Simulation of Coherent Phase Optical Time-Domain Reflectometry

By

Michael Brown

A Thesis submitted to the Faculty of Graduate Studies of

the University of Manitoba,

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

University of Manitoba

Winnipeg

Acknowledgments

I dedicate this work to my family. To my wife Natasha, who supported me throughout all my endeavors and had motivated me when I needed it most. And to my children Ethan and Rachel, to whom I owe many hours of father-son and father-daughter time.

I would also like to thank Dr. Sherif for our many useful discussions, friendship, and his many kind words.

Table of Contents

List of Acronyms

ADC – Analog to Digital Converter

AOM – Acousto-Optic Modulator

AR – Auto-Regressive

BW – Bandwidth

DOFS – Distributed Optical Fiber Sensing

FFT – Fast Fourier Transform

FUT – Fiber Under Test

GUI – Graphical User Interface

IF – Intermediate Frequency

LO – Local Oscillator

LPF – Low Pass Filter

MI – Michelson Interferometer

NA – Numerical Aperture

OTDR – Optical Time Domain Reflectometer

OFDR – Optical Frequency Domain Reflectometer

PDR – Polarization Diversity Receiver

PGC – Phase Generated Carrier

PMD – Polarization Mode Dispersion

PSD – Power Spectral Density

RIN – Relative Intensity Noise

SMF – Singlemode Fiber

SNR – Signal to Noise Ratio

SOP – State of Polarization

TVAR – Time Varying Auto-Regressive

α_S – Scattering Coefficient

α_{RS} – Rayleigh Scattering Coefficient

α_A – Attenuation Coefficient

$B(z)$ – Backscatter Capture Fraction

c – Speed of light

λ - Laser wavelength

k – wavevector of light

ε_0 – Electric permittivity of free space

ε_r – Relative electric permittivity

ε - Electric permittivity

ε - Strain

E_0 – Magnitude of initial energy of electric field

μ - Magnetic permeability

χ - Electric susceptibility

$\Delta\chi$ - Fluctuation of electric susceptibility

α - Polarizability of a molecule

V – Scattering volume

ΔV – Scattering sub-volume

v – Optical frequency

Δv – Optical linewidth

m_{mol} – Mass per molecule

v_s - Specific volume

ρ - density

S – entropy

p – pressure

p – magnitude of electric dipole

β_S – Adiabatic Compressibility

β_T – Isobaric Compressibility

σ - Coefficient of volume expansion

K – Boltzmann's Constant

c_p – specific heat

γ_p – Photo-elastic coefficient

n – refractive index

β - Chirp rate of chirped signal

T_P – Pulse duration

W_P – Pulse width

v_g – group velocity

p_{12}, p_{11} – Photo-elastic coefficients of singlemode optical fiber

ξ - Poisson ratio of singlemode optical fiber

$U(t;z)$ – Electric Field at location z at time t

U_{PD} – Electric Field at photo-detector

I_{PD} – Intensity at photo-detector

M – Polarization state forward transformation matrix

$\mathbf{M_R}$ – Polarization state backward transformation matrix (reciprocal)

$\mathbf{R}(\theta\mathbf{n})$ – Rotation matrix (counter-clockwise rotation of angle θ about a vector $\mathbf{n}$)

$\mathbf{S}_L(0)$ – Polarization state vector of launched pulse

$\mathbf{S}_R(z)$ – Polarization state vector of light returned from location z

$\mathbf{S}_{LO}(z)$ – Polarization state vector of light of local oscillator mixed with light from location z

$f_p[n]$ – forward predictor stage, p, of lattice filter

$g_p[n]$ – backward predictor stage, p, of lattice filter

k_p – reflection coefficients of lattice filter

a_k – AR coefficients

List of Tables

List of Figures

Introduction

Distributed optical fiber sensing (DOFS) is a set of technologies that are characterized by their ability to use an optical fiber as the sensing mechanism [1]. This allows for measurements to be obtained from any spatial location along the fiber. This differs from discrete spatial sensors that only allow information to be obtained from discrete points.

The field of distributed optical sensing has seen increased activity in recent years [2]. This is not surprising since optical sensing offers several benefits over other sensing technologies for certain applications. It can be used in environments where wired, or wireless technologies cannot always be effectively deployed, such as in high temperature environments or in magnetic fields. It also allows for sensing over extremely long distances without the need for batteries or power distribution to remote locations. In many cases, the sensitivity of optical sensors can also exceed that of other technologies [3], [4]. And with the decreasing cost of high speed electronics and processing, as well as the ability to perform real-time analysis without the need for more expensive analog means, it is no surprise that interest in this technology continues to grow and is seeing its use expand into several fields, such as intrusion detection [5], [6], structural health monitoring [7]–[10], environmental monitoring [11]–[13], oil & gas pipeline monitoring [14], [15], and train monitoring [16], just to name a few.

There are several methods used for distributed optical sensing; however, the majority of them operate by detection of backscattered light. In particular, there has been increasing interest in systems that operate by detection of coherent Rayleigh backscatter. These systems are generally referred to as phase-based reflectometers since the detected intensity of the backscattered light is

highly dependent on its phase components due to its coherent nature. These phase-based systems represent a large portion of the current research into distributed optical sensing and will be the focus of this thesis.

Research Motivation

Although there are many papers describing the development and use of distributed optical sensing systems for various applications, most of the literature fails to provide an adequate description of the fundamental physics of light scattering required to understand the technology used in these systems.

In addition to the absence of background literature covering the physics, there is also a lack of detail regarding simulations for distributed optical sensing systems. Many papers discuss the results of simulations, but details of how the simulations were constructed are generally scarce.

This thesis is the authors an attempt to fill the apparent gaps in the existing literature as well as a providing a detailed description of a simulator created for simulating a coherent phase-based optical time domain reflectometer (OTDR).

This thesis is comprised of five chapters. The first chapter gives an overview of the physics of optical scattering in an optical fiber. The second chapter gives an in-depth explanation of the different kinds of phase-based optical reflectometry systems, how they work, and an overview of some typical system configurations. The third chapter describes performance characteristics of coherent phase-OTDR specifically, the different configurations used, their strengths and weaknesses, as well as a look at the current state of the various methods for mitigating the deficiencies of phase-OTDR. The fourth chapter details the development of a coherent phase-OTDR simulator and the fifth and final chapter briefly outlines some conclusions and possible directions for future work.

Thesis Contributions

The thesis aims at providing a thorough explanation of the physics of light scattering as well as a clear description of how it relates to the implementation of optical reflectometry systems, focusing particularly on phase-based optical reflectometers. The author would argue that this contribution helps provide a more coherent description that bridges the gap between knowledge presented in papers focused on light scattering and papers focused on distributed optical fiber sensing. Details of this work will be submitted as a review paper to the Journal of Optical Fiber Technology.

A more novel contribution is the development of a simulator for a coherent phase-based optical time domain reflectometer. Specifically, the ability to simulate external forces on the optical fiber has particularly useful applications for verifying experimental results with theory. A flexible and functional simulator for a coherent phase based OTDR written in the common MATLAB language is a useful tool for distributed optical fiber sensing research. The developed simulator provides a useful framework upon which new signal processing methods may be tested, or which may be altered or expanded to provide simulations for similar types of sensors. Details of this work will be submitted soon for journal and/or conference publication.

Chapter 1

Review of Light Scattering in Optical Fiber

The use of phase-OTDR for optical fiber sensing applications has received significant attention over recent years; however, the majority the current literature fails to cover much of the fundamental theory and operation of phase-OTDR, or optical reflectometry in general. This section is an attempt at filling the apparent gap in the current literature by providing a complete background to the physics of light scattering in single mode optical fiber (SMF) and the associated mathematical models typically introduced in many papers discussing distributed sensing over optical fiber.

1.1 Propagation of Light in Singlemode Optical Fiber

Developed many decades ago [17], low loss optical fiber is commonly used as a transmission medium owing to its ability to carry light over long distances with little attenuation or distortion to the original signal. Various types of optical fiber have been produced for various applications, and their properties were extensively studied. For this thesis, we will only discuss standard telecom grade single mode fiber (SMF), such as SMF-28 [18].

The propagation of light in SMF has been extensively investigated [19]–[22] mostly in an effort to understand the loss mechanisms and how to minimize them. It has long been known that scattering and absorption are the two prevalent loss mechanisms in optical fibers. Absorption in optical fibers is wavelength dependent with absorption peaks in the ultra-violet region [23] as

well as the infrared region [24]. As manufacturing processes improved into the 1990s, the material impurities that contributed to the absorption peaks had been essentially eliminated, with the exception of Hydroxyl bonds in the glass [25]; however, in 1999, a “dry” fiber process had been used to produce optical fiber with near ideal absorption spectra [26]. This leaves scattering as the only significant contributor to signal attenuation in modern SMF. Details of light scattering in SMF will be discussed in detail in the following section.

1.2 Light Scattering in Singlemode Optical Fiber

In general, the intrinsic scattering losses in SMF are either the result of Rayleigh scattering or Mie scattering [27]. Mie scattering is caused by scatterers that are comparable in size to the wavelength of the incident light, whereas Rayleigh scattering is caused by scatterers that are much smaller in size than the wavelength of the incident light. Modern fabrication methods of SMF aim to produce glass that has low absorption and low scattering of light for a range of wavelengths. The quality of modern SMF is such that the number larger inhomogeneities are reduced to an amount such that Mie scattering is nearly insignificant. Rayleigh scattering is more difficult to reduce since it is not related to impurities, but is due to the molecular disorder that remains after the glass transitions from a vitreous state to a crystalline state during the manufacturing process [28]. Therefore, Rayleigh scattering is the dominant source of scattering in modern low loss SMF.

Several techniques have been developed over the years for fabricating low loss glass, such as modified chemical vapor process (MCVP) [29], plasma activated chemical vapor process (PCVD) [30], vapor phase axial deposition (VPAD) [31] and outside vapor deposition (OVD) [32]. After the glass is fabricated, it is drawn into a thin fiber and coated with various materials to provide protection [33], [34]. While the molten glass is being drawn, it is cooled at a very

specific rate until it reaches a glassy state, which occurs at the glass transition temperature T_g. At this temperature, the glass molecules are essentially "frozen" in place [35]. During the cooling process, some amount of structural disorder remains among the molecules. It is this disorder at the microscopic level that causes light scattering within the medium.

Further discussion of the parameters that affect the light scattering properties of SMF is outside the scope of this paper but can be found in papers by Saito *et al.* [35]–[37].

1.2.1 Microscopic Light Scattering of a Molecule

Let us first consider the scattering of light by a single molecule. Referring to Fig. 1, an incident electric field, $\mathbf{E} = E_0 \exp\left[-j(kz - \omega t)\right]$, will induce a dipole in the molecule such that,

$$\mathbf{p} = \alpha \mathbf{E} \quad (C \cdot m) \tag{1.1}$$

where α is the polarizability of the molecule. As shown in Fig. 1, the vector components of an induced dipole are dependent on the angle between the incident field and the scatterer.

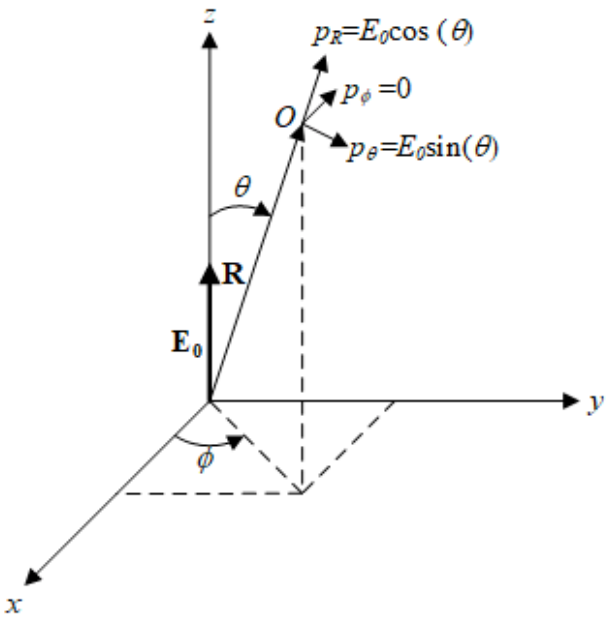


Fig. 1. Electric Field of Dipole Radiation

Since we are interested in detecting the light scattering in the far-field, the only component of the induced dipole that will be considered significant is the E_ϑ component; the other components are attenuated by $1/R^3$ and $1/R^2$ and therefore become negligible. The E_ϑ component can be expressed as,

$$E_\vartheta = -E_0 \frac{k_S^2 \sin\vartheta}{4\pi R}\left(\frac{p}{\varepsilon}\right) exp\left[-j\left(k_S R - \omega t\right)\right] \quad \left(V/m\right) \tag{1.2}$$

where k_s is the wavevector for the scattered wave, R is the distance between the source and the scatterer, p is the magnitude of the dipole and ω is the angular frequency of the incident wave.

The measured intensity of the field is given by the magnitude of the Poynting vector [39] which can be expressed as,

$$I_\vartheta = \frac{1}{2}\left(\varepsilon/\mu\right)^{1/2}\left\langle \mathbf{E}_\vartheta \cdot \mathbf{E}_\vartheta^* \right\rangle \quad \left(W/m^2\right) \tag{1.3}$$

where the angle brackets indicate time average of the electric field. We note that a photo-detector cannot react at optical frequencies, therefore, the resulting output will be a time average of the incident field. Substituting Eq. 1.2 into Eq. 1.3 yields,

$$\begin{aligned} I_\vartheta &= \frac{1}{2}\left(\frac{\varepsilon}{\mu}\right)^{1/2} \frac{k_S^4 \sin^2\vartheta}{16\pi^2 R^2}\left(\frac{\left\langle p^2 \right\rangle}{\varepsilon^2}\right) = \frac{1}{2}\left(\frac{\varepsilon}{\mu}\right)^{1/2} \frac{k_S^4 \left\langle \alpha^2 \right\rangle E_0^2 \sin^2\vartheta}{16\pi^2 R^2}\left(\frac{\varepsilon_0}{\varepsilon}\right)^2 \\ &= \left(\frac{\varepsilon}{\mu}\right)^{1/2} \frac{k_S^4 \left\langle \alpha^2 \right\rangle E_0^2 \sin^2\vartheta}{32\pi^2 R^2 n^4} \quad \left(W/m^2\right) \end{aligned} \tag{1.4}$$

Note that the only variable that is averaged is the square of the polarizability α.

We now determine the effective scattering intensity from a large collection of molecules. As discussed by Boyd [40], the intensity of the light scattering from a volume, V, consisting of a large number molecules is directly related to the variation in the number of molecules within the smaller volumes, ΔV shown in Fig. 2.

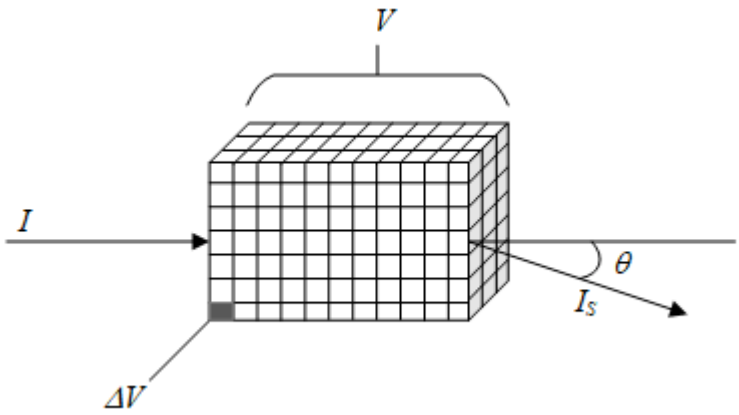


Fig. 2. Scattered Intensity from Scattering Volume

Assuming each volume, ΔV, contains on average $<n>$ molecules, the average intensity of the scattered light can be expressed as,

$$I_{\Delta V} = \langle n \rangle^2 I_S \tag{1.5}$$

where I_s is the intensity of scattered light from a single molecule from Eq. 1.4. However for every volume, ΔV, we can assume that there exists another volume that emits radiation that will interfere destructively. If a volume contains a number of molecules, Δn, which deviates from the average, there will be a net scattered intensity,

$$I_{\Delta n} = \langle \Delta n \rangle^2 I_S \tag{1.6}$$

The total scattered intensity from the entire volume is, therefore

$$I_V = \frac{V}{\Delta V} \langle \Delta n \rangle^2 I_S \tag{1.7}$$

Assuming that the molecules within the material follow a Poisson distribution, the variance of the number of molecules is equal to the average number of molecules, therefore,

$$I_V = \frac{V}{\Delta V} \langle n \rangle I_S \tag{1.8}$$

This result shows that the light scattering from a homogeneous isotropic material is directly proportional to the average number of molecules per unit volume.

1.2.2 Relation Between Microscopic and Macroscopic Light Scattering of Molecules

In the previous section a model for light scattering at the microscopic level was reviewed, however; for practical purposes, it is useful to have a model for light scattering at a macroscopic level. Using the Lorenz-Lorentz equation, the average dielectric constant of a material can be determined using knowledge of the polarizabilities of the material's molecules such that,

$$\frac{1}{3\varepsilon_0}\sum_i n_i\alpha_i = \frac{\varepsilon_r - 1}{\varepsilon_r + 2} \qquad (1.9)$$

where α_i are the polarizabilities of the individual molecules and n_i is the number of molecules per cubic meter. For homogeneous materials Eq. 1.9 can be expressed as

$$\frac{N\alpha}{3\varepsilon_0} = \frac{\varepsilon_r - 1}{\varepsilon_r + 2} \qquad (1.10)$$

which is known as the Clausius-Mossotti relation. The overall polarization per unit volume, **P**, is expressed as,

$$\mathbf{P} = \varepsilon_0\chi\mathbf{E} \quad \left(C/m^2\right) \qquad (1.11)$$

where **E** is the electric field and χ is the susceptibility of the material. The susceptibility is related to the dielectric constant by $\chi = \varepsilon_r - 1$, where ε_r can be calculated from Eq. 1.10. We note that $\chi + \Delta\chi = \varepsilon_r + \Delta\varepsilon_r - 1$, meaning that fluctuations in the susceptibility are related to the fluctuations in the dielectric constant such that $\Delta\chi = \Delta\varepsilon_r$. When n is close to unity Eq. 1.10 can be written as,

$$N\alpha = \chi_e \varepsilon_0 \tag{1.12}$$

Considering fluctuations in the susceptibility we can write,

$$N(\alpha + \Delta\alpha) = (\varepsilon_r + \Delta\varepsilon_r - 1)\varepsilon_0 \tag{1.13}$$

and finally,

$$N\Delta\alpha = \Delta\varepsilon \tag{1.14}$$

The overall change in polarization per unit volume, $\Delta\mathbf{P}$, due to a fluctuation in the dielectric of the material can be expressed as

$$\Delta\mathbf{P} = \varepsilon_0 N(\Delta\alpha)\mathbf{E} \quad \left(C/m^2\right) \tag{1.15}$$

This important result describes the macroscopic behavior of material given changes in its microscopic properties.

