaluated at an integer multiple of 2π equals one. Thus, we need only consider frequency to have a value in some unit-length interval.

1.1.3 Sinusoids

Discrete-time sinusoids have the obvious form $s(n) = A\cos(2\pi fn + \phi)$. As opposed to analog complex exponentials and sinusoids that can have their frequencies be any real value, frequencies of their discrete-time counterparts yield unique waveforms **only** when f lies in the interval $\left(-\left(\frac{1}{2}\right), \frac{1}{2}\right]$. This choice of frequency interval is arbitrary; we can also choose the frequency to lie in the interval $[0, 1)$. How to choose a unit-length interval for a sinusoid's frequency will become evident later.

1.1.4 Unit Sample

The second-most important discrete-time signal is the **unit sample**, which is defined to be

$$\delta(n) = \begin{cases} 1 \text{ if } n = 0 \\ 0 \text{ otherwise} \end{cases} \tag{1.3}$$

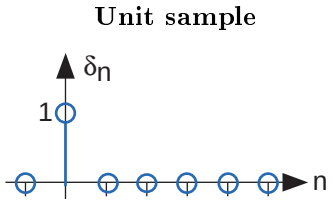

Figure 1.2: The unit sample.

Examination of a discrete-time signal's plot, like that of the cosine signal shown in Figure 1.1 (Cosine), reveals that all signals consist of a sequence of delayed and scaled unit samples. Because the value of a sequence at each integer m is denoted by $s(m)$ and the unit sample delayed to occur at m is written $\delta(n-m)$, we can decompose **any** signal as a sum of unit samples delayed to the appropriate location and scaled by the signal value.

$$s(n) = \sum_{m=-\infty}^{\infty} (s(m)\,\delta(n-m)) \tag{1.4}$$

This kind of decomposition is unique to discrete-time signals, and will prove useful subsequently.

1.1.5 Unit Step

The **unit sample** in discrete-time is well-defined at the origin, as opposed to the situation with analog signals.

$$u(n) = \begin{cases} 1 \text{ if } n \geq 0 \\ 0 \text{ if } n < 0 \end{cases} \tag{1.5}$$

1.1.6 Symbolic Signals

An interesting aspect of discrete-time signals is that their values do not need to be real numbers. We do have real-valued discrete-time signals like the sinusoid, but we also have signals that denote the sequence of characters typed on the keyboard. Such characters certainly aren't real numbers, and as a collection of possible signal values, they have little mathematical structure other than that they are members of a set. More formally, each element of the **symbolic-valued** signal $s(n)$ takes on one of the values $\{a_1, \ldots, a_K\}$ which comprise the **alphabet** A. This technical terminology does not mean we restrict symbols to being members of the English or Greek alphabet. They could represent keyboard characters, bytes (8-bit quantities), integers that convey daily temperature. Whether controlled by software or not, discrete-time systems are ultimately constructed from digital circuits, which consist **entirely** of analog circuit elements. Furthermore, the transmission and reception of discrete-time signals, like e-mail, is accomplished with analog signals and systems. Understanding how discrete-time and analog signals and systems intertwine is perhaps the main goal of this course.

1.1.7 Discrete-Time Systems

Discrete-time systems can act on discrete-time signals in ways similar to those found in analog signals and systems. Because of the role of software in discrete-time systems, many more different systems can be envisioned and "constructed" with programs than can be with analog signals. In fact, a special class of analog signals can be converted into discrete-time signals, processed with software, and converted back into an analog signal, all without the incursion of error. For such signals, systems can be easily produced in software, with equivalent analog realizations difficult, if not impossible, to design.

1.2 Systems in the Time-Domain[2]

A discrete-time signal $s(n)$ is **delayed** by n_0 samples when we write $s(n - n_0)$, with $n_0 > 0$. Choosing n_0 to be negative advances the signal along the integers. As opposed to analog delays[3], discrete-time delays can **only** be integer valued. In the frequency domain, delaying a signal corresponds to a linear phase shift of the signal's discrete-time Fourier transform: $\left(s(n - n_0) \leftrightarrow e^{-(j2\pi f n_0)} S\left(e^{j2\pi f}\right)\right)$.

Linear discrete-time systems have the superposition property.

Superposition

$$S(a_1 x_1(n) + a_2 x_2(n)) = a_1 S(x_1(n)) + a_2 S(x_2(n)) \tag{1.6}$$

A discrete-time system is called **shift-invariant** (analogous to time-invariant analog systems) if delaying the input delays the corresponding output.

Shift-Invariant

$$\text{If} \quad S(x(n)) = y(n), \text{ Then} \quad S(x(n - n_0)) = y(n - n_0) \tag{1.7}$$

We use the term shift-invariant to emphasize that delays can only have integer values in discrete-time, while in analog signals, delays can be arbitrarily valued.

