The Sonar Simulation Toolset, Release 4.6: Science, Mathematics, and Algorithms

by Robert P. Goddard

Technical Report

APL-UW TR 0702

October 2008

Applied Physics Laboratory University of Washington
1013 NE 40th Street Seattle, Washington 98105-6698

ONR Contracts N00014-07-G-0557, N00014-01-G-0460,
and N00014-98-G-0001

Acknowledgments

The Sonar Simulation Toolset was developed with sponsorship from several U.S. Navy sources, most recently ONR Codes 333 (Donald McDowell) and 321US (John Tague and Michael Vaccaro). Earlier sponsors included the Naval Surface Warfare Center, Carderock (William Beatty) and the Naval Underwater Systems Center, New London, CT (Walter Hauck and Thomas Wheeler).

The SST development team at APL-UW consists of the author, Bill Kooiman, Keith Wiley, John Pyle, Mike Boyd, and Julia Hsieh. Jim Luby, Warren Fox, and Chris Eggen have provided guidance, testing, and leadership. Earlier team members included Beth Kirby, Kou-Ying Moravan, Pete Brodsky, Brandon Smith, Don Percival, Patrick Tewson, Megan Hazen, Gordon Bisset, and undergraduates Pat Lasswell and Jason Smith. Kou-Ying Moravan and Pierre Mourad supplied the original Fortran implementations from which several of the boundary models started. Mike Boyd, Warren Fox, Greg Anderson, and Chris Eggen have contributed CASS expertise. The REVGEN project, where SST got its start, was led by Dave Princehouse.

The Comprehensive Acoustic System Simulation (CASS) program is available through the support of the Naval Undersea Warfare Center, Newport (NUWCDIVNPT). Permission to use and distribute CASS and GSM is granted by NUWC (Emily McCarthy).

SST users are the ingredients that make SST a useful tool instead of an academic exercise. Special thanks are due to the many SST users at ARL/PSU and NUWC (Newport, RI), with whom we have enjoyed a long and fruitful collaborative relationship.

Typeset February 5, 2008

Abstract

The Sonar Simulation Toolset (SST) is a computer program that produces simulated sonar signals, enabling users to build an artificial ocean that sounds like a real ocean. Such signals are useful for designing new sonar systems, testing existing sonars, predicting performance, developing tactics, training operators and officers, planning experiments, and interpreting measurements. SST's simulated signals include reverberation, target echoes, discrete sound sources, and background noise with specified spectra. Externally generated or measured signals can be added to the output signal or used as transmissions. Eigenrays from the Generic Sonar Model (GSM) or the Comprehensive Acoustic System Simulation (CASS) can be used, making all of GSM's propagation models and CASS's Gaussian Ray Bundle (GRAB) propagation model available to the SST user. A command language controls a large collection of component models describing the ocean, sonars, noise sources, targets, and signals. The software runs on several UNIX computers, Windows, and Macintosh OS X. SST's primary documentation is the SST Web (a large HTML "web site" distributed with the SST software), supported by a collection of documented examples.

This report emphasizes the science, mathematics, and algorithms underlying SST. It is intended to be updated often and distributed with each release of SST as an integral part of the SST documentation.

Contents

Acknowledgments ii

Abstract iii

Contents iv

List of Figures viii

List of Tables ix

1 Introduction 1
- 1.1 Purpose . . . 1
- 1.2 Objectives and Attributes . . . 2
- 1.3 History . . . 3
- 1.4 Release . . . 5
- 1.5 Outline . . . 5
- 1.6 Notation . . . 5

2 Overview 6
- 2.1 Physical Assumptions . . . 6
- 2.2 Outputs . . . 7
- 2.3 Inputs . . . 8
- 2.4 Models . . . 9
- 2.5 Units and Coordinate Systems . . . 10
- 2.6 Computers . . . 11

3 Example 11

4 Signals and Signal Transformations 13
- 4.1 Signal Representations . . . 13
  - 4.1.1 Real Samples . . . 13
  - 4.1.2 Complex Envelope . . . 14
  - 4.1.3 Windowed Frequency Domain . . . 15
- 4.2 Second Moment Time Series: Power Spectra and Scattering Functions 17
  - 4.2.1 Power Spectra . . . 17
  - 4.2.2 Scattering Functions and Doppler Densities . . . 18

4.3 Data Flow Design . . . 19
4.4 Data Flow Classes . . . 22
4.5 Basic Signal Operations . . . 26
4.5.1 Variable Delays . . . 26
4.5.2 Variable Finite Impulse Response Filters . . . 28
4.6 Generating Signals . . . 29
4.6.1 Generating Gaussian Noise . . . 30
4.6.2 Generating K-Distributed Noise . . . 31
4.6.3 Generating Harmonic Tone Families . . . 32
4.6.4 Generating Modulated Tones . . . 33
4.7 Window Functions . . . 34
4.8 Grids . . . 35

5 The Eigenray Model **35**
5.1 Eigenray Properties . . . 36
5.2 Eigenray as Local Expansion . . . 36
5.3 Eigenray as Sound Transformation . . . 38
5.4 Straight-line Eigenray Model . . . 38
5.5 CASS/GRAB Eigenrays . . . 40
5.6 Generic Sonar Model (GSM) Eigenrays . . . 41
5.7 Eigenray Interpolation and Ray Identity . . . 42

6 Ocean Model **43**
6.1 Ocean Depth . . . 43
6.2 Ocean Sound Speed . . . 44
6.3 Ocean Volume Attenuation . . . 44
6.4 Surface and Bottom Models . . . 45
6.4.1 Reflection Coefficients . . . 45
6.4.2 Bistatic Scattering Strength . . . 46
6.4.3 Boundary Classes . . . 47
6.5 Volume Scattering Strength . . . 48

7 Sonar and Source Models **48**
7.1 Trajectories and Coordinate Transformations . . . 49
7.2 Beam Patterns . . . 50
7.3 Sonar Transformation . . . 52
7.4 Source Transformation . . . 53

8 Direct Sound Propagation Models **53**
8.1 DirectSignal 54
8.2 DirectSpectrum 55
8.3 Simple Direct Model 56

9 Target Echo Model **58**
9.1 TargetEcho 58
9.2 Target Models 59
9.2.1 PointTarget 59
9.2.2 HighlightTarget 60
9.2.3 ExternalTarget 61
9.3 Simple Target Echo Model 61

10 Reverberation Theory **63**
10.1 Assumptions 63
10.2 Local Impulse Response 64
10.3 The Poisson Scattering Assumption 65
10.4 Gaussian Statistics 66
10.5 Scattering Field Generation 67
10.6 Spreading and Coherence 68
10.6.1 Time Spread vs. Frequency Coherence 68
10.6.2 Doppler Spread vs. Time Coherence 70
10.6.3 Direction Spread vs. Spatial Coherence 71
10.6.4 Spread vs. Coherence Summary 73
10.7 Separability: Critique and Justification 73
10.8 Complex Envelope Representation 75

11 Reverberation Generation Algorithms **76**
11.1 Point Scatterer Approaches 76
11.2 Directional Doppler Density Method for Reverberation 79
11.2.1 Assumptions 79
11.2.2 Objectives 79
11.2.3 The Doppler Density: Factoring Out the Source Signal 80
11.2.4 Directional Doppler Density: Factoring Out the Receiver 83
11.2.5 Point Scatterer Directional Doppler Density 84
11.2.6 Directional Scattering Function 85
11.2.7 Computing the Directional Scattering Function 87
11.2.8 Summary of the Directional Doppler Method 88
11.3 The Scattering Function Approach 89
11.3.1 Generating Reverberation 89
11.3.2 Computing the Scattering Function 90

12 Time Spreading **93**
12.1 Overview 93
12.2 Example Without Time Spreading 95
12.3 Critique: Why We Need Time Spread 97
12.4 The Time Spreading Algorithm 99
12.5 Example With Time Spreading 101
12.6 Time Spreading Critique 101

13 Summary and Plans **104**

REFERENCES **107**

List of Figures

1 SST Component Models 8
2 Total Signal for Pursuit Example 12
3 ASIAEX Simulation Without Time Spreading 96
4 ASIAEX Simulation With Time Spreading 102
5 ASIAEX Comparison With and Without Time Spreading 103

List of Tables

1 Signal Classes 23
2 Spectrum Classes 24
3 Scattering Function Classes 24
4 Doppler Density Classes 25
5 Window Function Classes 34
6 Surface and Bottom Models 47
7 Beam Pattern Models 52

1 Introduction

1.1 Purpose

The Sonar Simulation Toolset (SST) is a computer program that produces simulated sonar signals as “heard” by a user-specified active or passive sonar in a user-specified ocean environment. It enables a user to create an “artificial ocean” that may be used to test new or proposed sonar systems or tactics, to train sonar operators, to plan experiments, or to validate models of underwater acoustic phenomena by comparing simulation results with measurements.

This report focuses on the science, mathematics, and algorithms used in SST. It is intended, in part, as a tutorial to introduce scientists and engineers to the technical issues that must be addressed by any sonar signal simulation system. It is also intended as a supplement to the other SST documentation, adding information on “how it works” and “why” to the existing descriptions of “how to do it”. And it is intended to help scientists, engineers, trainers, and technical managers decide whether SST might be useful in their projects.

This report is updated often as the software changes. In the version distributed with SST Release 4.6, Sec. 10 was re-written to add more reverberation theory, and Sec. 11 was extended to document the new Directional Scattering Function method for generating reverberation.

A separate on-line document, the SST Web [SST Web], gives details about how to use SST. It includes working examples of SST simulations for several different kinds of active and passive sonar systems. Yet another on-line document, which is generated from the source code and comments therein using the Doxygen [Doxygen] software, covers the internal design of SST.

The SST software, together with all of the documentation just mentioned (including this report), are delivered to DoD agencies and U.S. DoD contractors via a secure web site.

We will assume that the reader is familiar with underwater sound at the level of Urick [Urick 1983], and with digital sonar processing at the level of Knight *et al.* [Knight 1981]. It will be helpful, but not required, for the reader to be familiar with discrete-time signal processing [Oppenheim Schafer 1989] and the fundamentals of object-oriented software design [Page-Jones 2000].

1.2 Objectives and Attributes

SST is certainly not the only sonar simulation system available to the Navy community. It may or may not be appropriate for a given application, depending on the purpose of the simulation. General modeling tools like CASS [Weinberg et al. 2001] or SWAT [Sammelmann 2002], real-time simulators like WAF [Correia 1988,Katyl 2000], tactical simulators like TRM [Cerenzia et al. 2005], or application-specific tools may have a place in a simulation tool kit. To understand where SST fits, consider the following objectives and attributes of SST:

Signal Simulation: SST produces sound, suitable as input for human ears, for the front end of a sonar system, or for a computer model of an existing or proposed sonar front end. It does not produce plots or predict performance by itself.

Portable: SST is designed to run on nearly any modern general-purpose computer. Each distribution includes pre-built versions for SPARC/Solaris, Intel/Linux, Intel/Windows/Cygwin [Cygwin], and PowerPC/Macintosh OS X systems, plus source code and porting tools to make it as easy as possible to port it to other systems.

General: SST is suitable for simulating a wide variety of active or passive sonar systems, sound sources, and targets in many different environments and scenarios. Multiple sound sources, multiple targets, complicated sonar systems and signals, arbitrary trajectories, variable bathymetry, various surface and bottom models, and many other details can be specified using SST's flexible command language.

Broadband: SST is suitable for signals of any bandwidth. The frequency range is limited primarily by its ray-based propagation models, which can be useful as low as a few kHz in shallow water or even lower in deep water, depending on the requirement for fidelity.

Multistatic: There can be any number of sound sources, receivers, and targets on any number of platforms.

Multi-channel: The sonar can have any number of channels, each of which is characterized by its own sensitivity pattern and offset. A channel may represent the signal behind one transducer (element-level) or for one beam behind the beam former (beam-level). Correlations between channels are controlled carefully.

Non-Real-Time: SST is not constrained to run in real time.

Flexible Fidelity: SST's lack of real-time constraint allows the user to trade off speed for fidelity and detail, striking whatever balance is consistent with the user's requirements, budget, and patience. The idea is to support whatever level of fidelity and detail is required for each application, without unnecessarily slowing the simpler simulations. The level of realism can, if necessary, be quite high, and simple simulations can achieve much faster than real-time throughput.

Embeddable: SST may be used as a signal generation component within a higher-level simulation. It is being used that way within TRM at ARL-PSU.

Streamable: SST's results are produced in time order. The early parts of the output are written as soon as possible, usually well before the simulation is finished. This feature is essential for embedded applications and for parallel processing (currently in development), and it helps minimize memory requirements for long runs. Another advantage is that users can examine partial results before deciding whether to continue a long simulation run.

Object Oriented: SST's command language, its primary implementation language (C++), and SST's design are organized around the concepts of *objects*, *classes*, *encapsulation*, *inheritance*, and *polymorphism*. The advantages of this approach are documented in standard software engineering texts [Page-Jones 2000].

Unclassified: SST's distribution is limited to the Department of Defense (DoD) and U. S. DoD contractors only (critical technology); exceptions must be approved by the Office of Naval Research, Code 33. However, the code and documentation are not classified.

1.3 History

Most of the support for SST's development has come through projects supporting specific applications. Hence SST's current capabilities reflect, to a dominant degree, the union of the requirements specified for those applications. In response to changing requirements, innovations were introduced in roughly the following order:

- SST's roots trace back to the REVGEN (Reverberation Generator) [Princehouse 1975,Princehouse 1978,Goddard 1986] project of the 1970s and early 1980s. SST itself started in 1989. The focus was on high-frequency, narrow-band, monostatic active sonar systems. The initial few versions generated reverberation and target echoes using a narrow-band point scatterer model (Sec. 11.1).

- In the early 1990s the emphasis shifted to broadband sonars. This led to updated high-frequency environmental models [APL Models 1994], a scattering-function reverberation model [Luby Lytle 1987], frequency-dependent environmental models and beam patterns, and the "data flow" architecture (Sec. 4.3).

- Another thrust during the middle 1990s was toward multistatic operation. SST's monostatic reverberation models were replaced by a fully bistatic, broadband one based on scattering functions (Sec. 11.3). Bistatic surface and bottom models were also introduced (Sec. 6).

- During this period, both the number of SST users and the size and complexity of the code rose dramatically. In response, we replaced the earlier text documentation with an extensively cross-linked HTML Web version [SST Web], and C++ replaced C as the primary programming language.

- Starting in 1997 several applications involved frequencies below 10 kHz. This led to a new mid-frequency bistatic surface model [Gilbert 1993], an upgraded bottom model, an external target model (Sec. 9.2.3), and better support for passive sonars.

- The Navy's concern shifted from deep water to littoral environments. In response, we added support in SST for range-dependent propagation using eigenrays computed using the GRAB [Weinberg Keenan 1996] eigenray model.

- In broadband, shallow-water applications, SST users recognized that the processing gain for simulated target echoes versus reverberation and countermeasures was too optimistic. The main culprit was identified as near-specular scattering from the surface and bottom. In response, we added time and frequency spreading (reduced coherence in frequency and time) (Sec. 12).

- Several users wanted to use SST as a signal generation component in a higher-level tactical simulation. Others wanted to use some of SST's models in other environments. These requirements have led to a continuing push toward interoperability.

- Current efforts, driven by user concerns, include improved coherence control, faster element-level simulations for systems with a very large number of hydrophones, user interface support for combined active and passive processing, ship wakes, very long active transmissions, and better realism at lower frequencies.

- Everyone always wants more speed, better documentation, fewer software errors, a simpler and more intuitive user interface, more checks to prevent user mistakes, interoperability with other systems, and more and better support for users. We are constantly working to improve SST and our services in all of these measures.

SST has played a significant role in many studies over the years. Unclassified published ones include Eggen and Goddard [Eggen Goddard 2002], Goddard [Goddard 2000], and Rouseff [Rouseff et al. 2001].

1.4 Release

The version of SST described here is Release 4.6, dated October 2007.

1.5 Outline

Section 2 is a general overview of SST and its component models. Section 3 presents a simple example showing the results of an SST simulation. The bulk of the paper describes the science and mathematics underlying SST. The order in which SST's various sub-models are described is intended to support linear reading by placing the background needed to understand a model's requirements before that model's description. We conclude, in Section 13, with a brief description of current and planned projects to improve SST's realism, scope, and ease of use.

1.6 Notation

In nearly all of this paper, we treat the signals, transformations, and models as *continuous* in time, space, frequency, and direction. This is a physicist's point of view, not a software engineer's. Of course, as in any digital implementation, conceptually continuous functions are sampled at discrete values of their independent variables, and integrations are implemented as finite sums, and precision is limited. We discuss digital samples when it is necessary to do so. In our view, however, the physical and mathematical concepts are easier to understand in the continuous domain, so we remain there whenever possible. The mapping between continuous and digital domains is covered well by standard texts [Oppenheim Schafer 1989,Hamming 1973].

SST is object oriented on two levels, the SST command language and the implementation language. For the most part, an SST input file consists of statements that define objects that are instances of built-in SST classes, and assign values to named attributes (parameters) of those objects. Each of the classes visible through the command language is implemented in terms of a C++ class having the same name, with attributes corresponding to C++ member variables. We do not distinguish here between the two kinds of classes.

Within paragraph text, class names are displayed in bold *sans serif* font and capitalized; examples are **Signal**, **PistonBeam**, and **JacksonBottom**. Names of class attributes are displayed in slanted *sans serif* font and uncapitalized; examples are *frequency*, *radius*, and *soundSpeedRatio*.

This document is designed to be easily readable and navigable in printed form. It also contains several features to enhance on-line browsing using a PDF viewer such as the free Adobe Reader [Adobe Reader]. These features include active (click-through) references to equations, figures, related text sections, and citations.

2 Overview

2.1 Physical Assumptions

The simulated sound produced by SST consists of a digital representation of the predicted signal in each channel of the sonar receiver's processing path. This sound may contain components from four types of sources:

Discrete sound sources: "Passive targets," such as ships or countermeasures, that radiate noise from a compact region

Diffuse sound sources: Environmental noise and self noise

Discrete scatterers: "Active targets," such as submarines or rocks, that echo an active sonar's transmitted pulses (or other sound) back to the sonar receiver

Reverberation: Diffuse scatterers, such as the ocean bottom, that send back many overlapping echoes of an active sonar's pulse

The sonar system can be passive (receiver only), or it can be active (listening for echoes of its own transmissions). Active sonars can be monostatic (transmitter and receiver on the same platform), bistatic (transmitter and receiver on two different platforms), or multistatic (employing several transmitters or receivers).

The receiver can have any number of channels (transducers or beams). There can be any number of sound sources and targets, and any number of sound paths (eigenrays) connecting sources, scatterers, and receivers. For both passive and active sonars, the signals can have an arbitrarily wide range of frequencies. The ocean, sound scatterers, and beam patterns act as filters that alter the frequency content of the sound. Transmissions and listening intervals can be arbitrarily long. Of course, all of these parameters will have an impact on size and speed, so the practical limits depend on available disk space, memory, processor speed, and the patience of the user.

Over the last decade, SST's most active users (and most of its financial support) have come from the torpedo development community. Therefore, its high-frequency environmental models are more complete and up-to-date than its lower-frequency models. The current propagation models are based on eigenrays, not on direct solutions of the wave equation. These factors limit the realism of SST's simulations for low frequencies, especially in shallow water.

2.2 Outputs

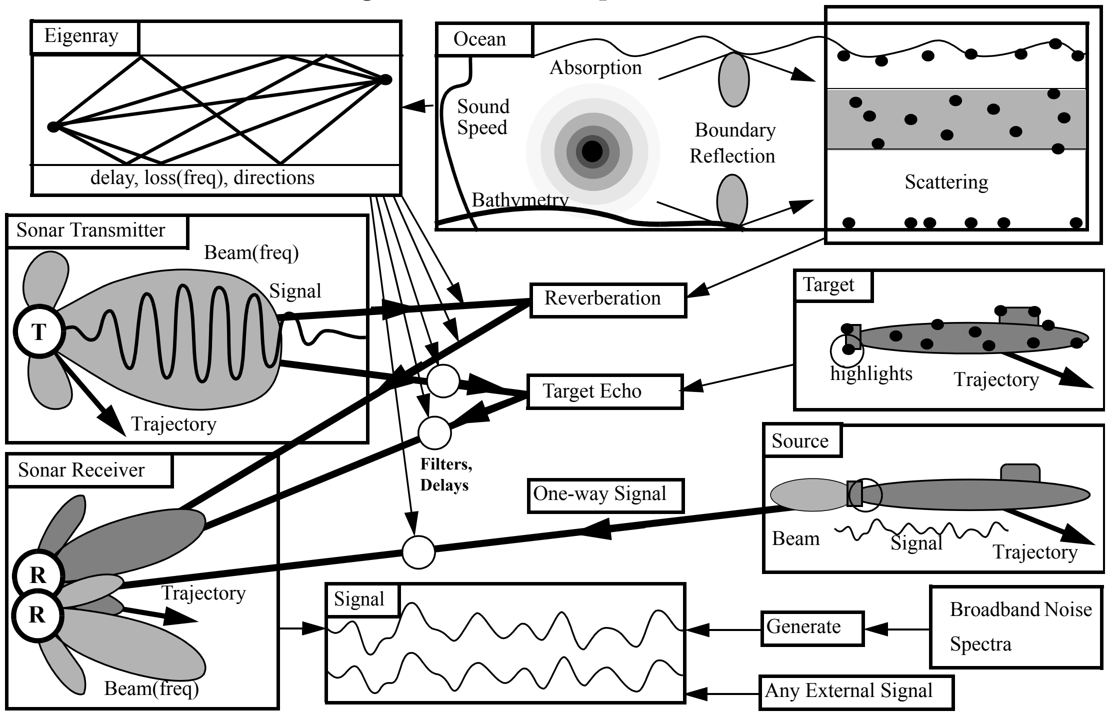
SST's primary output is multi-channel sampled sound: a digital representation of the simulated signal somewhere in the sonar receiver. A channel can represent either the signal behind a transducer (for element-level simulations) or an output of the beamformer (for beam-level simulations).

SST can also produce several other types of data, including the reverberation scattering function, various types of spectra and cross-spectra, and synthesized signals intended for use as transmissions or noise sources. Some of these are intermediate products used in the simulation, and some result from general-purpose signal processing tools (like a spectrum analyzer) that the user can apply to any signal.

These simulated signals and related data can be written into an external file in any of several user-specified forms, binary or text. Any of these forms can be read by SST as well as written. The ways that SST can represent or store signals are described in Sec. 4.

SST does not produce plots or other displays. Post-run analysis and plotting can be done using commercial packages like Mathematica [Mathematica] or Matlab [Matlab], or public-domain tools like Octave [Octave] or Gnuplot [Gnuplot], specialized tools like the SIO package from Scripps [Hodgkiss 1989], or many other tools. A large and growing set of useful Matlab scripts are provided with SST.

Figure 1: SST Component Models



2.3 Inputs

The primary inputs to an SST simulation consist of commands for generating the various types of signals, plus assignment statements in which the user specifies the characteristics of the ocean environment, the sonar transmitter and receiver, active and passive targets, and the format of the simulated signal. These specifications and commands are expressed in a simple but flexible language with an object-oriented flavor reminiscent of Python or C++. They may be entered from the keyboard, read from text files, or passed via a pipe from a higher-level program. The language supports user-defined variables and comments to help make the scripts readable.

Signals used as transmissions or noise sources can come from external files having the same file formats as SST's outputs. SST also includes tools for specifying and generating these signals internally (Sec. 4.6).

Eigenrays and beam patterns can come from external files, text or binary.

2.4 Models

SST is based on a large number of underlying *models* — mathematical representations of physical phenomena — which specify how each element of the environment affects the sound. Figure 1 summarizes those component models and the relationships among them.

Eigenray Model: SST's sound propagation models are based on *eigenrays* — paths through the water along which sound propagates between two specified end-points. Sound in the neighborhood of a receiver or scatterer is a sum of copies of the original transmitted sound, each of which arrives at a different time from a different direction, and each of which has been attenuated in a frequency-dependent way. SST provides three choices (plus variants) for eigenray models: a straight-line model with reflections (the base class), one based on eigenrays from the GRAB eigenray model in the CASS software, and one based on any of several eigenray models provided by the GSM software. The eigenray models are discussed in Sec. 5.

Ocean Model: The SST user specifies the ocean environment by describing characteristics of the surface and bottom, the depth, the volume scattering strength, and the sound speed and absorption rate of the water itself. These models act as inputs to the eigenray model and the reverberation model. The various components of the ocean model are discussed in Sec. 6.

Sonar and Source Models: The sonar receiver is specified by giving its trajectory through the water, the location of each channel's phase center, and the beam pattern giving each channel's sensitivity versus direction and frequency. Each discrete sound source is described in terms of its trajectory, the signal it transmits, and the directional behavior of the transmission (beam patterns). An active sonar's transmitter is treated just like any other sound source; hence the "Sonar Transmitter" model in Fig. 1 is the same as the "Source" model. These models are discussed in Sec. 7.

Target Model: Each target (for active sonars) is an object (with a trajectory) that receives a signal from a source and re-transmits it to a sonar; hence it acts like a group of receivers and sources back to back. Several target models are available; each one provides a different way to specify the relationship between the received sound and the re-transmitted sound. The current target models describe that relationship in terms of *highlights*. They are described in Sec. 9.2.

Direct (One-way) Sound Propagation: The heavy arrows in Fig. 1 represent the various ways that a source signal is transformed on its way to being received by a sonar system. The simplest one, marked "One-way Signal" on the diagram, is also called "direct" sound propagation. This transformation combines properties of the source model, the eigenray model, and the sonar model, and reduces them to a set of filters and delays that transform the signal emitted by a source into the signal received by a sonar via paths that may involve reflection and refraction but not scattering. This transformation is described in Sec. 8.

Target Echoes: The heavy arrows in Fig. 1 marked "Target Echo" represent sound that scatters from a discrete target. It is essentially two "one-way signals" in series, with the target model in between. This transformation is described in Sec. 9.

Reverberation: The heavy arrows in Fig. 1 marked "Reverberation" represent sound that scatters from a very large number of very small scatterers distributed throughout the environment. This is conceptually very similar to a lot of target echoes, but SST treats them statistically, so the implementation of the transformation is quite different from target echoes. Reverberation is described in Secs. 10 and 11.

Noise and Other Sound: Sound from any source can be added to SST's one-way signals, target echoes, and reverberation. SST can produce broadband Gaussian noise having a user-specified power spectrum, as described in Sec. 4.6.1; this type of noise is useful to represent ambient noise, self-noise, and electronic noise. Broadband Gaussian noise can also be used as the emission from any sound source, for which it may be combined with harmonic families of tones or active transmit pulses. Signals for any of these purposes may be generated internally by SST, or read in from an externally generated file.

Signal Processing: All of the signal transformations are built on top of a diverse set of general-purpose signal processing tools for generating, summing, filtering, delaying, scaling, Fourier analyzing, and otherwise transforming signals. All of these signal processing tools can be hooked together like plumbing, or they can be used separately. The "data flow" architecture that makes this flexibility possible is described in Sec. 4.3, and the components themselves are described in Sec. 4.4.

2.5 Units and Coordinate Systems

In SST and in this paper, acoustic pressure is expressed in micro-Pascals (μPa), acoustic intensity is in μPa^2 or in dB//μPa^2, and angles are in degrees. All other measurements are expressed in MKS units.

In the *Earth-centered* coordinate system, vector components are in the order (North, East, Down) with an origin at an arbitrary point on the surface of the ocean. Earth curvature is ignored, except insofar as it is included in the CASS and GSM eigenray models (Sec. 5). In *platform-centered* coordinate systems (used for beam patterns, element offsets, and target highlights), vector components are in the order (Forward, Starboard, Below) relative to an arbitrary origin on the platform (sonar, source, or target). The mapping between the two types of coordinate systems is defined by the *trajectory* attribute of the vehicle, as described in Sec. 7.1.

2.6 Computers

This release of SST has been tested on Intel/Linux, SPARC/Solaris, PowerPC/Macintosh OS X, and Intel/Windows under the Cygwin [Cygwin] environment. A version for the new Intel/Macintosh OS X platform will be available soon. An Intel/Windows version that does not require Cygwin is planned. SST is designed to be portable, and previous versions of SST have been ported to several other UNIX-like systems.

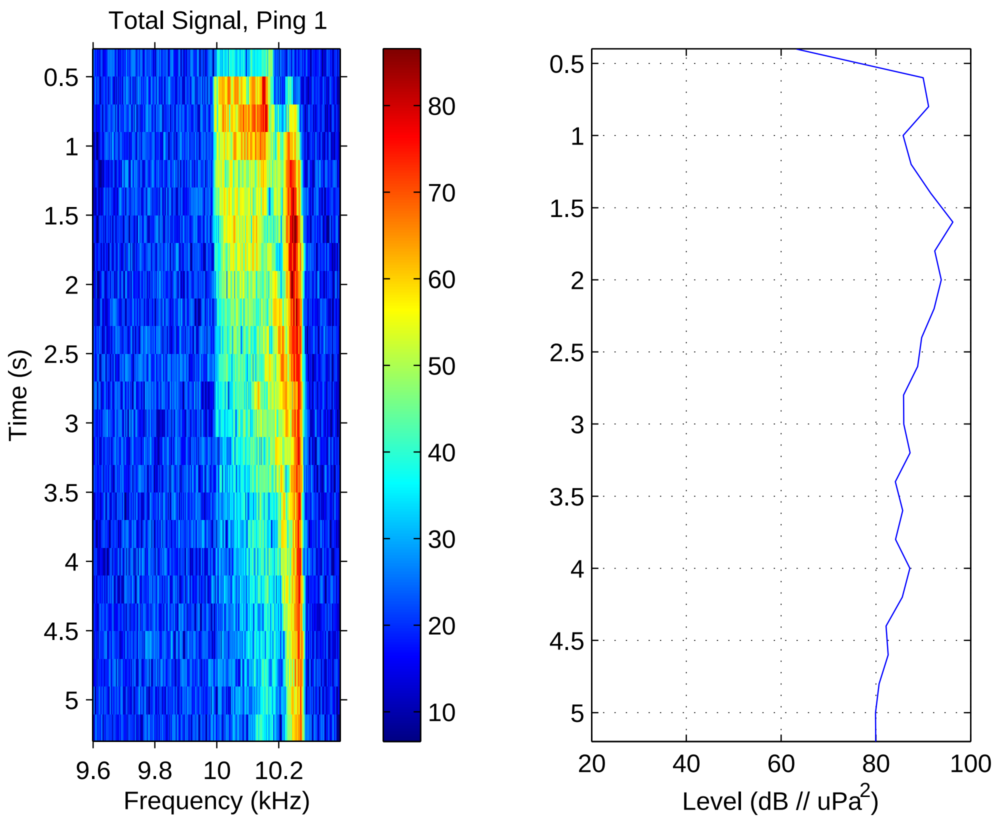
3 Example

Figure 2 is a Matlab display showing the result of SST's "Pursuit" example, which is one of the standard examples provided with SST. The Pursuit scenario features a high-speed vehicle carrying a monostatic, forward-looking active sonar chasing after a simple three-highlight target. SST produced the multi-channel signal heard by the sonar, which includes the target echo, reverberation, and noise produced by the fleeing target. The Matlab scripts that produced the figure sliced that signal into short segments, computed the power spectral density for each segment, and displayed the results, both as a time-dependent spectrum and as total signal level.

The dominant feature in the figure is the reverberation, which comes primarily from the surface in this scenario. The early part of the signal comes from almost directly overhead, via the sidelobes of the beam pattern; hence it has very little Doppler shift. The later part of the reverberation comes from the main lobe, almost straight ahead; hence it has a high upward Doppler shift (the main ridge in the upper part of the spectrum). This obvious "reverberation hook" from low early Doppler to high late Doppler is characteristic of forward-looking sonars. Less obvious is the streak that starts at about 1.7 seconds and merges with the main ridge; that feature results from a path that reflects from the bottom.

The target echo is the blip at about 2.8 seconds and 10.13 kHz. It has a lower Doppler shift than the main reverberation ridge because the target is running away. This is an "easy" high-Doppler detection despite the fact that the level of the target return is well below the level of the reverberation (the target doesn't show at all on the level plot on the right).

Figure 2: Total Signal for Pursuit Example



Note that this is a very simple analysis of a very simple scenario. Only one channel is shown (although the receiver in the example has five channels); complications like inter-channel correlations are in the SST-generated signal but not shown by this analysis. The pulse is a narrow-band pure tone, so the Fourier analysis used here is nearly optimal; a broadband pulse would require more complex analysis, probably featuring replica correlation. The sonar is monostatic, and only a few eigenrays are significant. These limitations are not imposed by SST, which can support much more complicated scenarios requiring much more complicated processing, both by SST and by the post-processing algorithms needed to analyze the results.

The SST script for this example can run in a few seconds on a modern workstation — faster than real-time throughput (but not real-time in terms of response deadlines). More complicated scenarios would take more time.

4 Signals and Signal Transformations

The inputs, outputs, and much of the intermediate data of SST consist of *signals*. In this section we address the question, “What is a signal?” from several points of view, including abstract meaning, sampled representations, and software components. These issues are addressed early because they form the scaffolding on which much of the SST system is built.

4.1 Signal Representations

Conceptually, a signal $x_c(t)$ is a continuous, band-limited, multi-channel function of time t, with channels indexed by c. A signal may represent (for example) voltage on a set of wires or sound pressure at a set of locations in the water. Within SST each signal is represented as a sequence of sets of sample values $x_c(t_n)$ corresponding to discrete values t_n of the time. SST supports three distinct, equivalent representations of a signal: real samples, complex envelope, and windowed frequency domain. All signals in SST, whether they are used for input, output, or in between, are represented in one of these three ways.