1.2.3 Light Scattering in a macroscopically homogeneous medium

In section 1.2.1 light scattering at a microscopic level was reviewed and in section 1.2.2 a relation between the fluctuations of particle density and material susceptibility was derived. In this section, the scattering of light from a macroscopically homogenous volume is investigated.

Given a completely homogenous material, where no fluctuations in the material properties occur, light scattering will only occur in the forward direction (see Fabelinskii [41]). This is due to the fact that in a perfectly homogeneous medium, for every radiating scatterer in a volume there would exist another scatterer with which it would destructively interfere. However, as reviewed in section 1.2.1, it was shown that even given a small volume of molecules, ΔV, a net scattered intensity is generated. We now revisit the radiating dipole Eq. 1.2 and consider the effective scattering from multiple dipoles in a macroscopic volume.

1.2.3.1 Radiating Dipole Model of Macroscopic Light Scattering

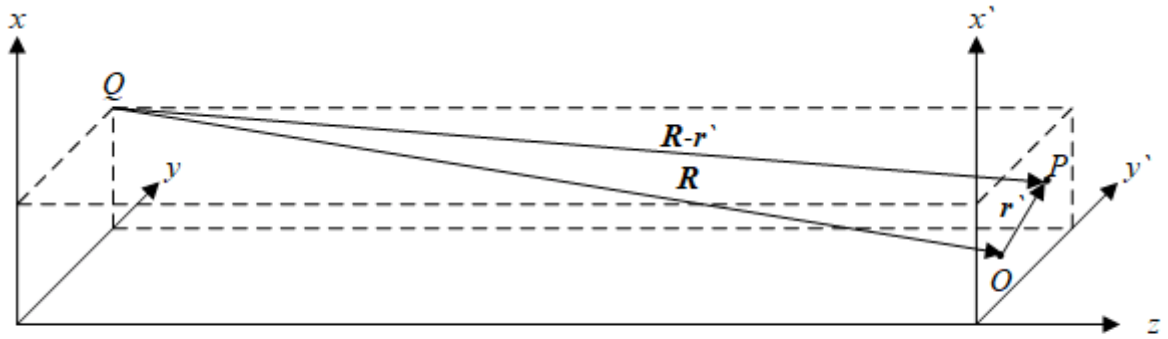
Considering that scattering occurs only due to material fluctuations, we express Eq. 1.2 as a function of the local fluctuation,

$$E_{\vartheta} = -E_0 \frac{k_S^2 \sin\vartheta}{4\pi R}\left(\frac{\varepsilon_0}{\varepsilon}\right)\Delta\chi \exp\left[-j\left(k_S R - \omega t\right)\right] \quad \left(V/m\right) \tag{1.16}$$

Referring to Fig. 3. the aggregate of the scattered field from a volume V is [39]–[41],

$$\mathbf{E}_S = \mathbf{E}_{\vartheta} = -E_0 \frac{k_s^2 \sin\vartheta}{4\pi R}\left(\frac{\varepsilon_0}{\varepsilon}\right) exp\left[j\left(k_s R - \omega t\right)\right]\int_V \Delta\chi\left(\mathbf{r}',t\right) exp\left[j\left(\mathbf{k}_s\right)\cdot\mathbf{r}'\right] d\mathbf{r}' \tag{1.17}$$

where **r** = **R** - **r**'.



O – Center of scattering volume
Q – Observer
P – Scatterer

Fig. 3. Detection of Light Scattering from a Volume in the Far-Field

Since a photo-detector will convert the electric field to an intensity, we substitute Eq. 1.17 into Eq. 1.3 to obtain,

$$I_S = \frac{1}{2}\left(\frac{\varepsilon}{\mu}\right)^{1/2} \frac{k^2 \sin^2\vartheta}{\left(4\pi R\right)^2}\left(\frac{\varepsilon_0}{\varepsilon}\right)^2 * E_0^2 \left\langle \iint_V \Delta\chi\left(\mathbf{r}_1',t\right)\Delta\chi\left(\mathbf{r}_2',t\right) exp\left[-jk\left(z_2' - z_1'\right)\right] exp\left[-jk\,\hat{\mathbf{R}}\cdot\left(\mathbf{r}_2' - \mathbf{r}_1'\right)\right] d\mathbf{r}_1' d\mathbf{r}_2' \right\rangle \quad \left(V/m^2\right) \tag{1.18}$$

It is assumed that the field being detected is in the far field and the following approximation applies,

$$\left\|\mathbf{R}-\mathbf{r}'\right\| = R - \mathbf{r}'\cdot\frac{\mathbf{R}}{R} = R - \mathbf{r}'\cdot\hat{\mathbf{R}} \qquad (1.19)$$

Eq. 1.18 can be rewritten as,

$$I_S = \frac{1}{2}\left(\frac{\varepsilon}{\mu}\right)^{1/2}\frac{k^2\sin^2\vartheta}{(4\pi R)^2}\left(\frac{\varepsilon_0}{\varepsilon}\right)^2 * E_0^2\iint_V\left\langle\Delta\chi\left(\mathbf{r}_1',t\right)\Delta\chi\left(\mathbf{r}_2',t\right)\right\rangle exp\left[-jk\left(z_2'-z_1'\right)\right]exp\left[-jk\hat{\mathbf{R}}\cdot\left(\mathbf{r}_2'-\mathbf{r}_1'\right)\right]d\mathbf{r}_1'd\mathbf{r}_2' \qquad (1.20)$$

due to the fact that the only statistical elements are the density fluctuations.

The term $\langle\Delta\chi(\mathbf{r}_1',t)\ \Delta\chi(\mathbf{r}_2',t)\rangle$ represents the ensemble average of the product of the susceptibility fluctuations between all locations within the volume. This term can be interpreted as a correlation between fluctuations at different locations. This correlation can be expressed as a delta function [42], we can, therefore, consider scatterers within a volume to behave as delta functions whose amplitude are equal to the average fluctuation within the material, expressed as $\langle\Delta\chi^2\rangle\delta(x)\delta(y)\delta(z)$, where x, y, and z are the location of the scatterer in a volume and δ represents the Kronecker delta function.

1.2.4 Material Fluctuations as a Thermodynamic Process

The physical reason for the fluctuations in the dielectric constant, ε, of a material can be attributed to fluctuations in material density, $\Delta\rho$, as stated by Einstein [43]. The relation between the material density and the number of molecules per unit volume can be expressed as,

$$\frac{m_{mol}}{\Delta v_S} = \Delta N \qquad (1.21)$$

Where m_{mol} is the mass per molecule and v_S is the specific volume of the material. Clearly then,

$$\frac{m_{mol}}{\Delta v_S} = \Delta N \qquad (1.22)$$

Since material density is inversely proportional to the specific volume, $v = 1/\rho$, and the dielectric constant is related to N through Eq. 1.10, a clear relation between fluctuations in dielectric constant and thermodynamic variables can be drawn. We now assume that $\Delta\varepsilon$ is a function of the thermodynamic variables of density, ρ, and temperature, T, such that [41]

$$\Delta\varepsilon(\rho,T) = \left(\frac{\partial\varepsilon}{\partial\rho}\right)_T \Delta\rho + \left(\frac{\partial\varepsilon}{\partial T}\right)_\rho \Delta T \qquad (1.23)$$

The second term is generally considered insignificant and can be ignored while obtaining results that are accurate to within 2% [41]. Furthermore, the variable $\Delta\rho$ can also be expressed as a function the thermodynamic variables of entropy, S, and pressure, p.

$$\Delta\rho(S,p) = \left(\frac{\partial\rho}{\partial S}\right)_p \Delta S + \left(\frac{\partial\rho}{\partial p}\right)_S \Delta p \qquad (1.24)$$

The first term in Eq. 1.24 corresponds to adiabatic fluctuations while the second term corresponds to isobaric fluctuations.

As a consequence of the Boltzmann principle, the joint probability density function of the thermodynamic variables in Eq. 1.23 and Eq. 1.24 are bivariate Gaussian with zero mean [42]. The fluctuations in temperature, density, pressure, and entropy can, therefore, be considered statistically independent and the variance of density fluctuations may be expressed as,

$$\left\langle \Delta\rho^2 \right\rangle = \left(\frac{\partial\rho}{\partial S}\right)_p^2 \left\langle \Delta S^2 \right\rangle + \left(\frac{\partial\rho}{\partial p}\right)_S^2 \left\langle \Delta p^2 \right\rangle \qquad (1.25)$$

Using the chain rule, Eq.1.25 and Eq.1.23 can be expressed as,

$$\left\langle \Delta\varepsilon^2 \right\rangle = \left(\frac{\partial \varepsilon}{\partial S}\right)_p^2 \left\langle \Delta S^2 \right\rangle + \left(\frac{\partial \varepsilon}{\partial p}\right)_S^2 \left\langle \Delta p^2 \right\rangle \qquad (1.26)$$

The fluctuations of pressure and entropy can be written as,

$$\left\langle \Delta p^2 \right\rangle = \frac{KT}{\Delta V \beta_S} \qquad (1.27)$$

$$\left\langle \Delta S^2 \right\rangle = \frac{Kc_P}{\Delta V} \qquad (1.28)$$

Where β_s is the adiabatic compressibility of the material, c_p is the specific heat at constant pressure, and K is Boltzmann's constant. Substituting Eq. 1.27 and Eq. 1.28 into Eq. 1.26, we obtain a description of the fluctuations in terms of the thermodynamic properties of the material.

$$\left\langle \Delta\varepsilon^2 \right\rangle = \left(\frac{\partial \varepsilon}{\partial S}\right)_p^2 \frac{Kc_P}{\Delta V} + \left(\frac{\partial \varepsilon}{\partial p}\right)_S^2 \frac{KT}{\Delta V \beta_S} \qquad (1.29)$$

Eq. 1.29 can be further simplified using the following relations,

$$\beta_T = \beta_S + \frac{T\sigma^2}{c_P} \qquad (1.30)$$

and

$$\left(\frac{\partial \varepsilon}{\partial S}\right)_p = \left(\frac{\partial \varepsilon}{\partial T}\right)_P \left(\frac{\partial T}{\partial S}\right)_P \quad \frac{\partial S}{\partial T} = \frac{c_P}{T} \qquad (1.31)$$

Where σ is the coefficient of volume expansion and β_T is the isothermal compressibility. And we also assume that,

$$\left(\frac{\partial \varepsilon}{\partial p}\right)_S = \left(\rho \frac{\partial \varepsilon}{\partial \rho}\right)_S \beta_S \qquad (1.32)$$

Eq. 1.29 then becomes

$$\left\langle \Delta\varepsilon^2 \right\rangle = \frac{KT\beta_T}{\Delta V}\left(\rho\frac{\partial\varepsilon}{\partial\rho}\right)^2 \qquad (1.33)$$

The differential term is related to the photo-elastic coefficient of the material through [44]

$$\left(\rho\frac{\partial\varepsilon}{\partial\rho}\right) = \varepsilon^2\gamma_P \qquad (1.34)$$

where γ_P is the photo-elastic coefficient. Finally, we obtain,

$$\left\langle \Delta\varepsilon^2 \right\rangle = \frac{KT\beta_T n^8 \gamma_P^2}{\Delta V} \qquad (1.35)$$

This result expresses the mean square fluctuation in the material dielectric as a function of the thermodynamic and photo-elastic properties of the material.

1.2.5 Calculation of Scattering Coefficient from Thermodynamic Properties of Material

Using the results obtained from the previous section, the scattering coefficient can be calculated. The scattering coefficient is a measure of the scattering efficiency and can be expressed as,

$$\alpha_S = \frac{I_S}{I_0}\frac{R^2}{\Delta V} \qquad (1.36)$$

where R is the distance from the point of observation to the scatterer, I_0 is the original light intensity, I_S is the intensity of the scattered light and ΔV is the scattering volume. In section 1.2.2 we established that the net scattering from a volume is due to the fluctuation in the dielectric constant so that we may write,

$$p = \varepsilon_0 \Delta\varepsilon \Delta V E_0 \qquad (1.37)$$

Substituting Eq. 1.37 into Eq. 1.4 we obtain,

$$I_S = \left(\frac{\varepsilon}{\mu}\right)^{1/2} \frac{k_S^4 \Delta V^2 \left\langle \Delta\varepsilon^2 \right\rangle E_0^2 \sin^2\vartheta}{32\pi^2 R^2 n^4} \quad \left(W/m^2\right) \tag{1.38}$$

From section 1.2.3 we found that the variance of the dielectric constant can be determined from the thermodynamic properties of the material, and thus Eq. 1.38 becomes,

$$I_S = \left(\frac{\varepsilon}{\mu}\right)^{1/2} \frac{k_S^4 \Delta V K T \beta_T n^4 \gamma_P^2 E_0^2 \sin^2\vartheta}{32\pi^2 R^2} \quad \left(W/m^2\right) \tag{1.39}$$

Since

$$I_0 = \left(\frac{\varepsilon}{\mu}\right)^{1/2} \frac{E_0^2}{2} \quad \left(W/m^2\right) \tag{1.40}$$

The scattering coefficient becomes

$$\alpha_S = \frac{k_S^4 K T \beta_T n^4 \gamma_P^2 \sin^2\vartheta}{16\pi^2} \tag{1.41}$$

This result agrees with Boyd. Furthermore to this result, it is of interest to calculate the scattering cross section over all angles, θ, so that the total amount of power scattered is known.

Integrating Eq. 1.41 over the surface of a sphere,

$$\alpha_{RS} = \int_0^{2\pi} d\phi \int_0^{\pi} \alpha_s \sin\vartheta \, d\vartheta \, d\phi \tag{1.42}$$

results in,

$$\alpha_{RS} = \frac{8\pi^3}{3\lambda^4} n^8 \gamma_p^2 \beta_T k_B T_F \tag{1.43}$$

The result[1] of Eq. 1.43 is the more commonly stated result for the Rayleigh scattering coefficient for SMF.

---

[1] T (temperature) is replaced with T_F, which is the fictive temperature of glass.

1.3 Rayleigh Backscatter of Light in Singlemode Fiber

In this section, the scattering theory detailed in section 1.2 is extended to consider the detection of Rayleigh *backscatter* when using SMF as the medium. For the purposes of optical reflectometry, it is useful to be able to estimate the light detected at the source end of an SMF being probed as light propagates down the fiber.

Light launched into an SMF with initial energy, E_0, will be attenuated at according to Eq. 1.44,

$$E(z) = E_0 \exp[-\alpha_A z] \qquad (1.44)$$

where α_A is the attenuation coefficient that accounts for all losses. The energy lost to scattering is determined by the scattering coefficient, often referred to as the Rayleigh scattering coefficient, which was described in section 1.2.5 and gives the fraction of the forward propagating energy that is scattered in all directions. As discussed by Hartog [45], it is only a small fraction of this scattered light that returns back to the source that can be detected, this fraction is known as the *backscatter capture fraction* which we represent as $B(z)$.

The backscatter capture fraction is determined by the power coupling between the radiating dipoles and the fundamental mode of the fiber (for SMF this is the HE11 mode). For a Gaussian beam, the backscatter capture fraction is expressed as,

$$B = \frac{3(NA)^2}{2n^2} \frac{1}{V\left(\frac{\omega_0^2}{a^2}\right)} \qquad (1.45)$$

where NA is the numerical aperture of the fiber, n is the refractive index, a is the core radius of the fiber, V is the normalized frequency and ω_0 is the spot size of the beam.

The power received from a propagating pulse then takes the form,

$$P(t) = \frac{v_g}{2} E_0 \alpha_{RS}(z) B(z) \exp[-2\alpha_A z] \quad W \qquad (1.46)$$

where α_{RS} is the Rayleigh scattering coefficient, and $v_g=2dz/dt$. Eq. 1.46 provides a general formula expressing the received power from scattered light in SMF.

1.4 Light Scattering Model for Singlemode Fiber under Coherent Radiation

In the previous section, the power received from backscattered light in an SMF was calculated. In that case, the assumption was that the source used is incoherent. In this section, we consider the effect of using a coherent source. When using a coherent source, the scattering from an SMF can be treated as an accumulation of scatterers with random phase and no spatial correlation[2]. The impulse response of the system can, therefore, be represented as,

$$h(t)=\sum_{n=1}^{N}\alpha_{RS}(\tau_n)B(\tau_n)A(\tau_n)exp\left[-v_g\tau_n\alpha_A\right]\delta(t-\tau_n) \qquad (1.47)$$

where $A(\tau_n)$ is the amplitude (reflectivity) of the n^{th} scatterer and τ_n is the time delay between the n^{th} scatterer and received light. Eq. 1.47 omits the polarization properties of the impulse response for simplicity. This model fits with that of various authors [39], [46]–[48], and the first three terms are generally absorbed into a single term resulting in,

$$h(t)=\sum_{n=1}^{N}a(\tau_n)exp\left[-v_g\tau_n\alpha_A\right]\delta(t-\tau_n) \qquad (1.48)$$

As noted by some authors [45], [46], the stochastic model of Eq. 1.48 is similar to the phenomena of speckle. An important result from this similarity is that the scatterers can be treated as circularly symmetric complex Gaussian random variables (CCRV) (see Appendix A for a full derivation).

---

[2] The spatial distribution of scatterers follows a random uniform distribution.

This property allows the use of a useful theorem when calculating the power spectral density of the received light scattering from the stochastic model of Eq. 1.48. This will be described in more detail in the section 2.2.2.

Chapter 2

Phase-Based Optical Reflectometry Systems

In this section, an overview of phase-based optical reflectometry systems is given, specifically systems that operate on Rayleigh scattering. An excellent overview of distributed optical fiber Rayleigh scattering sensing technologies, including phase based techniques, is given by Palmieri [2]. This is a very broad subject, and therefore this overview is not to be considered an exhaustive review, but it is meant to give a useful overview of the general techniques and processing methods specific to phase-based reflectometry systems.

When referring to a *phase*-based optical reflectometry system, phase refers to the fact that it is the phase component of the dipole radiation that causes a unique interference pattern in the detected signal that is useful for distributed sensing applications. As will be seen, extraction of the phase component can be accomplished using different types of systems and processing methods.