[2] This content is available online at <http://cnx.org/content/m0508/2.7/>.
[3] "Simple Systems": Section Delay <http://cnx.org/content/m0006/latest/#delay>

We want to concentrate on systems that are both linear and shift-invariant. It will be these that allow us the full power of frequency-domain analysis and implementations. Because we have no physical constraints in "constructing" such systems, we need only a mathematical specification. In analog systems, the differential equation specifies the input-output relationship in the time-domain. The corresponding discrete-time specification is the **difference equation**.

The Difference Equation

$$y(n) = a_1 y(n-1) + \cdots + a_p y(n-p) + b_0 x(n) + b_1 x(n-1) + \cdots + b_q x(n-q) \tag{1.8}$$

Here, the output signal $y(n)$ is related to its **past** values $y(n-l)$, $l = \{1, \ldots, p\}$, and to the current and past values of the input signal $x(n)$. The system's characteristics are determined by the choices for the number of coefficients p and q and the coefficients' values $\{a_1, \ldots, a_p\}$ and $\{b_0, b_1, \ldots, b_q\}$.

ASIDE: There is an asymmetry in the coefficients: where is a_0 ? This coefficient would multiply the $y(n)$ term in the difference equation (1.8: The Difference Equation). We have essentially divided the equation by it, which does not change the input-output relationship. We have thus created the convention that a_0 is always one.

As opposed to differential equations, which only provide an **implicit** description of a system (we must somehow solve the differential equation), difference equations provide an **explicit** way of computing the output for any input. We simply express the difference equation by a program that calculates each output from the previous output values, and the current and previous inputs.

1.3 Discrete Time Convolution[4]

1.3.1 Introduction

Convolution, one of the most important concepts in electrical engineering, can be used to determine the output a system produces for a given input signal. It can be shown that a linear time invariant system is completely characterized by its impulse response. The sifting property of the discrete time impulse function tells us that the input signal to a system can be represented as a sum of scaled and shifted unit impulses. Thus, by linearity, it would seem reasonable to compute of the output signal as the sum of scaled and shifted unit impulse responses. That is exactly what the operation of convolution accomplishes. Hence, convolution can be used to determine a linear time invariant system's output from knowledge of the input and the impulse response.

1.3.2 Convolution and Circular Convolution

1.3.2.1 Convolution

1.3.2.1.1 Operation Definition

Discrete time convolution is an operation on two discrete time signals defined by the integral

$$(f * g)(n) = \sum_{k=-\infty}^{\infty} f(k)\, g(n-k) \tag{1.9}$$

for all signals f, g defined on $\mathbb{Z}$. It is important to note that the operation of convolution is commutative, meaning that

$$f * g = g * f \tag{1.10}$$

[4]This content is available online at <http://cnx.org/content/m10087/2.21/>.

for all signals f, g defined on $\mathbb{Z}$. Thus, the convolution operation could have been just as easily stated using the equivalent definition

$$(f * g)(n) = \sum_{k=-\infty}^{\infty} f(n-k)\, g(k) \tag{1.11}$$

for all signals f, g defined on $\mathbb{Z}$. Convolution has several other important properties not listed here but explained and derived in a later module.

1.3.2.1.2 Definition Motivation

The above operation definition has been chosen to be particularly useful in the study of linear time invariant systems. In order to see this, consider a linear time invariant system H with unit impulse response h. Given a system input signal x we would like to compute the system output signal $H(x)$. First, we note that the input can be expressed as the convolution

$$x(n) = \sum_{k=-\infty}^{\infty} x(k)\, \delta(n-k) \tag{1.12}$$

by the sifting property of the unit impulse function. By linearity

$$Hx(n) = \sum_{k=-\infty}^{\infty} x(k)\, H\delta(n-k). \tag{1.13}$$

Since $H\delta(n-k)$ is the shifted unit impulse response $h(n-k)$, this gives the result

$$Hx(n) = \sum_{k=-\infty}^{\infty} x(k)\, h(n-k) = (x * h)(n). \tag{1.14}$$

Hence, convolution has been defined such that the output of a linear time invariant system is given by the convolution of the system input with the system unit impulse response.

1.3.2.1.3 Graphical Intuition

It is often helpful to be able to visualize the computation of a convolution in terms of graphical processes. Consider the convolution of two functions f, g given by

$$(f * g)(n) = \sum_{k=-\infty}^{\infty} f(k)\, g(n-k) = \sum_{k=-\infty}^{\infty} f(n-k)\, g(k). \tag{1.15}$$

The first step in graphically understanding the operation of convolution is to plot each of the functions. Next, one of the functions must be selected, and its plot reflected across the $k = 0$ axis. For each real t, that same function must be shifted left by t. The product of the two resulting plots is then constructed. Finally, the area under the resulting curve is computed.