4.1.1 Real Samples

The most straightforward way to represent a real signal $x_c(t)$ in SST is as a sequence of real samples on a uniform grid of time values:

$$x_c(t_n) = x_c(t_0 + nh), \tag{1}$$

where n is an integer and h is the time increment between samples.

If the Fourier transform of the original continuous signal $x_c(t)$ is negligible for frequencies greater than some maximum frequency $F_{\max}$, and the sample interval h is chosen such that $F_{\max}$ is within the Nyquist band:

$$h < 1/(2F_{\max}), \tag{2}$$

then the original continuous signal can be recovered (in principle) using band-limited interpolation:

$$x_c(t) = \sum_{n=-\infty}^{\infty} \left(\frac{\sin(\pi(t - t_n)/h)}{\pi(t - t_n)/h} \right) x_c(t_n). \tag{3}$$

The problematic infinities and long decay time of this equation will be dealt with in Section 4.5.1, when we describe the delay algorithm.

SST Class: The base class from which all time-domain signals are derived is **Signal**, which has attributes *frequency*, *isComplex*, and *times*. To specify the real sample representation in any class derived from **Signal**, set *isComplex* to *false* and *times* to a **UniformGrid** specifying the sample times. More details about this family of classes will be given in the subsections 4.3 and 4.4 below.

4.1.2 Complex Envelope

For a *band-limited* signal, a bandwidth W and a center frequency F exist such that the Fourier transform of the signal is negligibly small outside the frequency range $F_{\min} = F - W/2$ to $F_{\max} = F + W/2$ (considering only positive frequencies). If W is sufficiently small compared to F, it is often advantageous to express such signals in *complex envelope* notation [Knight 1981]. The real-valued signal $x_c(t)$ is expressed in terms of a complex envelope $\tilde{x}_c(t)$:

$$x_c(t) = \sqrt{2}\, \Re \left(\tilde{x}_c(t)\, \mathrm{e}^{2\pi \mathrm{i} F t} \right). \tag{4}$$

The inverse transformation is (ideally)

$$\tilde{x}_c(t) = \sqrt{2}\, \left[x_c(t)\, \mathrm{e}^{-2\pi \mathrm{i} F t} \right]_{\mathrm{LP}}, \tag{5}$$

where $\Re()$ denotes the real part of a complex number, and the subscript LP denotes application of an ideal low-pass filter with a passband of $-W/2$ to $W/2$ and unity gain. The complex envelope $\tilde{x}_c(t)$ is advantageous because it varies much more slowly with time than does the real signal $x_c(t)$ (to the extent that $W \ll F$). The complex envelope may be sampled and interpolated using Eq. (3) just like the original real signal, but now the maximum sample interval h required to fully recover the signal is determined by the bandwidth, not the maximum frequency $F_{\max}$:

$$h \leq 1/W, \tag{6}$$

i.e., the Nyquist band extends from $F - 1/(2h)$ to $F + 1/(2h)$. As a rule of thumb, the complex envelope notation is advantageous whenever the ratio of bandwidth to maximum frequency is less than about 25%.

Normalization: The constant $\sqrt{2}$ in Eqs. (4) and (5) is chosen to preserve power: $\mathbf{Av}\,\{|\tilde{x}_c(t)|^2\} = \mathbf{Av}\,\{[x_c(t)]^2\}$, where $\mathbf{Av}\,\{\}$ denotes a time average.

SST Class: To specify the complex envelope representation in any class derived from the base class **Signal**, set *frequency* to the center frequency F, *isComplex* to *true*, and *times* to a **UniformGrid** specifying the sample times.

SST Class: Transformations between real samples and complex envelope samples are implemented in SST class **ResampleSignal**, which uses approximate forms of Eqs. (4) and (5) employing window functions to combine finite-length filters with finite-order interpolation. **ResampleSignal** can also be used to change sample rate for real samples, or to change sample rate or center frequency for complex envelope samples, or to select a subset of a signal in time.

4.1.3 Windowed Frequency Domain

The third signal representation used in SST is a *windowed frequency domain* form. Currently, the primary application for the windowed frequency domain representation is in the implementation of class **DirectSpectrum**, which will be discussed in Sec. 8.

Conceptually, we break the continuous signal into blocks of a convenient size (one per *update cycle*) and perform a Fourier transform on each block. To avoid end effects, we enlarge the blocks so that they overlap, and multiply each one by a smooth "pre-window function" $w(t)$ whose properties are specified below. The transformation starting from a continuous signal is

$$X_c(f, t_u) = \int_{-\infty}^{\infty} w(t - t_u)\, x_c(t)\, \mathrm{e}^{-2\pi \mathrm{i} f (t - t_u)}\, \mathrm{d}t, \tag{7}$$

where the index u labels an update time on a uniform grid with interval Δ:

$$t_u = t_0 + u\Delta. \tag{8}$$

Note that the update interval Δ is normally much larger than the signal sampling frequency h used in Eq. (1).

The inverse operation, from windowed frequency domain to continuous signal, involves an inverse Fourier transform, multiplication by a "post-window function" $w'(t)$, a time shift, and a summation:

$$x_c(t) = \sum_u w'(t - t_u) \int_{-\infty}^{\infty} X_c(f, t_u)\, \mathrm{e}^{2\pi \mathrm{i} f (t - t_u)}\, \mathrm{d}f. \tag{9}$$

Of course, in the computer $X_c(f, t_u)$ is sampled on uniform grids in two dimensions, frequency f and update time t_u. The Fourier transforms of Eqs. (7) and (9) are implemented using fast Fourier transforms (FFTs). For real signals the values of $X_c(f, t_u)$ for negative frequencies are not stored because

$$X_c(f, t_u) = X_c^*(-f, t_u), \tag{10}$$

where the asterisk represents complex conjugation.

The requirement that Eqs. (7) and (9) are inverses of one another imposes the following requirement on the two window functions:

$$\sum_u w'(t - t_u)\, w(t - t_u) = 1 \tag{11}$$

for all times t. Typically, the pre-window $w(t)$ is a Hann (cosine squared) window, and the post-window $w'(t)$ is rectangular. We will return to window functions at the end of this section.

Normalization: The sampled version of $X_c(f, t_u)$ is normalized such that it converges to Eq. (7) as the sampling interval $h \to 0$. Thus, if $x_c(t)$ represents a pressure in μPa, the unit of measurement for $X_c(f, t_u)$ is $\mu\mathrm{Pa} - \mathrm{s}$.

SST Classes: To specify a signal in windowed frequency domain form, use a subclass of class **Spectrum** with attributes *isPower* $=$ *false*, *isComplex* $=$ *true*, and *isCorrelated* $=$ *false*. Attributes *times* and *frequencies* are **Grid** objects (Sec. 4.8) specifying the update times t_u and the frequencies f where the spectrum is sampled.

The transformations defined in Eqs. (7) and (9) are implemented by SST classes **SpectrumFromSignal** and **SignalFromSpectrum**, respectively.

4.2 Second Moment Time Series: Power Spectra and Scattering Functions

Each of the three representations in the previous subsection is "complete" in that it carries enough information to uniquely specify a continuous signal. SST also supports two types of time series, *power spectra* and *scattering functions*, that represent second-order statistical descriptors of a signal. These are "incomplete" in that they do not support unique reconstruction of the original signal.

4.2.1 Power Spectra

The *power spectral density* (PSD) of a stationary random signal is normally defined as the expectation of the Fourier transform of the autocovariance of the signal. For multi-channel signals, this definition is easily generalized to include covariances between channels. For nonstationary signals or those with non-random components (e.g., tones), the definition must be modified further to bring in window functions to ensure that the result is finite and to restrict it to a time interval over which the spectrum can be considered quasi-stationary.

A useful estimator of the PSD for multi-channel, non-stationary signals should be defined and normalized in such a way that it approximates the true PSD in cases where the signal is random and stationary. The expectation value of such an estimator is

$$P_{cc'}(f,t_u) = \frac{\mathbf{E}\left\{X_c(f,t_u)\,X^*_{c'}(f,t_u)\right\}}{\int_{-\infty}^{\infty}|w(\tau)|^2\,\mathrm{d}\tau}, \tag{12}$$

where $X_c(f,t_u)$ is given by Eq. (7), $w(\tau)$ is the window function used in that equation, and $\mathbf{E}\left\{\cdot\right\}$ denotes the statistical expectation value. For signals in μPa, the units of $P_{cc'}(f,t_u)$ are $\mu\text{Pa}^2/\text{Hz}$. The diagonal elements ($c=c'$) are measures of the power per unit frequency in a given channel in the neighborhood of the update time t_u, where the neighborhood is defined by the weighting function $w(\tau)$. The off-diagonal elements ($c \neq c'$) are measures of the cross-correlation between channels versus frequency in the same neighborhood.

For stationary signals, Eq. (12) gives the true PSD convolved with the PSD of the window function $w(\tau)$. Hence the interpretation of Eq. (12) as a "time dependent PSD" is most meaningful if the time series is random and a window function $w(\tau)$ can be found that satisfies both of the following conditions:

- The time series $x_c(t)$ is quasi-stationary over the time interval where the window function $w(\tau)$ is significantly nonzero.

- The power spectrum $P_{cc'}(f, t_u)$ is quasi-stationary in frequency f over a frequency interval whose width is that of the interval in which the Fourier transform of the window function is significantly nonzero.

The first criterion favors short windows (in the time domain), whereas the second one favors long windows. As a practical matter, Eq. (12) is a “good” PSD estimate if the window length lies within a range of window lengths over which the result doesn’t depend strongly on the window length.

SST Classes: SST class **Spectrum** is the base of a hierarchy from which all spectra derive — both complex vector-valued amplitude spectra $X_c(f, t_u)$ and matrix-valued power spectral densities $P_{cc'}(f, t_u)$. Their primary common feature is that they represent functions of both time and frequency. The two types of spectra are distinguished by a Boolean attribute *isPower*. More details about this family of classes will be given in subsections 4.3 and 4.4 below.

SST Classes: Equation (12) (omitting the expectation value operator) is implemented by SST class **SpectrumFromSignal** with attribute *isPower* set to *true*. (With *isPower* set to *false*, that class implements Eq. (7).)

4.2.2 Scattering Functions and Doppler Densities

A *scattering function* $Z_{rr's}(T, \Gamma, f)$ can be thought of as a generalization of the *intensity impulse response function* [Dahl 2001] for reverberation — generalized in that it includes Doppler spread, frequency dependence, and inter-channel correlation as well as time spread. It expresses the relationship between the PSD of a source signal and the PSD of the resulting reverberation signal received by a multi-channel sonar receiver. The details of that relationship are the subject of Sec. 11.3. For now, just consider it a matrix-valued function of two-way travel time T, Doppler shift Γ, and frequency f, for a given source channel s, where the function value is a non-negative definite square matrix (indices rr') whose dimension is the number of receiver channels. The Doppler shift Γ is defined as the ratio of received frequency to transmit frequency, which can differ from unity due to motion of the source, receiver, and scatterers. The scattering function is introduced here because it shares the data-flow design of signals and spectra, as outlined in the following subsection.

A *Doppler density* $Q_{rs}(T, \Gamma, f)$ is a time-dependent, multi-channel, stochastic Doppler spectrum for reverberation. It is a random realization whose second-order statistical moments (suitably normalized) constitute a scattering function:

$$Z_{rr's}(T, \Gamma, f) = \mathbf{E}\left\{Q_{rs}(T, \Gamma, f)\, Q^*_{r's}(T, \Gamma, f)\right\}/h \tag{13}$$

where h is the time step on which the Doppler density is computed. A Doppler density describes how each frequency component of the transmitted signal is randomly spread into multiple received frequencies by the Doppler shift due to sonar motion (and sometimes scatterer motion).

Again, the details are covered in Sec. 11. For now consider it a stochastic (random) vector-valued function of two-way travel time T, Doppler shift Γ, and frequency f, for a given source channel s, where the function value is a vector (index r) whose length is the number of receiver channels.

SST Classes: SST class **ScatFun** is the base of a hierarchy from which all scattering functions derive. All of them represent matrix-valued functions of time, Doppler, and frequency. More details about this family of classes will be given in the next two subsections and in section 11.

SST Classes: SST class **DopplerDensity** is the base of a hierarchy from which all Doppler densities derive. All of them represent matrix-valued functions of Doppler, frequency, and time. More details about this family of classes will be given in the next two subsections and in section 11.

4.3 Data Flow Design

All of the SST classes that produce, modify, and store time series are based on a *data flow* design. They represent continuous functions of time, sampled on a uniform grid of time values. Each data-flow class is based on one of three base classes: **Signal**, which depends only on time, **Spectrum**, which depends on both frequency and time, and **ScatFun** and **DopplerDensity**, each of which depends on Doppler, frequency, and time. Subclasses within each hierarchy differ according to how samples are stored or computed. From the common "data flow" point of view, all such classes produce and/or consume a block of numbers (samples) for each value of time in some uniformly increasing sequence of times.

The samples themselves, however, may or may not be contained within an object of one of these classes. Instead, these objects should be regarded as sources or sinks of samples, which can be made to produce or accept blocks of samples, in time order, in response to *readBlock* or *writeBlock* requests on the object. Creating an object or assigning it to a variable merely establishes a link, and perhaps a path along which the samples will flow.

Most of SST's work is done under control of a **CopySignal** command, which simply reads successive blocks of data from one data-flow object and writes the resulting data into another such object. The object on the “write” side of a **CopySignal** command is typically rather simple: It just stores the data in a file or memory buffer in some specified format. The processing on the “read” side of a **CopySignal** command can be much more complex. Data-flow objects are typically organized into a network, in which each object might contain references to other data-flow objects that supply its inputs. For example, an object of class **VarDelay** contains references to at least two other data-flow objects: one to supply the signal to be delayed, and one to supply the time-varying delay by which the first signal is to be delayed.

The protocol is common to all such objects: In response to a “read” request, each object figures out which data it needs from each of its input data-flow objects, and issues corresponding “read” requests. These requests propagate upstream until they reach a “leaf” object that can satisfy its request, for example by computing its output or reading from a file. The resulting data propagate downstream as the requests are satisfied, as each object uses data from its input data-flow objects (if any) to compute its output. Eventually the block originally requested by the **CopySignal** command is stored by the object on the “write” side of the command, and the cycle repeats for the next block.

The following very simple SST script illustrates the concept:

```
# Variable Delay Example
insig = HarmonicFamily{
    isComplex = false
    times = UniformGrid:{ first=-2; last=5; rate=8000 }
    fundamental = 220  #Hz
    harmonics = (
      # number ampDB phaseDeg
        1       -3      0.0
        2       -6     90.0
    )
}
delay = InternalSignal{
    isComplex = false
    times = UniformGrid:{ first=0; last=5; interval=1 }
    buf = ( 1.00 1.10 1.25 1.45 1.7 2.0 )
}
outsig = SoundSignal{ file = "myDelayedSignal.snd" }
delayGenerator = VarDelay {
    isComplex = false
    times = UniformGrid:{ first=0; last=5; rate=8000 }
    inSignal = insig
    commonDelayBuf = delay
}
CopySignal delayGenerator outsig
```

This fragment creates four objects of various subclasses of **Signal**, and assigns them the user-selected names *insig*, *delay*, *outsig*, and *delayGenerator*. The **VarDelay** requires two input **Signal** objects as attributes: *inSignal* specifies the signal to be delayed, and *commonDelayBuf* specifies a time-dependent delay to be applied to all channels. In this case, the signal to be delayed consists of a **HarmonicFamily** (Sec. 4.6.3) specifying two tones sampled at 8 kHz, and the delay is an **InternalSignal** specifying a steadily increasing delay, sampled once per second.

The main point of this example is that, until the final **CopySignal** statement, the only object that contains samples of a signal is the **InternalSignal** called *delay*. The other objects merely specify how signals are to be produced or stored. The **CopySignal** statement works essentially like this: First **CopySignal** calls *openRead* on the input signal and *openWrite* on the output signal to get things started. Then it executes a loop that reads a block from *delayGenerator*, writes the result to *outsig*, and repeats. At the end of the input signal, it calls the *close* method on both input and output signals to shut them down. We have omitted complications related to starting, stopping, and managing buffers, but the central loop of **CopySignal** reduces to *readBlock*, *writeBlock*, repeat. Each subclass of **Signal** implements these operations in its own way:

- **VarDelay**::*readBlock* reads the necessary samples from each of the inputs (the delay and the signal to be delayed), interpolates the delay and the input signal to implement the delay, and returns the results. The block requested from the input signal is earlier than the output times, depending on the delay.
- **HarmonicFamily**::*readBlock* computes the requested samples and returns them to the caller.
- **InternalSignal**::*readBlock* returns the requested samples from its internal buffer.
- **SoundSignal**::*writeBlock* writes the samples into a file in a form that most computers can play through their speakers.

Requests for data propagate up the chain of *readBlock* operations and the results propagate back down, with a transformation at each step. The result, in this example, is an audio file with tones that get lower as the rate of change of delay gets larger.

The pattern illustrated by this example is typical of SST simulations, which have the following characteristics:

- Data-flow classes belong to any of four hierarchies, based at classes **Signal**, **Spectrum**, **ScatFun**, or **DopplerDensity**. All of them represent continuous functions of time. The hierarchies differ in whether they depend on frequency or Doppler as well as time. Subclasses within each hierarchy differ according to how samples are stored or computed.

- Data-flow objects can represent nearly any time-dependent quantity, including some that are not normally considered "signals". These include the time-varying delay used to control a **VarDelay** object (as shown in the example above) and the time-varying filter coefficients used by **VarFirFilter**.

- Classes that compute samples on demand are *read-only*; they implement *readBlock* and *openRead*, but not *writeBlock* or *openWrite*. Many of the read-only objects accept other data-flow objects as attributes; these supply streams of input samples to be transformed. Chains of transformations formed in this way can be of any length. Read-only objects can be used as the source of a **CopySignal** command or as an input to a transformation.

- Classes that contain or store samples are *read-write*; they implement both *readBlock* and *writeBlock* operations, plus the *openRead* and *openWrite* methods. They can be used in the same contexts as the read-only objects, and can also be specified as the destination of a **CopySignal** command.

- SST's data-flow processing chains are *demand driven*: the caller determines which samples are required, and the called function delivers those samples. This "pull" model is in contrast to the "push" model used in most real-time signal processing systems, which are *input driven*; they process input data as they arrive.

- The class hierarchies satisfy the Liskov Substitutability Principle [Stroustrup 2000] (mostly): if a role (e.g., an attribute) calls for an object of one of the four base classes, any member of the same hierarchy can be used there. The main exception is the obvious one: read-only classes cannot be used as the destination of **CopySignal**.

4.4 Data Flow Classes

The classes shown in Tables 1 through 4 represent various kinds of time series and are based on data-flow design.

Table 1: Signal Classes

Class	*Inputs*	*Summary*
Signal		Base class (abstract): multi-channel function of time
BareAsciiSignal	file	Signal in a simple ASCII formatted file
BinarySignal	file	Signal in a headerless encoded binary file
BroadbandNoise		Gaussian noise with given power spectrum
DirectSignal	Signal, Eigenrays	Sound propagated from source to receiver
EWASignal	file	Signal in Early Warning Array file format
FIRCoefBuf	Spectrum	Coefficients for variable FIR filter
FrequencyShiftSignal	Signal	Frequency-shifted copy of its input Signal
HarmonicFamily		Sum of harmonic tones
InternalSignal	memory	Signal in an internal buffer
LPFirCoefBuf		Coefficients for low-pass FIR filter
MergeSignal	Any DFs	Merges input flows into adjacent channels
ModulatedTone		Tone with frequency and/or amplitude modulation
ResampleSignal	Signal	Changes *isComplex*, *frequency*, *times*
ReverbSignal	ScatFun, Signal	Generate reverberation sound
SIOSignal	file	Signal in a binary SIO file (times etc. in header)
ScaleSignal	Any DF	Input flow times constant scale factor
SelectChannel	Any DF	One channel from a data flow
SignalFromSpectrum	Spectrum	Compute time series from amplitude spectrum
SoundSignal	file	Signal in .snd file format, for listening
SumSignal	Any DFs	Sum of other data flows
TargetEcho	Signal	Echo of active pulse from target
VarDelay	2-3 Signals	Delay input signal by time-varying amount
VarFirFilter	2 Signals	Variable finite impulse response (FIR) filter

Table 2: Spectrum Classes

Class	*Inputs*	*Summary*
Spectrum	(abstract)	Base class: multi-channel function of frequency and time
AsciiSpectrum	file	Spectrum in an ASCII text file
BinarySpectrum	file	Spectrum in a plain binary file
DirectSpectrum	Spectrum, Eigenrays	Spectrum of sound propagated from source to receiver
FactorSpectrum	Spectrum	Cholesky factorization (matrix square root) of a power spectrum
GaussianSpectrum	Spectrum	Generate Gaussian random realization of a spectrum
InternalSpectrum	memory	Spectrum in an internal buffer
ReverbSpectrum	ScatFun, Spectrum	Convolve reverberation ScatFun with pulse spectrum
SIOSpectrum	file	Spectrum in a binary SIO file (times, frequencies, etc. in header)
SpectrumFromSignal	Signal	Analyze a Signal into a time-dependent Spectrum
UnfactorSpectrum	Spectrum	Square a factored Spectrum to get back a power spectrum
VarSpectFilter	Spectrum, Signal	Frequency Domain Finite Impulse Response filter

Table 3: Scattering Function Classes

Class	*Inputs*	*Summary*
ScatFun	(abstract)	Base class: multi-channel function of Doppler, frequency, and time
AsciiScatFun	file	A ScatFun in a simple ASCII formatted file
BBBDirectionalScat	Eigenrays	Broadband Bistatic Scattering Function using spherical tesselation
BBBScatFun	Eigenrays	Compute a Broadband Bistatic Scattering Function
InternalScatFun	memory	A ScatFun in an internal buffer
SIOScatFun	file	A ScatFun in a binary SIO file

Table 4: Doppler Density Classes

Class	*Inputs*	*Summary*
DopplerDensity	(abstract)	Base class: Random multi-channel function of Doppler, frequency, and time
DirectionalDopplerDensity	Eigenrays	Compute a Doppler Density using spherical tesselation
SIODopplerDensity	file	A DopplerDensity in a binary SIO file

In each table the *Inputs* column gives the types of any stream-like inputs; other inputs are omitted. Eigenrays are considered stream-like because they generate internal streams of eigenray properties (loss spectra, delays, and directions), as discussed in Sec. 8.

Classes with "file" or "memory" in the *Inputs* column are read-write, and the rest are read-only. The "file" or "memory" referred to in that column can be used for input, output, or both. The samples in the "memory" classes can be used as input by setting the values from the command language as in the example in Sec. 4.3; they can be used as output by printing the object using SST's **print** command; or they can serve as temporary storage, written by one **CopySignal** command and read by another. The "file" classes can be uses similarly, except the samples are in a separate file.

Classes with "Any DF" or "Any DFs" in the *Inputs* column can accept as inputs classes from any of the four data flow hierarchies **Signal**, **Spectrum**, **ScatFun**, or **DopplerDensity**. They are "chameleons" that take on the logical character of their inputs. For example, a **SumSignal** object that has **ScatFun** objects as its inputs effectively becomes a **ScatFun**, and can be copied into an output **ScatFun** subclass such as **SIOScatFun**. The extra attributes that come with **ScatFun** (e.g. the *dopplers* grid) tunnel through the **SumSignal** from the inputs to the result.

A few of the classes in Tables 1 through 4 are high-level classes that are essential parts of the user's view of SST. These include **BBBScatFun** (or **BBBDirectionalScat** or **DirectionalDopplerDensity**), **DirectSignal**, **ReverbSignal** (or **DDReverbSignal**), **SumSignal**, and **TargetEcho**; each of these classes will be discussed in more detail in subsequent sections. The rest are storage options, classes used internally by higher-level classes, and utilities that SST users have found useful. All of them are available for use by SST users through the command language, and all of them can be substituted anywhere that an object of the base class is required (subject to read/write constraints).

4.5 Basic Signal Operations

Signal processing operations are handled by those classes in Tables 1 and 2 that accept only other signals or spectra as inputs. Some of these have already been described, and others are simple and obvious. That leaves a few operations that are central to SST's operation: delays, filters, and noise generation.

4.5.1 Variable Delays

Ideally, SST class **VarDelay** accomplishes the following:

$$y_c(t) = x_c(t - T_c(t)), \tag{14}$$

where $x_c(t)$ is the input signal to be delayed (attribute *inSignal*). The time-varying delay $T_c(t)$ is the sum of two input **Signal** objects: a single-channel *commonDelayBuf* to be applied to all channels, and a multi-channel *channelDelayBuf* to be applied separately to corresponding channels of the signal. Either of the two delays may be omitted. Typically, the delays vary much more slowly than the signal itself, so the sampling rate for the delays is much lower than that of the input signal (typically a few samples per second or less). For example, in one application $T_c(t)$ is the sound propagation delay along one ray path from the source to the receiver, which changes with time because the source and receiver are moving.

The algorithm involves two interpolations. First, the delay $T_c(t)$ is interpolated to the sample times required for the output signal $y_c(t)$. Because $T_c(t)$ varies slowly, this first interpolation is always linear. Second, the input signal $x_c(t - T_c(t))$ is interpolated to the times given by the output sample times minus the interpolated delay. Typically this second interpolation must be done using a relatively high order; i.e., one must make use of a relatively large number of samples of the input signal in the neighborhood of each desired time $t - T_c(t)$.

The ideal band-limited interpolation formula, Eq. (3), has the disadvantage that it has a very long decay time — all of the samples in the signal are required to compute a single interpolated value. The first of two compromises used by class **VarDelay** is to introduce a window function to limit the number of samples used:

$$y_c(t) = \sum_{k=-M/2}^{M/2-1} d\left(\frac{t - t_{n'}}{h} + k\right) x_c(t_{n'} - kh), \tag{15}$$

where $M-1$ is the interpolation order (determined by the *order* attribute of the input signal $x_c(t)$) and $t_{n'}$ is the largest input sample time less than the desired time:

$$n' = \lfloor (t - t_0)/h \rfloor. \tag{16}$$

The interpolation weights $d(z)$ are the ideal weights from Eq. (3) multiplied by a Hann window of length M samples:

$$d(z) = \cos^2(\pi z/M) \left(\frac{\sin(\pi z)}{\pi z} \right), \tag{17}$$

where $-M/2 \le z \le M/2$. When the order is 3 or less ($M \le 4$), polynomial interpolation is used instead of Eq. (17).

The second compromise (if $M > 4$) is to tabulate the interpolation coefficients to avoid re-calculating them, using a time grid that is L times finer than the grid used for the signal:

$$d(z) \approx d\left(\left\lfloor \frac{z}{L} + \frac{1}{2} \right\rfloor L \right). \tag{18}$$

The sub-sampling ratio L can be specified by the user, but the default value of 512 is almost always sufficient.

To determine what interpolation order to use, it is helpful to view the interpolation algorithm as a band-pass filter. The spectrum of a discrete-time sampled signal consists of repeated copies of the desired spectrum, one within the Nyquist band (Eq. (2) or (6)) and others above and below the Nyquist band [Oppenheim Schafer 1989]. Interpolating it to a band-limited continuous signal is equivalent to filtering out all of the extra copies outside the Nyquist band.

In the ideal version of Eq. (3), the filter has a spectral response of unity throughout the Nyquist band, and zero elsewhere. Using a window function to reduce the filter length makes this transition more gradual. The width of this transition zone, for an M-sample Hann window, is roughly $2.5/(Mh)$ (from table 7.2 of [Oppenheim Schafer 1989]). Our objective is for the filter to have a spectral response near unity over the part of the Nyquist band in which the signal has significant power. In addition, we want the response to decrease to essentially zero at the locations of those extra copies, outside the Nyquist band, that must be filtered out. In between, near the edge of the band, a finite-length filter produces an incorrect result — so we must ensure that there are clear zones near the band edges where the input signal does not have significant power.

The suggested rule of thumb: if the significant parts of the signal cover a fraction α of the Nyquist band (in the range $F \pm \alpha/(2h)$ for complex signals, or 0 to $\alpha/(2h)$ for real signals), use an interpolation order $M - 1$ satisfying

$$M \ge 2.5/(1 - \alpha). \tag{19}$$

For example, for the common case of 80% band coverage ($\alpha = 0.8$), at least $M = 13$ is required. Even values of M (odd values of the **Signal** attribute *order*) are slightly more efficient, so *order* should be 13 or more.

The default value of *order* is 5, which is appropriate for no more than 50% band coverage.

SST Class: Class **VarDelay** is described above.

SST Class: Class **ResampleSignal**, which we mentioned at the end of Sec. 4.1.2, needs to interpolate between samples of a band-limited signal, just as **VarDelay** does. In the case where the Nyquist band of the original signal lies within the Nyquist band of the resampled signal, **ResampleSignal** uses the same underlying code as **VarDelay**. In the more general cases, where the original Nyquist band extends outside the new one, the signal must be filtered to prevent aliasing. The filter and the interpolation are combined in a single, more complicated set of interpolation coefficients. I have not yet incorporated those equations in this document, but I should.

4.5.2 Variable Finite Impulse Response Filters

SST needs filters whose spectral response depends (slowly) on time as well as frequency:

$$y_c(t) = \int_{-\infty}^{\infty} h_c(\tau, t)\, x_c(t-\tau)\, \mathrm{d}\tau, \tag{20}$$

where the filter impulse function $h_c(\tau, t)$ should ideally satisfy

$$h_c(\tau, t) = \int_{-\infty}^{\infty} H_c(f, t)\, \mathrm{e}^{2\pi \mathrm{i} f \tau}\, \mathrm{d}f, \tag{21}$$

where $H_c(f, t)$ is the filter's specified time-dependent spectral response. This treatment differs from the usual textbook case in that both $H_c(f, t)$ and $h_c(\tau, t)$ depend parametrically on time t. As written, this is theoretically sloppy (what does $H_c(f, t)$ really mean?), but as a practical matter it can be rescued by requiring that the change in $H_c(f, t)$ is negligible over intervals of t comparable to the range of τ [the width of the impulse response $h_c(\tau, t)$]. Equivalently, we require that the frequency response $H_c(f, t)$ varies slowly in frequency on a scale given by the inverse of the scale of its time variation. For example, in one application $H_c(f, t)$ is the sensitivity of a beam pattern to broadband sound arriving along one ray path from the source to the receiver, which changes with time because the source and receiver are maneuvering.

To eliminate the infinities in the integration limits (making a *finite impulse response*, or FIR, filter), we use the *window method* [Oppenheim Schafer 1989]: Eq. (21) is replaced by

$$h_c(\tau, t) = w(\tau) \int_{F-1/(2h)}^{F+1/(2h)} H_c(f, t)\, \mathrm{e}^{2\pi \mathrm{i} f \tau}\, \mathrm{d}f\,, \tag{22}$$

where $w(\tau)$ is a smooth window function whose length is short compared to the scale on which $H_c(f,t)$ varies with time t, and whose Fourier transform is narrow on the scale on which $H_c(f,t)$ varies with frequency f.

The choice of $w(\tau)$ enforces the smoothness requirements of the previous paragraph. The actual frequency response of this filter is the convolution of the desired frequency response, $H_c(f,t)$, with the Fourier transform of the window, $W(f)$. Hence, $W(f)$ should have a narrow central peak and low sidelobes. This issue is discussed at length in signal processing texts [Oppenheim Schafer 1989]. SST uses a Hann (cosine squared) window.

SST Class: Class **VarFirFilter** implements the discrete version of Eq. (20), given an input signal $x_c(t)$ and a stream of filter coefficients $h_c(\tau,t)$ sampled in both τ and t. Normally the sample interval in t for the coefficients is much larger than the sample interval common to τ and the signal. For a given lag τ the coefficients are interpolated linearly in t between the input samples. The convolution is done directly, in the time domain, if the filter is short. For longer filters the convolution is done using fast Fourier transforms (FFTs). The break-even point, determined empirically, is currently at 48 samples in τ; the user can adjust it via a global parameter named *firFourierCutoff*.

SST Class: Class **FIRCoefBuf** is a data-flow class whose input stream is a time-dependent spectrum $H_c(f,t)$ giving the desired filter response, and whose output is a stream of filter coefficients $h_c(\tau,t)$. Most of its work is done by class **FIRCoef**. Together, these classes implement Eq. (22).

SST Class: Class **VarSpectFilter** is equivalent to **VarFirFilter**, except that its input and output signals are in the *windowed frequency domain* representation (Eq. (7)). Because **FIRCoefBuf** uses FFTs whenever it is faster than the time domain implementation, **VarSpectFilter** is advantageous only if the input is already in the frequency domain, or if the desired output is in the frequency domain, or if several filters are to be applied consecutively and the filters are relatively long. **VarSpectFilter** is used internally to implement class **DirectSpectrum**, which is discussed in Sec. 8.

4.6 Generating Signals

Most of SST is about what happens to signals between a source and a receiver. But where do the original signals come from? One option is from "outside": SST can read signals, spectra, and scattering functions in all of the same file formats that it uses for writing them (the entries with "file" or "memory" under the "Inputs" column in Tables 1, 2, and 3). If measured data or externally generated signals are available, simply put it in one of those forms to use it as an input to SST. If the external signal isn't quite right, use SST's signal processing tools (or external tool collections like Matlab) to filter, sum, delay, or resample them.