The method in which data is captured and analyzed over time draws several parallels to radar, where backscatter from a launched signal is measured over time and then repeated continuously. In radar, a single waveform is referred to as being captured in *fast time*, and the period between consecutively acquired waveforms is referred to as *slow time*. The same terminology will be used here when referring to phase-OTDR data analysis.

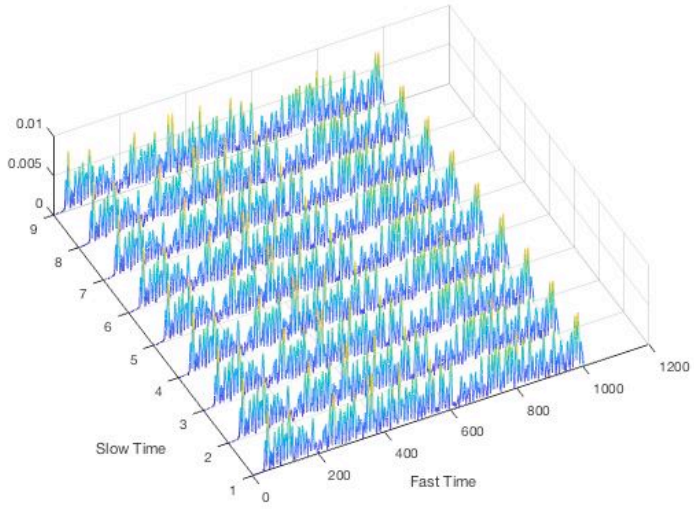


Fig. 4. Intensity Plots Showing Fast Time and Slow Time Axes

It is typical to plot the intensity or phase received by the photo-detector over time as the probe pulse propagates down the fiber. This is commonly referred to as a *trace* and is typically used to identify different events within the fiber at specific locations. Intensity traces acquired using phase-OTDR methods have a jagged appearance compared to traditional OTDR due to the coherent interference of the scatterers in the fiber. These inhomogeneities are unique to each fiber, and thus traces of intensity or phase may be considered a fingerprint [49] that may be used to detect changes the physical state of the fiber.

Consider a fiber illuminated by a monochromatic source. The field at a location, z, within the fiber at time, t, can be expressed as[3],

$$U(t;z) = E_0 \exp\left[-j\left(kz - 2\pi\nu t\right)\right]\exp\left[-\alpha_A z\right] \qquad (2.1)$$

[3] The convention used here is, $U(t;z)$ where t is a variable and z is a parameter

where E_0 is the original amplitude of the field, z is the spatial position along the fiber, k is the wavenumber of the field, α_A is the intensity attenuation coefficient and v is the frequency of the light.

In a phase based reflectometer it is the backscatter from the scatterers that is received by the photo-detector and analyzed. We, therefore, express the detected backscatter at time, t, from location, z, as,

$$U_{PD}(t;z) = E_0 B(z)\alpha_{RS}(z)exp\left[-j(2kz - 2\pi vt)\right]exp\left[-2\alpha_A z\right] \quad (2.2)$$

where the $2z$ term accounts for the round-trip distance covered by the scattered signal, α_{RS} is the Rayleigh scattering coefficient and the $B(z)$ term is the backscatter fraction coefficient. It is useful to rewrite Eq.2.2 as,

$$U_{PD}(t;z) = E_0 B(z)\alpha_{RS}(z)exp\left[-j\phi(t;z)\right]exp\left[-2\alpha_A z\right] \quad (2.3)$$

replacing the complex terms by a single phase term.

Since the source used in a phase-based reflectometer is highly coherent, the backscattered fields from the individual scatterers interfere coherently, causing the intensity detected over time to have a more jagged appearance (similar to speckle). This is different than in systems that use sources having shorter coherence times, resulting in the intensity detected over time to be more smoothed.

Assuming that N scatterers at z contribute to U_{PD}, we can write,

$$U_{PD}(t;z) = \sum_{i=1}^{N} E_0 B(z)\alpha_{RS}(z)exp\left[-j\phi_i(t;z)\right]exp\left[-2\alpha_A z\right] \quad (2.4)$$

It is clear that the detected field will depend on the relative phase and amplitude of the individual scatterers.

There exist many different types of systems that work on the principle of phase-based reflectometry. In this thesis, we will separate these systems into three categories based on the probe signal type; time domain, frequency domain or time-frequency domain.

2.1 Time Domain Optical Reflectometry

To date, the majority of phase based reflectometers operate in the time domain. In time domain based systems, an excitation signal that is finite in time is launched into a fiber under test. A sampling of the scattered signal is performed, and the acquired data is analyzed. One of the first (if not the first) phase-OTDR systems was described by Taylor [50].

Taylors' system is similar to a typical[4] OTDR type setup with the exception that the source has a long coherence time. In this type of system the excitation signal; typically a rectangular pulse, is launched into an optical fiber through an optical coupler or circulator. As the pulse propagates down the fiber, a fraction of the forward propagating pulse is scattered and travels back toward the coupler. This scattered light is then detected by a photo-detector, whose current is then converted to a voltage and subsequently digitized for further processing by a microprocessor or similar device. A typical OTDR system is shown in Fig. 5.

[4] Here we refer to typical as being an OTDR system with an incoherent source. This is quite typical in traditional OTDR applications such as break detection for telecommunication cables.

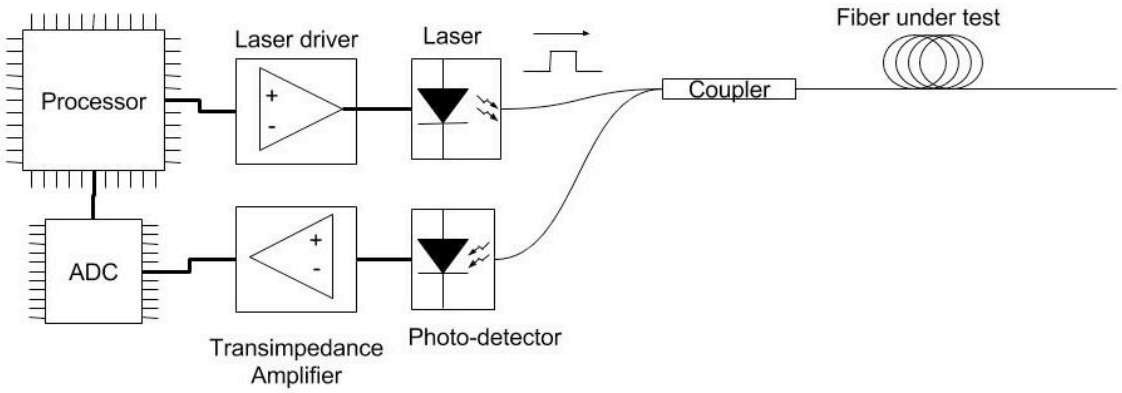


Fig. 5. Typical OTDR System Diagram

It should be apparent that the configuration shown in Fig. 5 is only one potential configuration that can be used to implement an OTDR.

2.1.1 Detection of Scattering from a Rectangular Pulse using Phase-OTDR

In time domain systems, it is commonplace to use a rectangular pulse to interrogate the system. Therefore, we now consider the backscatter received from a single scatterer at z, from a propagating rectangular pulse with width T_p, where $T_P v_g >> \Delta z$. We assume that the pulse enters the fiber at t=0, meaning that $kz_L - 2\pi vt = 0$ where z_L is the leading edge of the pulse. The field at location z, time t, is,

$$U(t) = E_0 rect\left(\frac{t-\tau}{T_p}\right) exp\left[-j2\pi v\tau\right] exp\left[-\alpha_A z\right] \qquad (2.5)$$

Where T_p is the duration of the pulse, and τ is the delay from the leading edge of the pulse. It must be noted that τ is related to the selected position z through the relation,

$$\tau = \frac{(z_L - z)}{v_g} \qquad (2.6)$$

Also, we define

$$rect(t) = \begin{cases} 1, & 0 \le t \le 1 \\ 0, & otherwise \end{cases} \qquad (2.7)$$

As the pulse propagates down the fiber, the leading edge of the pulse will illuminate the scatterer as it arrives at location z. This scatterer will radiate (as an induced dipole as described in Section 1.2.1) and will continue to radiate for the duration of the pulse, T_p. Scatterers separated by a distance, d, or less, will have their backscatter received at the same time, t, but from different delays, τ, within the pulse if the following condition is satisfied,

$$0 < d \le \frac{T_P v_g}{2} \qquad (2.8)$$

The period of time that the photo-detector will receive backscatter from two locations is,

$$T_S = \frac{T_P}{2} - \frac{d}{v_g} \qquad (2.9)$$

If the source has a coherence length longer than $T_p v_g$, the backscatter will interfere coherently. As described in section 1.4, the impulse response of the fiber can be treated as the accumulation of the radiating fields from the scattering centers under illumination. Using Eq. 2.5, the backscattered field received at the photo-detector at any time can be expressed as,

$$U_{PD}(t) = \sum_{i=1}^{N} rect\left(\frac{t-\tau_i}{T_p}\right) a_i \, exp\left[-j2\pi v \tau_i\right] exp\left[-2\alpha_A z_i\right] \qquad (2.10)$$

The detected intensity is, therefore

$$\begin{aligned} I_{PD}(t) &= \sum_{i=1}^{N} a_i^2 rect\left(\frac{t-\tau_i}{T_P}\right) exp\left[-4\alpha_A z_i\right] + \\ &2\sum_{i=1}^{N}\sum_{j=i+1}^{N} rect\left(\frac{t-\tau_i}{T_p}\right) rect\left(\frac{t-\tau_j}{T_P}\right) a_i a_j \, exp\left[-j2\pi v\left(\tau_i - \tau_j\right)\right] exp\left[-2\alpha_A\left(z_i + z_j\right)\right] \end{aligned} \qquad (2.11)$$

Eq. 2.11 is applicable to direct detection systems, which will be discussed in section 2.3.1.

2.1.2 Spatial Resolution of a Phase-OTDR

The spatial resolution of a phase-OTDR system is a function of the pulse width used for the probing signal.

$$\Delta z_S = \frac{T_P v_g}{2} \qquad (2.12)$$

The factor of ½ is due to the fact that as backscatter from the leading edge of the pulse begins travelling back to the source, the pulse is still travelling forward at the same velocity; therefore the given length within a fiber that contributes to the detected backscatter at any given time is ½ the width of the pulse.

2.1.3 Sensing Range & Repetition Rate of a Phase-OTDR

The sensing range for phase-OTDR applications can be anywhere from hundreds of meters to over 100 km. In general, the total sensing range of a phase-OTDR system is determined by the SNR of the system, which is determined by several factors, such as pulse energy, fiber attenuation coefficient, wavelength, and scattering coefficient just to name a few.

Total pulse energy is an easily adjusted parameter by either adjusting the peak power of the pulse or adjusting its width. There is a limit to how much peak power a pulse may have since increasing power will lead to non-linear effects such as self-phase modulation [51]. Adjusting pulse width may also have some limitations depending on the requirement for spatial resolution as discussed in the previous section. However, it has recently been shown that the dependence of spatial resolution on pulse width, as expressed in Eq. 2.12, may be broken through the use of pulse compression [52], [53].

The detection method used will also determine the maximum sensing range of a phase-OTDR system. Coherent detection methods offer higher SNR than incoherent methods, but they rely on

mixing the backscattered signal with a local oscillator signal, adding the additional requirement of a light source with a higher degree of coherence than the incoherent method. For these methods, the coherence length of the source needs to be more than twice as long as the maximum sensing distance to achieve proper demodulation.

The total sensing range of a phase-OTDR system will have an impact on the repetition rate of the system. The repetition rate is the frequency at which the system can generate probe pulses for interrogating the fiber. Because the probe pulse needs to travel the length of the fiber and back to the source, the maximum repetition rate for the system is defined as,

$$T_R = \frac{2L + W_P}{v_g} \tag{2.13}$$

This has implications for performance parameters of the system, in particular when considering periodic events on the fiber that need to be detected, such as vibration. Since changes in the fiber are typically detected based on trace-to-trace comparisons, the frequency at which traces may be acquired will dictate the maximum frequency of any periodic external events that can be detected.

2.2 Frequency Domain Optical Reflectometry

In frequency domain based reflectometry systems, the excitation signal used is continuously applied, but its frequency is changed over time. A sampling of the scattered signal is performed, and the acquired data is analyzed using the frequency information to relate to spatial information. A good overview of basic optical frequency domain reflectometry (OFDR) methods can be found in the paper by Yuksel *et al.* [54].

In an OFDR system, the excitation signal is launched into an optical fiber through an optical coupler or circulator, and the frequency of the signal is swept linearly over a frequency range. As the light propagates down the fiber, a fraction of the forward propagating pulse is scattered and travels back toward the coupler. This scattered light is then mixed with the source, which will have a different frequency due to the frequency sweep over time. This causes the signal detected by a photo-detector to be modulated down to an intermediate frequency. The current output of the photo-detector is then converted to a voltage and subsequently digitized for further processing by a microprocessor or similar device. A typical OFDR system is shown in Fig. 6.

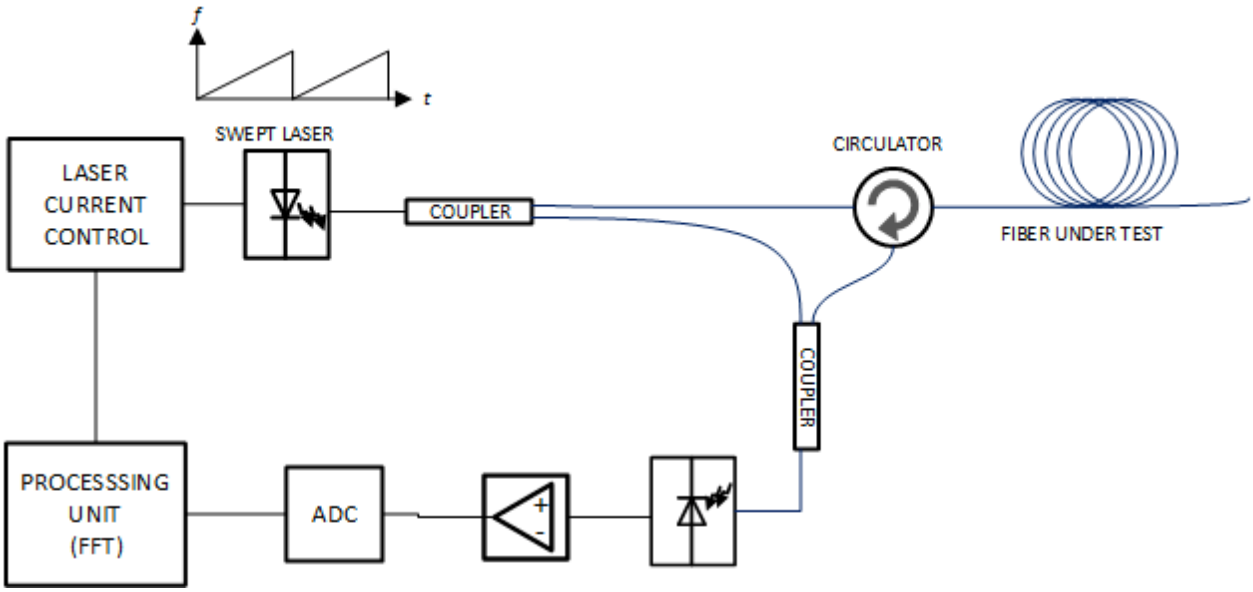


Fig. 6. Typical OFDR System Diagram

2.2.1 Detection of Light Scattering Using a Phase-OFDR

For an OFDR system, the output of the LO source can be expressed as a chirped signal,

$$U_{LO}(t) = E_0 \exp\left[-j\left(2\pi\nu t + \pi\beta t^2\right)\right] \qquad (2.14)$$

where β is the chirp rate of the source. The field at location z at time t can be expressed as,

$$U(t;z) = E_0 \exp\left[-j\left(kz - 2\pi\nu t + \pi\beta t^2\right)\right]\exp\left[-\alpha_A z\right] \quad (2.15)$$

where the wavenumber, k, for the chirp pulse is

$$k = \frac{2\pi n\nu}{c} + \pi\beta z \quad (2.16)$$

Since the system constantly applies the source to the fiber, the backscattered field detected by the receiver at any time, t, will be the summation of all the scatterers within the fiber.

$$U_S(t) = \sum_{i=0}^{N} E_0 B(z_i)\alpha_{RS}(z_i)\exp\left[-j\left\{2\pi\nu(t-\tau_i) + \pi\beta(t-\tau_i)^2\right\}\right]\exp\left[-2\alpha_A z_i\right] \quad (2.17)$$

In OFDR the sampled data is converted to the frequency domain using a Fast Fourier Transform (FFT) since the frequency data corresponds to the spatial locations.

2.2.2 Calculation of Power Spectral Density using phase-OFDR

As covered in section 1.4, the scattering in SMF is a random process and therefore to obtain the frequency information from the measured intensity, $I_{PD}(t)$, the Wiener-Khintchine theorem is used, which states that the power spectral density (PSD) of a stationary random process, $x(t)$, is the Fourier transform of the autocorrelation of $x(t)$.

$$S(f) = \int_{-\infty}^{\infty} \left\langle x^*(t)\, x(t-\tau)\right\rangle \exp\left[-j2\pi f\tau\right] d\tau \quad (2.18)$$

Since,

$$I_{PD}(t) = \left\langle U_{PD}^*(t) U_{PD}(t)\right\rangle \quad (2.19)$$

the autocorrelation of $I_{PD}(t)$ can be expressed as,

$$R(\tau) = \left\langle U_{PD}^{*}(t) U_{PD}(t) U_{PD}^{*}(t-\tau) U_{PD}(t-\tau) \right\rangle \quad (2.20)$$

Since, as discussed in section 1.4, $U_{PD}(t)$ is a circularly symmetric complex Gaussian random variable (CCRV), the complex Gaussian moment theorem can be applied [55]. Therefore, Eq. 2.20 can be written as,

$$\begin{aligned} &R(\tau) = \\ &\left\langle U_{PD}^{*}(t) U_{PD}(t) \right\rangle \left\langle U_{PD}^{*}(t-\tau) U_{PD}(t-\tau) \right\rangle + \\ &\left\langle U_{PD}^{*}(t) U_{PD}(t-\tau) \right\rangle \left\langle U_{PD}(t) U_{PD}^{*}(t-\tau) \right\rangle \end{aligned} \quad (2.21)$$

Calculating the PSD of received signal now becomes a tractable problem.