Example 1.1

Recall that the impulse response for a discrete time echoing feedback system with gain a is

$$h(n) = a^n u(n), \tag{1.16}$$

and consider the response to an input signal that is another exponential

$$x(n) = b^n u(n). \tag{1.17}$$

We know that the output for this input is given by the convolution of the impulse response with the input signal

$$y(n) = x(n) * h(n). \tag{1.18}$$

We would like to compute this operation by beginning in a way that minimizes the algebraic complexity of the expression. However, in this case, each possible coice is equally simple. Thus, we would like to compute

$$y(n) = \sum_{k=-\infty}^{\infty} a^k u(k) b^{n-k} u(n-k). \tag{1.19}$$

The step functions can be used to further simplify this sum. Therefore,

$$y(n) = 0 \tag{1.20}$$

for $n < 0$ and

$$y(n) = \sum_{k=0}^{n} (ab)^k \tag{1.21}$$

for $n \geq 0$. Hence, provided $ab \neq 1$, we have that

$$y(n) = \begin{cases} 0 & n < 0 \\ \frac{1-(ab)^{n+1}}{1-(ab)} & n \geq 0 \end{cases}. \tag{1.22}$$

1.3.2.2 Circular Convolution

Discrete time circular convolution is an operation on two finite length or periodic discrete time signals defined by the integral

$$(f * g)(n) = \sum_{k=0}^{N-1} \hat{f}(k) \hat{g}(n-k) \tag{1.23}$$

for all signals f, g defined on $\mathbb{Z}[0, N-1]$ where $\hat{f}, \hat{g}$ are periodic extensions of f and g. It is important to note that the operation of circular convolution is commutative, meaning that

$$f * g = g * f \tag{1.24}$$

for all signals f, g defined on $\mathbb{Z}[0, N-1]$. Thus, the circular convolution operation could have been just as easily stated using the equivalent definition

$$(f * g)(n) = \sum_{k=0}^{N-1} \hat{f}(n-k) \hat{g}(k) \tag{1.25}$$

for all signals f, g defined on $\mathbb{Z}[0, N-1]$ where $\hat{f}, \hat{g}$ are periodic extensions of f and g. Circular convolution has several other important properties not listed here but explained and derived in a later module.

Alternatively, discrete time circular convolution can be expressed as the sum of two summations given by

$$(f * g)(n) = \sum_{k=0}^{n} f(k)\, g(n-k) + \sum_{k=n+1}^{N-1} f(k)\, g(n-k+N) \tag{1.26}$$

for all signals f, g defined on $\mathbb{Z}[0, N-1]$.

Meaningful examples of computing discrete time circular convolutions in the time domain would involve complicated algebraic manipulations dealing with the wrap around behavior, which would ultimately be more confusing than helpful. Thus, none will be provided in this section. Of course, example computations in the time domain are easy to program and demonstrate. However, disrete time circular convolutions are more easily computed using frequency domain tools as will be shown in the discrete time Fourier series section.

1.3.2.2.1 Definition Motivation

The above operation definition has been chosen to be particularly useful in the study of linear time invariant systems. In order to see this, consider a linear time invariant system H with unit impulse response h. Given a finite or periodic system input signal x we would like to compute the system output signal $H(x)$. First, we note that the input can be expressed as the circular convolution

$$x(n) = \sum_{k=0}^{N-1} \hat{x}(k)\, \hat{\delta}(n-k) \tag{1.27}$$

by the sifting property of the unit impulse function. By linearity,

$$Hx(n) = \sum_{k=0}^{N-1} \hat{x}(k)\, H\hat{\delta}(n-k). \tag{1.28}$$

Since $H\delta(n-k)$ is the shifted unit impulse response $h(n-k)$, this gives the result

$$Hx(n) = \sum_{k=0}^{N-1} \hat{x}(k)\, \hat{h}(n-k) = (x * h)(n). \tag{1.29}$$

Hence, circular convolution has been defined such that the output of a linear time invariant system is given by the convolution of the system input with the system unit impulse response.