Most SST simulations, however, have no need of external input signals because SST provides a useful collection of simple tools to generate signals having specified properties. The remainder of this section outlines those tools.

4.6.1 Generating Gaussian Noise

Gaussian noise with a specified power spectrum can be used as a component of the signal put into the water by a source like a submarine or ship. To do that, use it as the ***signal*** component of a **Source** (sec. 7). Such noise can also be used as "background" noise, including sonar self-noise and other distributed noise sources for which SST has no explicit model. To do that, use **SumSignal** to add it to the output signal from the simulation.

Given a power spectral density (PSD) $P_{cc'}(f, t_u)$, our objective is to generate a multi-channel Gaussian random signal $x_c(t)$ such that applying Eqs. (7) and (12) returns a close approximation to the original PSD. Such a signal is a "realization" of the original PSD. To be more precise, if the expectation operator $\mathbf{E}\{\cdot\}$ in Eq. (12) is replaced by an average over independent realizations, that average should converge to the original PSD as the number of realizations increases. This is a well-defined and common problem for stationary, single-channel signals, and extending the usual methods to multiple channels is straightforward. *Stationary*, in this context, means the PSD $P_{cc'}(f, t)$ is independent of time t.

To extend it efficiently to a nonstationary PSD, an additional assumption is required: that an update interval Δ exists such that the time variation of $P_{cc'}(f, t_u)$ is slow on a scale of Δ and the frequency variation is slow on a scale of $1/\Delta$. Under those conditions, the following variant of the method of Mitchell and McPherson [Mitchell McPherson 1981] generates acceptable Gaussian realizations of $P_{cc'}(f, t_u)$.

The first step is to *factor* the PSD: A *generator* $G_{cc'}(f, t_u)$ is needed that satisfies

$$P_{cc'}(f, t_u) = \sum_{c''} G_{cc''}(f, t_u)\, G^*_{c'c''}(f, t_u). \tag{23}$$

This factorization always exists because, for any given frequency f and time t_u, the matrix $P_{cc'}(f, t_u)$ is non-negative definite (its eigenvalues are positive or zero). This factorization is not unique, and there are several good ways to compute $G_{cc'}(f, t_u)$;

SST uses Cholesky factorization [Golub Van Loan 1996], which is fast, reasonably stable, and produces a triangular result.

The second step is to multiply the generator by a vector of independent, complex, unit-variance Gaussian random numbers $g_c(f, t_u)$

$$X_c(f, t_u) = \sum_{c'} G_{cc'}(f, t_u)\, g_{c'}(f, t_u), \tag{24}$$

where the random numbers satisfy

$$\mathbf{E}\left\{g_c(f, t_u) g^*_{c'}(f', t_{u'})\right\} = \delta_{cc'}\, \delta_{uu'}\, \delta(f - f'), \tag{25}$$

where $\delta_{cc'}$ is a Kronecker delta function and $\delta(x)$ is a Dirac delta function. Of course, in the discrete domain the Dirac delta is effectively replaced by the Kronecker delta.

The third step is the same as Eq. (9): Inverse Fourier transform, window, and add. However, for this application the requirement on the post-window function is

$$\sum_u |w'(t - t_u)|^2 = 1 \tag{26}$$

instead of Eq. (11). This is satisfied, for example, by a set of cosine windows with 50% overlap. However, the Mitchell-McPherson window function [Mitchell McPherson 1981], which also satisfies Eq. (26), has somewhat better spectral properties.

SST Classes: The Mitchell-McPherson algorithm fpr generating nonstationary Gaussian noise is implemented by the sequence of **FactorSpectrum** (Eq. (23)), **GaussianSpectrum** (Eq. (24)), and **SignalFromSpectrum** (Eq. (9)). This same sequence is used within class **ReverbSignal** to generate realizations of reverberation. For stationary, single-channel noise, a simpler implementation is provided by class **BroadbandNoise**.

4.6.2 Generating K-Distributed Noise

Sometimes noise isn't Gaussian. Noise that is "spikier" (more impulsive) than Gaussian noise is of special concern because it can lead to false target rates (threshold crossings) higher than the rates predicted for Gaussian noise with the same level and power spectrum. For example, the noise made by snapping shrimp, under some conditions, can contain individual clicks that stand out from the background.

In statistical terms, a key measure is the kurtosis:

$$\gamma_2 = \frac{m_4}{m_2^2} - 3 \tag{27}$$

where the second and fourth *moments* of the zero-mean probability density $p(x)$ are given by

$$m^n = \int x^n p(x) \mathrm{d}x. \tag{28}$$

For a normal (Gaussian) distribution, the fourth moment is three times the second moment, so the kurtosis is zero. A distribution with positive kurtosis (higher fourth moment) has higher tails, so it is more impulsive in character.

The K distribution [Devroye 1986, Abraham Lyons 2002] has some useful properties. It has a "shape" parameter that allows the user to select the impulsiveness of the distribution. For large values of this parameter, the K distribution approaches a Gaussian distribution; the kurtosis approaches zero. For a shape parameter above about 10, the distribution is essentially Gaussian. As you decrease the shape parameter, the kurtosis increases, and the signal becomes more impulsive.

An algorithm like Mitchell-McPherson, in which the random numbers are introduced in the frequency domain and later transformed to the time domain, is inappropriate for non-Gaussian distributions. The Central Limit Theorem assures us that linear combinations of independent random numbers are always closer to Gaussian than the original distributions. If we simply substituted "spiky" random numbers for the Gaussian ones in Eq. (24), most of the spikiness would be lost in the subsequent inverse Fourier transform, Eq. (9). Therefore, we must use a more "brute force" approach to generate impulsive noise: We simply generate white (frequency independent) noise using a fresh, independent K-distributed random number for each sample. To give it the required frequency dependence, the user must apply a filter (e.g. **VarFirFilter**) to the output of **KNoise**.

SST Class: Class **KNoise** produces a stationary, single-channel stream of white, K-distributed random noise. The user specifies the level (in dB) and the shape parameter. In most applications, the **KNoise** should be used as input to a **VarFirFilter** to give it whatever frequency dependence is required. The filter should go to zero near the edges of the Nyquist band to prevent aliasing.

4.6.3 Generating Harmonic Tone Families

Harmonic tone families normally represent machinery noise. The usual way to generate the noise emitted by a submarine, ship, or weapon is to use **SumSignal** to combine several **HarmonicFamily** objects with a **BroadbandNoise** object (or perhaps a **KNoise** object and a filter). The result is used as the *signal* component of a **Source** object representing the vehicle.

The signal generated by a **HarmonicFamily** object has the following form:

$$x(t) = \sum_n A_n \cos(2\pi n f_1 t + \phi_n), \tag{29}$$

where the frequency of each term is an integer multiple of f_1. The user specifies f_1 (attribute *fundamental*) and a three-column table giving the harmonic number n, the amplitude in decibels ($20\log(A_n)$), and the phase at $t = 0$ in degrees ($(180/\pi)\phi_n$) for each harmonic.

4.6.4 Generating Modulated Tones

Class **ModulatedTone** is designed primarily to generate the pulses transmitted by the transmitter of an active sonar system. A **ModulatedTone** object, or several **ModulatedTone** objects combined using a **SumSignal**, may be used to generate almost any of the pulses commonly used by active sonar systems.

The signal generated by a **ModulatedTone** has the following form:

$$\begin{aligned} x(t) &= A\, e(t) \cos(\phi(t)) \qquad (30)\\ \phi(t) &= \phi_0 + \int_{t_0}^{t} m(\tau)\, \mathrm{d}\tau \\ e(t) &= \mathit{envelope}(t) \\ m(\tau) &= 2\pi\, \mathit{frequencyModulation}(\tau) \\ A &= 10^{\mathit{level}/20} \\ \phi_0 &= (\pi/180)\, \mathit{startingPhase} \end{aligned}$$

where *envelope*, *frequencyModulation*, *level*, and *startingPhase* are user-specified attributes of class **ModulatedTone**. In particular, *envelope* and *frequencyModulation*, which specify the time-dependent amplitude and frequency of the generated tone, are *function objects* (objects of any subclass of base class **Function**) that can represent any arbitrary function of time. Often objects of class **TableFunction** are used here. The window functions described in Sec. 4.7 are often useful for the *envelope* attribute. If *envelope* is omitted, it is a constant function with value 1.0. If *frequencyModulation* is omitted for complex envelope signals, it is a constant function whose value is the signal's center frequency (F in Eq. (4)).

The integration used to compute the phase $\phi(t)$ in Eq. (30) is done numerically using two-point (third order) Gaussian integration from each output sample to the next. This is exact for polynomial frequency modulation functions up to cubic. The phase is always continuous throughout the pulse sequence.

Table 5: Window Function Classes

Class	*Sum Rule*	*Summary (see Eq. 31)*
Function		Base class: real function of one real argument
TableFunction		Function specified as table of value vs. argument
CosineWindow	Squared, Eq. (26)	$\cos(\pi x/2)$
HannWindow	Linear, Eq. (11)	$\cos^2(\pi x/2)$
LinearWindow	Linear, Eq. (11)	$1 - x$
MitchellMcPhersonWindow	Squared, Eq. (26)	Mitchell-McPherson [Mitchell McPherson 1981]
RectangularWindow		1.0 in window, else 0
TaylorWindow		Taylor (low sidelobes) [Taylor 1955]

The SST Web contains examples showing how to use **ModulatedTone** to specify shaded pure-tone pulses, FM sweeps, and sequences of shaded or unshaded tones or sweeps. When used with **SumSignal**, **ModulatedTone** can also generate chords and other non-sinusoidal signals.

4.7 Window Functions

Table 5 shows the window functions that SST provides for use with **Spectrum-FromSignal**, **SignalFromSpectrum**, or **ModulatedTone**. Those marked "Squared, Eq. (26)" in the "Sum Rule" column are appropriate for generating noise or reverberation. Those listing "Linear, Eq. (11)" can be used as either the pre-window $w(t)$ or the post-window $w'(t)$ for transforming signals between time-domain and windowed frequency-domain representations (Eqs. (7) and (9)), provided the other window is a **RectangularWindow**. Window functions used for power spectrum analysis (Eq. (12)) or as the *envelope* of a shaded **ModulatedTone** need not satisfy any particular sum rule; for those applications, **TaylorWindow** or **HannWindow** is often preferred to achieve a narrow spectral peak and low sidelobes. Because window functions are derived from base class **Function**, you can use **TableFunction** to enter any window you want. For a general discussion of window functions, refer to Oppenheim and Schafer [Oppenheim Schafer 1989].

In the equations in Table 5, the variable x is a normalized form of the independent variable:

$$\begin{aligned} x &= (|t-\bar{t}| - (t_1 - \bar{t}))/L + 1 \\ \bar{t} &= (t_1 + t_0)/2 \\ L &= F(t_1 - t_0) \end{aligned} \tag{31}$$

where t_0 is the lower limit of the window (attribute ***start***, default -1), t_1 is the upper limit (attribute ***end***, default +1), and F is the fraction of the window length over which it is tapered (attribute ***taperFraction***, default and maximum 1/2). The equations in the table apply only to the tapered region, $0 \le x \le 1$. Each of the window functions has the value 0 if $x \ge 1$ and 1.0 if $x \le 0$.

4.8 Grids

In many places, SST represents continuous functions in terms of samples at discrete values of an independent variable. SST provides a uniform mechanism for specifying these independent values: class **Grid** and its subclasses. They are used, for example, to specify the *times* attribute in a **Signal**, the *times* and *frequencies* of a **Spectrum**, or the *times*, *frequencies*, and *dopplers* of a **ScatFun** or **DopplerDensity**. They are also used to specify the independent variables of **TableFunction** or **TableFunction2** objects, which can be used to specify window functions, the power spectrum in **BroadbandNoise**, and environmental parameters such as sound speed versus depth, volume absorption versus frequency, or ocean depth versus location.

Class **Grid** is the base class for a family of classes designed to specify an ordered set of values of an independent variable where some function is to be evaluated, or sampled. **UniformGrid** is the simplest subclass of **Grid**, in which the interval between adjacent values is constant (Eq. (1)). Other members of the family include **GeometricGrid** (in which the ratio between adjacent values is constant), **ListGrid** (for arbitrary values), and **SubGrid** (a subset of values from some other **Grid**).

5 The Eigenray Model

The model that describes how sound is transformed as it propagates from one part of the ocean to another — the eigenray model — is central to SST's operation. This transformation requires two time-dependent sets of inputs: the sound to be transformed, and the set of losses and delays to be applied to that sound. The eigenray model supplies the second set in the form of a finite list of eigenrays, each of which represents a distinct path by which sound can travel from one place in the ocean to another.

5.1 Eigenray Properties

Each eigenray takes as its input the source and receiver locations in the ocean, $\boldsymbol{r}_S$ and $\boldsymbol{r}_R$. Given that pair, eigenray p produces the following information:

- $T_p(\boldsymbol{r}_R, \boldsymbol{r}_S)$ is the propagation delay for path p; i.e., the time required for sound to travel from one end of the path to the other.
- $L_p(f, \boldsymbol{r}_R, \boldsymbol{r}_S)$ is the complex-valued propagation loss factor for path p including spreading loss, boundary reflection loss, and volume absorption. Its unit is inverse length (m^{-1}). It gives the ratio of the sound pressure at $\boldsymbol{r}_R$ to the transmitted sound pressure at $\boldsymbol{r}_S$ reduced to a distance of one meter, as a function of acoustic frequency f.
- $\boldsymbol{S}_{Sp}(\boldsymbol{r}_R, \boldsymbol{r}_S) = \nabla_S T_p(\boldsymbol{r}_R, \boldsymbol{r}_S)$ is the *slowness vector* at the source, which is defined as the spatial gradient of the time delay with respect to the source location $\boldsymbol{r}_S$. Its direction is opposite the direction of propagation along the eigenray at the source, and its magnitude is the inverse of the local sound speed at the source.
- $\boldsymbol{S}_{Rp}(\boldsymbol{r}_R, \boldsymbol{r}_S) = \nabla_R T_p(\boldsymbol{r}_R, \boldsymbol{r}_S)$ is the slowness vector at the receiver R; i.e., the spatial gradient of the time delay with respect to the receiver location $\boldsymbol{r}_R$. It points in the direction of propagation along the eigenray at the receiver, and its magnitude is the inverse of the local sound speed at the receiver.

5.2 Eigenray as Local Expansion

At one level, the eigenray model consists of a finite set of maps whose inputs are two end-point locations, $\boldsymbol{r}_R$ and $\boldsymbol{r}_S$, and whose outputs are the four pieces of information listed above: T_p, $L_p(f)$, $\boldsymbol{S}_{Sp}$, and $\boldsymbol{S}_{Rp}$.

At a slightly higher level, the presence of the gradients $\boldsymbol{S}_{Sp}$ and $\boldsymbol{S}_{Rp}$ suggests use of a first-order Taylor expansion to compute the time delay, not just between the given end points, but between two points in small neighborhoods around the given end points:

$$T_p(\boldsymbol{r}_R + \boldsymbol{\rho}_R, \boldsymbol{r}_S + \boldsymbol{\rho}_S) = T_p(\boldsymbol{r}_R, \boldsymbol{r}_S) + \boldsymbol{S}_{Rp} \cdot \boldsymbol{\rho}_R + \boldsymbol{S}_{Sp} \cdot \boldsymbol{\rho}_S + \cdots , \tag{32}$$

where the offset vectors $\boldsymbol{\rho}_R$ and $\boldsymbol{\rho}_S$ are assumed to be small compared to the total propagation distance. This is useful because sources and receivers have finite size. Given an eigenray from one point on the source to one point on the receiver, this expansion can be used to compute (approximately) the delay from any part of the source to any part of the receiver.

For most purposes the "local plane wave" approximation (first-order Taylor expansion) of Eq. (32) is sufficient. However, for a long array and a source at short range, wave-front curvature may be significant. More to the point, large-aperture passive sonars like towed arrays use wave-front curvature as an important clue for estimating target range. The eigenray model does not give enough information for a full second-order expansion of the time delay, but SST gets part way there using a *spherical wave* approximation. In this approximation, the wave front at each end is locally spherical, with a center located at the apparent location of the other end as inferred from the time delay and the local sound speed. Each of the dot products in Eq. (32) is replaced by a second-order expression of the form

$$\tau_p(\boldsymbol{S}, \boldsymbol{\rho}) = \boldsymbol{S} \cdot \boldsymbol{\rho} + \frac{|\boldsymbol{S}|^2 |\boldsymbol{\rho}|^2 - (\boldsymbol{S} \cdot \boldsymbol{\rho})^2}{2T_p}. \tag{33}$$

This approximation is good for straight-line propagation, and remains useful for horizontal curvature in most scenarios. It is less trustworthy for vertical curvature in the presence of ray bending.

Another important property is that the delay T_p does not depend on frequency because the ocean is *nondispersive*; i.e., the dependence of the sound speed on frequency is very weak [Urick 1983]. Small violations of the nondispersive assumption can be absorbed into the phase of the complex loss $L_p(f, \boldsymbol{r}_R, \boldsymbol{r}_S)$.

It is equally important to note that the eigenray model does not give gradients of the propagation loss $L_p(f, \boldsymbol{r}_R, \boldsymbol{r}_S)$. This is a value judgment, not a fundamental limitation. The eigenray model computes the gradients of time delay because beam formers are very sensitive to variation in propagation time between one part of a source or sonar and another part. However, variation in propagation loss in a region the size of a source or sonar is not so important, so it is ignored. The loss $L_p(f, \boldsymbol{r}_R + \boldsymbol{\rho}_R, \boldsymbol{r}_S + \boldsymbol{\rho}_S)$ is assumed to be independent of $\boldsymbol{\rho}_R$ and $\boldsymbol{\rho}_S$ over such regions. Large sonar arrays or targets for which these assumptions do not hold can still be modeled, but to do so SST must treat them as compact groups of elements or highlights, and explicitly compute the eigenrays to the center of each group.

Thus, at the second level, the inputs to the eigenray model are not two points, but two *regions* around those points. For each eigenray, the eigenray model gives the loss and delay from any point in one region to any point in the other region.

5.3 Eigenray as Sound Transformation

There is a third level, too: the eigenray model specifies a *linear transformation* whose input is a sound source field $q_S(t, \boldsymbol{r})$ emitted from various parts of a source in the neighborhood of $\boldsymbol{r}_S$, and whose output is the resulting sound field $p(t, \boldsymbol{r})$ in the neighborhood of $\boldsymbol{r}_R$. SST assumes that this transformation has the following mathematical form (in the time domain):

$$\begin{aligned} p_R(t, \boldsymbol{r}_R + \boldsymbol{\rho}_R) \;&=\; \sum_p \int\int l_p(\tau, \boldsymbol{r}_R, \boldsymbol{r}_S) \\ &\times q_S\left(t - T_p(\boldsymbol{r}_R + \boldsymbol{\rho}_R, \boldsymbol{r}_S + \boldsymbol{\rho}_S) - \tau, \boldsymbol{r}_S + \boldsymbol{\rho}_S\right) \,\mathrm{d}\tau \,\mathrm{d}\boldsymbol{\rho}_S. \end{aligned} \tag{34}$$

The offset vectors $\boldsymbol{\rho}_R$ and $\boldsymbol{\rho}_S$ have the same scale as the size of the receiver and source, respectively, and are assumed to be small compared to the total propagation distance. The sum is over eigenrays (paths) p.

The propagation impulse response $l_p(\tau, \boldsymbol{r}_R, \boldsymbol{r}_S)$ in Eq. (34) is the inverse Fourier transform of the propagation loss:

$$l_p(\tau, \boldsymbol{r}_R, \boldsymbol{r}_S) = \int_{-\infty}^{\infty} L_p(f, \boldsymbol{r}_R, \boldsymbol{r}_S)\, \mathrm{e}^{2\pi \mathrm{i} f \tau} \,\mathrm{d}f. \tag{35}$$

Because $L_p(f, \boldsymbol{r}_R, \boldsymbol{r}_S)$ varies slowly with frequency, $l_p(\tau, \boldsymbol{r}_R, \boldsymbol{r}_S)$ is sharply peaked in time delay τ. Eq. (34) is combined with the source and receiver models in Sec. 8.

For eigenrays that reflect from the surface or bottom, Eq. (35) can be inaccurate, especially at high frequencies. In real life, sound that reflects from a rough boundary is spread in time, in arrival angle, and sometimes in frequency. **EigenrayModel** and its subclasses offer a switch, called ***doTimeSpread***, that causes the set of FIR filter coefficients, $l_p(\tau, \boldsymbol{r}_R, \boldsymbol{r}_S)$, to get considerably longer and take on a random, time-dependent character. The default value of ***doTimeSpread*** is *true*; i.e., this randomness is added unless you explicitly turn it off. Because the implementation involves components that have not yet been introduced, discussion of the time spread model is deferred to Sec. 12.

5.4 Straight-line Eigenray Model

Of SST's three choices for the eigenray model, the only one that is contained entirely within the SST software is the base class, **EigenrayModel**. This model is valid only if the sound speed is independent of depth and location and the ocean depth is constant. Sound travels in straight lines except where it reflects in the specular (mirror) direction from the surface or bottom.

SST computes the eigenrays using the *method of images*. Effectively, the sound travels in a straight line from the source to an *image* of the receiver, which is placed at the apparent location of the receiver as seen from the source, possibly via surface and bottom reflections. Each image is above or below the actual position of the receiver, by a distance that depends on the ocean depth and the number and order of the reflections. The vector from the source to the image of the receiver for path p is given by

$$\boldsymbol{R}_p = \boldsymbol{r}_R - \boldsymbol{r}_S - 2|S_p - B_p| z_R \hat{z} \pm 2 B_p D \hat{z}, \tag{36}$$

where B_p is the number of bottom reflections for path p, S_p is the number of surface reflections, D is the ocean depth, $\boldsymbol{r}_R$ is the receiver location, $\boldsymbol{r}_S$ is the source location, $\hat{z}$ is a unit vector in the z direction (down), and $z_R = \boldsymbol{r}_R \cdot \hat{z}$ is the receiver depth. The sign of the last term is $+$ if the reflection closest to the source is from the bottom, or $-$ if it is from the surface. There are four images (i.e., four paths p) for each nonzero value of B_p: $S_p = B_p - 1$, $S_p = B_p + 1$, and two with $S_p = B_p$.

The time delay for path p is

$$T_p = R_p / c, \tag{37}$$

where $R_p = |\boldsymbol{R}_p|$ is the slant range from the source to the image of the receiver and c is the sound speed.

The propagation loss for eigenray p includes spherical spreading, volume absorption, and reflection losses:

$$L_p(f) = R_p^{-1} \mathrm{e}^{\alpha(f) R_p} \prod_b \beta(s_p, f, \boldsymbol{r}_{pb}), \tag{38}$$

where the product is over bounces b from the surface or bottom, s_p is the sine of the grazing angle for eigenray p, f is the frequency, $\boldsymbol{r}_{pb}$ is the location on the surface or bottom of bounce b of eigenray p, $\beta(s_p, f, \boldsymbol{r}_{pb})$ is the reflection coefficient (from the surface or bottom model, Sec. 6.4.1), and $\alpha(f) \leq 0$ is the volume absorption rate per meter (from the volume attenuation model, Sec. 6.3).

If the user specifies a location-dependent bottom type using class **GriddedBoundary** (Sec. 6.4), the reflection coefficient can depend on the bounce location $\boldsymbol{r}_{pb}$. These locations are on two horizontal lines on the surface and bottom connecting locations above and below the source and receiver. The location of the bounce with indices pb is given by

$$\begin{aligned} \boldsymbol{r}_{pb} &= (1 - w_{pb}) \boldsymbol{r}_S + w_{pb} \boldsymbol{r}_R \\ \text{except} \quad z_{pb} &= 0 \quad \text{if bounce b is a surface bounce} \\ &= D \quad \text{if bounce b is a bottom bounce,} \end{aligned} \tag{39}$$

where $z_{pb} = \boldsymbol{r}_{pb} \cdot \hat{z}$ is the depth of the bounce. The horizontal fraction w_{pb} is given by

$$\begin{aligned} w_{pb} &= (z_S + bD)/|Z_p| \quad \text{if bounce 0 is a surface bounce} \\ &= (D - z_S + bD)/|Z_p| \quad \text{if bounce 0 is a bottom bounce,} \end{aligned} \tag{40}$$

where the bounce index b starts with 0 (nearest the source), $z_S = \boldsymbol{r}_S \cdot \hat{z}$ is the source depth, and $Z_p = \boldsymbol{R}_p \cdot \hat{z}$ is the total vertical distance traversed by the eigenray ($\boldsymbol{R}_p$ is from Eq. (36)). Bounces from the surface and bottom alternate.

The slowness vectors are given by:

$$\begin{aligned} \boldsymbol{S}_{Rp} &= \boldsymbol{R}_p/(cR_p) \\ \boldsymbol{S}_{Sp} &= -\boldsymbol{R}_p/(cR_p) \end{aligned} \tag{41}$$

except that the sign of the vertical (Z) component of $\boldsymbol{S}_{Rp}$ is reversed if the total number of bounces, $S_p + B_p$, is odd.

5.5 CASS/GRAB Eigenrays

The Comprehensive Acoustic System Simulation (CASS) [Weinberg et al. 2001] software, developed by Henry Weinberg and others at NUWC Newport, models oceans in which the sound speed may vary with all three dimensions, and the bathymetry and bottom type may depend on horizontal location. The eigenrays can bend, and the direction of the forward bottom reflection depends on the local bottom slope. CASS is a self-contained modeling environment that can compute and plot many different results of interest for sonar analysis and performance prediction. The only component of CASS used by SST is the Gaussian Ray Bundle (GRAB) [Weinberg Keenan 1996] eigenray model.

CASS produces eigenrays in external binary files, which SST reads and uses in its simulations. Two different SST classes may be used, depending on the relationship between the CASS run and the SST run. The recommended choice is to run only SST, and specify class **CASSEigenrayRun** for the eigenray model. In that case, SST generates the CASS input files based on its own input parameters, and runs CASS as a sub-process. This is recommended in most cases because SST ensures that the CASS and SST run streams are consistent. It is also less work than **CASSEigenrayModel** because you need to prepare only SST's input files, not CASS's.

SST users may choose to prepare CASS input files, run CASS to generate eigenrays, and then run SST independently. In that case, choose class **CASSEigenray-Model** and specify the names of the eigenray files that CASS wrote. This is strongly discouraged because it is very difficult to maintain consistency between CASS and SST. **CASSEigenrayModel** remains in SST primarily to facilitate testing.

CASSEigenrayRun offers two different mechanisms for handling forward reflection losses from heterogeneous bottoms. For the first option, simply specify a **GriddedBoundary** as the *bottom* component of the **Ocean**. SST tabulates the reflection coefficient for each of the component **Boundary** objects, and passes the result to CASS as a separate "ENVIRONMENT". Thereafter, the GRAB eigenray model uses them directly. Unfortunately, CASS imposes a hard upper limit on the number of "environments", so runs with very fine-scale bottom variation can fail. Also, CASS is much slower with a large number of environments.

To address these problems, **CASSEigenrayRun** offers an option in which some of the calculation is moved from CASS into SST. The user specifies the desired **GriddedBoundary** as above, and in addition specifies an optional **CASSEigenrayRun** attribute called *cassBottom* by assigning an object of a **Boundary** class other than **GriddedBoundary** (i.e., a location-independent one). The CASS run uses this bottom type. Then, as **CASSEigenrayRun** reads the resulting eigenrays, it modifies the amplitudes by multiplying by the ratio of the actual reflection coeffcient (from the **GriddedBoundary**) to the value CASS used. The exact same eigenrays will not result, because CASS eliminates some eigenrays depending on the amplitude. Hence it is advisable to choose one of the harder (less lossy) bottom types from the set used in the **GriddedBoundary**, and use that one for the *cassBottom* attribute.

To make this second option work, **CASSEigenrayRun** needs more information from CASS. In addition to the eigenray properties listed in Sec. 5.1, it needs the locations and angles of all of the bounces. Fortunately, recent releases of CASS can provide this information if the "SEPES" eigenray model is specified. The SEPES model is identical to the GRAB model, except that the extra information about reflections is saved in additional binary files. **CASSEigenrayRun** always uses the SEPES eigenray model.

5.6 Generic Sonar Model (GSM) Eigenrays

The Generic Sonar Model (GSM) [Weinberg 1985], a much older product of Henry Weinberg at NUWC, models oceans in which the sound speed may depend only on depth. The sound speed, ocean depth, and boundary properties are assumed to be independent of horizontal location (*range independent*). GSM includes five different user-selectable eigenray models, of which the most useful with SST (in our opinion) are the Multipath Expansion (MULTIP) and Fast Multipath Expansion (FAME) models.

Use of GSM is somewhat simpler than use of CASS, but otherwise very similar. Like CASS, GSM can be run separately before SST, using the SST class **GSMEigenrayModel**. Your other choice is to run only SST, specifying class **GSMEigenrayRun** for the eigenray model; SST then runs GSM as a subprocess. The same issues of consistency arise with GSM as with CASS, but because GSM is simpler, the probability of getting it right with **GSMEigenrayModel** is somewhat better than with **CASSEigenrayModel**. Nevertheless, **GSMEigenrayRun** is the recommended option.

NUWC no longer supports GSM. We continue to support its use in SST, at least until our users have time to convert their SST scripts to CASS.

5.7 Eigenray Interpolation and Ray Identity

For many reasons, it is important that the delay, loss, and slowness vectors for a given eigenray are continuous functions of the end points $\boldsymbol{r}_R$ and $\boldsymbol{r}_S$. Continuity of the delay T_p is especially important, in part because the first derivative of the delay is the Doppler shift, to which many sonar receivers are especially sensitive. Achieving continuity for the straight-line eigenray model is easy because its properties are computed as needed, and because the identity of each eigenray is uniquely specified by the number and order of its surface and bottom reflections.

For both CASS and GSM eigenrays, the eigenray files read by SST contain eigenrays (sets of eigenray properties) at discrete values of the end point locations. Values between those locations must be computed using interpolation. Given a list of eigenrays at one location and another such list at a neighboring location, we are faced with the problem of matching the members of one list with the members of the other list in such a way that it makes physical sense to interpolate eigenray properties between the matching eigenrays at different locations. Matching eigenrays are assigned to a single *ray* (an object of some subclass of base class **Ray**). The expectation is that for a given *ray* object identified by index p, the maps from the location pair $(\boldsymbol{r}_R, \boldsymbol{r}_S)$ to the properties T_p, $L_p(f)$, $\boldsymbol{S}_{Sp}$, and $\boldsymbol{S}_{Rp}$ are continuous and physically meaningful.

CASS and GSM help with this matching by providing a *signature* for each eigenray, consisting of the numbers of surface bounces, bottom bounces, upper vertexes, and lower vertexes for each one. Unfortunately, the signature is not sufficient to specify uniquely how to match eigenrays at different locations because there can be any number of eigenrays (including zero) with the same signature for the same location pair, and that number can change from location to location.

At the start of a simulation, SST reads the CASS or GSM eigenray files into its memory and organizes them into a set of internal tables designed to support interpolation in the tables. To do this, each set of eigenray properties read from the files is assigned a *ray identity*, and eigenrays with the same identity are eventually mapped into the same *ray* object. Eigenrays with the same signature are assigned their ray identities using a complex set of heuristics that take into account the closeness of the attributes (especially delay) of one eigenray to a simple extrapolation of the attributes of an adjacent eigenray. These heuristics work well in our tests, but our tasks would be much easier and our confidence higher if the eigenray models provided a unique signature to support unambiguous matching for interpolation.

6 Ocean Model

The primary inputs to the eigenray model (besides the locations of the end points) come from properties of the *ocean*, which is an object of class **Ocean**. The properties used as inputs to the eigenray model are the depth, sound speed, volume attenuation, and the reflection coefficients of the surface and bottom. These properties are used directly by the straight-line **EigenrayModel**, and they are passed in table form to CASS or GSM by class **CASSEigenrayRun** or **GSMEigenrayRun**, respectively. Classes **CASSEigenrayModel** or **GSMEigenrayModel** do not use them, since they get their inputs from the user-supplied CASS input file.

In addition, class **Ocean** supplies the scattering strengths used by SST's reverberation model, which will be described in Secs. 10 and 11.

6.1 Ocean Depth

The *depth* attribute of class **Ocean** is, in general, a function of horizontal location $D(x, y)$, in meters, with x and y in meters north and east of an arbitrary origin. Its type is class **Function2**, which is an abstract base class for describing any real-valued function of two real variables. That means the user can assign to *depth* an object of any subclass of **Function2**. By default, it is an object of class **ConstantFunction2**; the user simply assigns a value, e.g., "`ocean.depth = 200`".

If CASS eigenrays are used, the bathymetry can be specified by assigning to the *depth* attribute an object of class **TableFunction2**, which provides several ways to enter a table of depth on a rectangular grid versus horizontal location (meters north and east of an arbitrary origin). Interpolation in the table is normally bilinear (order 1), although the user can specify the order up to cubic (order 3).

Alternatively, assign to **Ocean**'s *depthFile* attribute the name of a file in CASS's input format containing a "`BOTTOM DEPTH TABLE`" section. SST will search for that section, parse it, extract the bathymetry, and build a **TableFunction2** object from the data.

6.2 Ocean Sound Speed

The *soundspeed* attribute of class **Ocean** is, in general, a function of depth $c(z)$ in meters per second, versus depth z in meters. Its type is class **Function**, which is an abstract base class for describing any real-valued function of one real variables. By default, it is an object of class **ConstantFunction**; the user simply assigns a value, e.g., "`ocean.soundspeed = 1500`". The default value is 1520 m/s.