2.2.3 Spatial Resolution of OFDR Systems

The spatial resolution of OFDR systems is related to the bandwidth of the probe signal and the frequency resolution of the system (which is related to the receiver bandwidth). Since the backscattered signal is mixed with an LO, we can express the resulting beat frequency from the location at a distance, d, from the source as,

$$f_b = \frac{2\beta d}{v_g} \quad (2.22)$$

The difference in beat frequencies between two locations can be expressed as,

$$\Delta f_b = \frac{2\beta \Delta z}{v_g} \quad (2.23)$$

Re-arranging to solve for Δz we obtain,

$$\Delta z_s = \frac{\Delta f_b}{2\beta v_g} \quad (2.24)$$

We see therefore that for an OFDR increasing the sweep rate can increase the systems spatial resolution, thereby increasing the difference between the beat frequencies. Unfortunately, this increases the bandwidth requirements of the detector.

2.2.4 Sensing Range & Repetition Rate of a phase-OFDR System

In an OFDR system, the source is swept over a period T_s which corresponds to the length of the fiber by,

$$T_S = \frac{2L}{v_g} \qquad (2.25)$$

The frequency output of the source over time is generally linearly swept over this time, as shown in Fig. 7.

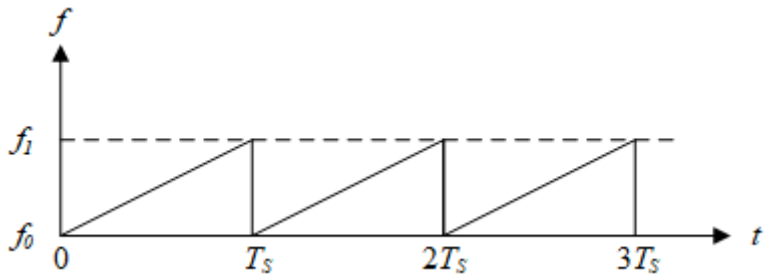


Fig. 7. OFDR Frequency Sweep Waveform

To obtain unique beat frequencies for each position on the fiber, the source frequency must be swept over a time equal to twice the propagation time to the end of the fiber since the signal must propagate to the end and back to be detected. Therefore, the sensing range can be increased or decreased by decreasing or increasing the sweep duration respectively.

2.3 Overview of Optical Detection Methods for Phase-OTDR

There are many configurations used for the detection of optical signals in reflectometers. In broad terms, these detection schemes can be categorized to be either coherent or incoherent[5] in nature. Both types have their strengths and weaknesses when applied to phase-OTDR applications.

2.3.1 Optical Receiver Using Incoherent Detection Method

Incoherent detection methods, which will henceforth be referred to as *direct detection***,** are those methods where the backscatter signal is fed directly into the photo-detector without mixing with another signal, as shown in Fig. 8.

[5] The term "incoherent detection" does not mean that the light source used is incoherent, but that the detection method does not rely on coherent mixing to enhance the detection.

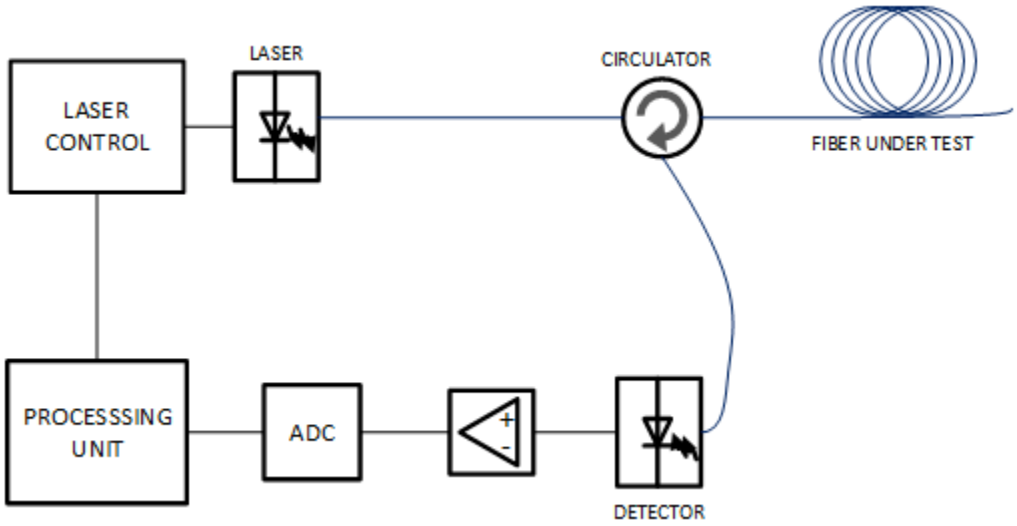


Fig. 8. System Diagram of a Reflectometer with Direct Detection Receiver

Since no mixing occurs, this method measures intensity only and thus cannot directly measure phase. Although recently a direct detection method has been developed that can extract phase information, assuming certain characteristics of the system are fulfilled. [56]. Table 1 lists the advantages and disadvantages of such a system.

Table 1. Advantages and Disadvantages when Using a Direct Detection Optical Receiver

Advantages	**Disadvantages**
Less expensive: Requires fewer parts to implement.	**Reduced SNR:** The backscattered signal has extremely low amplitude.
Shorter coherence time required: No need to be coherent with LO at the detector. Allows for longer sensing range.	**High frequency:** Unable to observe much of the higher frequency coherent “speckle.”
Polarization Insensitive:	**Nonlinear response:**

No fading from polarization mismatches.	Intensity-based measurement is implicitly nonlinear

Direct detection systems are often used where quantitative measurements are not required, and system state can be inferred from the measurements. A good example is intrusion detection systems, where it is often only necessary to categorize the change in the measurements over some time period to determine if an alarm should be sent or not. This is different than a temperature sensing application where quantitative measurements of temperature are required.

When using direct detection, the output signal from the system is a measurement of the intensity of the backscattered signal, which is inherently nonlinear, making quantitative measurements difficult.

2.3.2 Optical Receiver Using Coherent Detection Method

Coherent detection typically involves mixing the backscattered signal with a local oscillator (LO) at the photo-detector. A very useful overview of coherent detection methods is given by Ip [57] and by Kikuchi [58].

Table 2. Advantages and Disadvantages when Using a Coherent Detection Optical Receiver

Advantages	**Disadvantages**
Increased SNR: Backscattered signal is mixed with high strength LO.	**More expensive:** More components are required.
Frequency shifting to intermediate frequency:	**Longer coherence time required:** To have coherent interference, the source and

Backscattered signals are downshifted to a frequency that can be observed.	LO must have coherence time longer than the round trip time.
	Polarization Sensitive: Sensitive to polarization mismatches between LO and signal.

Since future sections focus mostly on systems that use of coherent detection, a review of some basic configurations of coherent phase-OTDRs is presented. This is followed by an analysis of system stability, and some mitigation techniques for stability issues are reviewed.

2.3.2.1 Optical Receiver Using Homodyne Detection

The first reported use of a homodyne receiver in an OTDR was by Healey [59]. The basic setup is shown in Fig. 9.

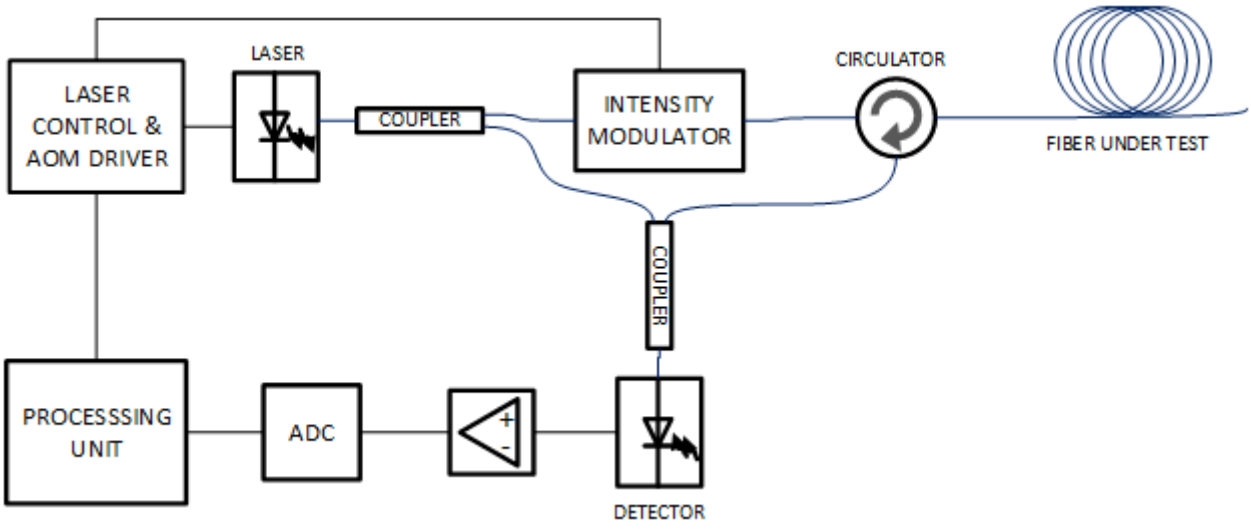


Fig. 9. System Diagram of Optical Reflectometer with Homodyne Detection Receiver

In the homodyne configuration, the laser source is used both as the probing light source and the LO for the receiver. In a homodyne system, the scattering signal is modulated back down to baseband by the LO. The intensity is expressed as,

$$I_{PD} = \left|U_S + U_{LO}\right|^2 = U_S U_S^* + 2U_S U_{LO} + U_{LO} U_{LO}^* \qquad (2.26)$$

The scattered wave and the LO are expressed as,

$$\begin{aligned} U_S &= a\,exp\left[-j2\pi vt\right] exp\left[-2\alpha_A z\right] exp\left[-j\phi(t)\right] \\ U_{LO} &= exp\left[-j2\pi vt\right] \end{aligned} \qquad (2.27)$$

Eq. 2.26 becomes,

$$I_{PD} = 1 + a^2\, exp\left[-4\alpha_A z\right] + 2a\, exp\left[-2\alpha_A z\right] exp\left[-j\phi\right] \qquad (2.28)$$

Homodyne detection is capable of obtaining a 3dB greater SNR than Heterodyne detection; however it requires that the LO is phase matched to the received signal. Due to the fact that with coherent phase-OTDR the phase of the backscattered signal is random, and that a phase lock is difficult to achieve on such fast signals, a homodyne receiver typically has no advantage over a heterodyne receiver in phase-OTDR.

2.3.2.2 Optical Receiver Using Heterodyne Detection

The first reported use of a heterodyne receiver in an OTDR was by Healey [60]. The basic setup is shown in Fig. 10.

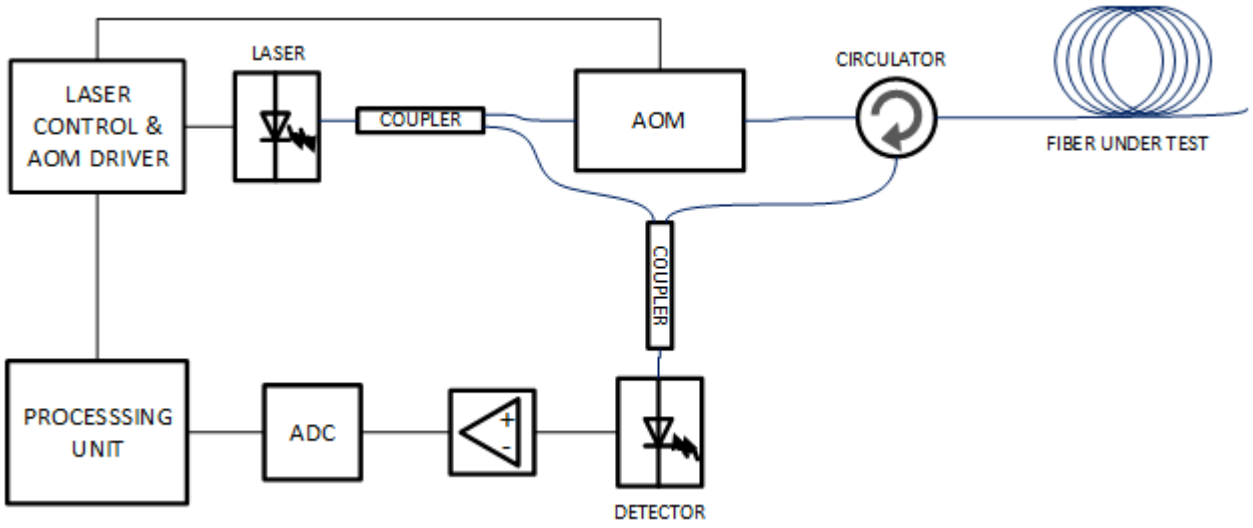


Fig. 10. System Diagram of Optical Reflectometer with Heterodyne Receiver

In the heterodyne configuration, the LO is a different frequency than the probe source. In this case, the scattered signal is demodulated down to an intermediate frequency (IF). The scattered wave and the LO are expressed as,

$$U_S = a\,exp\left[-j2\pi v_0 t\right]exp\left[-2\alpha_A z\right]exp\left[-j\phi(t)\right]$$
$$U_{LO} = exp\left[-j2\pi v_{LO} t\right] \qquad (2.29)$$

The resulting intensity is, therefore

$$I_{PD} = 1 + a^2\,exp\left[-4\alpha_A z\right] + 2a\,exp\left[-2\alpha_A z\right]exp\left[-j2\pi\left(v_0 - v_{LO}\right)t - \phi(t)\right] \qquad (2.30)$$

The third term of Eq. 2.30 contains the phase information, which is contained within the beat frequency v_0 - v_{LO}.

2.3.3 Phase Recovery from Optical Homodyne & Heterodyne Receivers

In both homodyne and heterodyne detection methods, the measured intensity carries a phase term that is due to the aggregate of the individual scattered fields from the fiber. For many applications, it is the phase term that is desired and must be extracted from the measured intensity. Several phase recovery methods have been developed. For example, using a Michelson

interferometer (MI) setup [61], phase generated carrier (PGC) method [62], *I/Q* demodulation using optical hybrid [63] and digital coherent detection [64] to name a few.

Methods using interferometric receiver type setups, similar to Fig. 11, operate by creating several phase-shifted versions of the detected signal, allowing for the phase to be recovered.

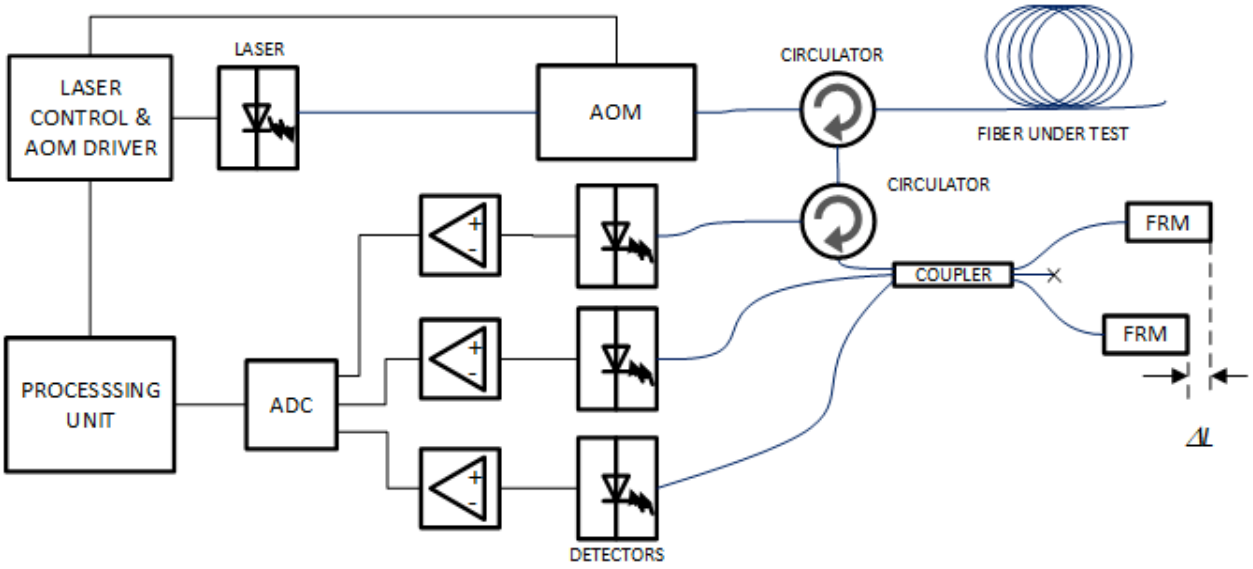


Fig. 11. System Diagram of Optical Reflectometer with Interferometric Receiver

This method is advantageous because the phase can be recovered without doing substantial signal processing. However, the obvious disadvantages are the increased component count and number of ADC detectors required.

PGC methods are similar to the interferometric methods, except on one of the arms of the MI a piezoelectric transducer is used to modulate the signal reflected from that arm.

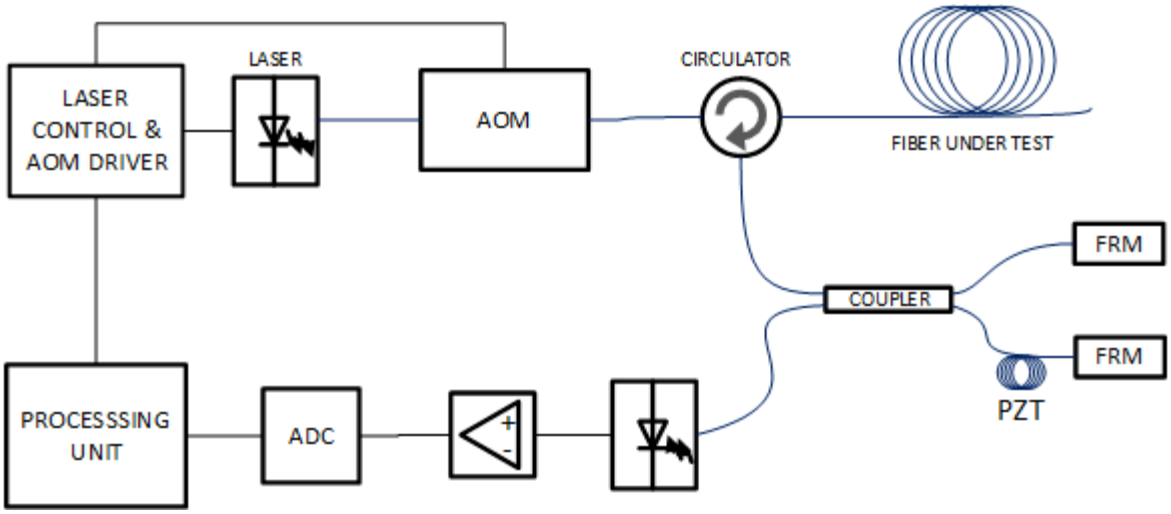


Fig. 12. System Diagram of Optical Reflectometer with PGC Detection

Details of the demodulation method are omitted here for brevity; however, it should be clear by Fig. 12 that fewer photo-detectors are required at the expense of greater processing requirements.