1.3.2.2.2 Graphical Intuition

It is often helpful to be able to visualize the computation of a circular convolution in terms of graphical processes. Consider the circular convolution of two finite length functions f, g given by

$$(f * g)(n) = \sum_{k=0}^{N-1} \hat{f}(k)\, \hat{g}(n-k) = \sum_{k=0}^{N-1} \hat{f}(n-k)\, \hat{g}(k). \tag{1.30}$$

The first step in graphically understanding the operation of convolution is to plot each of the periodic extensions of the functions. Next, one of the functions must be selected, and its plot reflected across the $k = 0$ axis. For each $k \in \mathbb{Z}[0, N-1]$, that same function must be shifted left by k. The product of the two resulting plots is then constructed. Finally, the area under the resulting curve on $\mathbb{Z}[0, N-1]$ is computed.

1.3.3 Convolution Summary

Convolution, one of the most important concepts in electrical engineering, can be used to determine the output signal of a linear time invariant system for a given input signal with knowledge of the system's unit impulse response. The operation of discrete time convolution is defined such that it performs this function for infinite length discrete time signals and systems. The operation of discrete time circular convolution is defined such that it performs this function for finite length and periodic discrete time signals. In each case, the output of the system is the convolution or circular convolution of the input signal with the unit impulse response.

1.4 Review of Linear Algebra[5]

Vector spaces are the principal object of study in linear algebra. A vector space is always defined with respect to a field of scalars.

1.4.1 Fields

A field is a set F equipped with two operations, addition and mulitplication, and containing two special members 0 and 1 ($0 \neq 1$), such that for all $\{a, b, c\} \in F$

1. a. $a + b \in F$
   b. $a + b = b + a$
   c. $(a + b) + c = a + (b + c)$
   d. $a + 0 = a$
   e. there exists $-a$ such that $a + (-a) = 0$
2. a. $ab \in F$
   b. $ab = ba$
   c. $(ab)\, c = a\,(bc)$
   d. $a \cdot 1 = a$
   e. there exists a^{-1} such that $aa^{-1} = 1$
3. $a\,(b + c) = ab + ac$

More concisely

1. F is an abelian group under addition
2. F is an abelian group under multiplication
3. multiplication distributes over addition

1.4.1.1 Examples

Q, R, $\mathbb{C}$

1.4.2 Vector Spaces

Let F be a field, and V a set. We say V **is a vector space over** F if there exist two operations, defined for all $a \in F$, $\mathbf{u} \in V$ and $\mathbf{v} \in V$:

- vector addition: $(\mathbf{u}, \mathbf{v}) \rightarrow \mathbf{u} + \mathbf{v} \in V$
- scalar multiplication: $(a, \mathbf{v}) \rightarrow a\mathbf{v} \in V$

and if there exists an element denoted $\mathbf{0} \in V$, such that the following hold for all $a \in F$, $b \in F$, and $\mathbf{u} \in V$, $\mathbf{v} \in V$, and $\mathbf{w} \in V$

[5] This content is available online at <http://cnx.org/content/m11948/1.2/>.

1. a. $\mathbf{u} + (\mathbf{v} + \mathbf{w}) = (\mathbf{u} + \mathbf{v}) + \mathbf{w}$
   b. $\mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}$
   c. $\mathbf{u} + \mathbf{0} = \mathbf{u}$
   d. there exists $-\mathbf{u}$ such that $\mathbf{u} + (-\mathbf{u}) = \mathbf{0}$
2. a. $a(\mathbf{u} + \mathbf{v}) = a\mathbf{u} + a\mathbf{v}$
   b. $(a + b)\mathbf{u} = a\mathbf{u} + b\mathbf{u}$
   c. $(ab)\mathbf{u} = a(b\mathbf{u})$
   d. $1 \cdot \mathbf{u} = \mathbf{u}$

More concisely,

1. V is an abelian group under plus
2. Natural properties of scalar multiplication

1.4.2.1 Examples

- $\mathbb{R}^N$ is a vector space over R
- $\mathbb{C}^N$ is a vector space over $\mathbb{C}$
- $\mathbb{C}^N$ is a vector space over R
- $\mathbb{R}^N$ is **not** a vector space over $\mathbb{C}$

The elements of V are called **vectors**.

1.4.3 Euclidean Space

Throughout this course we will think of a signal as a vector

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} x_1 & x_2 & \dots & x_N \end{pmatrix}^T$$

The samples $\{x_i\}$ could be samples from a finite duration, continuous time signal, for example.

A signal will belong to one of two vector spaces:

1.4.3.1 Real Euclidean space

$\mathbf{x} \in \mathbb{R}^N$ (over R)

1.4.3.2 Complex Euclidean space

$\mathbf{x} \in \mathbb{C}^N$ (over $\mathbb{C}$)

1.4.4 Subspaces

Let V be a vector space over F.

A subset $S \subseteq V$ is called a **subspace** of V if S is a vector space over F in its own right.

Example 1.2
$V = \mathbb{R}^2$, $F = \mathbb{R}$, $S =$ any line though the origin.