If CASS or GSM eigenrays are to be used, the sound speed profile can be specified by assigning to the *soundspeed* attribute an object of class **TableFunction**, which provides several different ways to enter a table of sound speed versus depth. This table is passed to CASS or GSM, which does its own interpolation.

CASS permits the sound speed to depend on horizontal location as well as depth. However, SST does not yet provide a way to specify this horizontal variability.

6.3 Ocean Volume Attenuation

The *volumeAttenuation* attribute of class **Ocean** is, in general, a function of frequency $A(f)$ in dB per km, versus frequency f in Hz. By default, it is an object of class **ConstantFunction**; the user simply assigns a value, e.g., "`ocean.volumeAttenuation = -2.1`". The default value is 0 dB/km.

To specify frequency-dependent attenuation, one option is to assign to the *volumeAttenuation* attribute an object of class **TableFunction**, which allows the user to enter a table of attenuation versus frequency using tables of the same formats used for the sound speed.

Another option is to assign **ThorpAttenuation** to the attribute. This option computes the absorption using the expression in [Urick 1983] page 108:

$$\alpha = \frac{0.1f^2}{1+f^2} + \frac{40f^2}{4,100+f^2} \tag{42}$$

where the frequency f is in kHz and the result α is in dB per kiloyard. The last two terms in the expression in [Urick 1983] are omitted, to match the implementation in CASS [Weinberg et al. 2001]. The input is converted from Hz to kHz, and the output from dB/kYd to dB/km, in the SST code surrounding the Thorp formula. This expression applies for a temperature of 4 deg C and a depth of about 1 km; it is approximate but useful elsewhere.

6.4 Surface and Bottom Models

Class **Ocean** contains attributes *surface* and *bottom*, both of which are objects of class **Boundary**. SST users can assign to them any object of any class derived from **Boundary**, including **APLBottom**, **JacksonBottom**, **APLSurface**, **GilbertSurface**, **McDanielSurface**, and **GriddedBoundary**.

6.4.1 Reflection Coefficients

Each of these classes defines two complex-valued member functions *TotalForwardAmp* and *CoherentForwardAmp*, each of which is a function of three arguments, *sinAngle* (the sine of the grazing angle) and *frequency* (in Hz), and *pos* (a 3-vector giving the position on the boundary).

The straight-line **EigenrayModel** class uses these functions to compute the factors $\beta(s_p, f, \boldsymbol{r}_{pb})$ in the eigenray propagation loss, Eq. (38); classes **CASSEigenrayRun** and **GSMEigenrayRun** use them similarly. For each boundary, SST calls the member function *CoherentForwardAmp* when the eigenrays are being used for one-way propagation or target echoes, but it calls *TotalForwardAmp* when the eigenrays are being used for reverberation. The theory behind this practice is based on the following distinction:

- For reverberation, the distinction between near-specular forward scattering and specular reflection is unimportant; all that matters is how much energy is removed from the total. Hence, SST calls *TotalForwardAmp*, which treats forward-scattered energy as if it were specularly reflected.
- For passive reception or target echoes, scattered energy is effectively lost because scattering reduces the coherence of the signal (reducing processing gain), stretches it out in time, and spreads it in angle, all of which tend to push it down under the background. Hence, SST calls *CoherentForwardAmp*, which treats forward-scattered energy as if it were absorbed.

This theory is flawed; for many purposes *TotalForwardAmp* is too large and *CoherentForwardAmp* is too small. As an option, SST addresses this issue in an approximate way by controlling the time-frequency coherence, replacing the "coherent versus incoherent" dichotomy by explicit and quantitative control of coherence. Coherence control is discussed in Sec. 12

For most purposes the distinction is unimportant anyway because *CoherentForwardAmp* is used only by the straight-line **EigenrayModel**. If **CASSEigenrayRun** or **GSMEigenrayRun** is selected, the *TotalForwardAmp* function is called repeatedly with different values of frequency and grazing angle to build a "`SURFACE REFLECTION COEFFICIENT TABLE`" and a "`BOTTOM REFLECTION COEFFICIENT TABLE`" to be passed to CASS or GSM. This tends to overestimate the reflection coefficient, except for reverberation. The table is passed in decibels; CASS and GSM ignore the phase.

(*Note*: CASS accepts a table of phase shifts too, but **CASSEigenrayRun** doesn't provide it. This needs to be fixed soon.)

6.4.2 Bistatic Scattering Strength

Each **Boundary** subclass also provides a function called *BistaticStrength*, denoted $S_l(f, \boldsymbol{S}_p, \boldsymbol{S}_q, \boldsymbol{r}_S)$. It computes the scattering differential cross section per unit area of the boundary; it is dimensionless (an area divided by an area). It takes as its input a frequency f, a scattering location $\boldsymbol{r}_S$, and two slowness vectors, $\boldsymbol{S}_q$ giving the direction from which incident sound arrives at the boundary, and $\boldsymbol{S}_p$ giving the direction toward which it scatters. The subscript l denotes a scattering layer (surface or bottom) and the subscripts p and q denote the outgoing and incoming eigenrays.

Most of the **Boundary** subclasses are fully bistatic. That means the two slowness vectors reduce to at least three independent parameters: two *grazing angles* (from the boundary plane to each of the two slowness vectors) and a *bistatic angle* (from a vertical plane containing one vector to the other vector). Under most conditions, the bistatic strength is strongly peaked in the region of the specular direction.

The older **Boundary** subclasses, **APLBottom** and **APLSurface**, are monostatic. That means they define a backscattering strength $S_l(f, \sin(\theta))$, which is a function of only one angle, the grazing angle θ. For those classes, the *BistaticStrength* method computes the backscattering strength for each of the two grazing angles (incoming and outgoing) and returns the geometric average (the square root of the product). This is the same form used by CASS. It is a reasonably good approximation for geometries that are close to backscattering (i.e., if the two slowness vectors are nearly equal) and for bubble-dominated surface scattering. It is a particularly bad approximation for the region near the specular direction because it does not produce a peak in that region. The monostatic models may be removed in a future SST release.

Table 6: Surface and Bottom Models

Class	*References*	*Use*
Boundary		Monostatic, table driven
APLBottom	[APL Models 1994, Mourad Jackson 1989]	Bottom, monostatic, high frequency
JacksonBottom	[Williams Jackson 1998, APL Models 1994, Mourad Jackson 1993, Mourad Dahl Jackson 1991, Moe Jackson 1994, Schulten Anderson Gordon 1979]	Bottom, bistatic, mid to high frequency
APLSurface	[APL Models 1994]	Surface, monostatic, high frequency
GilbertSurface	[Gilbert 1993, Kulbago 1994]	Surface, bistatic, low to mid frequency
McDanielSurface	[McDaniel 1990, Lang Culver 1992, Donelan Hamilton Hui 1985]	Surface, bistatic, high frequency
GriddedBoundary		Location dependent

6.4.3 Boundary Classes

The **Boundary** classes are listed in Table 6 with the literature references from which they were taken and a brief indication of applicability. In the table "high frequency" means over 10 kHz and "mid frequency" means 1 to 10 kHz (roughly).

The base class **Boundary** allows the user to enter either or both of the reflection coefficient functions and the monostatic backscattering strength as tables versus angle and frequency. By default the reflection coefficients are unity (no loss) and the backscattering strength is zero in energy terms (no scattering). **Boundary** may be used for either the surface or the bottom.

For the surface models **APLSurface**, **GilbertSurface**, and **McDanielSurface**, *TotalForwardAmp* includes all of the incident intensity except the fraction that is absorbed by bubbles and converted to heat. *CoherentForwardAmp* also removes scattered intensity using a simple model of surface roughness [APL Models 1994]; this loss can be very substantial whenever the wave height is comparable to or greater than the acoustic wavelength. The only environmental input to these models is the wind speed.

For the bottom models **APLBottom** and **JacksonBottom**, the *TotalForwardAmp* and *CoherentForwardAmp* are identical. The reflected intensity includes all of the incident intensity except the fraction that is refracted into the bottom, as estimated using a lossy Rayleigh coefficient [Mackenzie 1959]. Bottoms of these classes may be specified using sets of parameters describing surface roughness, sediment sound speed, absorption rate, and scattering within the sediment. Sets of named bottom types (e.g., *MediumSand* or *CoarseSilt*) are also provided.

Class **GriddedBoundary** provides a mechanism to add location dependence. The user provides a table containing other **Boundary** (or subclass) objects, each of which is associated with a two-dimensional location. The locations must be on a rectangular grid in x and y (meters north and east, respectively, of the origin of the World coordinate system). The properties of the boundary at a given location $\boldsymbol{r}$ are those of the contained **Boundary** object at the closest location in the table. Normally **GriddedBoundary** is used to model heterogeneous bottoms, although it could be used for the surface too (e.g. to model a headland wind shadow or a local storm).

6.5 Volume Scattering Strength

Volume scattering tends to occur in *layers* because sea life tends to congregate at restricted ranges of depth. Therefore, volume scattering is specified in SST by defining a list of **ReverbLayer** objects, each of which specifies the volume scattering strength (assumed constant) between specified upper and lower depth limits. Currents are specified by giving the average horizontal velocity of the scatterers in each layer.

Currently, SST's reverberation model (Sec. 11) treats each layer as if all of the scatterers were concentrated in a thin sheet in the center of the layer. Therefore, sometimes realism can be improved by defining separate volume layers covering depths with different propagation conditions, even if the volume scattering strength is nearly uniform. This "sheet" approximation can be unrealistic at very short times, when reverberation from a scattering layer first starts up too late and too suddenly, but it is usually acceptable for fully-ensonified scattering layers.

7 Sonar and Source Models

The sonar model (the box labeled "Sonar Receiver" in Fig. 1) describes any compact object that can receive sound from the ocean. The source model (the boxes labeled "Sonar Transmitter" and "Source") describes any compact object that emits sound into the ocean. These are represented in SST by objects of classes **Sonar** and **Source**, respectively. For passive sonar scenarios the **Source** is whatever the sonar is listening to. For active sonar scenarios the **Source** is the transmitter. Other **Source** objects may represent countermeasures, the sonar's own vehicle, interfering ships in the area, pile drivers, explosions, whales, or anything else that produces sound from a small region. **Sonar** and **Source** objects are also used internally to model target echoes (Sec. 9.1).

Classes **Sonar** and **Source** are almost identical. Each object of either of those classes contains the following attributes:

- *trajectory*: an object of a class derived from base class **Traject**. It determines the location and orientation of the sonar or source platform as a function of time.
- *beams*: a list of objects derived from base class **Beam**. They determine the spatial properties of each channel, including its directional sensitivity and the location of its phase center.
- *signal*: an object of a class derived from base class **Signal**. For a **Source**, this is an input representing transmitted sound. For a **Sonar**, it is an output representing received sound. (The association of a **Signal** with a **Sonar** is implicit; it is not formally an attribute of class **Sonar**.)

An active sonar system consists of a **Sonar** and at least one **Source**. It is *monostatic* if those components share a single trajectory, and it is *bistatic* if they have different trajectories.

7.1 Trajectories and Coordinate Transformations

In **Sonar**, **Source**, and **Target** objects, the local geometric properties (the locations and beam patterns of the transducers, sources, or highlights) are expressed in a *platform-centered* coordinate system whose components are in the directions (Forward, Starboard, Below) relative to some arbitrarily defined "center" of a particular platform (its *origin*). Global properties of the ocean are expressed in the *Earth-centered* coordinate system, whose components are in the directions (North, East, Down) relative to an origin at an arbitrary point on the surface of the ocean. The time-dependent mapping between the local and global coordinate systems is defined by the *trajectory* attribute of each vehicle. An SST user specifies the trajectory by assigning to this attribute an object of any subclass of class **Traject** — usually a **Trajectory**, but sometimes a **CombinedTraject**.

Two kinds of vectors need to be transformed between local and coordinate systems: locations and directions (slowness vectors). These transformations take the following linear form:

$$\begin{aligned} \boldsymbol{r}' &= \boldsymbol{M}_X(t)\;(\boldsymbol{r} - \boldsymbol{r}_X(t)) \\ \boldsymbol{S}' &= \boldsymbol{M}_X(t)\,\boldsymbol{S}\,, \end{aligned} \tag{43}$$

where the primed vectors are expressed in platform coordinates and the unprimed vectors are in Earth-centered coordinates. The vector $\boldsymbol{r}_X(t)$ is the location of the origin of platform X, and $\boldsymbol{M}_X(t)$ is the 3-by-3 rotation matrix determined by the orientation of the platform (sonar, source, or target). Each vehicle's trajectory supplies the time-dependent location $\boldsymbol{r}_X(t)$ and rotation matrix $\boldsymbol{M}_X(t)$ used in these coordinate transformations.

A **Trajectory** (the most commonly used subclass of **Traject**) specifies a body's motion using a list of **Snapshot** objects. Each **Snapshot** is a "picture" of the position, velocity, orientation, and rotation rate of a body (four attributes, three numbers each) at a single specified time. For intermediate times the trajectory is computed using self-consistent cubic interpolation, in which all four attributes are continuous in time. For times outside the range of the list, each **Trajectory** is extrapolated using the assumption that the velocity and the rotation rate remain constant in the body's own coordinate system. Thus, a **Trajectory** containing a single **Snapshot** can be used to specify motion in a straight line, a circle, or a helix.

The other **Traject** subclass is **CombinedTraject**, which is specified in terms of two other **Traject** objects: one to specify the motion of a body with respect to an intermediate coordinate system, and another to specify the motion of that intermediate coordinate system with respect to a global system. A **CombinedTraject** can be used anywhere that a **Traject** is required, but its primary application is internal, in the implementation of SST's target models.

Internally, orientations and rotations are represented using *quaternions* [Dean 1966], also known as *Cayley-Klein parameters* [Goldstein 1950]. This representation is chosen for its combination of compactness with efficient support of coordinate transformations, composition of rotations, interpolation, and extrapolation.

7.2 Beam Patterns

Each **Sonar** and each **Source** contains an attribute called *beams*, which consists of a user-specified list of objects belonging to classes derived from the base class **Beam**, one object per channel in the associated *signal*. Each **Beam** object provides member functions that compute the following quantities:

- $B_c(f, \boldsymbol{S}')$: the directional sensitivity pattern used in Eq. (45) (or the corresponding one for source beams), given the frequency f and the slowness vector $\boldsymbol{S}'$ in platform coordinates. For element-level simulations, this is the sensitivity pattern of one element, as modified by the physical supports and baffles surrounding it and by any preamplifiers or filters between the element and the injection point chosen for the simulation. For beam-level simulations, this is the effective sensitivity pattern of the array plus surrounding hardware, as modified by any signal processing steps from the array through the beamformer.

- $\boldsymbol{r}'_c$: the channel offset. This is the location, in platform coordinates, of the phase center of channel c. For element-level simulations, this is the location of a transducer relative to the array center. For beam-level simulations, it is often zero, or sometimes the center of a sub-array used to form an "offset phase center" beam.

- $\tau_p(\boldsymbol{S}', \boldsymbol{r}'_c)$: the offset delay used in Eq. (34). The expression used is essentially Eq. (33), using the channel offset $\boldsymbol{r}'_c$ in place of $\boldsymbol{\rho}_R$ and with inputs in platform coordinates.

Subscript c stands for the receiver channel r or the source channel s.

The various subclasses of class **Beam** differ from one another in the algorithm used to compute $B_c(f, \boldsymbol{S}')$. They are listed in Table 7.

The first section in the table contains simple, self-contained beam patterns, most of which are based on equations in Chapter 3 of [Urick 1983]. For example, **StickBeam**, **PistonBeam**, and **LineBeam** come from the first three rows of Urick's Table 3.2 (second column, omitting the squaring operation). Each beam pattern object is constructed using input parameters that differ from class to class. For example, **PistonBeam** requires a piston diameter and axis direction, whereas **LineBeam** requires the number and spacing of the elements, axis direction, and steering delay.

The classes in the second section in the table accept tables of numbers; $B_c(f, \boldsymbol{S}')$ is computed by interpolation. **EFIntensityBeam** and **SIOBeam** optionally accept another beam pattern as input, in which case the table is computed by sampling the input pattern; the result may be used in this or subsequent runs to speed up the simulation.

The classes in the third section of the table are transformations that accept one or more input beam patterns and transform them in some way.

All beam patterns accept, in addition to their class-specific input parameters, an offset vector $\boldsymbol{r}'_c$ and an additional delay to be added to the offset delay $\tau_p(\boldsymbol{S}', \boldsymbol{r}'_c)$.

Table 7: Beam Pattern Models

Class	*Summary*
Beam	Abstract base class
OmniBeam	Omnidirectional (1.0 everywhere)
BinomialBeam	Binomial weighted, steered line array
ConeBeam	1.0 inside a cone, 0 outside
DCLineBeam	Dolph-Chebyshev weighted line array [Albers 1965]
DipoleBeam	Cosine shape, as for a vector sensor
ElementSumBeam	Sum of weighted elements beam pattern
LineBeam	Uniformly weighted, steered line array
PistonBeam	Circular piston transducer
RecPistonBeam	Rectangular piston transducer
StickBeam	Continuous, uniform line transducer
EBFTableBeam	Interpolated from table vs. elevation, bearing, frequency
EFIntensityBeam	Intensity vs. sin(elevation) and frequency
SIOBeam	Table vs. elev, bear, freq in binary SIO file
DecibelBeam	Beam pattern transformed from decibels to pressure ratio
ProductBeam	Product of input beam patterns
RotatedBeam	Beam pattern rotated with respect to the platform coordinates
SumBeam	Sum of element beam patterns
WeightedBeam	Beam pattern multiplied by a weight and phase-shifted for delay

7.3 Sonar Transformation

The sonar model has a more abstract interpretation: it represents a transformation whose output is the signal $y_r(t)$ in all channels r of receiver R, and whose input is the sound field in the water, $p_R(t, \boldsymbol{r})$, for $\boldsymbol{r}$ in the neighborhood of the sonar origin $\boldsymbol{r}_R$. Such a sound field is produced as the output of the eigenray transformation (Eq. (34)). The sonar transformation takes the following form (in the time domain and platform coordinates):

$$y_r(t) = \int\!\!\int b_r(\tau, \boldsymbol{\rho}'_r)\, p'_R(t-\tau, \boldsymbol{r}'_r + \boldsymbol{\rho}'_r)\, \mathrm{d}\tau\, \mathrm{d}\boldsymbol{\rho}'_r\,, \tag{44}$$

where $\boldsymbol{r}'_r$ is the channel offset vector provided by the beam pattern model for receiver channel r (Sec. 7.2). The time-space domain kernel $b_r(\tau, \boldsymbol{\rho}'_r)$ for receiver channel r is related to the receiver's beam pattern $B_r(f, \boldsymbol{S}')$ by a four-dimensional spatiotemporal Fourier transform:

$$b_r(\tau, \boldsymbol{\rho}'_r) = \int\!\!\int B_r(f, \boldsymbol{\nu}'/f)\, \mathrm{e}^{2\pi\mathrm{i}(f\tau + \boldsymbol{\nu}'\cdot\boldsymbol{\rho}'_r)}\, \mathrm{d}f\, \mathrm{d}\boldsymbol{\nu}'\,, \tag{45}$$

where $\boldsymbol{\nu}'$ is the wave number vector in vehicle coordinates. The spatial kernel $b_r(\tau, \boldsymbol{\rho}'_r)$ is nonzero over the region $\boldsymbol{\rho}'_r$ where the receiver is sensitive. For example, for an ideal piston beam the sensitive region is the disk at the face of the transducer, although in practice effects like baffling and shadowing tend to make it more complicated. SST starts from the frequency-direction form, $B_r(f, \boldsymbol{S}')$, where $\boldsymbol{S}' = \boldsymbol{\nu}'/f$ (beam patterns versus frequency and look direction).

The primed coordinates in Eqs. (44) and (45) are expressed in a platform-centered coordinate system whose origin is at $\boldsymbol{r}_R(t)$. Equation (43) defines their relationship to the unprimed, Earth-centered coordinates used in the eigenray transformation (Eq. (34)).

7.4 Source Transformation

The source model has a similarly abstract interpretation: it represents a transformation whose input is the signal $x_s(t)$ emitted through each channel s of the source S, and whose output is the sound source field $q_S(t, \boldsymbol{r}_S + \boldsymbol{\rho}_S)$ in the neighborhood of $\boldsymbol{r}_S$. This sound source field is the input of the eigenray transformation (Eq. (34)). The source transformation takes the following form (in the time domain):

$$q_S(t, \boldsymbol{r}_S + \boldsymbol{\rho}_S) = \sum_s \int b'_s(\tau, \boldsymbol{M}_S(t)\, \boldsymbol{\rho}_S - \boldsymbol{r}'_s)\, x_s(t-\tau)\, \mathrm{d}\tau\,, \tag{46}$$

where $\boldsymbol{r}'_s$ is the channel offset vector provided by the beam pattern model for source channel s (Sec. 7.2). The time-space domain kernel $b'_s(\tau, \boldsymbol{\rho}'_s)$ for source channel s is related to the source's beam pattern $B_s(f, \boldsymbol{S}')$ as per the receiver (Eq. (45)) except for the sign in the relationship $\boldsymbol{S}' = -\boldsymbol{\nu}'/f$. The relations between platform-centered (primed) and Earth-centered (unprimed) coordinates are given by Eq. (43).

8 Direct Sound Propagation Models

The "direct" sound propagation model is a transformation whose input is the sound $x_s(t)$ emitted by all channels s of a given source, and whose output is that portion of the received sound $y_r(t)$ that does not scatter from objects or irregularities on its way to receiver channel r. Conceptually, it consists of the successive application of the source model, the eigenray model, and the receiver model, whose time-domain expressions are given by Eqs. (46), (34), and (44), respectively. If we combine those equations in series, the spatial integrations reduce to Dirac delta functions and drop out. The remaining operations can be arranged into successive signal transformations, as follows:

$$\begin{aligned}
x_{sp}(t) &= x_s\left(t - T_p - \tau_p(\boldsymbol{M}_S \boldsymbol{S}_{Sp}, \boldsymbol{r}'_s)\right) && (47)\\
x_p(t) &= \sum_s \int \left[\int \mathrm{e}^{2\pi \mathrm{i} f_S \tau'_S}\, B_s(f_S, \boldsymbol{M}_S \boldsymbol{S}_{Sp}) \mathrm{d}f_S\right] x_{sp}(t - \tau'_S)\, \mathrm{d}\tau'_S && (48)\\
y_p(t) &= \int \left[\int \mathrm{e}^{2\pi \mathrm{i} f \tau}\, L_p(f) \mathrm{d}f\right] x_p(t - \tau)\, \mathrm{d}\tau && (49)\\
x_{rp}(t) &= \int \left[\int \mathrm{e}^{2\pi \mathrm{i} f_R \tau'_R}\, B_r(f_R, \boldsymbol{M}_R \boldsymbol{S}_{Rp}) \mathrm{d}f_R\right] y_p(t - \tau'_R)\, \mathrm{d}\tau'_R && (50)\\
y_{rp}(t) &= x_{rp}\left(t - \tau_p(\boldsymbol{M}_R \boldsymbol{S}_{Rp}, \boldsymbol{r}'_r)\right) && (51)\\
y_r(t) &= \sum_p y_{rp}(t). && (52)
\end{aligned}$$

Thus, the transformation of a source channel signal $x_s(t)$ to a receiver channel signal $y_r(t)$ involves three filters [source beam pattern $B_s(f, \boldsymbol{S}'_{Sp})$, eigenray loss $L_p(f)$, and receiver beam pattern $B_r(f, \boldsymbol{S}'_{Rp})$] plus three delays [source channel offset $\tau_p(\boldsymbol{S}'_{Sp}, \boldsymbol{r}'_s)$, eigenray T_p, and receiver channel offset $\tau_p(\boldsymbol{S}'_{Rp}, \boldsymbol{r}'_r)$], plus sums over eigenrays and source channels. The offset delays are given by Eq. (33). The three inverse Fourier transforms (in square brackets) compute the time-domain impulse responses for the filters, and the three time integrations convolve those impulse responses with the signal.

Note that all of the filters and delays depend parametrically on time t because the trajectory attributes $\boldsymbol{r}_R$, $\boldsymbol{r}_S$, $\boldsymbol{M}_R$, and $\boldsymbol{M}_S$ depend on time (the sonar and source can move), and the eigenray attributes $\boldsymbol{S}_{Sp}$, T_p, $L_p(f)$, and $\boldsymbol{S}_{Rp}$ depend on the eigenray's end points $\boldsymbol{r}_R$ and $\boldsymbol{r}_S$.

8.1 DirectSignal

Equations (47) through (51) represent "ideal" filters and delays. Their implementation by SST class **DirectSignal** looks very much like those equations, except that the convolutions and interpolations are forced to have finite length. For each path p, **DirectSignal** sets up a chain of five "data flow" objects (Sec. 4.3) in series: two **VarDelay** objects (Sec. 4.5.1) implementing Eqs. (47) and (51), sandwiching three **VarFirFilter** objects (Sec. 4.5.2) implementing Eqs. (48), (49), and (50). The chains p feed into a **SumSignal** object implementing Eq. (52). The **VarDelay** and **VarFirFilter** objects are fed by lower-volume internal data flow objects carrying the time-varying delays and filter responses from the eigenray model and the source and receiver models.

This entire network of interconnecting objects is set up by the *openRead* operation at the start of the **CopySignal** operation. In the **CopySignal** main loop,each *readBlock* operation activates all of the objects in that network as required to compute that block. At the end of the **CopySignal** operation, the *close* operation of **DirectSignal** shuts down and destroys the network of data flow objects.

In the description so far, the signal computed by a **DirectSignal** is deterministic; i.e., **DirectSignal** uses no random numbers. In real life, sound that reflects from a rough boundary is randomly spread in time, in arrival angle, and sometimes in frequency. If the *doTimeSpread* attribute of the **EigenrayModel** is *true* (which it is by default), **DirectSignal** adds some random spreading in time and frequency. This option was introduced briefly in Sec. 5.3, and it is described more fully in Sec. 12.

8.2 DirectSpectrum

As the signal bandwidth increases, the lengths of the required FIR filters may increase to the point where it becomes advantageous to do all three filters (Eqs. (48) through (50)) in the frequency domain. To do this, SST offers class **DirectSpectrum** as an alternative to **DirectSignal**. **DirectSpectrum** produces its results in the windowed frequency domain form as defined by Eq. (7), as follows:

$$\begin{aligned} Y_r(f, t_u) &= \sum_p B_r(f, \boldsymbol{M}_R \boldsymbol{S}_{Rp})\, L_p(f) \sum_s B_s(f, \boldsymbol{M}_S \boldsymbol{S}_{Sp}) \\ &\quad \times X_s(f, t_u - T_p - \tau_p(\boldsymbol{M}_S \boldsymbol{S}_{Sp}, \boldsymbol{r}'_s))\,. \end{aligned} \tag{53}$$

The main approximation used to go from Eqs. (47) through (52) to Eq. (53) is the following:

$$w(t - t_u + \tau'_R + \tau + \tau'_S) \approx w(t - t_u)\,, \tag{54}$$

where $w(t-t_u)$ is the smooth window function used in Eq. (7). To satisfy this requirement, we must choose the window function to be long compared to the FIR filters that result from the Fourier transforms of $B_r(f, \boldsymbol{S}'_{Rp})$ in Eq. (50), $L_p(f)$ in Eq. (49), and $B_s(f, \boldsymbol{S}'_{Sp})$ in Eq. (48). Equivalently, we assume that the Fourier transform of the window function $w(t - t_u)$ is narrow in frequency, compared to the frequency scale on which $B_r(f, \boldsymbol{S}'_{Rp})$, $L_p(f)$, and $B_s(f, \boldsymbol{S}'_{Sp})$ vary significantly. Since we are free to choose the length of the window, we can always make that a good approximation. The window length is twice the increment between values of the update time t_u.

In addition, in using only one frequency f throughout Eq. (53), we are assuming that Doppler shifts due to motion of the source and receiver do not significantly change the values of the beam patterns or propagation loss. Like the previous approximation, this one rests on the smoothness of those factors in frequency. More about Doppler shifts is presented in the next subsection.

In **DirectSpectrum**, for each eigenray p the initial delay, which includes the source channel offset delays $\tau_p(\boldsymbol{S}'_{Sp}, \boldsymbol{r}'_s)$ and the eigenray delay T_p, is done in the time domain using class **VarDelay**, just as it is in **DirectSignal**. Following that delay **DirectSpectrum** transforms the signal into the windowed frequency domain form (class **SpectrumFromSignal**, Eq. (7)). The three successive filters (source beams, eigenray, and receiver beams) are done in the frequency domain, and the result remains in the frequency domain. For cases where the filters are long, this saves the time needed to transform back and forth between time and frequency domains between filters.

Unfortunately, SST does not (yet) have a frequency-domain delay operation, so **DirectSpectrum** cannot be used if the receiver channels have offsets. That necessitates the second approximation in Eq. (53): $\tau_p(\boldsymbol{S}'_{Rp}, \boldsymbol{r}'_r) = 0$ in Eq. (51). Adding small, slowly-varying delays (e.g., $\tau_p(\boldsymbol{S}'_{Rp}, \boldsymbol{r}'_r)$ or $\tau_p(\boldsymbol{S}'_{Sp}, \boldsymbol{r}'_s)$) in the frequency domain is not especially difficult; it just has not yet been implemented. Adding large or rapidly varying delays (e.g., to model the eigenray delay T_p) in the frequency domain is much trickier, and probably not worthwhile.

DirectSpectrum was used heavily in a passive-sonar application several years ago, and (to my knowledge) has not been used since.

8.3 Simple Direct Model

This subsection describes a simpler but less powerful way to model the sound transformation due to one-way propagation. This is *not* the algorithm used in **DirectSignal** or any other SST class. However, it is the starting point for the simple scattering model of Sec. 9.3, which is itself the starting point for the discussion of reverberation in Sec. 10. This version is most appropriate in cases where the source signal $x_s(t')$ is relatively short.

We start by modeling one-way propagation in the frequency domain using a modified form of the **DirectSpectrum** equation, Eq. (53):

$$\begin{aligned} Y_r(f, t_u) \quad = \quad & \sum_p \mathrm{e}^{-2\pi \mathrm{i} f \tau_p(\boldsymbol{M}_R \boldsymbol{S}_{Rp}, \boldsymbol{r}'_r)} B_r(f, \boldsymbol{M}_R \boldsymbol{S}_{Rp}) \, L_p(f) \\ & \times \sum_s B_s(f, \boldsymbol{M}_S \boldsymbol{S}_{Sp}) \, \mathrm{e}^{-2\pi \mathrm{i} f \tau_p(\boldsymbol{M}_S \boldsymbol{S}_{Sp}, \boldsymbol{r}'_s)} X_s(f, t_u - T_p)) \,, \end{aligned} \tag{55}$$

where we have pulled the offset delays $\tau_p(\boldsymbol{M}_R \boldsymbol{S}_{Rp}, \boldsymbol{r}'_r)$ and $\tau_p(\boldsymbol{M}_S \boldsymbol{S}_{Sp}, \boldsymbol{r}'_s)$ into the frequency domain as phase shifts.

When we transform this back to the time domain (Eq. (9)), the result is

$$y_r(t) = \sum_p \sum_s \int \mathrm{d}\tau \, h_{rps}(\boldsymbol{R}_R, \boldsymbol{R}_S, \tau) \, x_s(t - T_p(\boldsymbol{R}_R, \boldsymbol{R}_S) - \tau) \, , \tag{56}$$

where

$$h_{rps}(\boldsymbol{R}_R, \boldsymbol{R}_S, \tau) = w(\tau) \int \mathrm{d}f \, H_{rps}(\boldsymbol{R}_R, \boldsymbol{R}_S, f) \, \mathrm{e}^{\mathrm{i}2\pi f\tau} \, , \tag{57}$$

where $w(\tau)$ is a window function (Sec. 4.7) and

$$\begin{aligned} H_{rps}(\boldsymbol{R}_R, \boldsymbol{R}_S, f) \;\; = \;\; & \mathrm{e}^{-2\pi\mathrm{i}f\tau_p(\boldsymbol{M}_R\boldsymbol{S}_{Rp}, \boldsymbol{r}'_r)} \, B_r(f, \boldsymbol{M}_R\boldsymbol{S}_{Rp}) \\ & \times L_p(f) \, B_s(f, \boldsymbol{M}_S\boldsymbol{S}_{Sp}) \, \mathrm{e}^{-2\pi\mathrm{i}f\tau_p(\boldsymbol{M}_S\boldsymbol{S}_{Sp}, \boldsymbol{r}'_s)} \, . \end{aligned} \tag{58}$$

Note that the eigenray attributes T_p, $L_p(f)$, $\boldsymbol{S}_{Sp}$, and $\boldsymbol{S}_{Rp}$ depend implicitly on the locations of the source $\boldsymbol{R}_S$ and of the receiver $\boldsymbol{R}_R$. Not also (and here's the main complication) that if the source and receiver are moving, the eigenray attributes depend on time. Furthermore, the time that is relevant for the source location is not the independent variable t but the earlier time $t - T_p$ when the sound left the source. Explicitly, to compute the received signal $y_r(t)$ at time t, we need to evaluate the source location $\boldsymbol{R}_S(t')$ at time t', where

$$t' = t - T_p(\boldsymbol{R}_R(t), \boldsymbol{R}_S(t')) \, , \tag{59}$$

where t' appears on both sides. Thus we have an implicit equation with no closed-form solution.