One of the most popular methods in recent years is the use of an optical hybrid to perform *I/Q* demodulation. A schematic depiction of the optical hybrid is shown in Fig. 13.

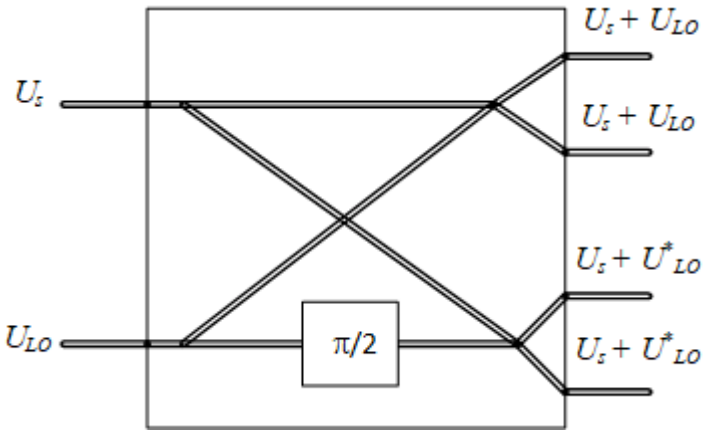


Fig. 13. Diagram of Optical Hybrid Receiver

Details on the use of an optical hybrid for coherent *I/Q* demodulation in the digital domain are explained in the paper by Taylor [65]. In Taylor's method, the *I* & *Q* signals are optically generated, while demodulation, filtering, and all other signal processing are done in the digital domain. For homodyne detection, the use of the optical hybrid is particularly appealing since it generates the *I* & *Q* signals in the optical domain at baseband inherently.

Digital coherent detection is also becoming a very popular method for implementing *I/Q* demodulation. With digital coherent detection, a single balanced detector receives the mixed light from a local oscillator (LO) and the backscatter signal, resulting in an intensity signal,

$$I \propto 2U_s(t)U_{LO}(t)cos\left(2\pi\left(v - v_{LO}\right)t + \vartheta_n(t)\right) \qquad (2.31)$$

The output of the balanced detector is sampled at a high sampling rate (typically several hundred MSPS) and the acquired data is processed to extract the phase since,

$$\vartheta(z) = arg\left(\int_{z-L/2}^{z+L/2} I\, exp\left[-j2\pi\left(v - v_{LO}\right)z\right]dz\right) \qquad (2.32)$$

Eq. 2.32 shows that the phase of the scattered fields in the area z-$L/2$ to z+$L/2$ is equal to the phase of the Fourier transform of the measured intensity, which elucidates the requirement for high sampling rates since the spatial resolution is then limited by the time required to obtain enough samples for the FFT.

Chapter 3

Performance Analysis of Coherent Phase-OTDR

In this chapter, a brief overview of the various performance limitations of coherent phase-OTDR systems is given, as well as some of the latest developments to address these issues. In particular, issues involving signal fading are investigated.

3.1 Effects of Laser Source on Phase-OTDR Systems

The properties of the laser source used in a phase-OTDR system influence the behavior of a phase-OTDR system in several ways. Both the laser stability and linewidth are the main properties of interest and shall be reviewed in the following sections.

3.1.1 Effects of Laser Frequency Drift on Phase-OTDR Systems

Phase-OTDR systems work primarily by comparing acquired traces over time. As previously covered, the values in the particular trace acquired are a function of the frequency of the source used. This implies that phase-OTDR requires a source that is extremely stable over time, as a change in the source frequency will change the phase relationship of the scatterers. This fact is well known and is perhaps one of the biggest challenges in achieving a system that can operate reliably for long periods of time [66].

The phase difference between two scatterers separated by a distance, Δz, is

$$\phi = 4\pi v(\Delta z)/v_g \qquad (3.1)$$

Therefore a change in the optical frequency, Δv, will result in a change in phase,

$$\Delta\phi = 4\pi n(\Delta v \Delta z)/c \qquad (3.2)$$

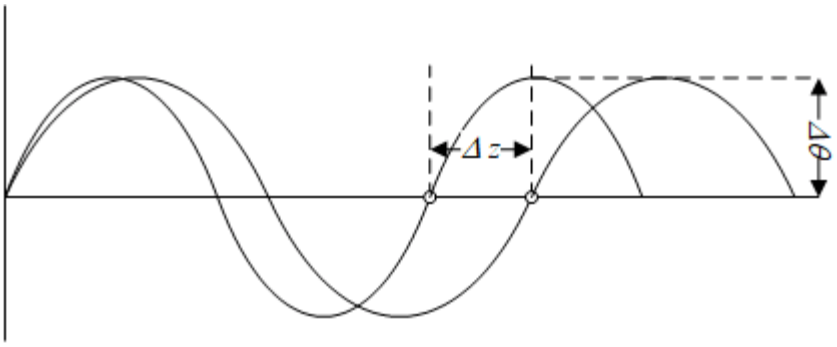


Fig. 14. Relationship Between Frequency and Phase of Sinusoid

As investigated by Zhong [67], we can see that for a direct detection system, the trace fluctuation due to frequency drift of the source can be given by,

$$\Delta I = 2\sum_{i=0}^{N}\sum_{j=i+1}^{N} a_i a_j \exp\left[-\alpha_A(z_i - z_j)\right]\left\{\cos\left(2\pi(v+\Delta v)(z_i - z_j)\right) - \cos\left(2\pi v(z_i - z_j)\right)\right\} \qquad (3.3)$$

This difference in received intensity can easily be equivalent or greater than the effect of external forces trying to be detected.

3.1.2 Effects of Laser Phase Noise on Phase-OTDR Systems

Phase noise refers to the error in the phase term that a laser source may have at any given time. The phase noise of a laser is directly related to its linewidth and has a variance that can be expressed as [68],

$$\sigma_\phi^2(t) = 2\pi\Delta v t \qquad (3.4)$$

Where Δv is the linewidth of the source, and t is the time over which the error is accumulated. To explain this differently, assume that we measure the phase of light from a source at time *0* and

again at time t. The mean square error in the phase term between these measurements will be $2\pi\Delta vt$.

This type of noise is particularly problematic for coherent detection systems where scattered light originally emitted by the laser source at time t_1 is mixed with light emitted at time t_2. The larger the difference in time t, the larger the phase noise component.

3.2 Signal Fading in Phase-OTDR

In phase-OTDR, the differences between acquired traces are used to determine whether physical changes in the fiber under test have occurred, be they from vibration, strain or other forces. In most cases, the largest impediment faced when comparing these measurements is the issue of fading. The definition of fading in phase-OTDR is somewhat overloaded and is used to refer to different phenomena.

Fading can be treated as a source of noise, and this noise corrupts the desired signal, essentially reducing the overall SNR of the system. In the remainder of this section, some different fading mechanisms are reviewed.

3.2.1 Visibility Fading in Phase-OTDR

To obtain coherent interference between scatterers, the interrogating light must have a high degree of coherence otherwise it will suffer from visibility fading. It is common to express the degree of coherence of light in terms of the visibility of the fringes created by interfering waves. Visibility is expressed as,

$$V = \frac{I_{max} - I_{min}}{I_{max} + I_{min}} \qquad (3.5)$$

where I_{max} is the maximum intensity of the detected interference and I_{min} is the minimum intensity.

The intensity is a function of the degree of coherence, ρ, and is expressed as,

$$I = I_1^2 + I_2^2 + 2\rho\sqrt{I_1 I_2}\cos\vartheta \qquad (3.6)$$

where θ is the phase difference between the interfering fields. As the degree of coherence reduces, the visibility of the interference also reduces, which for phase-OTDR causes a reduction in SNR. For phase-OTDR systems that use direct detection methods, the coherence time must, therefore, be greater than the length of the probing pulse, $T_{coh} > T_p$. This ensures that SNR is maintained between light scattered from the leading edge and trailing edge of the pulse. For coherent detection systems, the coherence time of the light needs to be proportional to 2x the length of the fiber being monitored, $T_{coh} > (2L/v_g)$, so that the backscattered light can be mixed at the photo-detector.

This elucidates an advantage for direct detection systems since the requirements for coherence may be much less aggressive than coherent detection systems, especially for applications where a long sensing range is desired.

3.2.2 Polarization Fading in Phase-OTDR

As discussed by Healey [69], systems using coherent detection methods are subject to polarization fading. A standard SMF is birefringent, and light propagating down such fiber will undergo a random change in its state of polarization (SOP). Under static conditions, the transform of the SOP, **M**, is considered to be reciprocal. The SOP of the backscatter can, therefore, be described as rotation, reflection, and rotation [70],

$$\mathbf{S}_{\mathrm{PD}}(z) = \mathbf{R}^{\mathrm{T}}(\vartheta\mathbf{n})\mathbf{M}_{\mathrm{R}}\mathbf{R}(\vartheta\mathbf{n})\mathbf{S}_{\mathrm{L}}(0) \qquad (3.7)$$

where $\mathbf{R}(\theta\mathbf{n})$ describes a counter-clockwise rotation of angle θ about a vector **n**, $\mathbf{M}_{\mathrm{R}}$ is a reflection, and $\mathbf{S}_{\mathrm{L}}(0)$ is the SOP of the launched pulse.

When the polarization of the source and the backscatter are the same, they are able to interfere completely; otherwise, the signals suffer from fading relative to the angle between their SOPs,

$$A = \left\|\mathbf{S_R}(z)\right\|\left\|\mathbf{S_{LO}}(z)\right\| cos\vartheta \qquad (3.8)$$

This issue is compounded by the fact that both temperature and strain will change the birefringence at the location being affected, which can either increase or reduce the polarization fading effect, making event measurements difficult to analyze.

3.2.3 Nonlinear Fading Effects in Phase-OTDR

Commonly referred to as *modulation instability*, the Kerr effect and the associated self-phase modulation are nonlinear effects that cause fading in phase-OTDR systems. This type of fading primarily affects coherent detection schemes. Both the effects of modulation instability on the SNR of the receiver [51], as well as the overall impact on the acquired trace [71] have been studied.

The Kerr effect describes the dependence of the refractive index of the single mode fiber on optical intensity,

$$n(I) = n + n_2 I \qquad (3.9)$$

where n_2 is the optical Kerr coefficient.

This change in refractive index leads to self-phase modulation, where the light will undergo a phase shift,

$$\Delta\phi = -2\pi n_2 \frac{LP}{\lambda A} \qquad (3.10)$$

where P is the optical power, L is the distance traveled, and A is the cross-sectional area of the fiber. The related change in the frequency content of the light is given by

$$\Delta\omega = -2\pi n_2 \frac{PL}{\lambda A}\frac{\partial I(t)}{\partial t} \quad (3.11)$$

This frequency shift between the local oscillator signal in a coherent receiver setup and the backscattered light will cause signal fading.

Martins considers this effect on the received signal over the entire length of a fiber. Since the phase shift is a factor of the distance traveled, it is likely that for long lengths of fiber the frequency shift will evolve and return to the center frequency of the source several times over the length of the fiber. In this case, it was found that the fading effect becomes position dependent.

3.2.4 Phase Bias Fading

Several papers identify the problem of fading that can occur in heterodyne receivers [69], [72]. This fading occurs due to the random nature of the phases of the received from coherent backscatter in SMF. Specifically, when the summation of all the contributing scatterers over the region causes the resulting intensity to be zero or near zero [73]. In this case, the SNR of the system is near zero, and system performance is compromised.

3.3 Fading Mitigation in Phase-OTDR

To combat the different sources of fading in phase-OTDR several different methods have been proposed and tested. The following sections give some examples of these methods.

3.3.1 Frequency & Phase Diversity Methods for Phase-OTDR

In section 3.2.4 the problem of phase bias fading was described. An obvious method for mitigating this type of fading is by implementing a system with frequency diversity. By changing the frequency or the source, the phase relationship between scatterers is also changed since,

$$\phi = 4\pi(\Delta z)/\lambda \quad (3.12)$$

This helps identify locations where fading is present at one frequency and allows for the comparison of the results of multiple frequencies to mitigate these effects.

Introducing multiple probing frequencies has been done in several fashions. Alekseev uses a single laser with a phase modulator to generate two pulses with distinct frequencies that are launched into the fiber with some separation [74], while Shi uses two distinct frequencies within the same pulse by using multiple lasers [75]. Zhou also generates multiple frequencies within the same pulse, but only uses a single laser and phase modulation [76].

Similar to frequency diversity, a system that utilizes multiple phases have also been developed [77]. This system works under the same principle as those using frequency diversity.

3.3.2 Polarization Diversity Methods for Phase-OTDR

There are two frequently used methods to mitigate polarization fading in coherent detection systems. The first is by using a polarization diversity receiver (PDR). A PDR typically consists of at least two (but can use up to four) photo-detectors that are used to detect light at orthogonal polarizations to each other. A typical two detector PDR is shown in Fig. 15.

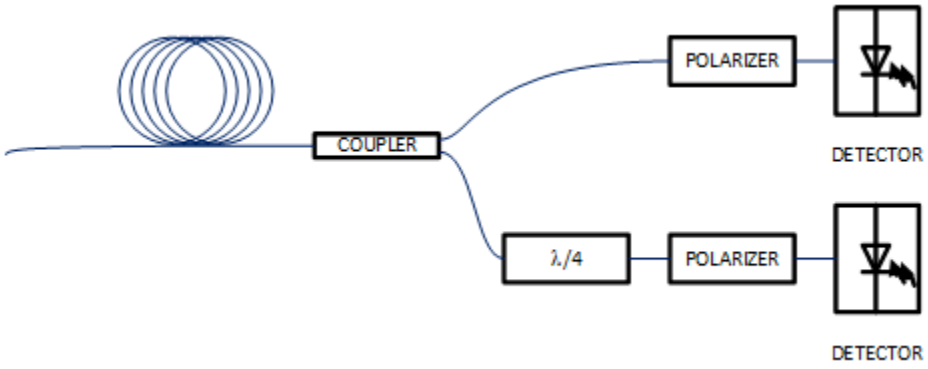


Fig. 15. Diagram of Polarization Diversity Receiver

Using this method, any detected fields that are orthogonal to one detector will be detectable by the other detector.

The second method used is to perform polarization scrambling. In this method, the probing pulse passes through a polarization controller before entering the FUT. Over time the polarization controller randomly scrambles the SOP of the outgoing pulse. This method requires that several trace acquisitions be averaged in order to obtain the mean value over many SOPs.

3.3.3 Utilization of Modulation Waveforms for Non-Linear Fading Mitigation in phase-OTDR

As described in section 3.2.3, modulation instability occurs due to a nonlinear response of the fiber to high-intensity light. This is problematic as a seemingly simple method to achieve a better SNR for the system is to increase the energy of the probing pulse. However, there is a limit to how much power can be used in a phase-OTDR system. The reader is referred to the work by Izumita [51] which provides a thorough discussion regarding this exact problem.

One method to mitigate this issue is proposed by Martins [78], where different pulse shapes are used to reduce the effect of modulation instability. It is interesting to note that rectangular shaped pulses tend to suffer from MI the most while Gaussian or triangular pulse shapes perform better.

Chapter 4

Development of a Simulator for a Coherent Phase-OTDR System

In this chapter, the development of a simulator of a coherent detection based phase-OTDR is described. As described in Chapter 3, the factors affecting the performance of a coherent phase-OTDR system are many, and generating a system simulation based on all parameters analytically can easily become difficult if not intractable. It is attractive with these types of problems to find the solutions numerically using computer simulations. A detailed, flexible and easily modifiable simulation is therefore desired for further research. This is the goal of the work conducted.

The first section of this chapter will describe the assumed system configuration for the simulator, such as transmitter, receiver and data acquisition. The second section provides a description of the software architecture of the simulator and its components, including models for the optical fiber, noise sources and perturbation generation. The third and fourth sections provide some examples of results obtained from the simulator for various types of simulated perturbations.

4.1 Description of Coherent Phase-OTDR System Configuration

The coherent phase-OTDR configuration that will be assumed by the simulator program is shown in Fig. 16.

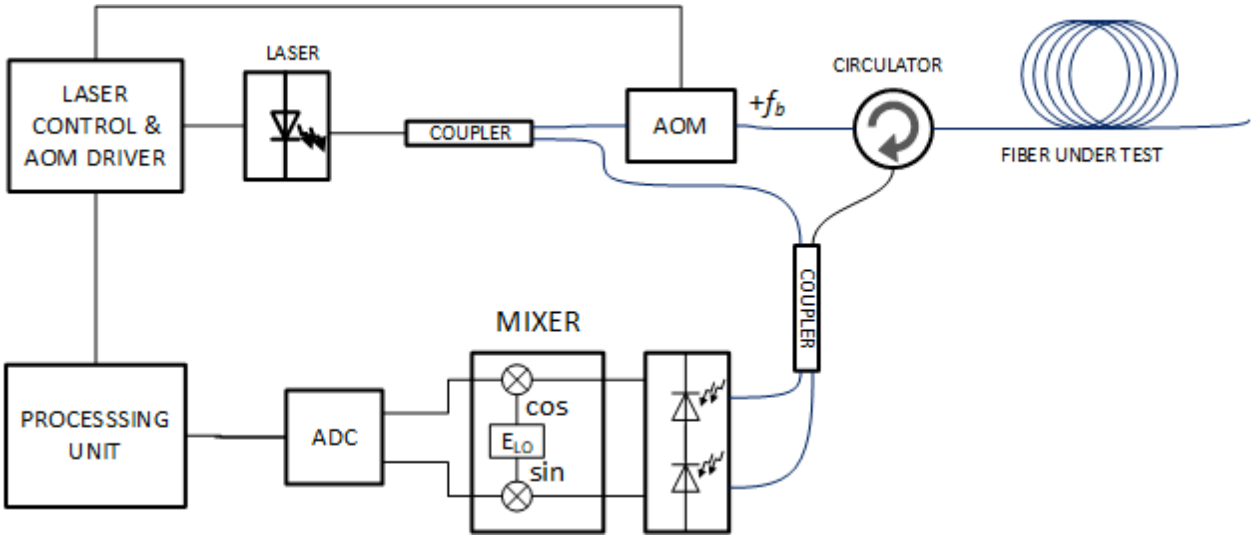


Fig. 16. System Diagram of Simulated OTDR

In this configuration, the output of a laser is split using a coupler, with one output feeding an acoustic optic modulator (AOM) and the other being used as the LO. The AOM will generate a square pulse while imparting a frequency shift to the outgoing light. Both the duration and the frequency shift are configurable by the user through the simulator. The output of the AOM is fed into an optical circulator, which is connected to the fiber under test (FUT). The output of the circulator and the LO are both fed into a balanced photo-detector. This will provide an IF modulated signal which is converted to a voltage by a trans-impedance amplifier stage. The output of the amplifiers is then demodulated down to baseband by an electrical demodulator where an ADC, whose data is then read by the processor and further analyzed, samples it.