Let us assume, for this method, that we have solved this implicit equation iteratively once: For each eigenray p, we have found a pair of times t_0 and t'_0 such that

$$t'_0 = t_0 - T_p(\boldsymbol{R}_R(t_0), \boldsymbol{R}_S(t'_0)) \tag{60}$$

and then use a first-order expansion to evaluate the signal $y_r(t)$ for values of the time t near t_0.

In this simple version, we will ignore the problem in calculating $L_p(f)$, $\boldsymbol{S}_{Sp}$, and $\boldsymbol{S}_{Rp}$ by assuming that those eigenray attributes, evaluated at time t, do not differ significantly from the values computed by evaluating the source location at our base time t_0. However, we will be slightly more careful when evaluating the propagation delay T_p: We will use a first-order expansion to do that job.

To first order in $t - t_0$ and $t' - t'_0$, the end-point locations are given by

$$\begin{aligned} \boldsymbol{R}_R(t) \;\; &\approx \;\; \boldsymbol{R}_R(t_0) + (\boldsymbol{V}_R \cdot \boldsymbol{S}_{Rp})(t - t_0) \\ \boldsymbol{R}_S(t') \;\; &\approx \;\; \boldsymbol{R}_S(t'_0) + (\boldsymbol{V}_S \cdot \boldsymbol{S}_{Sp})(t' - t'_0) \, , \end{aligned} \tag{61}$$

where $\boldsymbol{V}_R$ and $\boldsymbol{V}_S$ are the velocities of the receiver and source, respectively. This expansion follows directly from the definition of the slowness vectors (Sec. 5.1) as the gradients of the time delay with respect to the two end point locations. If we substitute Eq. (61) into Eq. (59), subtract Eq. (60), and solve, we get

$$t' - t'_0 = \Gamma_p \left[t - t_0 - T_p(\boldsymbol{R}_R(t_0), \boldsymbol{R}_S(t'_0)) \right] , \tag{62}$$

where the *Doppler shift* Γ_p is given by

$$\Gamma_p = 1 - \frac{\mathrm{d}T_p}{\mathrm{d}t} = \frac{1 - (\boldsymbol{V}_R \cdot \boldsymbol{S}_{Rp})}{1 + (\boldsymbol{V}_S \cdot \boldsymbol{S}_{Sp})} . \tag{63}$$

Traditionally, the first-order expansion is done relative to a time base at or near the start of the transmit signal, and that point is chosen as the time origin; i.e. $t'_0 = 0$. In that case, Eq. (56) simplifies to

$$y_r(t) = \sum_p \sum_s \int \mathrm{d}\tau \, h_{rps}(\tau) \, x_s(\Gamma_p[t - T_p - \tau]) , \tag{64}$$

where the location arguments of $h_{rps}(\tau)$, T_p, and Γ_p are now implicit, and are understood to be given by the "base" locations $\boldsymbol{R}_R(t_0)$ and $\boldsymbol{R}_S(0)$.

Thus, seen in the time domain, the Doppler shift is a *time compression* operation if $\Gamma_p > 1$. In the frequency domain, it is easily seen that the Doppler shift is also a *frequency expansion* operation: If the source spectrum contains a line at some frequency f_S, then the receiver will hear that line at frequency $f_R = \Gamma_p f_S$. This important result is the main motivation for the first-order expansion for the eigenray delay T_p.

9 Target Echo Model

9.1 TargetEcho

SST class **TargetEcho** is a data-flow class (Sec. 4.3) whose attributes include a **Source**, a **Sonar**, and a **Target**. The output represents sound that has been transmitted by the **Source**; received, altered, and re-transmitted by the **Target**; and finally received by the **Sonar**.

From **TargetEcho**'s point of view, a target consists of a group of receivers (**Sonar** objects) and a group of transmitters (**Source** objects) back to back, with a target-specific transformation that determines what happens to the received sound before it is re-transmitted.

Class **TargetEcho** is implemented using one or more **DirectSignal** objects to carry sound from the source to the target, and one or more **DirectSignal** objects to carry sound from the target to the receiver. The transformation from the sound received by the **Target** to the sound re-transmitted by the **Target** is determined by the **Target** subclass. The **TargetEcho** object sets up the network of **DirectSignal** objects at the start of a **CopySignal** operation, and tears it down at the end of that operation.

This algorithm is inherently *bistatic* — nowhere is it assumed that the source and the receiver are at the same location. The common *monostatic* case is modeled by assigning the same trajectory to both the source and the receiver. The code that SST executes is exactly the same for either monostatic or bistatic systems. In fact, the source need not be a conventional active transmitter, and the signal need not be a pulse. This flexibility enables SST to model echoes from a target of sound from any source, including a continuously emitting source. That situation is not what is conventionally thought of as a "target echo", but SST's **TargetEcho** class will handle it.

The sound from **TargetEcho**, like that from **DirectSignal**, can be randomly spread in time and frequency to model near-specular scattering from rough boundaries. Users control this option by setting the *doTimeSpread* attribute of the **EigenrayModel**. This option was introduced briefly in Sec. 5.3, and it is described more fully in Sec. 12.

9.2 Target Models

All of SST's target models are derived from the base class **Target**. Every **Target** contains a trajectory (Sec. 7.1). Each subclass of **Target** adds other attributes, which determine the relationship between the sound received by the target and the sound it re-transmits. All of the existing subclasses of **Target** express that relationship in terms of *highlights*, which are point scatterers that move as a group along the target's trajectory. However, the **Target** interface used by **TargetEcho** (summarized above) is general enough to support not only highlight-based target models but also artificial targets and active countermeasures.

The three existing target models differ in how the placement and properties of their highlights are determined.

9.2.1 PointTarget

Class **PointTarget** is SST's simplest target model. The user specifies a list of point scatterers, each of which is characterized by its scattering strength and its position and velocity relative to the target's local coordinate system. The signal transmitted from each highlight's location is simply a scaled copy of the signal received at that highlight.

The user can choose between *common center* processing and *multiple center* processing. In the *common center* case the **PointTarget** creates only one receiver and one transmitter located at the target's origin, each of which has multiple channels, one per highlight. This causes **TargetEcho** to create only two **DirectSignal** objects, with multiple channels at the target end. In the *multiple center* case the **PointTarget** creates a separate, single-channel receiver and transmitter for each highlight. This causes **TargetEcho** to create two separate **DirectSignal** objects for each highlight.

The *common center* option is always faster, but it introduces an additional approximation: the offsets from the target center to the individual highlight locations affect only the arrival times of the echoes, and not the directions from which they arrive. Therefore, the *multiple center* option should be chosen whenever the target is close enough for the sonar system to detect and use the target's cross-range extent (e.g., to compute "line-like" classification clues).

PointTarget also allows the user to specify a *randomPosition* value, which gives the root mean square value of a Gaussian random component to be added to the specified highlight locations. This can be used to break up unrealistic grating effects from regularly spaced highlights.

9.2.2 HighlightTarget

Class **HighlightTarget** improves on **PointTarget** in three ways:

- Each highlight may have a complex, frequency-dependent response.
- Each highlight may have a delay between reception and transmission.
- Each highlight is assigned a *group number*. All highlights with the same group number are treated as channels in a single receiver-transmitter pair. This provides the flexibility to model cross-range extent (e.g., by placing stern highlights in one group and bow highlights in another) without paying the performance price of treating each highlight as a separate target. Assigning each highlight a different group number is equivalent to setting *commonCenter* to *false* in **PointTarget**. Assigning the same group number to all highlights is equivalent to setting *commonCenter* to *true* in **PointTarget**.

9.2.3 ExternalTarget

Class **ExternalTarget** is functionally identical to **HighlightTarget** except for the source of its highlights. When you create an **ExternalTarget** object, it starts up an external program provided by the user. At the start of each ping (when the **TargetEcho** object is opened for reading), the **ExternalTarget** object sends to the external program the distance and direction to the active sonar's transmitter and receiver and the range of signal frequencies. The external program sends back a list of highlights. Once that exchange is complete, **ExternalTarget** behaves exactly like a **HighlightTarget**.

This information exchange occurs via a pair of pipes connected to the external program's standard input and standard output streams. The protocol involves text commands, and the data are in text form. Hence the external program can be tested in isolation, without SST. The SST distribution includes a Fortran skeleton to serve as a starting point, or the protocol can be implemented in the user's favorite language.

Some target models in use by the Navy take the form of subroutines that return lists of highlights. Several of these models have been wrapped for use with SST using class **ExternalTarget**.

9.3 Simple Target Echo Model

This subsection describes a simpler but less powerful way to model echoes from a point scatterer. This is *not* the algorithm used in **TargetEcho**, or any other SST class. However, it is the starting point for the discussion of reverberation in the following sections.

In the frequency domain, we view target scattering as two applications of Eq. (55) with a single scattering event (one highlight) between them. For the transmit leg, the "receiver" is the target, which has an omnidirectional "beam pattern" and an offset of zero. For the receive leg, the "transmitter" is the target, which is similarly simple. We model the scattering event at target highlight T using a complex scale factor $A_T(f, \boldsymbol{S}_o, \boldsymbol{S}_i)$ relating the sound emitted by the target to the sound received at the target. This amplitude can depend on frequency f, the slowness (direction) of the outgoing sound $\boldsymbol{S}_o$, and the slowness of the incoming sound $\boldsymbol{S}_i$. The spectrum of the target echo is thus

$$\begin{aligned} Y_{rT}(f,t_u) \;=\; & \sum_{pq} \mathrm{e}^{-2\pi \mathrm{i} f \tau_p(\boldsymbol{M}_R \boldsymbol{S}_{Rp}, \boldsymbol{r}'_r)}\, B_r(f, \boldsymbol{M}_R \boldsymbol{S}_{Rp})\, L_p(f) \\ & \times A_T(f, \boldsymbol{S}_{Tp}, \boldsymbol{S}_{Tq}) \\ & \times L_q(f) \sum_s B_s(f, \boldsymbol{M}_S \boldsymbol{S}_{Sq})\, \mathrm{e}^{-2\pi \mathrm{i} f \tau_q(\boldsymbol{M}_S \boldsymbol{S}_{Sq}, \boldsymbol{r}'_s)}\, X_s(f, t_u - T_p - T_q))\,, \end{aligned} \tag{65}$$

where the sum now extends over both receive-leg eigenrays p and transmit-leg eigenrays q. As usual, the eigenray attributes depend implicitly on the end-point locations – the locations of the source, target, and receiver.

The transformation into the time domain and the introduction of the Doppler shift parallel the development of Sec. 8.3. The result is

$$y_{rT}(t) = \sum_{pq} \sum_s \int \mathrm{d}\tau\, h_{rpTqs}(\tau)\, x_s(\Gamma_{pTq}[t - T_p - T_q - \tau])\,, \tag{66}$$

where $h_{rpTqs}(\tau)$, T_p, T_q, and Γ_{pTq} are evaluated at the three "base" locations $\boldsymbol{R}_R(t_0)$ (receiver), $\boldsymbol{R}_S(t'_0)$ (source), and $\boldsymbol{R}_T(t''_0)$ (target). Those base locations and times are related by

$$\begin{aligned} t''_0 &= t_0 - T_p(\boldsymbol{R}_R(t_0), \boldsymbol{R}_T(t''_0)) \\ t'_0 &= t''_0 - T_q(\boldsymbol{R}_T(t''_0, \boldsymbol{R}_S(t'_0)) \end{aligned} \tag{67}$$

and conventionally $t'_0 = 0$ at the start of the short source signal. The Doppler shift includes contributions from motion of the target, as well as of the source and receiver:

$$\Gamma_{pTq} = 1 - \frac{\mathrm{d}(T_p + T_q)}{\mathrm{d}t} = \left[\frac{1 - (\boldsymbol{V}_R \cdot \boldsymbol{S}_{Rp})}{1 + (\boldsymbol{V}_S \cdot \boldsymbol{S}_{Sq})}\right] \left[\frac{1 - (\boldsymbol{V}_T \cdot \boldsymbol{S}_{Tq})}{1 + (\boldsymbol{V}_T \cdot \boldsymbol{S}_{Tp})}\right]\,. \tag{68}$$

The target echo impulse response is given by

$$h_{rpTqs}(\boldsymbol{R}_R, \boldsymbol{R}_S, \boldsymbol{R}_T, \tau) = w(\tau) \int \mathrm{d}f\, H_{rpTqs}(\boldsymbol{R}_R, \boldsymbol{R}_S, \boldsymbol{R}_T, f)\, \mathrm{e}^{\mathrm{i}2\pi f \tau} \tag{69}$$

and the transfer function is given by

$$\begin{aligned} H_{rpTqs}(\boldsymbol{R}_R, \boldsymbol{R}_S, \boldsymbol{R}_T, f) \;=\; & \mathrm{e}^{-2\pi \mathrm{i} f \tau_p(\boldsymbol{M}_R \boldsymbol{S}_{Rp}, \boldsymbol{r}'_r)}\, B_r(f, \boldsymbol{M}_R \boldsymbol{S}_{Rp})\, L_p(f) \\ & \times A_T(f, \boldsymbol{S}_{Tp}, \boldsymbol{S}_{Tq}) \\ & \times L_q(f) \sum_s B_s(f, \boldsymbol{M}_S \boldsymbol{S}_{Sq})\, \mathrm{e}^{-2\pi \mathrm{i} f \tau_q(\boldsymbol{M}_S \boldsymbol{S}_{Sq}, \boldsymbol{r}'_s)}\,. \end{aligned} \tag{70}$$

This model is conceptually simpler than the one used by **TargetEcho**, but that simplicity is only skin deep. Note, in particular, that it involves a separate impulse response $h_{rpqs}(\tau)$ (and hence a separate filter operation) for each combination of receiver channel r, receive-leg eigenray p, transmit-leg eigenray q, and source channel s. That's a poor scaling law $(N_r N_p N_q N_s)$ compared to the operation chains used by **TargetEcho**, which scale as $(N_r N_p + N_q N_s)$, more or less. (It's not quite that clear-cut because the signals in the receive leg are a bit longer for **TargetEcho**, and because the various filters can have different lengths, but **TargetEcho** eventually wins as those numbers get bigger.) Also, the linear Doppler approximation used by the simple model is appropriate only for short signals, whereas **TargetEcho** can be used for very long signals too. Also, the target models supported by **TargetEcho** are more general than the simple point scatterer model of the simple model. There are undoubtedly cases for which the simple model would be faster, but those cases are the simple ones (few eigenrays, few channels, simple targets, short signals) where **TargetEcho** is quite fast anyway.

The main reason for including the simple model here is that it forms the starting point for our discussion of reverberation, which follows.

10 Reverberation Theory

SST offers several different methods for generating reverberation. To help you understand where they come from, which approximations are inherent in each method, and how and why they differ from one another, we start with the basic theory and assumptions that are common to all of them.

Reverberation can be thought of as the sum of echoes from a very large number of discrete scatterers, each of which sends back a delayed, filtered, Doppler-shifted copy of the source signal. Other descriptions are also valid, and mostly equivalent. Reverberation scatterers are on the surface (waves and bubbles), the bottom (roughness and embedded inhomogeneities), and the ocean volume (mostly marine life). It is useful to start from a model of reverberation as a sum of a very large number of target echoes, each of which has the form of Eqs. (66) to (70), summed over targets T.

10.1 Assumptions

The physical assumptions are the same as those used for target echoes: linearity, eigenray propagation, single scattering, and restriction to the far field of the transmitter and receiver.

A few more assumptions are added for reverberation: the scatterers that contribute to reverberation are randomly distributed, and their density is so high that the sonar system cannot resolve them as individual scatterers. Moreover, each cell in space, down to a scale smaller than the sonar can resolve, is assumed to contain scatterers with a distribution of Doppler shifts due to scatterer motion. (Of course, the bottom is stationary, so the corresponding distribution of scatterer Doppler is a delta function at 1.0.)

10.2 Local Impulse Response

We start with Eq. (66), summed over scatterers (targets) T, and take it to the limit in which each scatterer represents an infinitesimally small region of the ocean. Each of these notional scatterers has a random amplitude and a random Doppler due to scatterer motion. For this purpose, we split the total Doppler shift of Eq. (68) into the product $\Gamma_{pq}\gamma$, where

$$\Gamma_{pq} = \frac{1-(\boldsymbol{V}_R\cdot\boldsymbol{S}_{Rp})}{1+(\boldsymbol{V}_S\cdot\boldsymbol{S}_{Sq})} \tag{71}$$

is the *platform Doppler* due to motion of the source and receiver, and

$$\gamma = \frac{1-(\boldsymbol{V}_T\cdot\boldsymbol{S}_{Tq})}{1+(\boldsymbol{V}_T\cdot\boldsymbol{S}_{Tp})} \tag{72}$$

is the *scatterer Doppler* due to motion of the scatterer itself. The platform Doppler Γ_{pq} is geometric in nature, depending on the location of a scattering patch but not on its properties. The scatterer Doppler γ is treated as an independent variable; any scattering patch will contain scatterers having a distribution of values of γ, with random amplitudes.

In this limit, the sum over discrete scatterers T becomes an integration over both scatterer location $\boldsymbol{R}$ and scatterer Doppler γ. In the integrand, the multiplier of the source signal is a random function of both of those variables:

$$y_r(t) = \int \mathrm{d}^3\boldsymbol{R} \sum_{pqs} \int \mathrm{d}\gamma \int \mathrm{d}\tau\, h_{rpqs}(\gamma,\boldsymbol{R},\tau)\, x_s(\gamma\Gamma_{pq}(t-T_p-T_q-\tau))\,, \tag{73}$$

where the sums over p, q, and s are over the receive-leg and transmit-leg eigenrays and sound sources (transmitters), respectively.

The convolution over τ is a filtering operation that accounts for the frequency dependence of all of the underlying processes. We saw in Sec. 4.5.2 (Eq. (22)) how to get a time-domain filter impulse response from its frequency-dependent transfer function:

$$h_{rpqs}(\gamma, \boldsymbol{R}, \tau) = w(\tau) \int \mathrm{d}f \, H_{rpqs}(\gamma, \boldsymbol{R}, f) \, \mathrm{e}^{\mathrm{i}2\pi f \tau} \,, \tag{74}$$

where $w(\tau)$ is a window function (Sec. 4.7).

The *local Doppler impulse response* $h_{rpqs}(\gamma, \boldsymbol{R}, \tau)$ and its Fourier transform, the *local Doppler transfer function* $H_{rpqs}(\gamma, \boldsymbol{R}, f)$, are stochastic (random) functions of scattering location and Doppler. The ingredients that go into them are familiar from the target echo model, Eq. (70):

$$\begin{aligned} H_{rpqs}(\gamma, \boldsymbol{R}, f) \;&=\; \mathrm{e}^{-\mathrm{i}2\pi f \boldsymbol{S}'_{Rp}\cdot \boldsymbol{r}_r} \, B_r(f, \boldsymbol{S}'_{Rp}) \, L_p(f) \\ &\quad \times A_{pq}(\gamma, \boldsymbol{R}, f) \, L_q(f) \, B_s(f, \boldsymbol{S}'_{Sq}) \, \mathrm{e}^{-\mathrm{i}2\pi f \boldsymbol{S}'_{Sq}\cdot \boldsymbol{r}_s} \,, \end{aligned} \tag{75}$$

where $B_r(f, \boldsymbol{S}'_{Rp})$ and $B_s(f, \boldsymbol{S}'_{Sq})$ are the beam patterns (or element patterns) for receiver channel r and source channel s, respectively; $L_p(f)$ and $L_q(f)$ are the propagation losses along receive-leg eigenray p and transmit-leg eigenray q, respectively; and the phase factors at the ends are delays due to offsets of the channels (or elements) from the centers of the receive and source arrays, $\boldsymbol{r}_r$ and $\boldsymbol{r}_s$, respectively. The eigenray attributes T_p, L_p, $\boldsymbol{S}_{Sp}$, and $\boldsymbol{S}_{Rp}$, which are defined in Sec. 5, depend implicitly on the scatterer location $\boldsymbol{R}$ as well as on the locations of the receiver and the source. The primes on $\boldsymbol{S}'_{Rp}$ and $\boldsymbol{S}'_{Sq}$ indicate that they have been transformed into vehicle-based coordinates.

10.3 The Poisson Scattering Assumption

The scattering amplitude $A_{pq}(\gamma, \boldsymbol{R}, f)$, in the middle of Eq. (75), is a strange beast. It is a *stochastic*, or random, function with an essentially *fractal* behavior with respect to its arguments γ and $\boldsymbol{R}$. This idea is expressed mathematically using a strong and separable form of the *Poisson scattering assumption*:

$$\begin{aligned} \mathbf{E}\left\{A_{pq}(\gamma, \boldsymbol{R}, f) \, A^*_{p'q'}(\gamma', \boldsymbol{R}', f')\right\} \;&=\; \delta_{pp'} \, \delta_{qq'} \, \delta(\gamma - \gamma') \, \delta(\boldsymbol{R} - \boldsymbol{R}') \\ &\quad \times S(\gamma, \boldsymbol{R}, \boldsymbol{S}_{Sp}, \boldsymbol{S}_{Rq}, f) \, S^*(\gamma, \boldsymbol{R}, \boldsymbol{S}_{Sp}, \boldsymbol{S}_{Rq}, f') \,, \end{aligned} \tag{76}$$

where $\mathbf{E}\{\cdot\}$ denotes the statistical expectation value, and $|S(\gamma, \boldsymbol{R}, \boldsymbol{S}_{Sp}, \boldsymbol{S}_{Rq}, f)|^2$ is the scattering differential cross section (an area) per unit volume ($\mathrm{d}^3\boldsymbol{R}$) per unit scatterer Doppler shift ($\mathrm{d}\gamma$). When integrated over Doppler, it is the non-decibel form of the volume scattering strength.

The Dirac delta function $\delta(\boldsymbol{R} - \boldsymbol{R}')$ expresses the usual form of the Poisson assumption, which postulates that the random scatterers are uncorrelated with one another no matter how close they are in scatterer location $\boldsymbol{R}$. As a practical matter, it really means that the correlation distance is assumed smaller than the range resolution of the sonar system. The other Dirac delta function, $\delta(\gamma - \gamma')$, extends the Poisson assumption by postulating that scatterers at the same location with different Doppler shifts are uncorrelated with one another. The two Kronecker deltas, $\delta_{pp'}\,\delta_{qq'}$, express the separate assumption that the eigenrays are far enough apart that the signals arriving along any two different eigenray pairs are uncorrelated with one another – or at least that the sonar system is insensitive to any such correlation if it exists. This last assumption is a high-frequency approximation, and will need to be discarded at lower frequencies where Lloyd's Mirror effects in reverberation can be observed as striations in the range-Doppler map.

The scattering strength is assumed to be *separable*. That means the dependence of Eq. (76) on the frequencies f and f' takes the form of the product shown on the second line. This assumption is added because it simplifies a lot of the math that follows. The underlying observation is that the frequency dependence of the scattering strength is weak, which follows from the assumption that each individual scatterer is small – which has the consequence that the precise form of the dependence doesn't matter much, so we might as well make it separable. Further justification will be given in Sec. 10.7. The phase of $S(\gamma, \boldsymbol{R}, \boldsymbol{S}_{Sp}, \boldsymbol{S}_{Rq}, f)$ doesn't matter as long as it varies slowly with frequency. Hence we will assume it is real; it is just the square root of the scattering strength per unit Doppler.

One can use this formalism to describe surface and bottom scattering, as well as volume scattering, by inserting a Dirac delta function $\delta(z - Z_L)$ to select out the depth Z_L of a particular scattering layer L. The integration over depth (the Z component of $\boldsymbol{R}$) then becomes a sum over L. In fact, SST uses that trick to treat each volume scattering layer as if all of the scatterers were concentrated in a thin sheet at the center of the layer. This is a reasonable approximation at most ranges, but it can be inaccurate early in the ping (especially for bistatic sonars) because, for each volume layer and each two-way path to that layer, the volume reverberation starts up too suddenly, too strongly, and late. This ought to be fixed.

10.4 Gaussian Statistics

In many situations, the number of scatterers in each resolution cell of the sonar is high enough to reach the Gaussian limit. The consequence, in those cases, is that *reverberation has Gaussian statistics.*

Certain tactically important phenomena that are conventionally regarded as reverberation, most notably rock outcrops, often do not satisfy that assumption. For now this problem is defined away by saying, "Reverberation is Gaussian, and echoes that are not Gaussian are target echoes." If false targets are tactically important, one recourse is to model them explicitly as targets. A new method that relaxes the Gaussian limitation will be described in a separate document.

The assumption of Gaussian statistics has a corollary: If we can compute all of the relevant second-order statistics (second moments) of a Gaussian process, we know everything there is to know about that process. Given those second moments, there are well-defined (if not always efficient) ways to generate any number of realizations of the process. To put it another way, if we can invent a method of generating reverberation having a Gaussian distribution, and if we show that all of the relevant second-order moments of the distribution are correct, that constitutes a demonstration that the generation method is correct. We will take advantage of this property repeatedly in the following discussion.

10.5 Scattering Field Generation

In principle, we can immediately write down a Gaussian scattering field that satisfies the separable Poisson property, Eq. (76):

$$A_{pq}(\gamma, \boldsymbol{R}, f) = S(\gamma, \boldsymbol{R}, \boldsymbol{S}_{Sp}, \boldsymbol{S}_{Rq}, f)\, g_{pq}(\gamma, \boldsymbol{R})\,, \tag{77}$$

where $g_{pq}(\gamma, \boldsymbol{R})$ is a complex Gaussian process satisfying

$$\mathbf{E}\left\{g_{pq}(\gamma, \boldsymbol{R})\, g^*_{p'q'}(\gamma', \boldsymbol{R}')\right\} = \delta_{pp'}\, \delta_{qq'}\, \delta(\gamma - \gamma')\, \delta(\boldsymbol{R} - \boldsymbol{R}')\,. \tag{78}$$

It is easy to verify that Eq. (77) satisfies Eq. (76). To generate reverberation, we substitute $A_{pq}(\gamma, \boldsymbol{R}, f)$ from Eq. (77) into the basic generation equations, Eqs. (73) through (75).

It is important to note that it is the assumption of separability of the scattering strength that makes this possible. We will say more about separability in Sec. 10.7.

Of course, we have just pushed the problem down a level: We don't have a "random field generator" that satisfies Eq. (78). The way out of that is to work in terms of integrals of Eq. (77) over small, disjoint regions $(\Delta\gamma)_j\, (\Delta\boldsymbol{R})_i$ that are small enough that the multiplier of the random field $g_{pq}(\gamma, \boldsymbol{R})$ is essentially constant within each region:

$$G_{pq}(\gamma_j, \boldsymbol{R}_i) = \int_{(\Delta\gamma)_j} \mathrm{d}\gamma \int_{(\Delta\boldsymbol{R})_i} \mathrm{d}^3\boldsymbol{R}\, g_{pq}(\gamma, \boldsymbol{R})\,, \tag{79}$$

which satisfies

$$\mathbf{E}\left\{G_{pq}(\gamma_j, \boldsymbol{R}_i)\, G^*_{p'q'}(\gamma_{j'}, \boldsymbol{R}_{i'})\right\} = \delta_{pp'}\, \delta_{qq'}\, \delta_{jj'}\, \delta_{ii'}\, (\Delta\gamma)_j\, (\Delta\boldsymbol{R})_i\,, \tag{80}$$

which describes the output from an ordinary Gaussian random number generator, weighted by the size of the integration interval. This is the basis for the Point Scatterer approach, which we will examine in Sec. 11.1.

Before we dive into specific algorithms for generating reverberation, we will follow some consequences of the theory so far.

10.6 Spreading and Coherence

An important set of properties of reverberation is that reverberation has relatively low coherence in frequency, time, and direction, relative to non-reverberation signals such as target echoes. The low coherence in each of those dimensions is a consequence of random spreading of the signal in the three "Fourier complements" of those dimensions, namely time, Doppler shift, and spatial separation, respectively. In this subsection we will explain these assertions by working through some very simple "toy" cases that illustrate what they mean and why they arise.

10.6.1 Time Spread vs. Frequency Coherence

For the first simple case, we reduce the reverberation problem to a single dimension, which we choose to be the two-way propagation time T, and assume that the scattering strength is localized to a particular region in T and has a very specific shape – a Gaussian, versus T. Then Eq. (73) reduces to the following form:

$$y(t) = \int \mathrm{d}T\, h(T)\, x(t-T)\,, \tag{81}$$

where the Poisson scattering assumption (Eq. (76)) reduces to:

$$\mathbf{E}\left\{h(T)h^*(T')\right\} = \delta(T-T')\, U^2(T)\,. \tag{82}$$

We postulate that the "scattering strength" has a Gaussian form:

$$U^2(T) = \frac{U_0^2}{\sigma_T\sqrt{2\pi}}\, \mathrm{e}^{-\frac{(T-\bar{T})^2}{2\sigma_T^2}}\,, \tag{83}$$

where $\bar{T}$ is the mean and σ_T is the standard deviation of the distribution of scattering strength in T. With that normalization, U_0^2 is the total target strength of that lump of scatterers.

Now we look at the result in frequency space. The Fourier transform of Eq. (81) with respect to t reduces to

$$Y(f) = \int \mathrm{d}T \, h(T) \mathrm{e}^{\mathrm{i}2\pi fT} \, X(f) \, . \tag{84}$$

Now consider the autocovariance of the frequency-space result:

$$\mathbf{E}\left\{Y(f)Y^*(f')\right\} = \int \mathrm{d}T \int \mathrm{d}T' \, \mathbf{E}\left\{h(T)h^*(T')\right\} \, \mathrm{e}^{\mathrm{i}2\pi(fT - f'T')} \, X(f) \, X^*(f') \, . \tag{85}$$

If we plug in Eq. (82), the delta function eliminates the integration over T', and the remaining integration over T is recognizable as another Fourier transform:

$$\mathbf{E}\left\{Y(f)Y^*(f')\right\} = U_0^2 C(f - f') \, X(f) \, X^*(f') \, , \tag{86}$$

where

$$U_0^2 C(f - f') = \int \mathrm{d}T \, U^2(T) \mathrm{e}^{\mathrm{i}2\pi(f-f')T} \, . \tag{87}$$

But the Fourier transform of a Gaussian is another Gaussian:

$$C(f - f') = \mathrm{e}^{-\frac{(f-f')^2}{2\sigma_f^2}} \, , \tag{88}$$

where $C(f - f')$ is the *frequency coherence* of the signal, which in this case has an RMS width – the *coherence bandwidth* – of

$$\sigma_f = \frac{1}{2\pi\sigma_T} \, . \tag{89}$$

This is a contrived example, but it illustrates an important general principle: If random scattering produces a time spread with a particular width, then the coherence bandwidth of the result will be proportional to the *inverse* of that time spread width.

This has an important practical consequence: Many systems use *replica correlation* (or *matched filtering*) to detect target echoes in the presence of reverberation. In other words, the system computes the correlation of the received signal against a copy (or *replica*) of the transmit signal, where the latter is delayed and Doppler shifted by various amounts. This process produces a high value when a segment of the signal most closely resembles the replica. For broadband signals, a target echo has a relatively large frequency coherence width, which results in a large output from replica correlation, whereas reverberation's narrow frequency coherence implies that the output from replica correlation is low. As the bandwidth of the signal increases, the "gain" from replica correlation makes target echoes stand out increasingly from the reverberation. In the ideal situation (single direct path, simple target), that gain in signal to reverberation ratio increases by 3 dB per octave of increased pulse bandwidth.

The low coherence width of reverberation also makes simulation simpler. It is the reason why SST's Scattering Function algorithms (Sec. 11.3), which assume a vanishingly small coherence width, work well in regions where reverberation is smooth (which is almost everywhere). Of course, this principle also explains why those algorithms are inaccurate in the neighborhood of transient events like near-specular "startup" of surface or bottom reverberation, where they under-estimate the signal coherence.

10.6.2 Doppler Spread vs. Time Coherence

Our second simple case follows the same pattern. We reduce the reverberation problem to a single dimension, which we choose this time to be the Doppler shift γ, and assume that the scattering strength is localized to a particular region in γ and has a Gaussian shape versus γ. Then Eq. (73) reduces to the following form:

$$y(t) = \int \mathrm{d}\gamma\, h(\gamma)\, x(\gamma(t-T))\,, \tag{90}$$

where this time T is a constant (no time spreading). This time, we also postulate that the transmit signal is a long, pure tone with frequency f:

$$x(t') = x_0 \mathrm{e}^{\mathrm{i}2\pi f t'}\,. \tag{91}$$

The Poisson statistical assumption is

$$\mathbf{E}\left\{h(\gamma)h^*(\gamma')\right\} = \delta(\gamma-\gamma')\, U^2(\gamma)\,. \tag{92}$$

and we postulate a Gaussian shape:

$$U^2(\gamma) = \frac{|U_0|^2}{\sigma_\gamma\sqrt{2\pi}}\, \mathrm{e}^{-\frac{(\gamma-\bar{\gamma})^2}{2\sigma_\gamma^2}}\,, \tag{93}$$

where $\bar{\gamma}$ is the mean and σ_γ is the standard deviation of the distribution of scattering strength in γ. Again, $|U_0|^2$ is the total target strength of that lump of scatterers.