The entire simulation can be broken into the steps into as shown in Fig. 17.

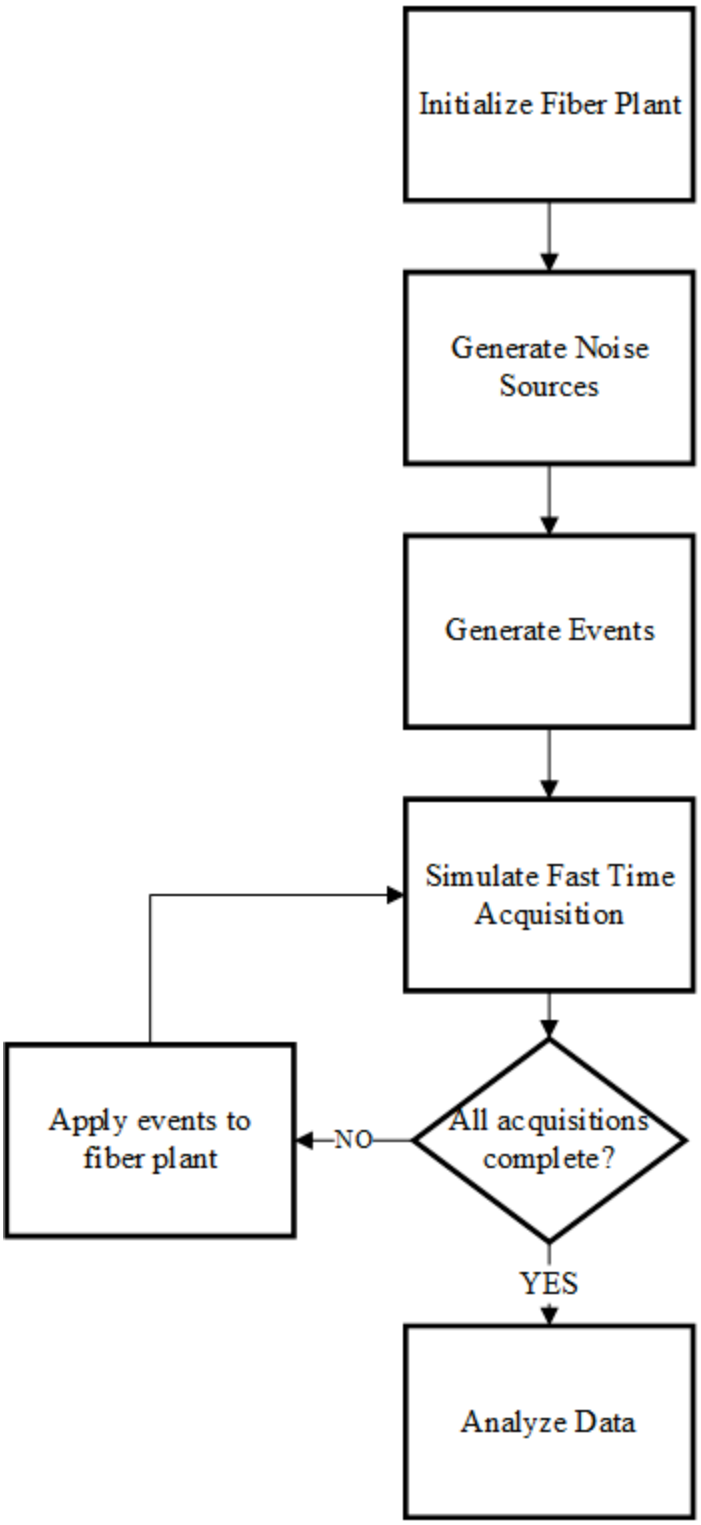


Fig. 17. Flowchart of Simulator Program Execution

4.2 Architecture of Simulator

The simulator was developed using MATLAB as the programming language. This allowed for the use of an object-oriented design approach as well as allowing for simpler graphical user interface (GUI) development and data visualization.

A high-level block diagram of the simulator is shown in Fig. 18.

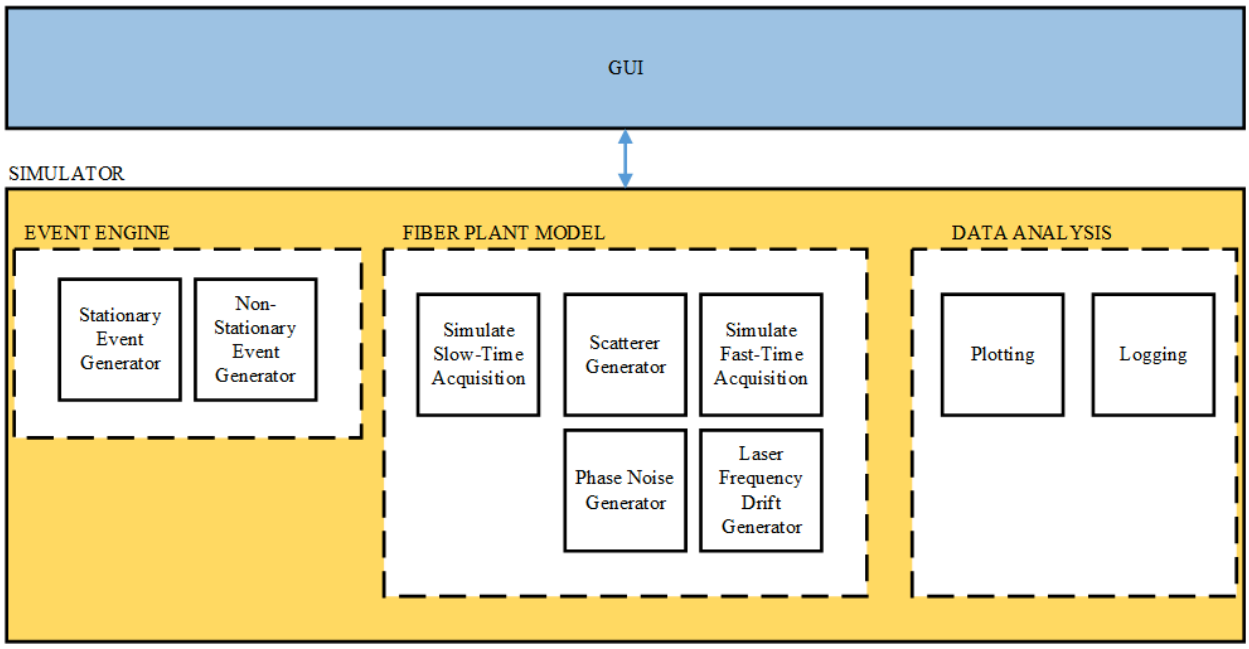


Fig. 18. High-Level Block Diagram of Simulator

The major blocks include:

1) **Graphical User Interface** - The GUI provides a more convenient means of interacting with the simulator by providing simple menus where parameters can be entered. In addition, it allows for better organization of the simulator functions, making it easier for the user to understand how to use the simulator and interpret the results.
2) **Fiber Plant Model** - The fiber plant model block contains all the information pertaining to the physical representation of the optical fiber and functions for manipulating this representation given changes in system parameters or external perturbations.

3) **Event Generation Engine** - The event generation engine block is responsible for generating the disturbance events given the user selected parameters.
4) **Data Analysis** - The data analysis block is responsible for any trace analysis functionality and data preparation.

4.3 Simulator Graphical User Interface

The GUI uses a simple tab-based approach to organize the interface into logical sections. These sections are:

1) **Configuration Panel** – The configuration panel (shown in Fig. 19), allows the user to specify parameters related to the fiber plant itself, as well as the OTDR system. All of the fiber plant parameters are listed in Table 3.

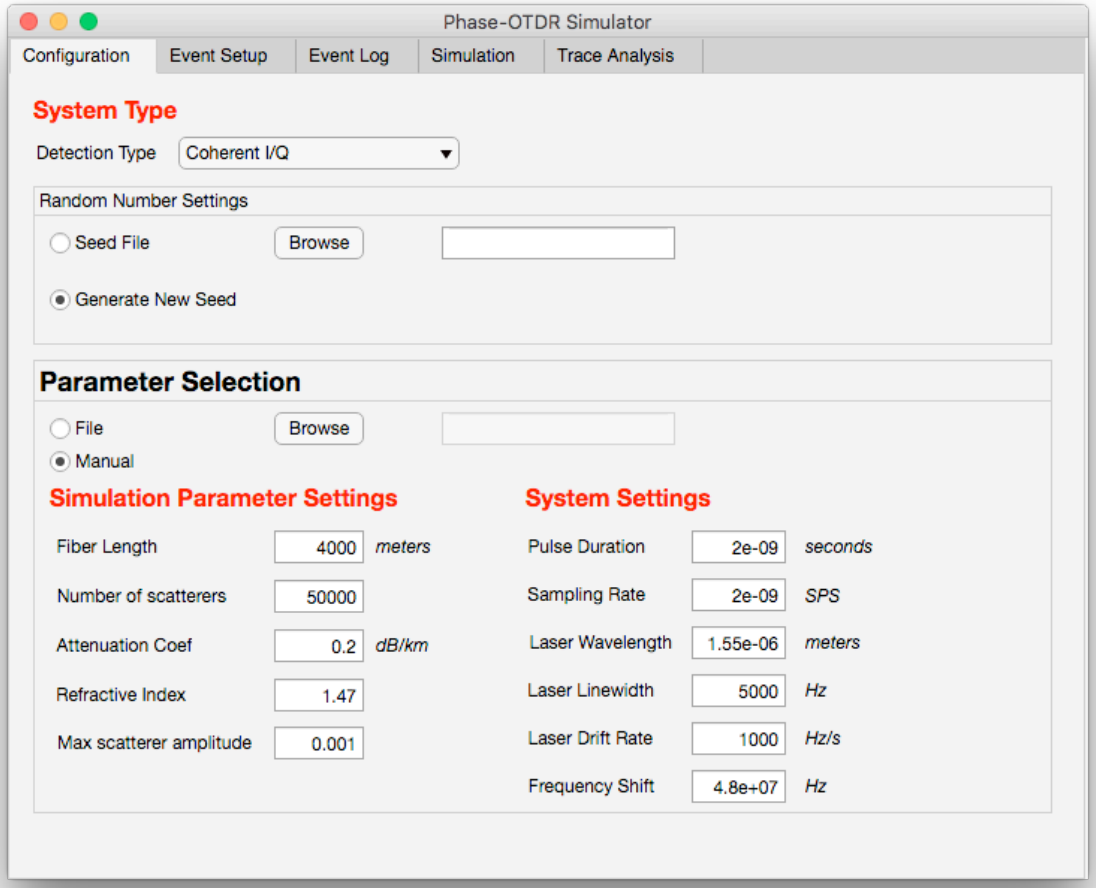


Fig. 19. Configuration Panel of Simulator

2) **Event Generation Panel** – The event generation panel (shown in Fig. 20), allows the user to select which types and how many different events will be generated for the simulation. It also provides fields to place restrictions on certain parameters such as:

   a. Enabling or disabling event types

   b. Number of frequencies for each event type

   c. Min/Max frequencies for each event type

   d. Min/Max span of each event type

e. Min/Max strain applied as each event type

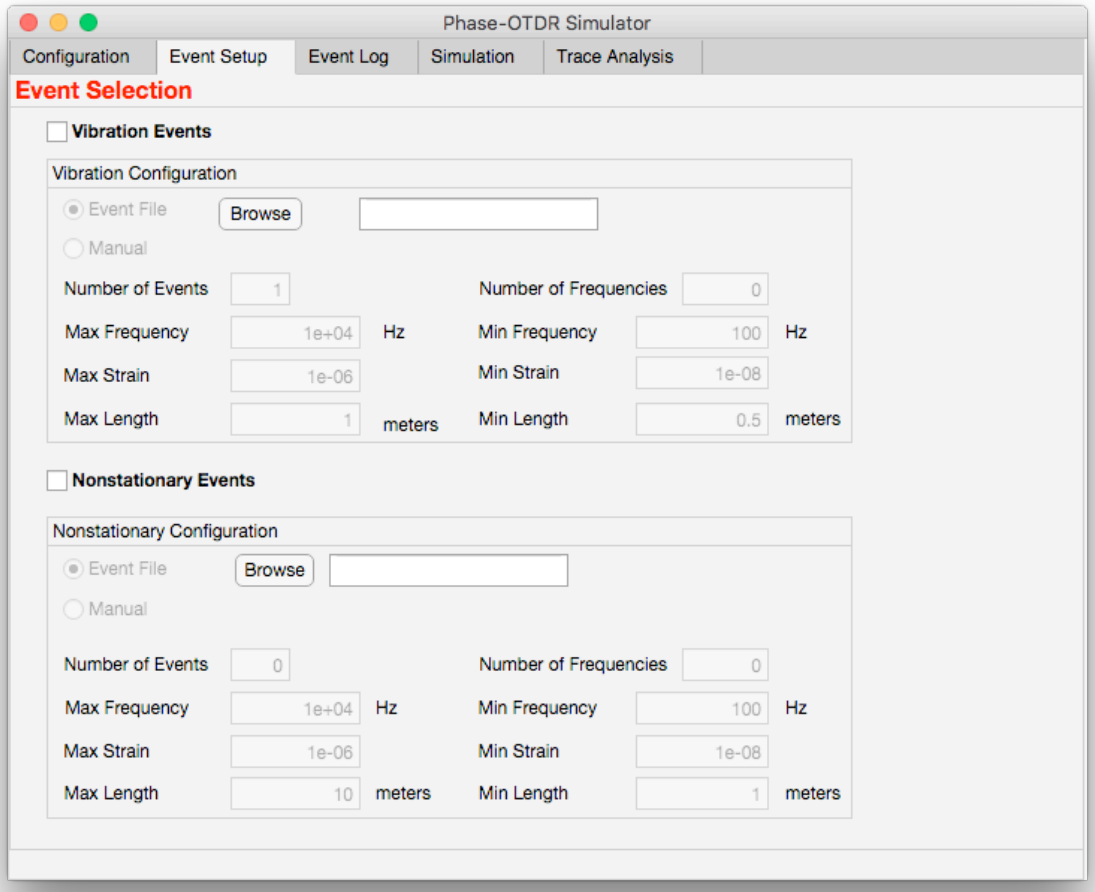


Fig. 20. Event Generation Panel of Simulator

3) **Simulation Panel** – The simulation panel (shown in Fig. 21), allows the user to stop or start a simulation, specify how many trace acquisitions to perform and contains a simulation log where status information is displayed.

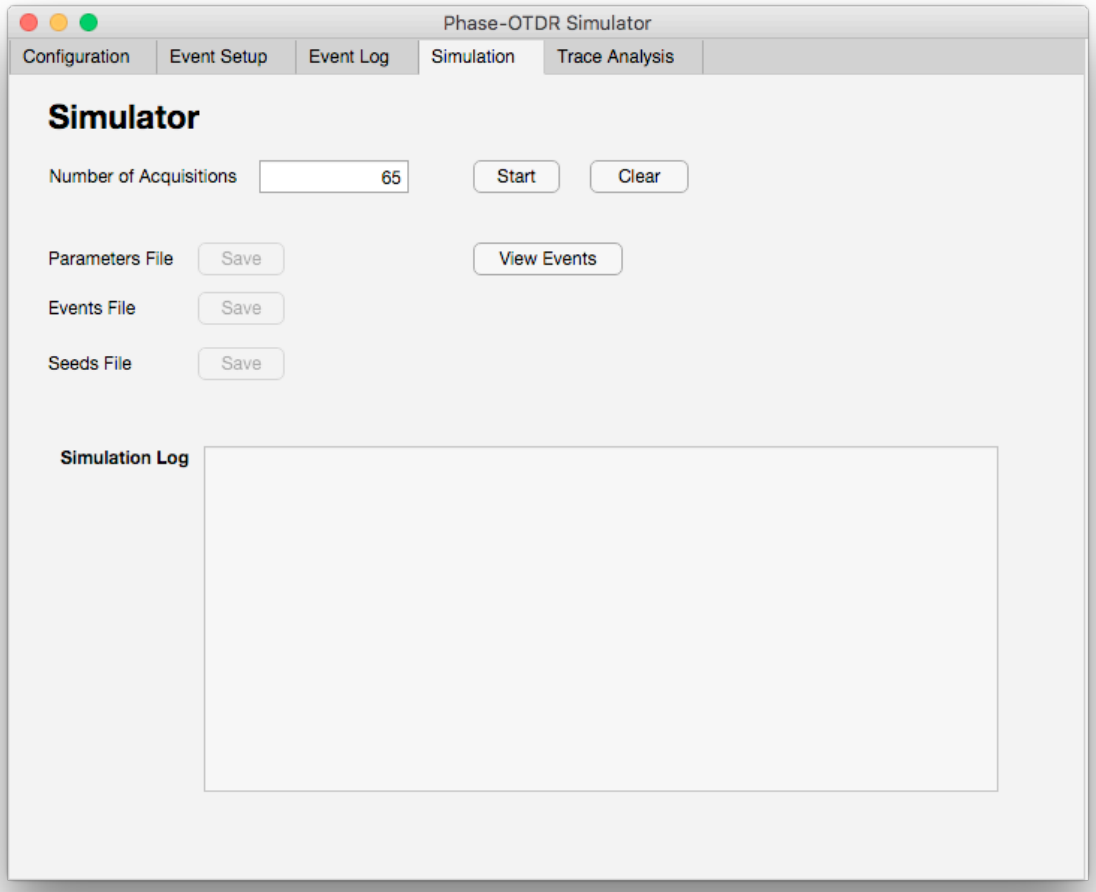


Fig. 21. Main Simulation Panel for Simulator

4) **Trace Analysis Panel** – The analysis panel (shown in Fig. 22), allows the user to view several different plots such as:

   a. Phase plot / deltas (difference between consecutive slow time traces)

   b. Intensity plot / deltas (difference between consecutive slow time traces)

   c. Phase differential plot / deltas (difference between consecutive slow time traces)

   d. Phase differential average plot / deltas (difference between consecutive slow time traces)