This time, we look at the autocovariance in time:

$$\begin{aligned}
\mathbf{E}\left\{y(t)y^*(t')\right\} &= |x_0|^2 \int \mathrm{d}\gamma \int \mathrm{d}\gamma'\, \mathbf{E}\left\{h(\gamma)h^*(\gamma')\right\}\, \mathrm{e}^{\mathrm{i}2\pi f[\gamma(t-T)-\gamma'(t'-T)]} \qquad (94)\\
&= |x_0|^2 \int \mathrm{d}\gamma\, U^2(\gamma)\, \mathrm{e}^{\mathrm{i}2\pi f\gamma(t-t')}\\
&= |U_0 x_0|^2\, C(t-t')\,,
\end{aligned}$$

where the *time coherence* of the signal is

$$C(t-t') = \mathrm{e}^{-\frac{(t-t')^2}{2\sigma_t^2}} \tag{95}$$

with an RMS width (or *coherence time*) of

$$\sigma_t = \frac{1}{2\pi f \sigma_\gamma}\,. \tag{96}$$

This example illustrates this general principle: If random scattering produces a Doppler spread with a particular width, then the coherence time of the result will be inversely proportional to the product of that Doppler spread width and the frequency. For a broadband signal, this isn't as clean as the previous example because the coherence time decreases with increasing frequency.

The practical consequence is that the gain in target-reverberation ratio expected from replica correlation improves with increasing pulse length in a manner similar to its improvement with bandwidth. This extra gain is available if the reverberation is spread in Doppler due to either scatterer motion (e.g., wave motion for surface reverberation) or platform motion. It is absent for stationary sonars in situations where bottom reverberation dominates.

The low coherence time of reverberation also makes simulation simpler. SST's Scattering Function algorithms (Sec. 11.3) assume that the coherence time is less than the "update interval" Δ chosen for the algorithm.

10.6.3 Direction Spread vs. Spatial Coherence

For our third simple case, the receiver consists of two hydrophones separated by some horizontal distance d along the x axis. Our signal, again, is a long tone (Eq. (91)). We again reduce the reverberation problem to a single dimension, which we choose this time to be the x component of the slowness vector (Sec. 5.1), which is related to the direction from which the sound is coming by:

$$S_x = -\sin(\theta)/c\,, \tag{97}$$

where θ is the angle from the $y-z$ plane (bisecting the line between the hydrophones) to the direction from which the sound is coming. Then Eq. (73) reduces to the following form:

$$y_\pm(t) = \int \mathrm{d}S_x\, h(S_x)\, x(t - T \mp S_x d/2))\,, \tag{98}$$

where T is a constant (no time spreading) and the $\mp$ indicates that the $-$ or $+$ sign is to be used for the hydrophone in the $+x$ $(y_+(t))$ or $-x$ $(y_-(t))$ direction respectively.

The Poisson statistical assumption is

$$\mathbf{E}\left\{h(S_x)h^*(S'_x)\right\} = \delta(S_x - S'_x)\, U^2(S_x)\,. \tag{99}$$

We postulate that the shape is Gaussian in S_x:

$$U^2(S_x) = \frac{|U_0|^2}{\sigma_S\sqrt{2\pi}}\,\mathrm{e}^{-\frac{(S_x-\bar{S_x})^2}{2\sigma_S^2}} \tag{100}$$

with its peak at some value (direction) $\bar{S_x}$. This might arise if the transmit beam pattern has a Gaussian shape. Here, $|U_0|^2$ is the total target strength of the lump of scatterers illuminated by that transmit beam.

Of course, a true Gaussian shape is impossible since the independent variable S_x has a finite range, but we will ignore that fact and extend the tails of the Gaussian out to infinity. This approximation is best for relatively narrow beams, with a direction close to broadside ($\bar{S_x} \approx 0$).

This time we are interested in the cross-covariance, at zero time lag, between the signals in the two hydrophones, as a function of the distance d between them:

$$\begin{aligned}\mathbf{E}\left\{y_+(t)y_-^*(t)\right\} &= |x_0|^2 \int \mathrm{d}S_x \int \mathrm{d}S'_x\, \mathbf{E}\left\{h(S_x)h^*(S'_x)\right\} \\ &\quad \times x(t-T+S_xd/2)x^*(t-T-S'_xd/2) \\ &= |x_0|^2 \int \mathrm{d}S_x\, U^2(S_x)\,\mathrm{e}^{\mathrm{i}2\pi f S_x d} \\ &= |U_0x_0|^2\, C(d)\,,\end{aligned} \tag{101}$$

where the *spatial coherence* of the signal is

$$C(d) = \mathrm{e}^{-\frac{d^2}{2\sigma_d^2}} \tag{102}$$

with an RMS width (or *coherence distance*) of

$$\begin{aligned}\sigma_d &= \frac{1}{2\pi f\sigma_S} \\ &= \frac{\lambda}{2\pi\sigma_\theta}\,,\end{aligned} \tag{103}$$

where σ_θ is the RMS width of the Gaussian transmit beam in *sinθ*, and λ is the wavelength.

Since the width of the transmit beam decreases as the size of the transmit aperture increases relative to the wavelength, the behavior of the spatial coherence can be expressed this way: If the spread in arrival direction is due to the width of the transmit beam for a monostatic sonar, then the coherence distance at the receive array is approximately equal to the size of the transmit array.

The general principle this time is that if random scattering produces a spread in arrival direction, then the spatial coherence will be proportional to the ratio of the wavelength to the spread in the sine of the angle, measured from the plane perpendicular to the vector connecting the receiver elements.

The spatial analog of replica correlation is beam forming. An important clue differentiating a target echo from reverberation is that the target echo is more localized in direction – which is equivalent to saying that the correlation distance in the perpendicular direction is longer for target echoes than for reverberation.

The low coherence distance of reverberation could, in principle, make simulation simpler. SST's Scattering Function algorithms (Sec. 11.3) could gain speed if they took advantage of the fact that the cross-power matrix is often sparse. We might do this in the future.

10.6.4 Spread vs. Coherence Summary

Spatial coherence is a three-dimensional concept, since the separation vector between elements is a three-vector. However, the component parallel to the mean arrival direction is essentially the same as the time coherence, and the arrival direction has only two independent dimensions, so spatial coherence adds only two independent dimensions to the set of coherence functions.

Reverberation, then, is quadruply spread: in time, Doppler shift, and two dimensions of direction. As a consequence, the coherence of the reverberation signal is reduced in four independent dimensions: frequency, time, and two dimensions of spatial separation.

We will look at this same set of issues from a different point of view in Sec. 12.

10.7 Separability: Critique and Justification

Equation (76) is postulated to have *separable* form; i.e., the dependence on the two frequencies has the form of a product of two instances of a function, each evaluated at one frequency. The convenience of this form is obvious, since it enables us to use the simple generation technique of Eq. (77). But one must ask: Is it accurate?

We start with the simplest case, a single scattering event seen via a single eigenray pair. In that case, the transfer function is given by Eq. (70), which clearly has separable form, since the expectation of Eq. (76) reduces trivially to a simple product:

$$\mathbf{E}\left\{A(f)\,A^*(f')\right\} = A(f)\,A^*(f')\,. \tag{104}$$

If there are multiple independent scattering events in a resolution cell, the scattering amplitude is the sum of individual scattering amplitudes. The expectation of Eq. (76) is then

$$\begin{aligned}\mathbf{E}\left\{A(f)\,A^*(f')\right\} &= \mathbf{E}\left\{\left[\sum_T A_T(f)\right]\left[\sum_{T'} A^*_{T'}(f')\right]\right\} \\ &= \sum_T A_T(f)\,A^*_T(f')\,,\end{aligned} \tag{105}$$

where the two sums reduce to one because the scattering events are statistically independent. This does *not*, in general, have separable form because a sum of products is not the same as the product of sums.

However, if the shape versus frequency is the same for all scatterers and only a frequency-independent multiplier M_T is random, it looks like this:

$$\begin{aligned}\mathbf{E}\left\{A(f)\,A^*(f')\right\} &= \mathbf{E}\left\{\left[\sum_T M_T B(f)\right]\left[\sum_{T'} M^*_{T'} B^*(f')\right]\right\} \\ &= \left[\sum_T |M_T|^2\right] B(f)\,B^*(f') \\ &= S(f)\,S^*(f')\,,\end{aligned} \tag{106}$$

which is separable, with

$$S(f) = \sqrt{\sum_T |M_T|^2} B(f)\,. \tag{107}$$

So scattering from an ensemble of scatterers is separable if the individual scatterers are similar to one another, such as fish of one size and species in a school.

However, that's not a very general model. For example, a bubble plume consists of bubbles of many sizes, and resonant scattering from those bubbles has a size-dependent frequency dependence. Roughness scattering is similar, since scattering at a particular frequency is dominated by roughness of a particular scale related to the acoustic wavelength.

We are rescued by the very narrow frequency coherence of reverberation, which we discussed in Sec. 10.6.1. If our answer is wrong for frequencies f and f' that are far apart, *it doesn't matter*! Those combinations don't contribute significantly to reverberation anyway. All that matters is that the result *looks* separable for frequencies that are very close to one another. The criterion we need for that to be true is easy to satisfy: The scattering amplitudes of the individual scatterers must vary slowly relative to the coherence bandwidth of the reverberation. If that is true, any errors introduced by treating the scattering strength as separable will have a negligible effect on the outcome.

This analysis suggests the limits of the approximation: In cases where the coherence bandwidth is large *and* the ensemble of scatterers is heterogeneous *and* highly resonant, our assumption of the separable form might introduce errors. The place to look for such errors is near transients, where the time dependence of reverberation is not smooth.

10.8 Complex Envelope Representation

The discussion so far dealt with real signals: The time-domain quantities $y(t)$, $x(t)$, and $h(\gamma, \boldsymbol{R}, \tau)$ are real, and the frequency-domain quantities $H_{rpqs}(\gamma, \boldsymbol{R}, f)$, $B(f, \boldsymbol{S}')$, $L(f)$, and $A(\gamma, \boldsymbol{R}, f)$ are conjugate-symmetric about $f = 0$ (e.g. $L(-f) = L^*(f)$) (subscripts omitted).

Now let us assume that the source signal and the reverberation signal are represented using complex envelope form, as described in Sec. 4.1.2. Further, for generality we permit those two complex signals to have two different center frequencies: the source signal $\tilde{x}(t)$ has center frequency F_x, and the reverberation signal $\tilde{y}(t)$ has center frequency F_y. Then the equations starting with Eq. (73) acquire some extra phase factors, as follows:

$$\begin{aligned}\tilde{y}_r(t) \;=\; & \mathrm{e}^{-\mathrm{i}2\pi(F_y-F_x)t} \int \mathrm{d}^3\boldsymbol{R} \sum_{pqs} \int \mathrm{d}\gamma \int \mathrm{d}\tau \, \tilde{h}_{rpqs}(\gamma, \boldsymbol{R}, \tau) \\ & \times \tilde{x}_s(\gamma\Gamma_{pq}(t - T_p - T_q - \tau)) \, \mathrm{e}^{-\mathrm{i}2\pi F_x[T_p+T_q+(1-\gamma\Gamma_{pq})(t-T_p-T_q-\tau)]} \, ,\end{aligned} \tag{108}$$

where

$$\tilde{h}_{rpqs}(\gamma, \boldsymbol{R}, \tau) = w(\tau) \int_{-1/2h}^{1/2h} \mathrm{d}\tilde{f} \, H_{rpqs}(\gamma, \boldsymbol{R}, F_x + \tilde{f}) \, \mathrm{e}^{\mathrm{i}2\pi\tilde{f}\tau} \, . \tag{109}$$

The Fourier transform is now in terms of $\tilde{f}$, which is the frequency offset relative to the center frequency F_x. Its range is $-1/2h \leq \tilde{f} < 1/2h$, where h is the sampling interval.

Most of the discussion in the following section will deal with real signals. These extra phase factors will re-enter the discussion occasionally when we get to concrete descriptions of the algorithms.

11 Reverberation Generation Algorithms

All algorithms for generating reverberation involve, at some stage, introducing randomness by invoking sources of random (really pseudo-random) numbers. Before the randomization stage, the algorithm computes characteristics of some random process, typically second moments of a distribution. After the randomization stage, the algorithm works on an input stream of random quantities, transforming them to the multi-channel signal required by the application.

A useful way to classify these algorithms is by the stage in the calculation where the randomization takes place. For algorithms based on the point scatterer approach (Sec. 11.1), that point is very early in the process, and almost all of the computation goes into transforming random point scatterers into random signals. For algorithms based on scattering functions (Sec. 11.3), the randomization occurs late in the process; most of the computation involves computing second order moments describing the reverberation signal. Between those extremes lie many different algorithms, one of which is SST's Directional Doppler Density (DDD) algorithm (Sec. 11.2).

We will describe the methods in order of randomization stage, from early to late, because they are easiest to explain that way. This is quite different from the implementation order: The Point Scatterer method came first in SST, but is no longer used; the Scattering Function approach has been standard in SST for a long time; and the DDD method is new with SST 4.6 (and, frankly, still immature).

11.1 Point Scatterer Approaches

One very simple way to generate simulated reverberation, the "point scatterer" approach, is based directly on the simple target echo model of Sec. 9.3. SST's predecessor, REVGEN-4 [Goddard 1986], and the earliest releases of SST, used a version of this algorithm. All of the versions dicussed here are limited to monostatic sonar systems, and all but the "noiselet" algorithm are narrow-band (in that frequency dependence of propagation, scattering, and beam patterns is ignored). We start the discussion with a very simple version.

Scattering comes from *scattering layers*, as they do in the current SST: one layer for the surface, one for the bottom, and one representing a thin sheet of scatterers at the center of each volume scattering layer. For each scattering layer l, a cell size in range (δr) and azimuth $(\delta\theta)$ is chosen such that each resolution cell of the sonar (in range, azimuth, elevation, and platform Doppler) contains several such cells. Those parameters determine the range step size:

$$\rho = (\delta r)\,(\delta\theta)/(2\pi)\,. \tag{110}$$

Starting at range $r_0 = 0$, for each layer l, the algorithm marches outward in horizontal range, generating random scatterers:

$$\begin{aligned} r_i &= r_{i-1} + \rho\, \mathit{ranexp}() \\ \theta_i &= 2\pi\, \mathit{ranun}() \\ \boldsymbol{R}_i &= (r_i \cos\theta_i, r_i \sin\theta_i, z_l) \\ \mathsf{area}_i &= 2\pi\rho r_i \\ A_i &= \sqrt{\mathsf{area}_i}\, \mathit{ranorm}()\,, \end{aligned} \tag{111}$$

where *ranexp*(), *ranun*(), and *ranorm*() are random-number generators for exponential, uniform, and zero-mean normal distributions, respectively, and A_i is the random scattering amplitude assigned to each scatterer, for a uniform scattering strength of unity. Using an exponential distribution for the range step results in a uniform distribution for the range itself.

The algorithm then finds all of the two-way eigenray pairs pq to the newly generated scatterer. For each pq, the ray angles at the scatterer are used to compute the scattering strength, beam patterns, and Doppler shift. The Doppler shift for that two-way path is used to select one from a collection of pre-computed Doppler-shifted replicas of the transmit pulse. That replica is weighted by the product of A_i, the square root of the scattering strength, the eigenray amplitudes, and the transmit and receive beam patterns. The weighted replica (the echo) is then summed into the output buffer at a location determined by the eigenray pair's two-way travel time.

The algorithm just described has the disadvantage that the signal is not computed in time order. The whole output buffer stays in memory until the whole ping is finished. A relatively simple modification can reduce the buffer memory to the time window whose length is the difference between the shortest and longest two-way propagation delays – but in deep water this might not reduce the buffer length at all.

REVGEN-4 and the earliest SST used a more radical modification: As the scatterer generation sweeps outward in range, the minimal attributes describing each echo are used to create an **Echo** object, which is placed into a *priority queue* [Knuth 1973], which sorts the echoes by two-way travel time $T_{pq}(\boldsymbol{R}_i)$. At any stage in the outward sweep by range, there exists a time $T_<$ such that we know that we are finshed generating echoes with travel times less than $T_<$. As $T_<$ increases, the "back end" of the algorithm pulls out of the queue the echoes having times less than that limit, and finishes the calculation by computing the weighted, Doppler-shifted replica and summing it into the output buffer. Memory for the output buffer (which scales with the number of receiver channels and with the sample rate) is traded for memory for the priority queue (which does not).

Note that these versions of the algorithm do *not* assume that the signals along different eigenray pairs are statistically independent (i.e., the factors $\delta_{pp'}\,\delta_{qq'}$ in Eq. (76) are absent). This is an advantage at low frequencies, where sometimes interference effects (e.g., Lloyd's Mirror) in reverberation can be observed as striations in the range-Doppler map.

In yet another version of the Point Scatterer algorithm, the eigenray pairs are precomputed and sorted by two-way travel time at the start of the calculation. Then, instead of sweeping outward by range, the algorithm sweeps outward in two-way travel time, placing clusters of random scatterers at ranges determined by the stored eigenray pairs. The main advantage is that the main part of the calculation is done in time order. The exception is computing and sorting the eigenray pairs, which can be done by a separate process, even on another computer. One disadvantage is that the echoes from the same location via different eigenray pairs are statistically independent; the $\delta_{pp'}\,\delta_{qq'}$ factors are back in.

Another variant is the *noiselet* approach, which is under development by Prometheus Inc. for use in the WAF simulator at NUWC Newport. A *noiselet* is a sum of many copies of the transmit pulse, with random amplitudes, a short range of delays, and a particular Doppler shift. A large collection of noiselets is pre-computed before the main processing loop of the simulation. From there on, the algorithm proceeds like the previous version (with time-sorted eigenray pairs), except that it randomly selects a noiselet from its collection and uses it in place of the transmit pulse. The advantage is that this version requires far fewer "scatterers" because each noiselet represents echoes from a cluster of scatterers distributed over a somewhat larger area or volume. For extra speed, most of the processing is done in the frequency domain. This algorithm, unlike the others in this section, is "broadband" in that it supports frequency dependence in propagation, scattering, and beam patterns. Details will be published separately.

The point scatterer approach and its variants are examples of "early randomization" models, so called because much of the processing occurs after random numbers are introduced. The point scatterer algorithm was eliminated from SST in the early 1990s because it was slow, especially since signals with higher bandwidths required more and more point scatterers to "fill in" the distribution. A somewhat less extreme "early randomization" model is described in Sec. 11.2.

11.2 Directional Doppler Density Method for Reverberation

The Directional Doppler Density (DDD) method for generating reverberation is new with SST Release 4.6.

11.2.1 Assumptions

The physical assumptions listed in Sec. 10.1 apply here: linearity, eigenray propagation, single scattering, far field, and high scatterer density (Poisson scattering). In addition, we also assume that reverberation arising from different source channels s can be treated as uncorrelated – which means in practice that the receiver is not sensitive to such correlations if they exist. This is useful, for example, for "ripple transmits" in which each pulse in a sequence is sent in a different direction using a different transmit beam. This approach precludes interpreting each element of a transmit array as a "channel".

In this discussion, we continue to assume that reverberation has Gaussian statistics. The DDD method (like the Point Scatterer approach) is capable of generating non-Gaussian (e.g. K-distributed) reverberation. Discussion of this capability is deferred to a separate document.

We also assume, in this section, that the transmit signal is relatively short – short enough that it makes sense to describe a Doppler-shifted pulse as $x_s(\Gamma t)$ using a single Γ for the whole pulse. It would be relatively simple to get around this restriction by treating long signals as sequences of shorter ones, but we don't do that (yet).

11.2.2 Objectives

Let's review where we are so far. In the idealized, continuous view of Sec. 10, the procedure for generating reverberation is straightforward: Generate a random scattering field using Eq. (77), plug the result into Eq. (75), plug that result into Eq. (74), and plug that one into Eq. (73) – which involves integrating a random field over all space, among other things. A direct numerical approximation of that procedure involves breaking all space up into finite chunks (Eq. (79)) and summing them – which is essentially a description of the Point Scatterer approach.

That approach is simple and understandable, but it's weak on efficiency. Equation (73) is an integration over a very large region of the ocean, and the integrand depends on everything you can think of, and the answer doesn't come out in time order. This last criterion is most critical for real-time systems because latency before delivering the first results is crucial. It is also important for non-real-time systems because it seems that the entire answer must be kept in memory until the whole ping is finished.

In the next few subsections, we will break the problem into sub-problems and rearrange the pieces. Our main objectives in designing this algorithm are

- Move as much processing as possible outside of the spatial integration, to keep the integrand simple.
- Deliver the results in time order, with minimal time spent computing intermediate results that won't be needed until later in the listening interval.
- Avoid algorithms having super-linear scaling with the number of receiver channels, to maintain efficiency for element-level simulations.
- Plan for extension to non-Gaussian statistics (e.g. K distributions).

11.2.3 The Doppler Density: Factoring Out the Source Signal

Our approach here may seem backward. Our first step will be to define the *last* step in the algorithm: the one in which the final output signal is computed.

Our starting point is the Local Impulse Response of Sec. 10.2. Equation (73) is clearly a linear transformation operating on the signal $x_s(t')$. It looks sort of like a convolution, but the integration variables are only very distantly related to the independent variables of the input and output signals, namely t' and t.

We observe that reverberation is a *linear* but *non-stationary* function of the source signal. The most general such relationship has the following canonical form:

$$y_r(t) = \sum_s \int \mathrm{d}t'\, k_{rs}(t,t')\, x_s(t')\,, \qquad (112)$$

where the *reverberation kernel* $k_{rs}(t,t')$ depends on the ocean channel and on the characteristics and locations of the transmitter and receiver, but not on the source signal $x_s(t')$.

In principle, one could reduce Eq. (73) to its canonical form, Eq. (112). In fact, the original formulation of this algorithm did exactly that. However, we found it difficult to make the computation of the reverberation kernel efficient, so we settled on a slightly more complicated form for the final generation step:

$$y_r(t) = \sum_s \int \mathrm{d}\Gamma \int \mathrm{d}T'\, u_{rs}(T',\Gamma)\, x_s(\Gamma(t-T'))\,, \tag{113}$$

where $u_{rs}(T',\Gamma)$ is the *reduced Doppler density.*

When we are using complex envelope notation, most of the extra phase factors from Eq. (108) come in here:

$$\begin{aligned}\tilde{y}_r(t) \quad &= \quad \mathrm{e}^{-\mathrm{i}2\pi(F_y-F_x)t} \sum_s \int \mathrm{d}\Gamma \int \mathrm{d}T'\, u_{rs}(T',\Gamma) \\ &\qquad \times \tilde{x}_s(\Gamma(t-T'))\, \mathrm{e}^{-\mathrm{i}2\pi F_x(1-\Gamma)(t-T')}\,. \end{aligned} \tag{114}$$

Note that $u_{rs}(T',\Gamma)$ is real for real signals, and complex for signals having the complex envelope representation.

The transformation from Eqs. (73) – (77) to Eq. (113) is a change of integration variables, accompanied by a large reduction in the number of dimensions. With reckless use of Dirac delta functions, it can be expressed in the following form:

$$\begin{aligned}u_{rs}(T',\Gamma) \quad &= \quad \int \mathrm{d}^3\boldsymbol{R} \sum_{pq} \int \mathrm{d}\gamma \int \mathrm{d}\tau \\ &\qquad \times w(\tau) \int \mathrm{d}f\, H_{rpqs}(\gamma,\boldsymbol{R},f)\, \mathrm{e}^{\mathrm{i}2\pi f\tau} \\ &\qquad \times \delta(\Gamma - \gamma\Gamma_{pq}(\boldsymbol{R}))\, \delta(T' - T_p(\boldsymbol{R}) - T_q(\boldsymbol{R}) - \tau)\,, \end{aligned} \tag{115}$$

which you can verify by substituting Eq. (115) into Eq. (113) and observing that the Dirac delta functions eliminate the integrations over T' and Γ. Applying Eq. (74) then leaves us with Eq. (73).

We choose to pick Eq. (115) apart in the following order:

$$u_{rs}(T',\Gamma) = \int \mathrm{d}\tau\, q_{rs}(T'-\tau,\Gamma,\tau)\,, \tag{116}$$

where

$$q_{rs}(T,\Gamma,\tau) = w(\tau) \int \mathrm{d}f Q_{rs}(T,\Gamma,f)\, \mathrm{e}^{2\pi \mathrm{i} f\tau}\,, \tag{117}$$

where $Q_{rs}(T,\Gamma,f)$ is the *Doppler density*:

$$\begin{aligned} Q_{rs}(T,\Gamma,f) \quad &= \quad \int \mathrm{d}^3\boldsymbol{R} \sum_{pq} \int \mathrm{d}\gamma\, H_{rpqs}(\gamma,\boldsymbol{R},f) \\ &\qquad \times \delta(\Gamma - \gamma\Gamma_{pq}(\boldsymbol{R}))\, \delta(T - T_p(\boldsymbol{R}) - T_q(\boldsymbol{R}))\,, \end{aligned} \tag{118}$$

where the *local Doppler transfer function* $H_{rpqs}(\gamma,\boldsymbol{R},f)$ is the stochastic function given by Eq. (75). If $x(t')$ and $y(t)$ are real, then $q_{rs}(T,\Gamma,\tau)$ and $u_{rs}(T',\Gamma)$ are real, and the Doppler density is Hermitian in f (i.e. $Q_{rs}(T,\Gamma,f) = Q^*_{rs}(T,\Gamma,-f)$).

In the case where the signals are represented in complex envelope form, Eq. (116) acquires an extra phase factor:

$$u_{rs}(T',\Gamma) = \int \mathrm{d}\tau\, q_{rs}(T'-\tau,\Gamma,\tau)\, \mathrm{e}^{-\mathrm{i}2\pi F_x(T'-\tau)}\,. \tag{119}$$

It is worth noting that in the special case that $H_{rpqs}(\gamma,\boldsymbol{R},f)$ is independent of frequency f, the "reduction" steps of Eqs. (117) and (116) do nothing at all. For real signals, $u_{rs}(T,\Gamma) = Q_{rs}(T,\Gamma,\cdot)$ in the limit of a long window $w(\tau)$. For complex signals, they differ only by the phase factor $\mathrm{e}^{-\mathrm{i}2\pi F_x T}$ in this case.

The main thing we have accomplished here is to remove the source signal $x_s(t')$ from the complicated part of the computation. This is especially advantageous for signals having a large time-bandwidth product, which require many samples for their representation. We gain in efficiency by delaying processing of this potentially large structure until a late stage of the algorithm.

SST Class: Class **DDReverbSignal** is a "data flow" class (Sec. 4.3) that reads from a **DopplerDensity** object (Table 4) and produces multi-channel sound. It first "reduces" the Doppler density by applying Eqs. (117) and either (116) or (119). In a second step, it performs the two-dimensional integration involving the source signal, Eq. (113) or (114).

SST Class: Class **FIRCoef** implements Eq. (22) to compute FIR filter coefficients for a given frequency dependence. (It was also mentioned in Sec. 4.5.2, where it is used to generate inputs for Class **VarFirFilter**). In this application (Eq. (117)), it is applied separately to $Q_{rs}(T,\Gamma,f)$ for each value of T and Γ.

Units of Measure: The output signal $y(t)$ is a pressure, in units of μPa. The input signal is a pressure "reduced to one meter," which means units of μPa-m.

Working back via Eq. (113), that means the reduced Doppler density $u_{rs}(T', \Gamma)$ has units $\mathrm{m}^{-1}\mathrm{s}^{-1}$. It is also scaled "per unit Doppler", which doesn't change the units since Doppler is dimensionless, but we still have to remember it, so it is convenient to think of the units as $\mathrm{m}^{-1}\mathrm{s}^{-1}\gamma^{-1}$. The two reduction steps, Eqs. (117) and (116), don't change the units, so the Doppler density $Q_{rs}(T, \Gamma, f)$ also has units $\mathrm{m}^{-1}\mathrm{s}^{-1}\gamma^{-1}$.

Units of Measure in Code: Integrations are, of course, implemented as sums. In class **DDReverbSignal**, Eq. (113) is implemented as *just* a sum:

$$y_r(t_i) = \sum_s \sum_j \sum_k \left[h\,(\delta\Gamma_k)\, u_{rs}(T'_j, \Gamma_k)\right]\, x_s(\Gamma_k(t_i - T'_j))\,, \tag{120}$$

where h and $(\delta\Gamma_k)$ are the step sizes in T and Γ respectively. In the code, the array used to represent the reduced Doppler density is the quantity in square brackets; it already includes those step sizes. Its unit of measure is therefore m^{-1}. The same is true of the Doppler density $Q_{rs}(T, \Gamma, f)$: As computed and stored, it includes the step sizes h and $(\delta\Gamma_k)$, and thus its unit of measure is m^{-1}.

11.2.4 Directional Doppler Density: Factoring Out the Receiver

That leaves us with the hard part, Eq. (118). The next pieces we will peel off are the characteristics of the receive array: the receive beam (or element) patterns $B_r(f, \boldsymbol{S}')$ and the receive channel offsets $\boldsymbol{r}_r$. Again, we accomplish this through a change of integration variables, again through introduction of still more delta functions. (Don't worry, we will soon flick aside the veil and reveal the trick for dealing with those pesky Dirac delta functions.)

For each scattering location $\boldsymbol{R}$ and each eigenray pair pq, the incoming sound arrives from a direction defined by two angles, an elevation θ and an azimuth ϕ, expressed in vehicle coordinates. When we choose these as our integration variables, and insert the generation equations (75) and (77), Eq. (118) becomes:

$$Q_{rs}(T, \Gamma, f) = \int \mathrm{d}\sin(\theta) \int \mathrm{d}\phi\, \zeta_s(T, \Gamma, f, \theta, \phi)\, B_r(f, \boldsymbol{S}'(\theta, \phi))\, \mathrm{e}^{-\mathrm{i}2\pi f \boldsymbol{S}'(\theta,\phi)\cdot \boldsymbol{r}_r}\,, \tag{121}$$

where the stochastic *directional Doppler density* (DDD) is given by

$$\begin{aligned}\zeta_s(T, \Gamma, f, \theta, \phi) \;\; = \;\; & \int \mathrm{d}^3\boldsymbol{R} \sum_{pq} \int \mathrm{d}\gamma\, U_{pqs}(\gamma, \boldsymbol{R}, f)\, g_{pq}(\gamma, \boldsymbol{R}) \\ & \times \delta(\Gamma - \gamma\Gamma_{pq}(\boldsymbol{R}))\, \delta(T - T_p(\boldsymbol{R}) - T_q(\boldsymbol{R})) \\ & \times \delta(\sin(\theta) - \sin(\theta_{Rp}(\boldsymbol{R})))\, \delta(\phi - \phi_{Rp}(\boldsymbol{R}))\,,\end{aligned} \tag{122}$$

where

$$U_{pqs}(\gamma, \boldsymbol{R}, f) = L_p(f)\, S(\gamma, \boldsymbol{R}, \boldsymbol{S}_{Sp}, \boldsymbol{S}_{Rq}, f)\, L_q(f)\, B_s(f, \boldsymbol{S}'_{Sq})\mathrm{e}^{-\mathrm{i}2\pi f \boldsymbol{S}'_{Sq}\cdot \boldsymbol{r}_s} \tag{123}$$

and $|S(\gamma, \boldsymbol{R}, \boldsymbol{S}_{Sp}, \boldsymbol{S}_{Rq}, f)|^2$ is the scattering strength (Eq. (76)). Here, $\theta_{Rp}(\boldsymbol{R})$ and $\phi_{Rp}(\boldsymbol{R})$ are the elevation and azimuth angles of the incoming sound from receive-leg eigenray p, expressed in vehicle coordinates, for scatterer location $\boldsymbol{R}$. In case you were wondering, ζ is Greek *zeta*; I ran out of Roman letters.

In splitting the procedure into Eqs. (121) and (122), we move from an environment-centered view to a receiver-centered view. The right side of Eq. (122) depends on the scatterer location $\boldsymbol{R}$, the scatterer Doppler γ, and the propagation eigenrays indexed by pq, all of which have been eliminated from Eq. (121). Instead, the latter depends on variables that are directly relevant to the receiver, including the two-way time delay T, the arrival direction θ, ϕ, and the total Doppler shift Γ. In particular, breaking out the two-way travel time T allows us to arrange the downstream processing in time order.

By removing the receiver array characteristics from the DDD, we gain efficiency, especially for receivers having a large number of channels. This is especially important for element-level simulations. The key to realizing this efficiency is to take advantage of the fact that the DDD is very sparse. For a given time delay T, most of the energy comes from a few elevation angles representing sound arriving from a given scattering layer via a given return path. For times late in a ping, sound tends to come in from nearly horizontal directions. Moreover, for a given incoming direction, most of the sound falls in a very narrow range of Doppler values Γ. Hence the five-dimensional function $\zeta_s(T, \Gamma, f, \theta, \phi)$ has the value zero over nearly all of its range.

Again, we have factored out a potentially large data structure, $B_r(f, \boldsymbol{S}')\, \mathrm{e}^{-\mathrm{i}2\pi f \boldsymbol{S}'\cdot \boldsymbol{r}_r}$. We do most of the processing, including the sum over two-way paths and the spatial integration, before the beams and offsets are introduced.

SST Class: In class **DirectionalDopplerDensity**, the final section of the *doReadRecord* function carries out the integration of Eq. (121), transforming a *directional Doppler density* to a *Doppler density*. The DDD itself, $\zeta_s(T, \Gamma, f, \theta, \phi)$, doesn't exist anywhere as a data structure; rather, it is computed one element at a time and immediately consumed. Equation (121) is the consuming part; we will get to the producing part soon.

11.2.5 Point Scatterer Directional Doppler Density

This is an aside, but it may prove interesting in the future.

As it happens, Eq. (122) is the basis for a reasonable variant of the point scatterer approach, which we will call the "PSDDD" algorithm. One could generate random point scatterers as in Sec. 11.1, and accumulate their amplitudes in a big but sparse data structure representing $\zeta_s(T, \Gamma, f, \theta, \phi)$. Then, as a separate step, one could insert the receive beams, receive channel offsets, and transmit signal just as in the present implementations of Eqs. (121), (117), (116), and (113).