e. Laser phase noise

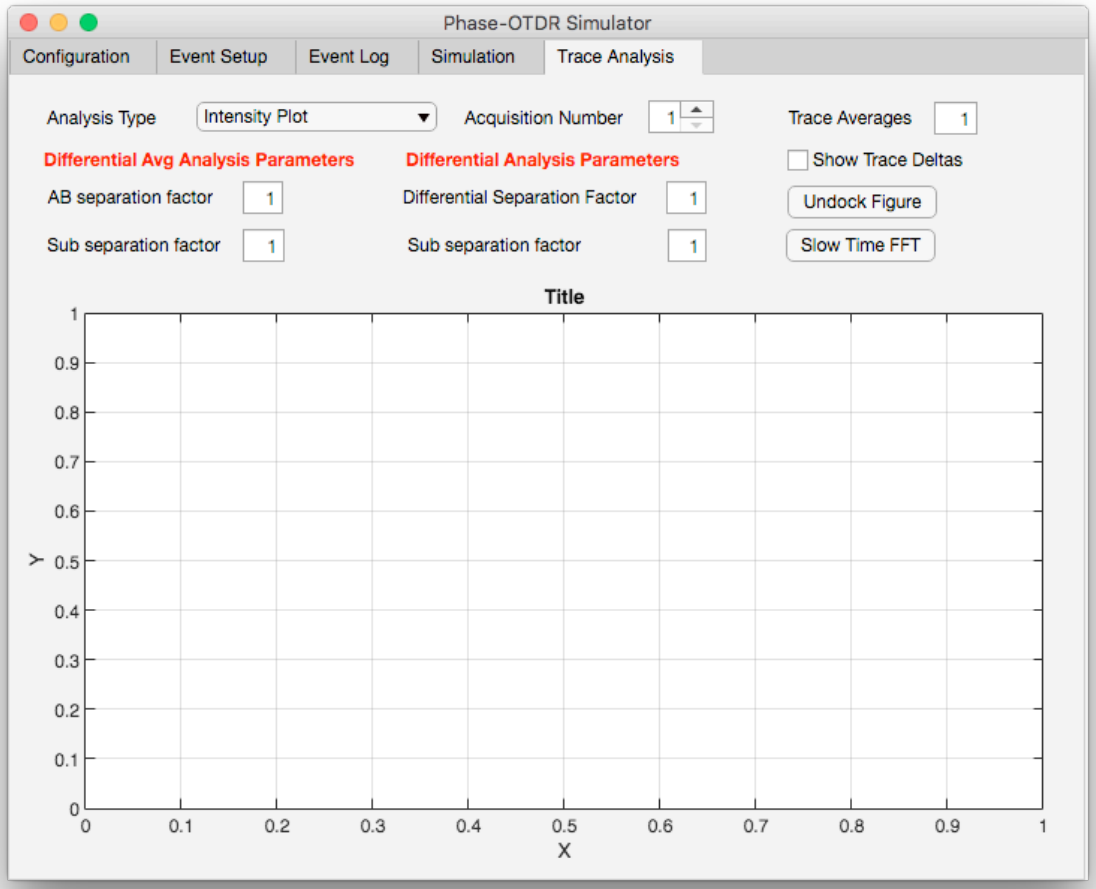


Fig. 22. Trace Analysis Panel of Simulator

4.4 Description of Fiber Plant Model

Although little detail is given in many of the papers that mention using simulations, the anecdotal evidence indicates that many numerical simulations operate on the model whereby scatterers are assigned random amplitudes and phase and are not given an explicit spatial location (since phase relates to position). This is problematic when there is a desire to simulate with different values of wavelength or refractive index, as these parameters will affect the phase value of the scatterers.

An excellent proposal for the modeling of scatterers for a phase-OTDR simulation has been proposed by Liokumovich *et al.* [79]. In their model, the scatterers are assigned a location on the z-axis (optic axis). Liokumovich does not consider how perturbations to the scatterers can be modeled in his work; therefore, this author will develop a model considering external perturbations. The rules for the fiber plant model proposed by Liokmovich are largely applied here and are summarized below:

1) Scatterers are defined by their amplitude and axial location $\{a_i, z_i\}$
2) A scatterer as defined in 1) (termed a *mini* scatterer) is considered to be an aggregation of many *micro* scatterers that satisfy the Rayleigh criteria where its size << λ. For a *mini* scatterer, it is also assumed that the distance between *mini* scatterers $\Delta z >> \lambda$. This helps to reduce the effective number of simulated scatterers to a number that is computationally achievable.
3) The distance between any two scatterers, $\Delta z << (v_g T_p) / 2$. This ensures that the number of scatterers that are scattered by the pulse is statistically significant.

One major difference between the model used by the Author and that of Liokmovich is in how changes of refractive index are handled. In Liokmovich's model, the total scattered field is represented as,

$$U_S = \sum_{z_1 < z_i < z_2} a_i \exp\left[j \frac{4\pi}{\lambda} \int_0^{z_i} n(x)\, dx \right] \qquad (4.1)$$

We see that the time delay (phase) of the scattered field is calculated taking into account the spatial variations in the refractive index up to the locations of the scatterers. This elucidates the fact that the propagation time is a function of refractive index and this leads to some additional complexities to the model. Firstly, the location of the propagating pulse is dependent on the

spatial variation of n, and secondly, the width of the propagating pulse will change depending on the spatial variations of n.

When considering the effects of external perturbations in the model, the largest effect to consider is spatial displacement of scatterers as will be discussed in section 4.2.4.2. Changes in refractive index will change the propagation constant of the probing pulse; however, this is equivalent to moving the spatial location of the scatterer when considering phase. For the purposes of this work, the spatial variations of n will be ignored.

4.4.1 Fiber Plant Parameters

The simulation allows for great flexibility on how the fiber plant is configured. This is accomplished by providing many parameters that can be changed within the GUI. The fiber plant parameters can either be considered to be belonging to the physical fiber model or part of the system model. Parameters related to the physical fiber model are listed in Table 3.

Table 3. List of Fiber Plant Parameters Used in Simulation

Parameter	**Description**
N – Number of scatterers	The total number of simulated scatterers distributed in the FUT
L – Length of FUT	The length of the optical fiber being tested.
α_A - Attenuation Coefficient	The attenuation of power per unit distance (dB/km)
n – Refractive Index	The refractive index of FUT

Parameters related to the system model are listed in Table **4**.

Table 4. List of System Model Parameters Used in Simulation

Parameter	**Description**
Tp – Pulse Duration	Temporal width of the probing pulse (s)
Ts – Sampling period	Sampling period of the system's ADC (SPS)
λ - Laser Wavelength	Wavelength of laser source (meters)
Δv – Laser linewidth	Linewidth of laser source (Hz)
$\Delta\lambda$ - Laser drift rate	Drift of laser frequency (Hz/s)
Fs – AOM frequency shift	The frequency upshift incurred by the AOM (Hz)

4.4.2 Algorithm for Generating Minimum Spaced Scatterers

The generated scatterers must be randomly positioned using a uniform distribution while maintaining a minimum separation between scatterers. The following algorithm was used to generate a set of scatterers that comply with this criterion.

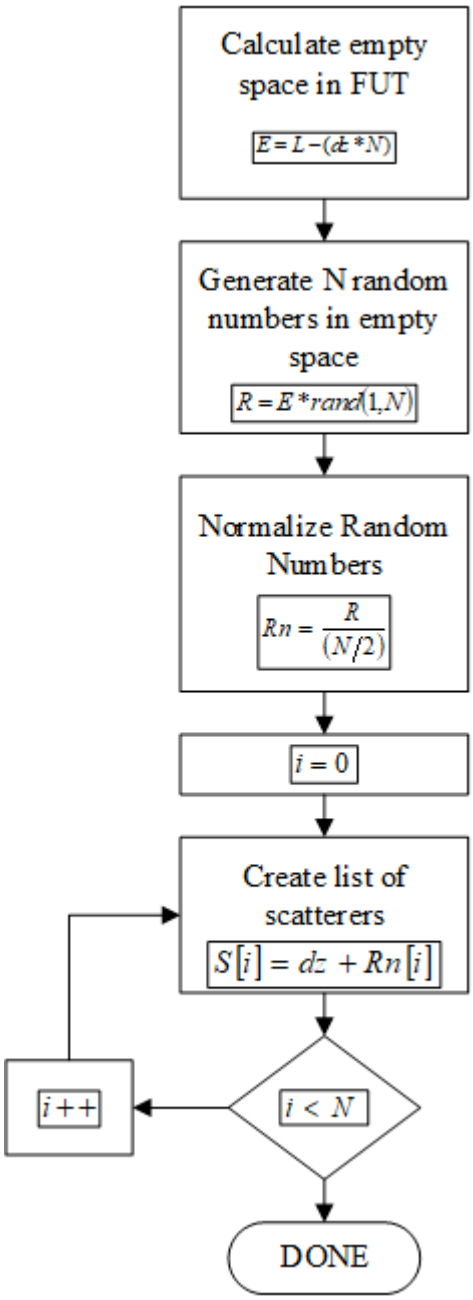


Fig. 23. Flowchart for Generating Scatterers with Minimum Spacing

4.4.3 Model of Noise Sources

Although a real system will include many different noise sources, such as electronic noise, environmental noise, and laser noise, for the purposes of this simulator only laser-related noise is considered. Other noise sources could be added to the simulator as future work.

4.4.3.1 Model of Laser Phase Noise

As covered in section 3.1.2, laser phase noise depends on the laser linewidth and the time since emission as expressed by Eq. 3.4. The simulator generates an array of normally distributed random variables with variance dependent on their spatial location. This noise is added as a phase term to Eq. 2.28.

4.4.3.2 Model of Laser Frequency Drift

All lasers experience frequency drift over time. This drift can be due to the effects of aging [80], [81] or thermal effects [82]. The measurement of frequency drift is a lengthy topic, which is not covered here. The reader is referred to the papers by Barnes [83] and by Walls [84].

As discussed by Barnes, the main problem with expressing a measure of frequency drift is that most often, the device exhibits noise that is the superposition of several types of non-deterministic random noise such as thermal, shot and flicker noise. Some standard noise measurements are therefore non-convergent (e.g., standard variance) and not very meaningful for larger time windows. A frequently used measure of frequency stability is the Allan variance, which is defined as,

$$\sigma_y^2 = \left\langle \frac{\left(\langle y_{n+1} \rangle - \langle y_n \rangle\right)^2}{2} \right\rangle \qquad (4.2)$$

Using the Allan variance, we interpret the laser drift frequency specification as the variance of the average frequency over the chosen averaging time τ.

For simulation purposes, we use a simple model for the laser frequency drift, where the drift is modeled as the superposition of two noise sources. One noise, v_s, is characterized by its variance over a short time scale and v_l, which is characterized by its variance over a long timescale.

$$v_d = v_l + v_s \tag{4.3}$$

The method used in the model to produce v_d is as follows:

1) Determine the total simulation time T_t. The total number of fast time samples will be $N=T_t/T_s$
2) Generate an array of short time noise values, $v_s[N]$, which are distributed as N(0, σ^2)
3) Generate an array of longtime noise values, $v_{l1}[2*T_r]$, which are distributed as N(0, σ^2), where T_r is the time for one slow time acquisition. This generates 2 points per slow time acquisition.
4) Using cubic splines interpolation, create an array of noise values, $v_l[N]$, that are derived from the values in v_{l1}.
5) Create the array for the total noise, v_d, as v_s+v_l

Examples of different laser frequency drifts created by the simulation over time are shown in Fig. 24.

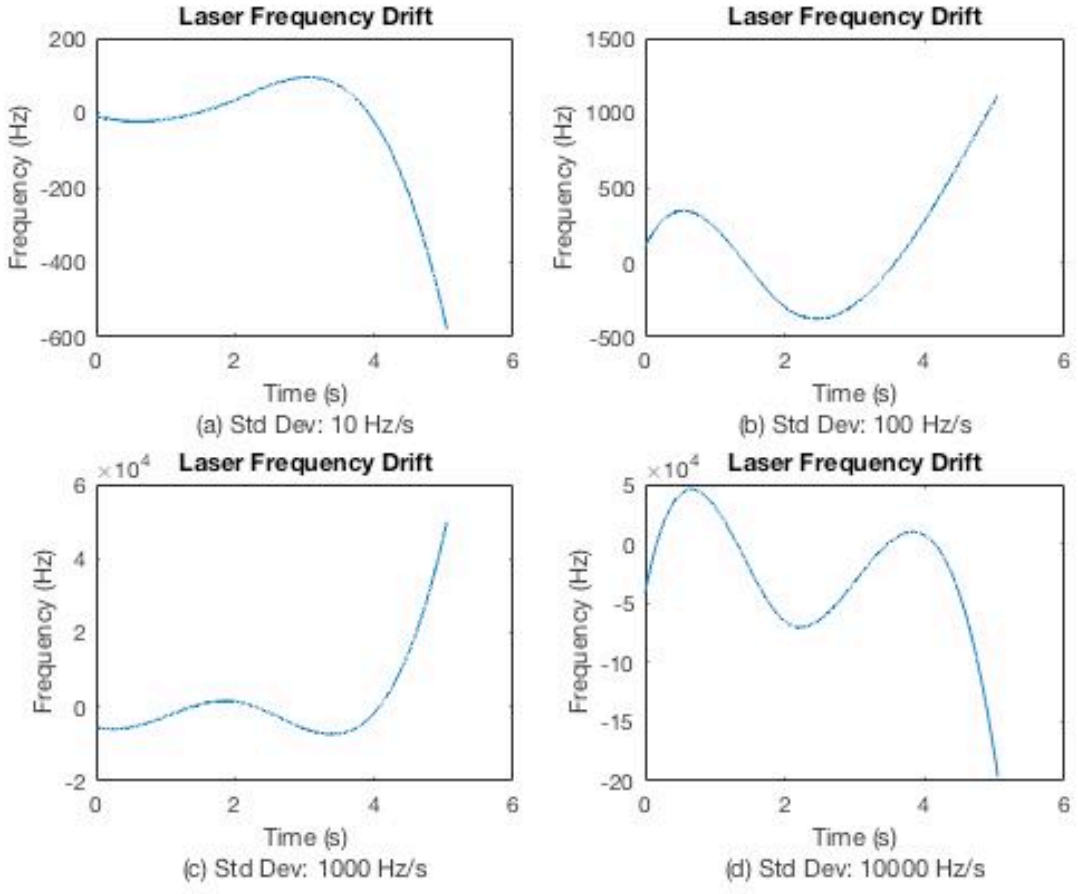


Fig. 24. Examples of Laser Frequency Drift with Various Drift Rates.

(a) Std Dev. 10 Hz/s (b) Std Dev. 100 Hz/s (c) Std Dev. 1000 Hz/s (d) Std Dev. 10000 Hz/s

4.5 Generation of Perturbation Models

Although many papers mention the results of simulated perturbations using a phase-OTDR system [85]–[88], to the Authors' knowledge, no particular details regarding how the events are mathematically modeled are available. When obtaining results by experiment, it appears commonplace to use a device such as a piezo-electric transducer to apply vibrations and strain as desired. This provides a very "clean" signal; however, in real-world applications we assume that the applied signal will be much noisier in nature. Therefore the goal will be to replicate something closer a disturbance found in real-world applications.

There are two particular considerations for the modeling of events, i) How closely the selected mathematical model estimates real-world perturbations and ii) how the changes are imparted to the physical model from the event.

4.5.1 Perturbation Event Generation Engine

As will be covered in section 4.5.6, events will be modeled as a strain that is applied to the fiber. This strain can be expressed as a time series, $x(t)$, where the amount of strain will vary over some time period.

The event processes used in the simulation will fall into one of two categories, stationary events or non-stationary events. Taking into consideration the types of applications that phase-OTDR is currently targeting we can make some arguments to their validity as simulated events and to derive reasonable constraints for their generation.

4.5.2 Synthesis of Stationary Perturbation Events Using AR Lattice Filter

The use of autoregressive (AR) lattice filters for modeling stationary random processes has broad applications in areas including, but not limited to, neurophysics, geophysics, speech and audio processing, economic forecasting and various industrial applications [89]. When considering phase-OTDR applications, we could argue that certain activities could produce perturbations that could be well modeled as stationary random processes. Essentially most mechanical type vibrations that are fairly consistent in frequency and sustained for several seconds or minutes would be good candidates; this could include:

1) Malicious activities – such as using tools to break through the protective sheath of an optical fiber.
2) Structural/pipeline monitoring – vibrations from structures or machinery.

3) Temperature changes – localized changes in temperature that cause thermal strain to the optical fiber.

For any real process, such as those listed above, we can assume that it will be corrupted by noise in some way. For the purposes of this simulation, we will assume this noise is white in nature.

One of the attractive properties of using AR processes for generating a random time series is that only the filter coefficients are needed to represent the process. It is convenient therefore to use this approach since an event can be stored as a set of coefficients which can then generate a random process of arbitrary length since the generation of process data is done in an iterative fashion.

An AR process is an all-pole filter and can be expressed as,

$$x[n] = -\sum_{k=1}^{p} a_k x[n-k] + w[n] \qquad (4.4)$$

where a_k are the AR coefficients and $w[n]$ is white noise.

The event model needs to describe a filter, $H(z)$, such that given white noise as the input, we obtain an output with the desired power spectrum as illustrated in Fig. 25.

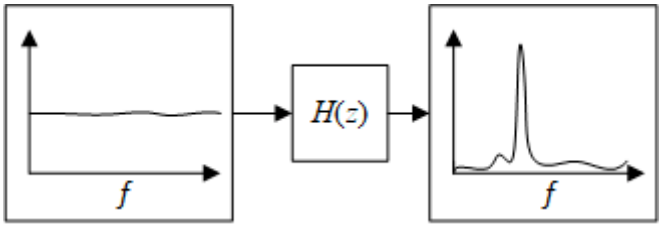


Fig. 25. Example of Effect of All-Pole Filter on White Noise Input

To create the desired perturbation process, the following method is used:

1) Generate the desired perturbation signal, $y(t)$, of a particular set of frequencies and amplitudes

$$y(t) = \sum_{k=1}^{p} A_k \sin(2\pi f_k t) \qquad (4.5)$$

2) Calculate the autocorrelation $\gamma_{yy}(t-\tau)$ of $y(t)$

3) Solve for the AR coefficients, a_k, using the Levinson-Durbin method.

One method that can be used to implement an AR process is to use a lattice filter structure [90]. For the developed simulator, the lattice filter was chosen for reasons that will become clear in section 4.5.4.

A lattice filter implementation for an AR process is shown in Fig. 26.