The PSDDD algorithm is probably faster than the more straightforward algorithms outlined in Sec. 11.1 because more of the processing is done on data structures that do *not* scale with the number of channels and with the number of samples in the pulse replica. On the other hand, this variant shares some of the disadvantages of those algorithms, including the difficulty of producing results in time order, and adds the complexity of managing a very large sparse data structure. The following algorithm for computing the DDD is, in our judgment, superior to PSDDD, at least for high frequencies. For low frequencies, where we may need to retain some correlation between signals arriving along different paths from the same scatterer, it might prove advantageous to implement PSDDD.

11.2.6 Directional Scattering Function

We return now to the problem of generating the directional Doppler density, Eq. (122). To simplify that formulation, we would like to find a different, equally valid second-order statistic that retains the diagonal and separable properties (for easy generation) while reducing the total processing load by reducing the number of dimensions. The hope is that this would reduce both the number of random numbers that we need to generate and the amount of processing needed after that stage.

Toward that end, consider a slightly restricted set of second-order moments of Eq. (122):

$$\begin{aligned}\mathbf{E}\left\{\zeta_s(T,\Gamma,f,\theta,\phi)\,\zeta_s^*(T',\Gamma',f,\theta',\phi')\right\} \;=\;& Z_s(T,\Gamma,f,\theta,\phi) \\ &\times\delta(T-T')\,\delta(\Gamma-\Gamma') \\ &\times\delta(\sin(\theta)-\sin(\theta'))\,\delta(\phi-\phi')\,,\end{aligned} \tag{124}$$

which is restricted in that the frequency f is the same in both factors in the expectation. The *directional scattering function* (in its differential form) is

$$Z_s(T,\Gamma,f,\theta,\phi) = \sum_{pq} \frac{\left|U_{pqs}\left(\gamma_{pq}(T,\Gamma,\theta,\phi), \boldsymbol{R}_{pq}(T,\Gamma,\theta,\phi), f\right)\right|^2}{\left|\frac{\partial(T,\Gamma,\theta,\phi)_{pq}}{\partial(\gamma,\boldsymbol{R})}\right|}\,, \tag{125}$$

where the delta functions of Eq. (78) have been used to eliminate some of the integrations. The arguments γ and $\boldsymbol{R}$ of $U_{pqs}(\gamma, \boldsymbol{R}, f)$ on the right side of Eq. (125) signify an inverse mapping from the receiver-oriented set $(T, \Gamma, \theta, \phi)$ back to the corresponding environment-oriented set $(\gamma, \boldsymbol{R})$. The denominator is the Jacobian determinant associated with the forward mapping.

Aside: The derivation of Eqs. (124) and (125) from Eq. (122) involves the following lemma about Dirac delta functions:

$$\int \mathrm{d}^n\boldsymbol{x}\, \delta(\boldsymbol{z}-\boldsymbol{y}(\boldsymbol{x}))\, \delta(\boldsymbol{z}'-\boldsymbol{y}(\boldsymbol{x})) = \frac{\delta(\boldsymbol{z}-\boldsymbol{z}')}{\left|\frac{\partial \boldsymbol{y}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right|}\,, \tag{126}$$

where $\boldsymbol{z}$, $\boldsymbol{z}'$, $\boldsymbol{x}$, and the function $\boldsymbol{y}(\boldsymbol{x})$ are all vectors of length n. This lemma follows from the definition of the Dirac delta function plus standard integral calculus, but it's not obvious.

In accordance with our "last first" approach, we will defer a discussion of Eq. (125) (which has serious problems) and focus first on inverting Eq. (124). The delta functions tell us that the second moments of the directional Doppler density are diagonal in all of its arguments except the frequency. In accordance with the discussion of Sec. 10.7, we choose to *assume* that it is separable in f. With that assumption, we can use our standard, simple method for generating Gaussian realizations of the DDD:

$$\zeta_s(T, \Gamma, f, \theta, \phi) = \sqrt{Z_s(T, \Gamma, f, \theta, \phi)}\, g_s(\Gamma, \theta, \phi, T)\,, \tag{127}$$

where $g_s(\Gamma, \theta, \phi, T)$ is a complex, unit-variance Gaussian random process satisfying

$$\mathbf{E}\left\{g_s(\Gamma, \theta, \phi, T) g_s^*(\Gamma', \theta', \phi', T')\right\} = \delta(\Gamma-\Gamma')\, \delta(\theta-\theta')\, \delta(\phi-\phi')\, \delta(T-T')\,. \tag{128}$$

Equation (127) is statistically equivalent to Eq. (122), provided the separability assumption is correct. The difference is that Eq. (127) introduces the random numbers later in the process. Arguably, fewer random numbers are required because the sum over two-way paths is done before the randomization step. More important, in most circumstances the directional scattering function is expected to be *smooth* in time T. This allows us to compute it on a relatively coarse time grid, and interpolate in between. This turns out to be a major time saver.

SST Class: In class **DirectionalDopplerDensity**, the initial section of the *doReadRecord* function invokes class **ReverbDirectionalScatFun** (described later) to compute $Z_s(T, \Gamma, f, \theta, \phi)$ on a relatively coarse time grid (the "update" grid). It interpolates these values to a much finer time grid, then takes a square root and multiplies the results by Gaussian random numbers to implement a discrete version of Eq. (127). (This "discrete version" involves integration of Eq. (127) over small volumes in its four-dimensional space, analogous to Eqs. (79) and (80).) The final section implements Eq. (121), as we already described in Sec. 11.2.4.

11.2.7 Computing the Directional Scattering Function

The differential form of the DSF defined in Eq. (125) is useless for computation because it relies on an explicit mapping from the variable set $(T, \Gamma, \theta, \phi)$ back to the set $(\gamma, \boldsymbol{R})$ – a different mapping for each two-way eigenray set pq. This mapping is ugly because the forward mapping involves the eigenray properties, which themselves come from a very complex piece of code (e.g. GRAB) that has no explicit inverse. Moreover, the Jacobian can be zero, so Eq. (125) can blow up.

We solve these computational problems by dividing the "ocean space," whose dimensions are location $\boldsymbol{R}$ and scatterer Doppler γ, into little cells that are small enough that $U_{pqs}(\gamma, \boldsymbol{R}, f)$ can be regarded as constant within a given cell. We also break up the sonar's "perception space," whose dimensions are travel time T, total Doppler Γ, and incoming direction θ, ϕ, into cells that are small enough to be unresolvable by the sonar. As we integrate over each cell of the "ocean space," we accumulate the results in the corresponding cell of the "perception space." The result, suitably normalized, is the *integral form* of the *directional scattering function*:

$$Z_s(T_\nu, \Gamma_k, f_m, \theta_i, \phi_j) = \frac{1}{H\,(\delta\Gamma_k)\,(\delta\Omega_{ij})} \sum_{pq} \int_{V_{ij\nu,pq}} \mathrm{d}\boldsymbol{R} \int_{\Gamma_k} \mathrm{d}\gamma\ \left|U_{pqs}(\gamma, \boldsymbol{R}, f_m)\right|^2 , \tag{129}$$

where $U_{pqs}(\gamma, \boldsymbol{R}, f_m)$ is defined in Eq. (123). The variables H and $\delta\Gamma_k$ are the widths of the bins in two-way travel time T_ν and Doppler Γ_k, respectively, and $\delta\Omega_{ij}$ is the solid angle subtended by the angle bins θ_i and ϕ_j.

The integration volumes $V_{ij\nu,pq}$ are defined implicitly by the requirement that sound transmitted via eigenray q, scattered by scatterers within that volume, and received via eigenray p will arrive in a small three-dimensional bin of elevation angles centered on θ_i, azimuth angles centered on ϕ_j, and two-way travel time $T_p + T_q$ centered on T_ν. We also integrate over bins in scatterer Doppler γ defined by requiring the total Doppler $\gamma\Gamma_{pq}$ falls within a bin centered on Γ_k. The sizes of the bins in Doppler, frequency, elevation, azimuth, and time are chosen such that variation in the result within any one bin can be neglected.

SST's approach is to lay out the locus of constant two-way travel time T, for a given two-way path pq, and then compute samples of the integrand along that locus. For each sample location, the corresponding direction and Doppler values are computed, and the corresponding bins of the directional scattering function are incremented by the product of the integrand and a normalization factor proportional to the volume or area $\mathrm{d}\boldsymbol{R}$ represented by the sample point.

The integral form of the DSF, Eq. (129), is equivalent to the differential form, Eq. (122), averaged over bins in time T_ν, direction (θ_i, ϕ_j), and total Doppler Γ_k.

It is important to note that the pieces of the integrand that end up in each bin in Γ_k, θ_i, ϕ_j, and T_ν are statistically independent of one another. That means the directional scattering function, viewed as a second-order statistic, is still *diagonal* in those variables. That is the property that enables us to use a simple square root in (127) instead of a more complicated factorization.

SST Classes: Class **ReverbDirectionalScatFun** implements the integration of Eq. (129). This class does not appear in the SST command language interface, but it is used to implement both Class **BBBDirectionalScat** (see Sec. 11.3.2) and the new Class **DirectionalDopplerDensity**.

11.2.8 Summary of the Directional Doppler Method

The DDD processing chain follows SST's usual "data flow" pattern: For each block of the final signal, requests for intermediate data flow "upstream", and the data themselves flow "downstream" through successive processing steps. Here we review those steps in the forward, or downstream, order:

1. The chain starts with class **ReverbDirectionalScatFun**, which implements the integration of Eq. (129) over space and scatterer Doppler. The result is the integral form of the Directional Scattering Function, $Z_s(T, \Gamma, f, \theta, \phi)$.
2. The DSF is passed to class **DirectionalDopplerDensity**, which interpolates it to a finer time grid and then randomizes it by implementing a discrete version of Eq. (127). The result is the stochastic Directional Doppler Density, $\zeta_s(T, \Gamma, f, \theta, \phi)$.
3. Class **DirectionalDopplerDensity** immediately consumes the DSF to implement Eq. (121). The result is the Doppler Density, $Q_{rs}(T, \Gamma, f)$.
4. The DD is passed to class **DDReverbSignal**, which applies Eqs. (117) and either (116) or (119). The result is the Reduced Doppler Density, $u_{rs}(T', \Gamma)$.
5. Class **DDReverbSignal** then computes the final signal by performing the two-dimensional integration involving the source signal, Eq. (113) or (114).

The efficiency of this algorithm depends critically on three observations.

- The DSF is smooth almost everywhere. This allows the first step to be done on a relatively coarse time grid.

- The DSF and the DDD are *sparse* in direction and Doppler. For any given two-way travel time T, only a few cells in θ and ϕ are nonzero. For large T, the nonzero cells represent sound arriving from nearly horizontal directions. In addition, for any given direction, only a small range of Doppler cells are populated. SST makes use of this sparsity to speed up the calculation.

- The characteristics of the receiver channels are introduced late in the process, in Eq. (121). Thus, most of the work does *not* scale with the number of receiver channels. This is especially important for element-level simulations, which sometimes involve 100 channels or more.

11.3 The Scattering Function Approach

SST's current reverberation model is based on the *scattering function approach*, which is less direct and obvious than the point scatterer approach, but much more efficient. The task of generating simulated reverberation is broken into two major steps:

- Compute the *scattering function*, a generalized intensity impulse response function introduced in Sec. 4.2.2.

- Combine the scattering function with the source signal and a stream of random numbers to generate a realization of the reverberation signal.

This approach is not new. The basic idea has been described by Luby and Lytle [Luby Lytle 1987], Hodgkiss [Hodgkiss 1984], and Chamberlain and Galli [Chamberlain Galli 1983], all of which built on the classic work of Faure [Faure 1964], Ol'shevskii [Ol-shevskii 1967], and Middleton [Middleton 1967]. SST's implementation is applicable to far more real systems than the versions described in those references because we have discarded most of the simplifying assumptions: short narrowband pulses, isovelocity single-path propagation, monostatic geometry, and others.

11.3.1 Generating Reverberation

We begin with the second of the two steps: given a scattering function and a source signal, generate a stochastic realization of the received reverberation.

SST computes the power spectral density for reverberation by performing a two-dimensional convolution (versus Doppler Γ and two-way travel time T) of the scattering function $Z_{rr's}(T, \Gamma, f)$ with the power spectral density of the transmit signal:

$$P_{rr's}(f, t_u) = \int \int \Gamma^{-2} \, Z_{rr's}(t_u - t_{u'}, \Gamma, f) \, P_s(f/\Gamma, t_{u'}) \, \mathrm{d}\Gamma \, \mathrm{d}t_{u'} \,. \tag{130}$$

The source PSD $P_s(f, t_u)$ comes from applying Eqs. (7) and (12) to the source signal.

Starting with this estimate for the PSD, SST generates Gaussian random realizations of the reverberation using the same Mitchell-McPherson algorithm used for Gaussian noise, described in Sec. 4.6.1. The validity of this step depends on the same assumptions required for noise: that the statistics are Gaussian and that an update interval Δ exists such that the time variation of $P_{rr's}(f, t_u)$ is slow on a scale of Δ and, simultaneously, the frequency variation is slow on a scale of $1/\Delta$. The resulting signal has a very narrow frequency coherence width (order $1/\Delta$) and a coherence time that is usually of the same order as the inverse of the signal bandwidth but no longer than Δ. This is realistic if the scattering function is sufficiently smooth.

SST Classes: Class **ReverbSignal** generates a realization of Gaussian reverberation starting from the scattering function and the transmit signal. It is implemented using an object of class **ReverbSpectrum**, which implements Eq. (130), together with **SpectrumFromSignal** [to compute $P_s(f, t_u)$, Sec. 4.2] and the Mitchell-McPherson noise generation classes **FactorSpectrum**, **GaussianSpectrum**, and **SignalFromSpectrum** (Sec. 4.6.1).

11.3.2 Computing the Scattering Function

A detailed derivation of the scattering function, and a discussion of the approximations and assumptions on which it is based, are beyond the scope of this report. Some of those issues are discussed in a technical report [Goddard 1993] written early in SST's development. Here, we will skip the derivation and go directly to the answer.

The scattering function $Z_{rr's}(T, \Gamma, f)$ is computed by evaluating the following integral:

$$\begin{aligned} Z_{rr's}(T_\nu, \Gamma_k, f_m) \quad &= \quad \frac{1}{H\,(\delta\Gamma_k)} \sum_{plq} \int_{A_{k\nu,plqs}} |U_{plqs}(f_m, \boldsymbol{r})|^2 \\ &\quad \times B_r(f_m, \boldsymbol{M}_R \boldsymbol{S}_{Rp}(\boldsymbol{R}))\, B^*_{r'}(f_m, \boldsymbol{M}_R \boldsymbol{S}_{Rp}(\boldsymbol{R}))\, \mathrm{e}^{2\pi \mathrm{i} f_m \, \tau_{rr'p}(\boldsymbol{R})} \, \mathrm{d}\boldsymbol{R} \,, \end{aligned} \tag{131}$$

where

$$\begin{aligned} U_{plqs}(f_m, \boldsymbol{R}) \quad &= \quad B_s(f_m, \boldsymbol{M}_S \boldsymbol{S}_{Sq}(\boldsymbol{R}))\, L_p(f_m, \boldsymbol{r}_R, \boldsymbol{R})\, L_q(f_m, \boldsymbol{R}, \boldsymbol{r}_S) \\ &\quad \times S_l(f_m, \boldsymbol{S}_{Tp}(\boldsymbol{R}), \boldsymbol{S}_{Tq}(\boldsymbol{R})) \end{aligned} \tag{132}$$

and $\tau_{rr'p}(\boldsymbol{R})$ is the difference between the first-order offset delays for receiver channels r and r':

$$\tau_{rr'p}(\boldsymbol{R}) = (\boldsymbol{M}_R \boldsymbol{S}_{Rp}(\boldsymbol{R})) \cdot (\boldsymbol{r}'_r - \boldsymbol{r}'_{r'}). \tag{133}$$

The other factors in the integrand are familiar: the beam patterns $B_x(f, \boldsymbol{S}')$, the propagation losses $L_p(f, \boldsymbol{r}_R, \boldsymbol{r}_S)$, and the bistatic scattering strength $S_l(f, \boldsymbol{S}_p, \boldsymbol{S}_q)$. The sum in Eq. (131) is over receive-path eigenrays p connecting the scattering field at $\boldsymbol{R}$ to the receiver, transmit-path eigenrays q connecting the source to $\boldsymbol{R}$, and a scattering layer indexed by l. These "layers" include the surface, the bottom, and any number of volume scattering layers. SST treats volume scattering layers as if all of the scatterers were concentrated on a thin sheet at the center of the layer; this reduces the spatial integration from three dimensions to two.

The scattering function has three conceptually continuous independent variables, all of which are divided into bins for sampling: the Doppler shift Γ, the frequency f, and the two-way travel time T. It also has three discrete indices: the source (or source channel) s and two receiver channels r and r'. The appearance of two receiver channels underscores the nature of the scattering function as a second-order statistic; the off-diagonal elements ($r \neq r'$) give rise to correlations between receiver channels.

The subtle part is defining the domain of integration. For each layer and pair of eigenrays, the domain of integration $A_{k\nu,plqs}$ in Eq. (131) is defined implicitly as the locus of locations $\boldsymbol{R}$ on layer l for which the total round-trip propagation time falls within time bin ν (centered on T_ν with width H) and for which the Doppler shift due to platform motion falls within Doppler bin k (centered on Γ_k with width $\delta\Gamma_k$) for a given receive-leg path p, scattering layer l, and transmit-leg path q.

For a straight-line propagation model with a flat or uniformly sloping bottom, the locus of constant two-way travel time is an ellipse. For a general eigenray model, there is no closed-form mapping from time and Doppler to location, so SST uses standard numeric root-finding techniques to define an approximately elliptical integration path and to partition it into segments that fall into each Doppler bin. The details are beyond the scope of this paper.

SST Classes: SST provides two classes, **BBBScatFun** and **BBBDirectionalScat**, which differ primarily in the strategy used to partition the scattering layers into time-Doppler bins. **BBBScatFun** does the integration of Eq. (131) separately for each eigenray pair and layer plq, and sums them. The partitioning and integration are done with considerable care for continuity and accuracy. Evaluation of the receive beam patterns is in the inner loop; the number of beam pattern evaluations scales with the product of the square of the number of beams, the square of the number of eigenrays, the number of scattering layers, and the number of source channels.

The second choice, **BBBDirectionalScat**, gains speed (sometimes) by factoring out the receive beam patterns and taking less care with the numerics. The first stage is computing the *directional scattering function*:

$$Z_s(T_\nu, \Gamma_k, f_m, \theta_i, \phi_j) = \frac{1}{H\left(\delta\Gamma_k\right)} \sum_{plq} \int_{A_{kij\nu,plqs}} U_{plqs}(f_m, \boldsymbol{R})\, \mathrm{d}\boldsymbol{R}\,, \tag{134}$$

where now the integrand (Eq. (132)) excludes the receive beam patterns, and the integration area $A_{kij\nu,plqs}$ is a patch on the scattering layer such that the eigenray direction at the receiver falls in a small cell centered on elevation angle θ_i and bearing angle ϕ_j in receiver-centered coordinates. For a given time bin T_ν, this large data structure is accumulated by evaluating the integrand for each ray pair and layer plq at a number of sample points around the locus of constant travel time T_ν, and adding a properly scaled increment to the closest cell in Γ, θ, and ϕ for all frequencies f_m.

The directional scattering function is almost always very sparse. For a given source s and time T_ν, four dimensions remain. The contribution for a given eigenray pair and layer plq falls on a thin, closed band on the unit sphere, and those bands move toward the equator (low elevation) as travel time increases. For a given direction cell ij, typically only one Doppler cell k is nonzero. **BBBDirectionalScat** uses a compromise memory management scheme that avoids wasting memory without a large performance penalty.

SST includes hooks and Matlab scripts to create a Matlab movie that shows how the directional scattering function evolves with time.

The second stage of **BBBDirectionalScat** inserts the receiver beam patterns and the phase shift due to receiver channel offsets:

$$\begin{aligned} Z_{rr's}(T_\nu, \Gamma_k, f_m) \;\;=\;\; & \sum_{ij} (\delta\theta_i)\,(\delta\phi_j)\, Z_s(T_\nu, \Gamma_k, f_m, \theta_i, \phi_j) \\ & \times B_r(f_m, \boldsymbol{S}_{ij})\, B^*_{r'}(f_m, \boldsymbol{S}_{ij})\, \mathrm{e}^{2\pi \mathrm{i} f_m \boldsymbol{S}_{ij}\cdot(\boldsymbol{r}'_r - \boldsymbol{r}'_{r'})}\,, \end{aligned} \tag{135}$$

where $\boldsymbol{S}_{ij}$ is the slowness vector corresponding to elevation θ_i and bearing ϕ_j in receiver-centered coordinates. The cell sizes in θ and ϕ are chosen by the user to be small compared to the resolution of the sonar. The beam patterns are evaluated only once for the required angles and frequencies, and used for all time steps.

The advantage of **BBBDirectionalScat** is that it is often faster than **BBBScat-Fun**, especially when the sonar has many channels and there are many eigenrays (e.g., in shallow water). The main disadvantage is that the resulting scattering function tends to be somewhat jagged; neighboring direction cells may have very different values depending on where the integration sample points happen to fall. Users are advised to start with **BBBScatFun** for its more careful numeric techniques. Those who switch to **BBBDirectionalScat** for speed should compare the results of the two methods and adjust the sample density until they give close to the same answers. Of course, only the user can judge how close is close enough.

12 Time Spreading

In the description so far, forward propagation (Sec. 8) is entirely distinct from reverberation (Sec. 10). Forward specular reflections can involve loss of energy but no loss of coherence, whereas the reverberation model produces almost complete loss of coherence. The truth is somewhere between those extremes. The solution is to integrate the forward reflection and forward scattering models into a single, consistent model that generates FAT (Frequency, Angle, Time) spreading for all forward-propagated signals. This section describes a model, based in large part on work by Dahl [Dahl 1996, Dahl 1999, Dahl 2001, Dahl 2002], that goes partway toward that goal. So far, this model includes time spreading and Doppler spreading, but not yet angle spreading. Applications include design and evaluation of longer, higher-bandwidth active transmit signals and acoustic communications.

12.1 Overview

Forward reflection from a rough surface or bottom produces spreading in multiple dimensions. For high-frequency surface reflection, an important physical mechanism for this spreading is reflection from "facets" in the neighborhood of the specular point. The facets act like random shards of a mirror that are not quite horizontal. This phenomenon is analogous to the spreading of an image of the sun reflected from the ocean surface, as seen from an airplane. The probability distribution of the slopes of these facets is largest for nearly horizontal ones, so the distribution of reflection angles is strongly peaked close to the specular (mirror) direction.

Directional spreading, the type of spreading that is most obvious to an airplane traveler, is also important for sound. Sound that arrives via a bounce path arrives from a random distribution of directions in the neighborhood of the specular direction.

In both active and passive sonar, an important clue for detecting a target in background is that the target signal is more localized in direction. A strategy for improving detectability is to increase the size of the receiver array, which decreases the widths of the beam patterns that can be formed, and hence improves its ability to distinguish localized targets from spread-out background. Directional spreading places an upper limit on the array size for which this strategy is effective: Once the beam patterns are as narrow as the image you are looking at, making them narrower (using a bigger array) ceases to help. Another way of looking at that is that the *spatial coherence* of the signal is reduced. When the signals received at opposite edges of the array have a low statistical correlation with one another, a bigger array no longer helps to separate the signal from the background.

Time spreading arises from the same physical mechanisms. The sound paths via near-specular facets are slightly longer than the path via the specular point, so they arrive slightly later. Compared to sound that propagates without reflecting, a short pulse that reflects from a rough surface has more ping-to-ping variability, and on average it has a slower rise time and a relatively long tail.

In active sonar, an important clue for detecting a target in background is that the target signal is more localized in arrival time. A strategy for improving detectability is to increase the bandwidth of the signal, and then use replica correlation to compress the echo in time. The width of the echo in time is, at narrowest, roughly the inverse of the signal bandwidth. Increasing the bandwidth makes the echo narrower in time, making it stand out better from the background – until the width gets down to the natural time-spreading width of the oceanic channel. After that, increasing the pulse bandwidth no longer helps. Another way of looking at that is that the *frequency coherence* of the signal is reduced. When the signal at the low end of the frequency band is no longer correlated with the signal at the high end of the band, replica correlation (and similar coherent processing schemes) cease to improve the signal-to-noise ratio with increasing bandwidth.

Doppler spreading arises because the channel is not stationary in time. For surface-reflected paths, motion of the surface itself broadens the distribution of Doppler shifts, and hence of frequencies. In addition, if the source or receiver is moving, the Doppler from one part of the near-specular scattering patch can differ from the Doppler from another part, simply due to the differences in angle. Further, the motion of the specular point along the rough boundary produces a random distribution of Doppler shifts.

In active sonar, an important clue for detecting a moving target in background is the Doppler shift of the echo. A strategy for improving detectablilty is to use a longer transmit pulse, and then using replica correlation to compress the echo in Doppler. The Doppler width of the echo is, at narrowest, roughly the inverse of the pulse length (in frequency terms). For a given center frequency, increasing the pulse length makes the echo narrower in Doppler, making it stand out from the background – until the Doppler resolution gets down to the natural Doppler width of the ocean channel.

After that, longer pulses no longer help. Another way of looking at that is that the *time coherence* of the signal is reduced. When the signal at the leading edge of the echo is no longer correlated with the signal at the trailing edge, coherent processing ceases to improve signal-to-noise ratio with increasing pulse length for moving targets.

Thus, for all three types of spreading, the notions of *spreading* and *coherence* are two sides of the same coin. Directional spreading implies reduced spatial coherence, limiting the gain available from increasing array size. Time spreading implies reduced frequency coherence, limiting the gain available from increasing signal bandwidth. Doppler spreading implies reduced time coherence, limiting the gain available from increasing signal length. Omitting these effects can result in overly optimistic predictions in which the probability of detection is too high.

SST now offers the option of adding some random spreading in time and frequency to **DirectSignal** and **TargetEcho**. The switch that controls this behavior is the *doTimeSpread* attribute of class **EigenrayModel**. In addition, the same switch causes the near-specular parts of the reverberation signal to be omitted. This is necessary because the time spreading arises from the same physical mechanism as near-specular reverberation, so we must omit it from the reverberation model to avoid double-counting it. The default value of *doTimeSpread* is *true*; i.e. this randomness is added (at least for **McDanielSurface**) unless you explicitly turn it off.

12.2 Example Without Time Spreading

Figure 3 shows an attempt to reproduce the forward-scattering results of Run 22 of the ASIAEX experiment [Dahl 2002]. We thank Peter Dahl for permission to use his data in this analysis. (That paper is mostly about spatial coherence whereas we are concerned here with frequency coherence, which is the same thing as time spread.)

The top panel of Fig. 3 shows the experimental data from ASIAEX Set 22, and the other three show various kinds of SST simulations designed to reproduce or explain those data. In the experiment, the transmitter repeatedly broadcast a three-millisecond pulse at a frequency of 8 kHz in shallow water on a day in which the wind speed was low. The receiver, located about 500 m from the transmitter, could distinguish four peaks corresponding to four eigenrays from the transmitter to the receiver: a direct path, one with a single surface bounce, one with a single bottom bounce, and two with one bounce from each boundary (surface-bottom, then bottom-surface).

The top panel shows the sound intensity observed in the experiment, as a function of time after the first arrival. The red lines represent individual pings from a set of twenty. The blue line running through the middle of that jumble is a 20-ping intensity

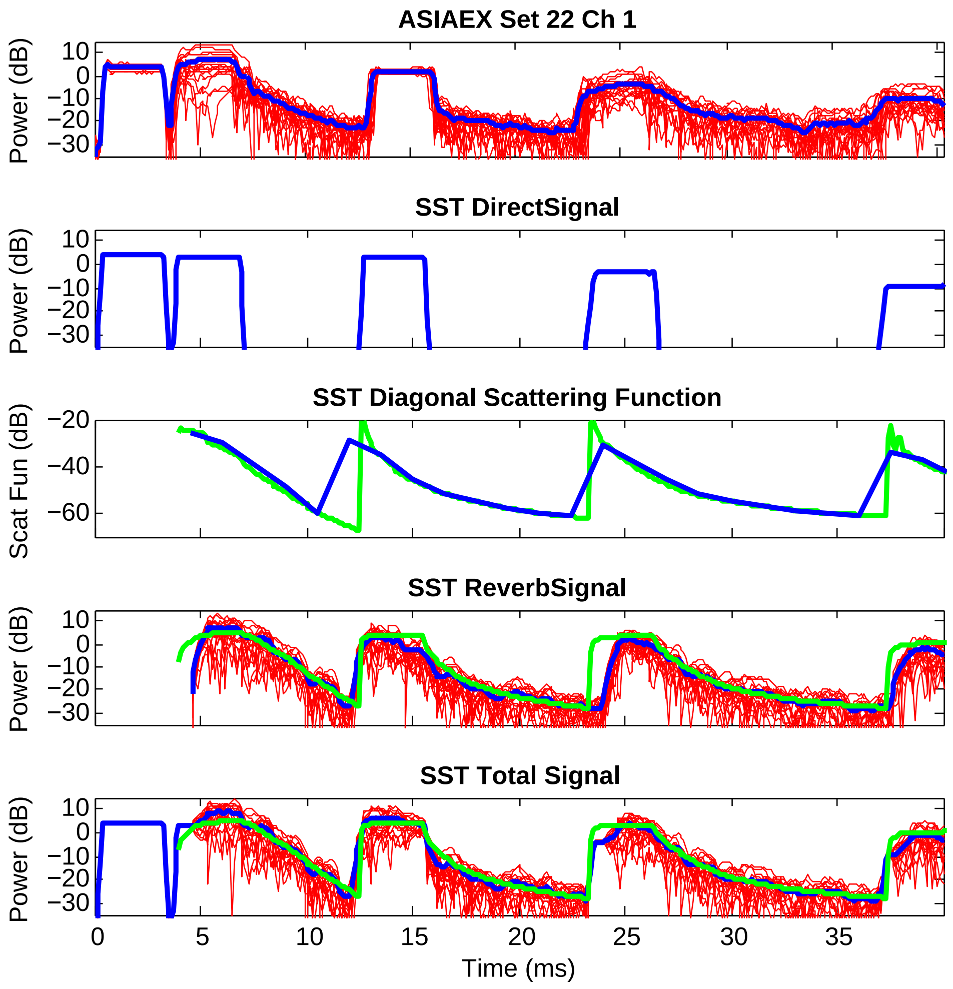


Figure 3: ASIAEX Simulation Without Time Spreading

average. The first and third peaks (direct path and bottom bounce) show very little ping-to-ping variation, whereas the others are highly variable because the surface moves. In addition, if you look at the average (blue line), all of the peaks except the direct one show some broadening in time: The leading edge rises more gradually than the direct peak, and the falling edge has a long tail. This "time spread" happens because the surface and bottom are rough. The specular reflection (the first arrival) is followed by a large number of smaller reflections from "facets" of the surface that are increasingly far from the specular point. This effect is more pronounced for the surface than for the bottom because the scale of surface roughness is much larger than that of the bottom roughness.

The second panel shows an SST simulation of the same scenario, using **DirectSignal** without time spread, as described in Sec. 8.1. This, too, is a 20-ping average, but there is no ping-to-ping variation. The leading and trailing edges of the peaks are all equally crisp. Physically, each reflection is treated as if the surface and bottom were slightly absorbing but otherwise perfect mirrors. The received signals are completely coherent.

The third panel shows the scattering function computed by SST's **BBBScatFun** (Sec. 11.3.2), again with time spread disabled. The blue and green lines are computed using update intervals of 1.5 and 0.1 ms respectively.

The fourth panel shows SST's simulated bistatic reverberation, as generated using **ReverbSignal** (Sec. 11.3.1). As in the top panel, the red lines are individual pings and the blue line is a 20-ping average. The green line is the expected average value, which was computed by convolving the high-resolution scattering function (the green line from the third panel) with the square pulse power envelope. The data were produced using an update interval of 1.5 ms (i.e., the scattering function used as input was the blue line in the third panel).

The fifth panel shows the total simulated signal, direct (second panel) plus reverberation (fourth panel).

12.3 Critique: Why We Need Time Spread

SST, in Release 4.1 and earlier, treated forward reflection and scattering as completely separate phenomena. Reflection was treated as specular, or mirror-like; the corresponding signal, produced by **DirectSignal**, had no randomness at all. It was completely coherent. As a consequence, the processing gain from replica correlation was the same for bounce paths as for the direct path.

On the other hand, scattering was handled entirely by the reverberation model: **BBBScatFun** and **ReverbSignal**. The resulting signal was completely incoherent. As a consequence, the processing gain from replica correlation was the same as for Gaussian noise with the appropriate power spectrum. In fact, that is how it was generated.

This is still true if time spreading is disabled (*doTimeSpread* is *false*).

The update interval used to generate reverberation is always a compromise. If it is too fast (as in the green curve in Fig. 3), it can't be used as input for the generation process of **ReverbSignal** because the resulting reverberation spectrum would be too wide. If it is too slow (as in the blue curve), it can't follow transients. In Fig. 3, the initial transient in the scattering function comes just a little too late for the surface and too early for the bottom, and the rise times are too slow. The consequences are obvious in the total (last panel): The first millisecond of the surface echo and the last millisecond of the bottom echo are dominated by the **DirectSignal**, which reduces the ping-to-ping variablility – which happens to be an improvement for the bottom but not the surface. In addition, both the surface and bottom returns have regularly spaced lumps in their tails.

The theory was that the "correct" signal should be the sum of the reflected and scattered components; i.e. the coherent sum of the second and fourth panels of the figure above. The hope was that the facts that the **DirectSignal** result was too coherent and the **ReverbSignal** result wasn't coherent enough should somehow average out to something reasonably realistic, without double-counting the energy.

That scheme works acceptably well for many scenarios and purposes, but it is obviously inadequate to reproduce coherence-sensitive forward scattering results like those shown here.

The fundamental problem is that SST's reverberation algorithm is built on the assumption of quasi-stationarity: It depends on the assumption (mentioned in Sec. 4.6.1) that an update interval Δ exists that simultaneously satisfies two conditions: The scattering function does not change significantly in a time Δ, and the frequency spreading due to the combination of Doppler and pulse-width effects is larger than $1/\Delta$. In this particular case, the Doppler spreading width for the surface peak is less than 10 Hz (Ref. [APL Models 1994], Eq. II-43a), so the frequency width is determined by the short pulse length. Our "slow" update interval of 1.5 ms (two updates per pulse length) gives the generated signal about the right frequency width. However, that interval is obviously too long to follow the near-specular transients. The final conclusion is:

Both SST's coherent forward reflection model and SST's reverberation model are inaccurate for near-specular forward scattering.

SST's time spread model is designed to fix that problem, at least in part.

12.4 The Time Spreading Algorithm

If the **EigenrayModel** attribute *doTimeSpread* is *true* (the default), algorithms used for both "direct" propagation (**DirectSignal**, including within **TargetEcho**) and reverberation (**BBBScatFun**, **BBBDirectionalScat**, and **DopplerDensityDirectional**) are modified. The effects of near-specular forward boundary scattering are removed from the reverberation model and added to the forward sound transformation model (**DirectSignal**). The main advantage of this shift is that the time spread algorithm added to **DirectSignal** is more nearly correct. The reverberation model in **ReverbSignal** is inaccurate for times when the scattering function is changing rapidly – which is true for the time period immediately following the specular reflection. Another advantage of this realignment is that it reduces the coherence of one-way signals (**DirectSignal**) and active target echoes (**TargetEcho**) that arrive via boundary-bounce paths.

In **DirectSignal** (and by extension in **TargetEcho**, which uses **DirectSignal** internally), the time spread algorithm effectively adds a stochastic FIR filter to the processing chain.

For each bounce from the surface or bottom, the total forward reflection coefficient is split into a coherent part and an incoherent part, as outlined in Sec. II-6 of Ref. [APL Models 1994]. The coherent part is treated deterministically, as described in Sec. 8.1, except the frequency dependence includes that of the division between coherent and incoherent parts.

The incoherent part of the signal is filtered by a stochastic FIR filter. The filter coefficients are Gaussian random numbers whose expected values are determened by an "intensity impulse response" function, which is computed as outlined in Sec. II.7a of Ref. [APL Models 1994]. The random numbers used to generate the filter coefficients change slowly, on a time scale determined by the frequency spreading width computed as in Sec. II.8 of Ref. [APL Models 1994]. This slow variation is accomplished by passing white Gaussian noise through an AR-1 filter. The resulting stochastic filter is convolved with a deterministic filter that implements the frequency dependence of the eigenray propagation, as modified by the division between coherent and incoherent parts.

The two filters, for the coherent and incoherent fractions, are summed. The resulting total filter is used as in Sec. 8.1 to modify the sound produced by **DirectSignal** or **TargetEcho**.

Qualitatively, the effect of the time spread is that the sound reflected from a boundary arrives not with a single delay but with a range of delays, from the "specular" (minimum) delay to a few milliseconds later. The characteristic spreading time depends on the reflection geometry and on the boundary roughness. In the frequency domain, the coherence width of the sound is reduced to roughly the inverse of the spreading time.

SST Classes: The master on-off switch (*doTimeSpread*) is located in **EigenrayModel**, which is used by both the direct propagation code and the reverberation code. It is inherited by all of the **EigenrayModel** subclasses. The time spreading model is logically part of the sound propagation model.

SST Classes: The parameters that control the time spreading algorithm are part of the **Boundary** model. For **McDanielSurface** (a subclass of **Boundary**), the time spread parameters are computed by that surface model, and are determined by the wind speed. For all other **Boundary** subclasses, three additional user-settable parameters control the time and frequency spread:

- *rmsHeight* controls the partition of the total reflected energy between coherent and incoherent fractions. The key quantity is the wavelength divided by the sine of the grazing angle. If *rmsHeight* is larger than that quantity, the reflection is mostly incoherent. If *rmsHeight* is smaller, it is mostly coherent. At weapon frequencies, incoherent reflection dominates for all but the calmest conditions.

- *meanSquareSlope* controls the time scale with which the incoherent fraction of the intensity decays after the initial onset (the length of the "tails" in the figures). As a rule of thumb, the characteristic spreading time is roughly equal to the mean square slope times half of the total one-way propagation time if the reflection is near the middle of the path. That time decreases for more asymmetric paths; for a reflection near one end of the path, the characteristic spreading time is roughly the mean square slope times twice the one-way propagation time along the shorter leg of the path.

- *rmsVerticalSpeed* controls the Doppler frequency spread. For moving sonars, Doppler spread comes from two components: Motion of the surface itself and motion of the reflection point along the surface. The *rmsVerticalSpeed* determines the former. The characteristic RMS speed for the latter component is given by the velocity of that motion times the square root of *meanSquareSlope*.

The total effective RMS vertical speed is the square root of the sum of squares of the two components. The effective Doppler frequency spread is the acoustic frequency times the total RMS vertical speed, divided by the product of the sound speed and the sine of the grazing angle.

The default values of all three paramters are zero, so by default only **McDanielSurface** produces time and frequency spread.

As a curiosity, for **McDanielSurface**, motion of the reflection point across the rough surface produces roughly the same Doppler broadening as a wind of comparable speed. As far as I know, that correspondence is entirely accidental.

12.5 Example With Time Spreading

Figure 4 is just like Fig. 3, except that time spreading has been turned on.

The most obvious difference from Fig. 3 is that the **DirectSignal** result (second panel) shows time spread and ping-to-ping variation. The second difference is that the peaks in the scattering function (second panel) have been surgically removed. The result is that the corresponding peaks in the generated reverberation are gone (fourth panel), leaving only relatively smooth, long tails that start several milliseconds later than the original abrupt peaks. The total signal (fifth panel) consists mostly of the **DirectSignal** result, which looks a lot more like the data (first panel) than in Fig. 3 due to its newly acquired variability and tails.

That last assertion is more obvious in Fig. 5. The top panel shows the data, yet again. The middle panel shows the simulation total without time spread (as in the bottom panel of Fig. 3), and the bottom panel shows the simulation total with time spread (as in the bottom panel of Fig. 4).

12.6 Time Spreading Critique

First let's focus on the surface return (the second peak). The level and decay rate of the tail are approximately right – which, I must confess, is because I adjusted the wind speed to make it right. The measured wind speed was quoted as 4 m/s, but 3 m/s fits the tail better. On the other hand, the ping-to-ping variability of the main part of the peak is lower than in the measurements; in that respect, the original wind speed of 4 m/s gave much better agreement.

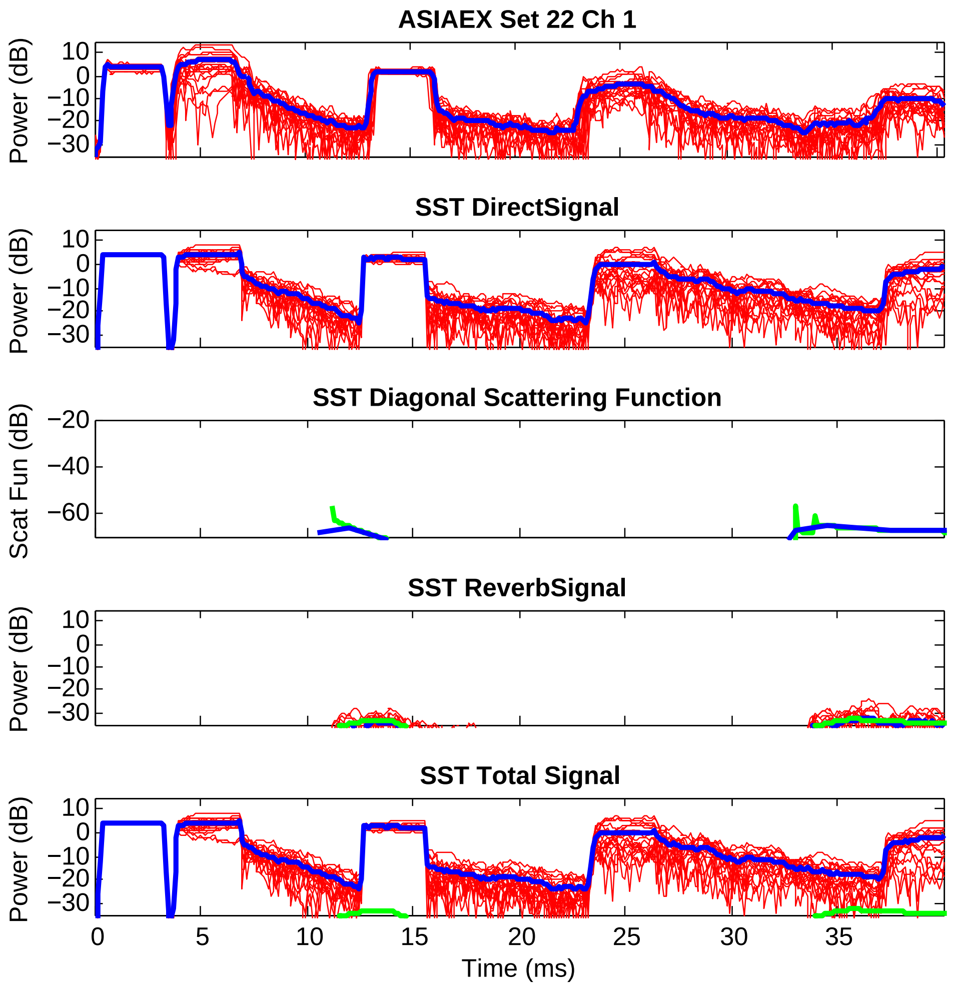


Figure 4: ASIAEX Simulation With Time Spreading

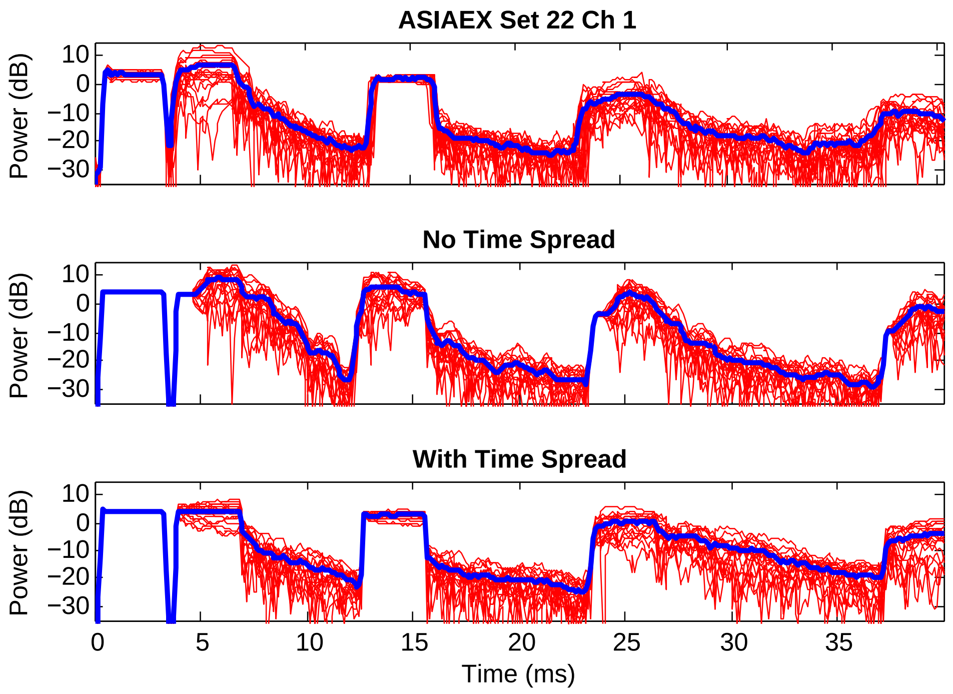


Figure 5: ASIAEX Comparison With and Without Time Spreading

Those two visually separable aspects of the data, as it happens, are controlled by two different parameters that are computed by the **McDanielSurface** model: The decay time of the tail is determined by the mean square slope, attribute *meanSquareSlope*. The coherent fraction, and hence the variability of the main peak, is determined by the root mean square surface height, attribute *rmsHeight*. In **McDanielSurface**, both of those parameters depend only on the wind speed. The fact that two different wind speeds are required to separately reproduce those two visible aspects of the signal can be explained on physical grounds: Apparently the medium-scale ripples (a few acoustic wavelengths in length) are characteristic of a wind speed of 3 m/s, whereas the average height of the waves is characteristic of a wind speed of 4 m/s. That is quite reasonable in the presence of a swell generated by earlier wind conditions or by winds far away. Both wind speeds are quite low, so we are not talking about large swell, and the difference is more or less typical of the variability of light breezes.

Now let us focus on the bottom return (the third peak). Here the simulation reproduces both visible aspects – the decay rate of the tail and the variablility of the main peak – reasonably well. There is a good reason: I cheated. The **JacksonBottom** model does not yield values for the two key parameters (or at least I have not figured out how to get those values out of it), so I adjusted the values to get a visual fit to the data. The following lines in the input file `asiaex22_common.sst` tell the story:

```
bottom = FineSand {
    rmsHeight = 0.012           # These are fit to the data by eye
    meanSquareSlope = 0.045
}
```

In fact, **McDanielSurface** is the only boundary model for which *rmsHeight* and *meanSquareSlope* are computed by the model. For all other **Boundary** subclasses, you are free to enter any values at all for those parameters. The default values are zero, which gives the old, fully coherent behavior for **DirectSignal** and fully incoherent reverberation.

Now turn your attention to the multiple-bounce peaks (the last two). The general degree of time spread is about right, but the shape is not: The intensity is too high immediately after the pulse and too low later. We understand this, and intend to fix it.

SST Example: The SST and Matlab scripts that generated the figures in this section are distributed with SST as the *timespread* example. The corresponding page in the SST Web [SST Web] explains how to run the example simulation and plot the results.

13 Summary and Plans

The Sonar Simulation Toolset is a mature, well-supported software product that has contributed to many Navy projects since its first release in 1989. It produces sound, suitable for listening or for feeding into a sonar front end, using commonly available computers. SST is portable, general, broadband, bistatic, multi-channel, embeddable, streamable, object oriented, unclassified, and offers flexible fidelity. It is available to any DoD agency or contractor.

SST is actively supported and continuously improving. Development is always driven by users' requirements and sponsors' priorities. We conclude with some directions that SST may move from here.

Performance Tuning and Parallel Processing: SST is not intended as real-time software. Nevertheless, speed is important to users. Current funding from ONR 333 includes performance tuning, including work toward parallel algorithms based on the MPI library [Gropp Lusk Skjellum 1999]. This will provide dramatic performance increases for multi-processor computers and Beowulf-style [Beowulf] clusters of commodity computers.

Interoperability: SST lies in the middle of the modeling and simulation hierarchy because it is both a consumer of other people's models and a producer of signals for input to higher-level simulations like TRM. Hence, portability and interoperability are especially important concerns. The TEAMS [TEAMS] (and other) standardization efforts will guide us in making SST more open, both from below and from above.

Server Mode: SST is currently "embeddable" in a limited sense. To use SST as a signal generation component within a higher-level simulation, the host simulator (e.g. TRM [Cerenzia et al. 2005]) must generate SST commands, feed them to SST, and read the signal files that SST produces. This approach is complex, inefficient, and insufficiently dynamic. For example, it does not allow the host simulator to "change its mind" and initiate a maneuver during a listening interval. We intend to improve this situation in several steps: first to allow more dynamic "closed-loop" behavior, then to return signals through inter-process communication channels instead of files, and eventually to allow SST to be treated as a library of software components, bypassing the command language.

Wakes: Ship wakes are a huge, complex problem. Current funding supports simple modeling of wakes, but that is only a start. Fundamental, innovative changes to current modeling approaches will be necessary to model wakes more accurately and comprehensively.

Updating Component Models: Improving the fidelity of SST's underlying models is a continuing task, driven both by improvements in our scientific understanding and by the requirements of new SST applications. Examples of improvements that should be included in SST are the small-slope approximation for rough-surface scattering [Thorsos Broschat 1995] and elastic bottom scattering [Jackson Ivakin 1998]. In addition, SST's support for very long passive scenarios is incomplete, and SST's practice of treating volume scattering layers as thin sheets of scatterers produces artifacts early in the return in some situations. These and similar shortcomings must be corrected.

Lower Frequencies: Users have asked for SST-like signal level simulation tools for lower frequencies, and a project to improve SST's low-frequency realism is under way. A bottom model with penetration to sub-bottom layers is being incorporated into SST, and other "incremental" improvements are planned. A more radical change would be to incorporate a wave-based propagation model.

Verification, Validation, and Accreditation: Making sure that SST simulations reflect reality with useful accuracy is a continuing effort. We are always looking for challenging data sets that are sensitive tests of SST's unique capabilities.

Support: We support SST users by answering their questions, helping them set up simulations, tracking down mysteries, fixing bugs, and distributing releases. In return, our users keep us in touch with what is important or not, what works well or doesn't, what seems easy or hard to do, and what needs fixing or enhancement. User support is one of the most important components in each SST contract.

More Applications: We feel that only a small fraction of the Navy users who could benefit from SST are using it. Our primary reward for the work that we do is to see it make a positive difference to the national defense. Torpedoes, torpedo defense, acoustic communications, shipboard and submarine sonars, bottom-mounted arrays, sonobuoys, and many other classes of systems are candidates for SST simulations. Early idea testing, advanced development, performance prediction, interpretation of experiments, operator training, and many other objectives can be served using realistic simulated sonar signals. We look forward to continuing and expanded service to the U.S. Navy.

REFERENCES

[Abraham Lyons 2002] D. A. Abraham and A. P. Lyons, "Novel Physical Interpretations of K-Distributed Reverberation," IEEE J. Ocean. Eng. **27**, 800-813 (2002).

[Adobe Reader] Adobe Reader, http://www.adobe.com/products/reader/.

[Albers 1965] V. C. Albers, *Underwater Acoustics Handbook – II* (Pennsylvania State University Press, University Park, PA, 1965), 188-189.

[APL Models 1994] "APL-UW High Frequency Ocean Environmental Acoustic Models Handbook," APL-UW TR 9407, Applied Physics Laboratory, University of Washington, Seattle, WA, October 1994.

[Beowulf] *Beowulf: Introduction, History, Overview*, NASA, http://beowulf.gsfc.nasa.gov/overview.html.

[Cerenzia et al. 2005] J. L. Cerenzia, P. H. Lallement, H. B. Miska, S. D. Sheaffer, and F. R. Menotti, "CARLEE Acoustic Signal Processing Architecture,"ARN/PSU Technical Memorandum 05-059, Applied Research Laboratory, The Pennsylvania State University, State College, PA, September 2005

[Chamberlain Galli 1983] S. G. Chamberlain and J. C. Galli, "A Model for Numerical Simulation of Non-Stationary Sonar Reverberation Using Linear Spectral Prediction," IEEE J. Ocean. Eng. **OE-8**, 21-36 (1983).

[Correia 1988] E. Correia, "Weapons Assessment Facility (WAF)", Technology Digest, Naval Undersea Warfare Center Division Newport, September 1988, pp. 85-86.

[Cygwin] *Cygwin^{TM}*, Red Hat, Inc., http://www.redhat.com/software/cygwin/.

[Dahl 1996] P. H. Dahl, "On the spatial coherence and angular spreading of sound forward scattered from the sea surface: Measurements and interpretive model," J. Acoust. Soc. Am. **100**, 748-758 (1996).

[Dahl 1999] P. H. Dahl, "On bistatic sea surface scattering: Field measurements and modeling," J. Acoust. Soc. Am. **105**, 2155-2169 (1999).

[Dahl 2001] P. H. Dahl, "High-Frequency Forward Scattering from the Sea Surface: The Characteristic Scales of Time and Angle Spreading," IEEE J. Ocean. Eng. **26**, 141-151 (2001).

[Dahl 2002] P. H. Dahl, "Spatial Coherence of Signals Forward Scattered from the Sea Surface in the East China Sea," in *Impact of Littoral Environmental Variability on Acoustic Predictions and Sonar Performance*,edited by N. G. Pace and F. B. Jensen (Kluwer Academic Publishers, Netherlands, 2002), pp. 55-62.

[Dean 1966] R. A. Dean, *Elements of Abstract Algebra* (John Wiley and Sons, New York, 1966).

[Devroye 1986] L. Devroye, *Non-Uniform Random Variate Generation* (Springer-Verlag, New York, 1986).

[Donelan Hamilton Hui 1985] M. A. Donelan, J. Hamilton and W. H. Hui, "Directional Spectra of Wind-Generated Waves," Phil. Trans. R. Soc. Lond. **A 315**, 509-562 (1985).

[Doxygen] D. van Heesch, *Doxygen*, http://www.doxygen.org.

[Eggen Goddard 2002] C. Eggen and R. Goddard, "Bottom Mounted Active Sonar for Detection, Localization, and Tracking," in *Proceedings Oceans '02 MTS/IEEE* (IEEE Publication 0-7803-7534-3, 2002), Vol. 3, pp. 1291-1298, http://ieeexplore.ieee.org/Xplore/DynWel.jsp.

[Faure 1964] P. Faure, "Theoretical Model of Reverberation Noise," J. Acoust. Soc. Amer. **36**, 259-268 (1964).

[Gilbert 1993] K. E. Gilbert, "A stochastic model for scattering from the near-surface oceanic bubble layer," J. Acoust. Soc. Am. **94**, 3325-3334 (1993).

[Gnuplot] T. Williams and C. Kelley, *gnuplot*, http://www.gnuplot.info/.

[Goddard 1986] R. P. Goddard, "REVGEN-4 High-Fidelity Simulation of Sonar Pulses," APL-UW 8505, Applied Physics Laboratory, University of Washington, Seattle, WA, June 1986.

[Goddard 1989] R. P. Goddard, "The Sonar Simulation Toolset," in *Proceedings Oceans '89, The Global Ocean* (IEEE Publication Number 89CH2780-5, 1989), Vol. 4, pp. 1217-1222.

[Goddard 1993] R. P. Goddard, "Simulating Ocean Reverberation: A Review of Methods and Issues," APL-UW 9313, Applied Physics Laboratory, University of Washington, Seattle, WA, October 1993.

[Goddard 2000] R. P. Goddard, "SST Bistatic Reverberation Modeling: North Sea Experiment, April-May 1998," APL-UW TM 5-00, Applied Physics Laboratory, University of Washington, Seattle, WA, May 2000.

[Goldstein 1950] H. Goldstein, *Classical Mechanics* (Addison-Wesley, Reading, MA, 1950).

[Golub Van Loan 1996] G. H. Golub and C. F. Van Loan, *Matrix Computations* (Johns Hopkins University Press, Baltimore, MD, 1996), 3rd ed.

[Gropp Lusk Skjellum 1999] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface* (MIT Press, Cambridge, MA, 1999), 2nd ed.

[Hamming 1973] R. W. Hamming, *Numerical Methods for Scientists and Engineers* (McGraw-Hill, New York, 1973), 2nd ed.

[Hodgkiss 1984] W. S. Hodgkiss, "An Oceanic Reverberation Model," IEEE J. Ocean. Eng. **OE-9**, 63-72 (1984).

[Hodgkiss 1989] W. S. Hodgkiss, "A Modular Approach to Exploratory Data Analysis," in *Proceedings Oceans '89, The Global Ocean* (IEEE Publication Number 89CH2780-5, 1989), pp. 1100-1104.

[Jackson Ivakin 1998] D. R. Jackson and A. N. Ivakin, "Scattering from elastic sea beds: First-order theory," J. Acoust. Soc. Am. **103**, 336-345 (1998).

[Katyl 2000] D. Katyl, "Distributed Systems at the Weapons Analysis Facility", Simulation Technology Magazine, Vol. 2, Issue 4b, June 28, 2000, http://www.sisostds.org/webletter/siso/Iss_62/.

[Knight 1981] W. C. Knight, R. G. Pridham, and S. M. Kay, "Digital Signal Processing for Sonar," Proc. IEEE **69**, 1451-1506 (1981).

[Knuth 1973] D. E. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching* (Addison-Wesley, Reading, MA, 1973), Sec. 5.2.3.

[Kulbago 1994] L. J. Kulbago, "A Study of Acoustic Backscatter from the Near-Surface Oceanic Bubble Layer," Master of Engineering thesis, The Pennsylvania State University, State College, PA, 1994.

[Lang Culver 1992] D. C. Lang and R. L. Culver, "A High Frequency Bistatic Ocean Surface Scattering Strength," ARL-PSU TM 92-342, Applied Research Laboratory, Pennsylvania State University, State College, PA, December 1992.

[Luby Lytle 1987] J. C. Luby and D. W. Lytle, "Autoregressive Modeling of Nonstationary Multibeam Sonar Reverberation," IEEE J. Ocean. Eng. **OE-12**, 116-129 (1987).

[McDaniel 1990] S. T. McDaniel, "Models for Predicting Bistatic Surface Scattering Strength," ARL-PSU TM 90-88, Applied Research Laboratory, Pennsylvania State University, State College, PA, March 1990.

[Mackenzie 1959] K. V. Mackenzie, "Reflection of sound from coastal bottoms," J. Acoust. Soc. Am. **32**, 221-231 (1959).

[Mathematica] *Mathematica*, Wolfram Research, Inc., http://www.wolfram.com/products/mathematica.

[Matlab] *Matlab*, The MathWorks, Inc., http://www.mathworks.com/products/matlab/.

[Middleton 1967] D. Middleton, "A Statistical Theory of Reverberation and Similar First-Order Scattered Fields. Part I: Waveforms and the General Process," IEEE Trans. Inf. Theory **IT-13**, 372-392 (1967).

[Mitchell McPherson 1981] R. L. Mitchell and D. A. McPherson, "Generating Nonstationary Random Sequences," IEEE Trans. Aerospace Electron. Syst. **AES-17**, 553-560 (1981).

[Moe Jackson 1994] J. E. Moe and D. R. Jackson, "First-order perturbation solution for rough surface scattering cross section including the effects of gradients," J. Acoust. Soc. Am. **96**, 1748-1754 (1994).

[Mourad Dahl Jackson 1991] P. D. Mourad, P. H. Dahl, and D. R. Jackson, "Bottom Backscatter Modeling and Model/Data Comparison for 100-1000 Hz," APL-UW TR 9107, Applied Physics Laboratory, University of Washington, Seattle, WA, September 1991.

[Mourad Jackson 1989] P. D. Mourad and D. R. Jackson, "High Frequency Sonar Equation Models for Bottom Backscatter and Forward Loss," in *Proceedings OCEANS 89, The Global Ocean* (IEEE Publication Number 89CH2780-5, 1989), pp. 1168-1175.

[Mourad Jackson 1993] P. D. Mourad and D. R. Jackson, "A model/data comparison for low-frequency bottom backscatter," J. Acoust. Soc. Am. **94**, 344-358 (1993).

[Octave] *Octave*, University of Wisconsin, http://www.octave.org/.

[Ol-shevskii 1967] V. V. Ol-shevskii, *Characteristics of Sea Reverberation* (Consultants Bureau, New York, NY, 1967).

[Oppenheim Schafer 1989] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing* (Prentice-Hall, Englewood Cliffs, NJ, 1989).

[Page-Jones 2000] M. Page-Jones, *Fundamentals of Object-Oriented Design in UML* (Addison-Wesley, New York, NY, 2000).

[Princehouse 1975] D. W. Princehouse, "REVGEN, A Real-time Reverberation Generator: Concept Development," APL-UW 7511, Applied Physics Laboratory, University of Washington, Seattle, WA, September 1975.

[Princehouse 1978] D. W. Princehouse, "Reverberation Generator Algorithm, A Status Report," APL-UW 7806, Applied Physics Laboratory, University of Washington, Seattle, WA, February 1978.

[Rouseff et al. 2001] D. Rouseff, D. R. Jackson, W. L. J. Fox, C. D. Jones, J. A. Ritcey, and D. R. Dowling, "Underwater Acoustic Communication by Passive-Phase Conjugation: Theory and Experimental Results," IEEE J. Ocean. Eng. **26**, 821-831 (2001)

[Sammelmann 2002] Gary Steven Sammelmann, "PC SWAT 7.0: Users' Manual," CSS Draft Report, Coastal Systems Station Code R22, Panama City, FL, February 2002.

[Schulten Anderson Gordon 1979] Z. Schulten, D. G. M. Anderson, and R. G. Gordon, "An Algorithm for the Evaluation of the Complex Airy Functions," J. Comp. Phys. **31**, 60-75 (1979).

[SST Web] R. P. Goddard, "The Sonar Simulation Toolset Web, Release 4.6", APL-UW TM 10-96 (Electronic Document), REVISED October 2007, Applied Physics Laboratory, University of Washington, Seattle, WA, October 2007.

[Stroustrup 2000] B. Stroustrup, *The C++ Programming Language* (Addison-Wesley, Reading, MA, 2000), Special Ed.

[Taylor 1955] T. T. Taylor, "Design of Line Sources Antennas for Narrow Beamwidth and Low Sidelobes," IRE Transaction on Antennas and Propagation **AP-3**, 16-28 (1955). Also in C. A. Balonis, *Antenna Theory — Analysis and Design* (Wiley & Sons, New York, 1997), pp. 358-362.

[TEAMS] *Undersea Warfare TEAMS*, led by ONR 333, ARL/PSU, and NUWC (Newport, RI), `http://uswteams.arl.psu.edu/homepage/Homepage.svlt`.

[Thorsos Broschat 1995] E. I. Thorsos and S. L. Broschat, "An investigation of the small slope approximation for scattering from rough surfaces. Part I. Theory," J. Acoust. Soc. Am. **97**, 2082-2093 (1995).

[Urick 1983] R. J. Urick, *Principles of Underwater Sound* (McGraw Hill, New York, NY, 1983), 3rd Ed.

[Weinberg 1985] H. Weinberg, "Generic Sonar Model," NUSC TD 5971D, Naval Underwater Systems Center, New London, CT, June 1985.

[Weinberg Keenan 1996] H. Weinberg and R. E. Keenan, "Gaussian ray bundles for modeling high-frequency propagation loss under shallow-water conditions," J. Acoust. Soc. Am. **100**, 1421-1431 (1996).

[Weinberg et al. 2001] H. Weinberg, R. L. Deavenport, E. H. McCarthy, and C. M. Anderson, "Comprehensive Acoustic System Simulation (CASS) Reference Guide", NUWC-NPT TM 01-016, Naval Undersea Warfare Center Division, Newport, RI, March 2001.

[Williams Jackson 1998] K. L. Williams and D. R. Jackson, "Bistatic bottom scattering: Model, experiments, and model/data comparison," J. Acoust. Soc. Am. **103**, 169-181 (1998).

REPORT DOCUMENTATION PAGE	*Form Approved* *OPM No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** October 2008	**3. REPORT TYPE AND DATES COVERED** Technical Report

4. TITLE AND SUBTITLE The Sonar Simulation Toolset, Release 4.6: Science, Mathematics, and Algorithms	**5. FUNDING NUMBERS** ONR N00014-07-G-0557, N00014-01-G-0460, and N00014-98-G-0001
6. AUTHOR(S) Robert P. Goddard	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Applied Physics Laboratory, University of Washington, 1013 NE 40th Street, Seattle, WA 98105-6698	**8. PERFORMING ORGANIZATION REPORT NUMBER** APL-UW TR 0702
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Donald J, McDowell, Jr., ONR Code 332, 875 N. Randolph Street, Ste. 1425, Arlington, VA 22203-1995	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION / AVAILABILITY STATEMENT *Approved for public release; distribution is unlimited*	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(Maximum 200 words)*

The Sonar Simulation Toolset (SST) is a computer program that produces simulated sonar signals, enabling users to build an artificial ocean that sounds like a real ocean. Such signals are useful for designing new sonar systems, testing existing sonars, predicting performance, developing tactics, training operators and officers, planning experiments, and interpreting measurements. SST's simulated signals include reverberation, target echoes, discrete sound sources, and background noise with specified spectra. Externally generated or measured signals can be added to the output signal or used as transmissions. Eigenrays from the Generic Sonar Model (GSM) or the Comprehensive Acoustic System Simulation (CASS) can be used, making all of GSM's propagation models and CASS's Baussian Ray Bundle (GRAB) propagation model available to the SST user. A command language controls a large collection of component models describing the ocean, sonars, noise sources, targets, and signals. The software runs on several different UNIX computers. The software runs on several UNIX computers, Windows, and Macintosh OS X. SST's primary documentation is the SST Web (a large HTML "web site" distributed with the SST software), supported by a collection of documented examples.

This report emphasizes the science, mathematics, and algorithms underlying SST. This report is intended to be updated often and distributed with SST as an integral part of the SST documentation.

14. SUBJECT TERMS sonar, signal, reverberation, target echo, Generic Sonar Model (GSM), Comprehensive Acoustic System Simulation (CASS), Gaussian Ray Bundle (GRAB)	**15. NUMBER OF PAGES** 121
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	SAR