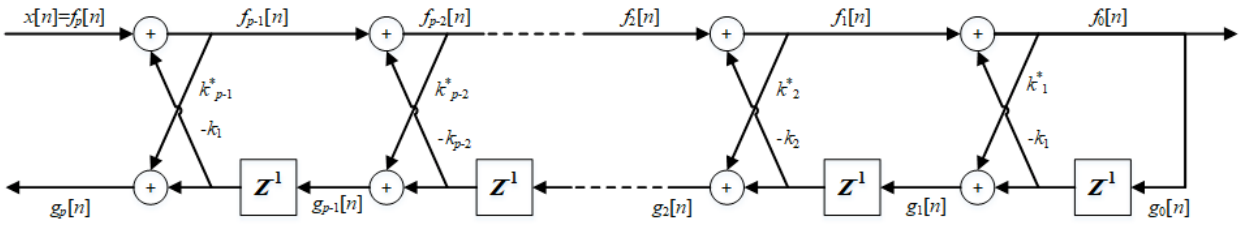


Fig. 26. Lattice Filter Representation of AR Process

As can be seen in Fig. 26 both the forward predictors and backward predictors are calculated when using the lattice filter. The reflection coefficients of the lattice filter, k_p, are equal to the AR coefficients so that $k_p = a_k$ where $p=k$.

The output of the lattice filter, $f_0[n]$, can be found using the following iterative process:

1) Assign input $x[n]$ to $f_p[n]$

2) Calculate the forward predictor stage, $f_{p-1}[n]$, as

$$f_{p-1}[n] = f_p[n] - k_p g_{p-1}[n-1] \qquad (4.6)$$

3) Calculate the backward predictor stage, $g_p[n]$, as

$$g_p[n] = k_p^* f_{p-1}[n] + g_{p-1}[n-1] \qquad (4.7)$$

4) Let $p = p - 1$ and repeat steps *2* & *3* until $p = 0$.

4.5.3 Results of Simulating Stationary Perturbation Events

The simulator randomly selects the parameters of the perturbation signal, $y(t)$, given constraints provided by the user, this includes:

- Number of frequencies in an event
- Frequency range of the frequencies
- Range of lengths for the perturbed region

Fig. 27 shows the power spectral density (PSD) of two generated perturbation events along with the PSD of the output of their associated AR lattice filters.

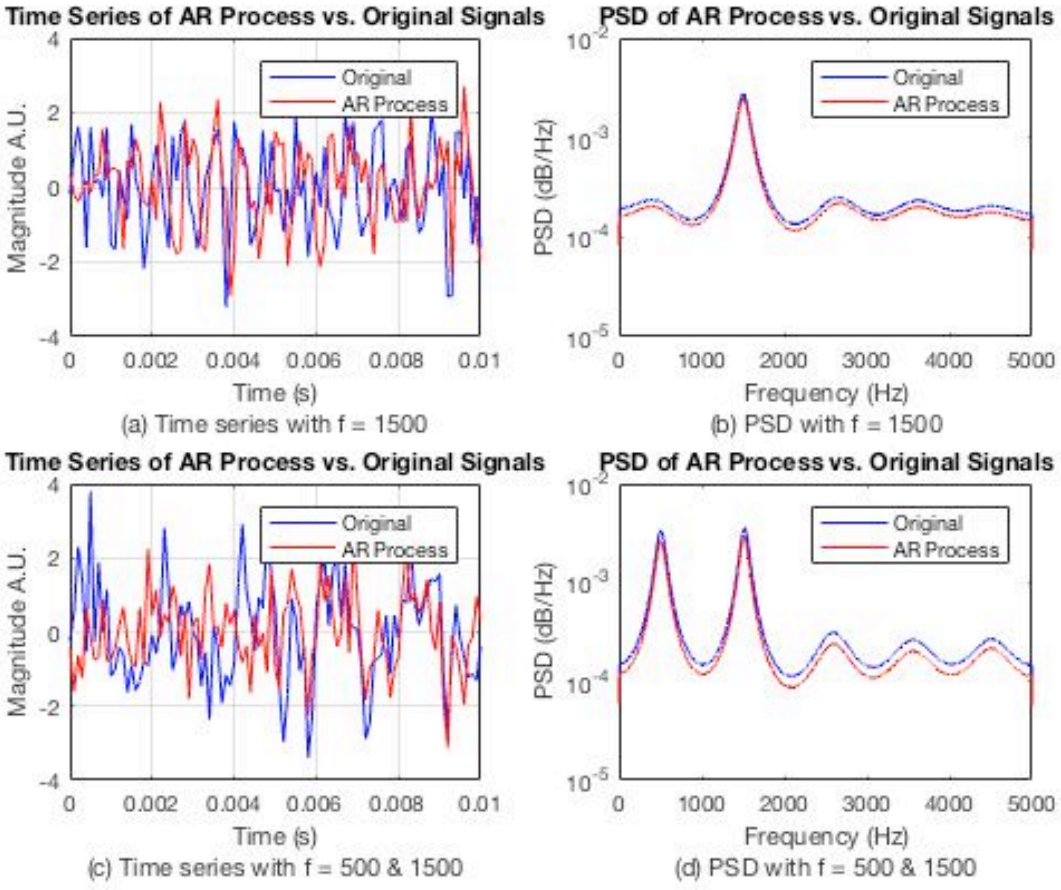


Fig. 27. Time Series & PSD of AR Process vs. Time Series & PSD of Original Process

(a) Process 1: Time Series f=1500 Hz (b) Process 1: PSD f=1500 Hz (c) Process 2: Time Series f=500Hz & 1500Hz (d) Process 2: PSD f=500Hz & 1500Hz

As can be seen, the PSD of the AR process matches closely to the desired frequency content for the event.

4.5.4 Synthesis of Non-Stationary Perturbation Events Using Frequency Warped AR Lattice Filter

Frequency warped filters were first postulated by Strube [91] as an efficient way to model processes with a nonlinear frequency scale. This has since found applications in speech and audio processing [92], [93]. Acoustic sensing using DOFS, such as phase-OTDR, operates using the same principles as vibration sensing applications [94]. With acoustic signals such as speech, however; the detected signal is more likely to be non-stationary in nature. The use of frequency warped filters is therefore an appealing approach for simulating these types of events.

The warped frequency lattice filter is implemented by replacing the delay component in a standard AR lattice filter with an all-pass filter [95]. The all-pass transfer function can be expressed as,

$$D(z) = \frac{\left(-\lambda + z^{-1}\right)}{\left(1 - \lambda z^{-1}\right)} \qquad (4.8)$$

The λ term affects the frequency shift applied by the filter with positive values causing an up-chirp of the original frequency and a negative value causing a down-chirp of the original frequency as shown in Fig. 28.

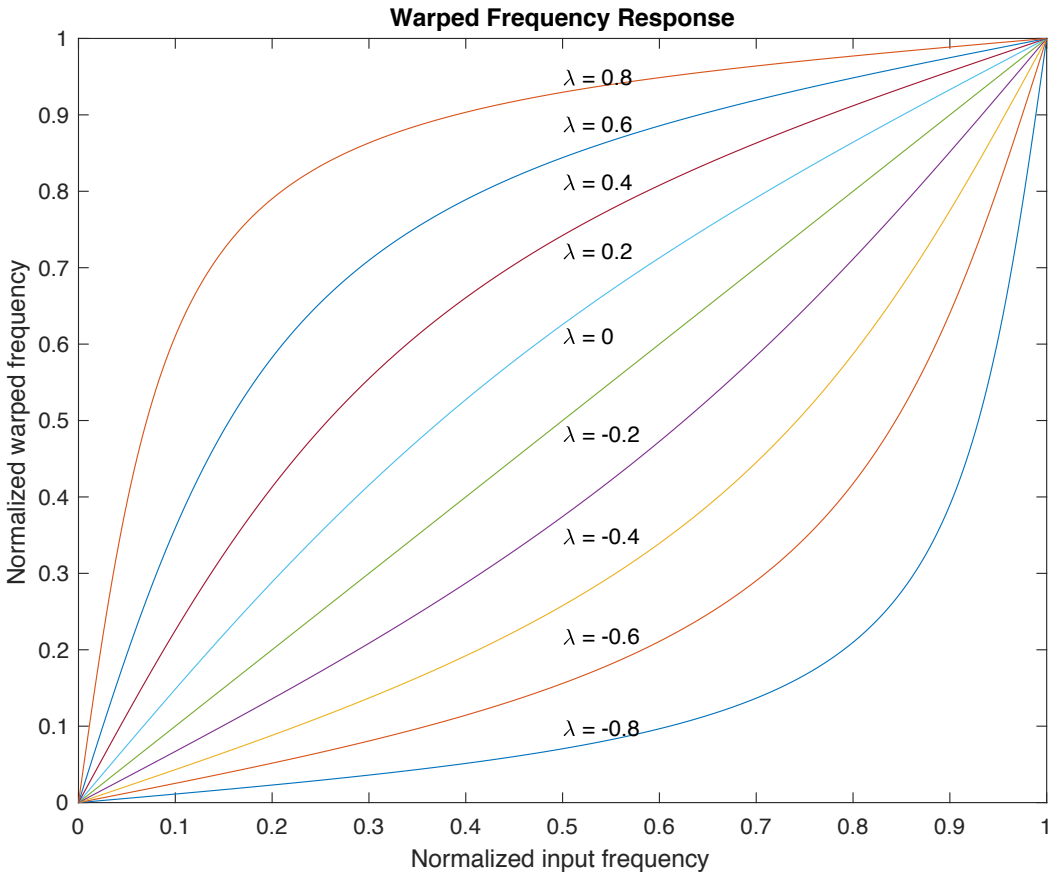


Fig. 28. All-Pass Filter Response for Various Warping Parameters

The actual implementation of the frequency warped AR lattice filter is shown in Fig. 29.

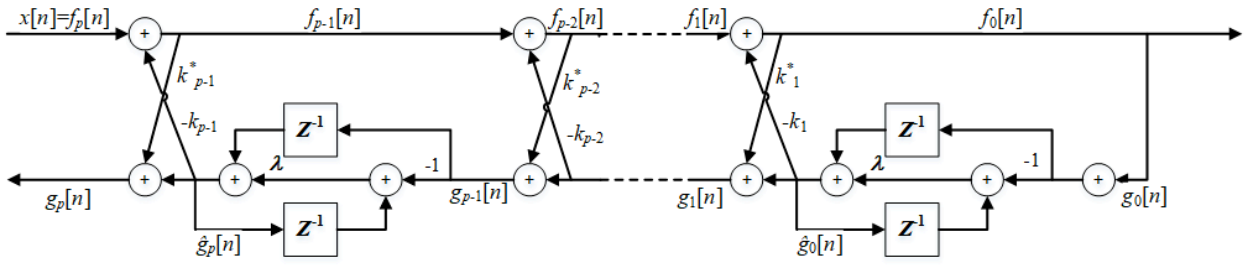


Fig. 29. Lattice Filter Representation of Frequency Warped AR Process

As shown in Fig. 29 the implemented filter is similar to that shown in Fig. 26, but with the delay blocks replaced by an all-pass component.

The output of the lattice filter, $f_0[n]$, can be found using the following iterative process:

1) Assign input $x[n]$ to $f_0[n]$

2) Calculate the backward predictor estimate for stage p, $g_p[n]$ as

$$\hat{g}_p[n] = g_{p-1}[n] - \lambda\left(g_{p-1}[n] - \hat{g}_p[n-1]\right) \quad (4.9)$$

3) Calculate the backward predictor for stage p as

$$g_p[n] = \hat{g}_p[n] + k_p^* f_{p-1}[n] \quad (4.10)$$

4) Calculate the forward predictor for stage p as

$$f_p[n] = -k_p \hat{g}_p[n] + f_{p-1}[n] \quad (4.11)$$

5) Let $p = p$ - 1 and repeat steps 2 through 4 until $p = 0$.

In step 2), if the warping parameter λ=0, we obtain the standard AR lattice filter described in section 4.5.2. From an implementation standpoint, it is convenient therefore to use the same structure for generating both stationary and non-stationary events and let λ=0 for stationary events and $\lambda \neq 0$ for non-stationary events.

It should be noted that implementing a frequency-warped filter does not impose non-stationarity of the generated signal. Assuming the original signal contains no time-varying frequency content, the output of the frequency-warped filter will simply shift the frequency content up or down. To address this issue, the simulation considers the warping factor, λ, to be a time-varying parameter. The all-pass filter model therefore becomes time-dependent,

$$D_n(z) = \frac{\left(-\lambda[n] + z^{-1}\right)}{\left(1 - \lambda[n] z^{-1}\right)} \qquad (4.12)$$

4.5.5 Results of Simulating Non-Stationary Perturbation Events

The simulator randomly selects the parameters of the perturbation signal, $y(t)$, given constraints provided by the user, this includes:

- Number of frequencies in an event
- Frequency range of the frequencies
- Range of lengths for the perturbed region

Fig. 30 shows the power spectral density (PSD) of three generated perturbation signal along with PSD of the output their associated AR lattice filters.

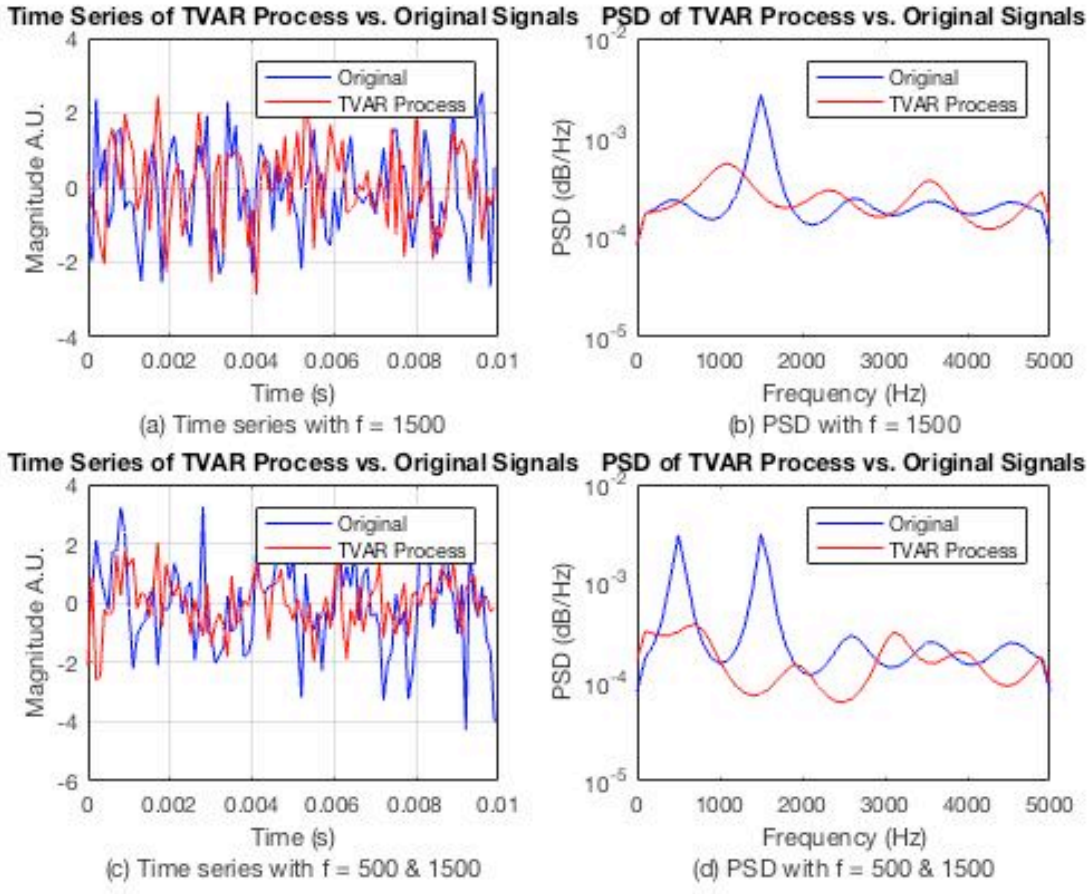


Fig. 30. Time Series & PSD of Frequency Warped AR Process vs. Time Series & PSD of Original Process (a) Process 1: Time Series f=1500 Hz (b) Process 1: PSD f=1500 Hz (c) Process 2: Time Series f=500Hz & 1500Hz (d) Process 2: PSD f=500Hz & 1500Hz

As can be seen, the PSD of the warped AR process shows some shifting in its spectrum compared to the original signal.

4.5.6 Changes to the Fiber Plant Model When Subjected to External Forces

As force is applied to an optical fiber, it undergoes physical changes that will change the physical dimensions of the fiber altering the propagation of light through the perturbed section. This change in propagation is due to the physical deformation of the fiber caused by stress/strain.

For optical materials, the change in phase of an optical wave due to applied stress is expressed through the photo-elastic coefficients.

The effects of axial strain on SMF has been investigated by Bertholds [96], specifically the induced phase change resulting from nonlinear deformation of the fiber in the axial and trans-axial directions. Other types of strain have also been investigated, including micro and macro bends, as well as twists; however, this research has mostly focused on the resulting changes in material birefringence which requires a polarimetric measurement [70], [97], [98].

To simplify the event modeling, the types of forces considered will be limited to axial strain. This seems reasonable since the current model has no way to describe the spatial distribution of the scatterers other than their axial position. Therefore only changes in length can be described given the current model.

The change in length of the fiber given an applied axial strain, ε, is given by [99],

$$\Delta L = \varepsilon L \tag{4.13}$$

In addition to the change in length, the refractive index of the perturbed section also changes due to change in size to the fibers' core radius.

$$\Delta n = -\frac{1}{2} n^3 \varepsilon \left(p_{12} - \zeta \left[p_{11} + p_{12} \right] \right) \tag{4.14}$$

The resulting change in phase is a function of both the change in length and the change in refractive index,

$$\Delta \phi = k \left(n \varepsilon L + \Delta n L \right) \tag{4.15}$$

substituting Eq. 4.14 into Eq. 4.15 yields,

$$\Delta\phi = kL\varepsilon n\left(1 - \frac{n^2}{2}\left[p_{12} - \zeta\left(p_{11} + p_{12}\right)\right]\right) \quad (4.16)$$

Where k is the wavenumber of the wave, L is the length over which the force is applied, ε is the applied strain, n is the refractive index, ζ is the Poisson ratio of the SMF, and p_{12}, p_{11} are the photo-elastic coefficients of the SMF. From Bertholds [100], we find that ζ=0.16, p_{12}=0.27 and p_{11}=0.121 for standard SMF. Eq. 4.16 then reduces to,

$$\Delta\phi = 0.78 * kL\varepsilon n \quad (4.17)$$

for n=1.47, which is a typical value for SMF.

Since we are not considering refractive index variations, the term from Eq. 4.14 needs to be considered as a change in length. From Eq. 4.17, we can rewrite the total phase change as,

$$\frac{2\pi n\Delta L_{eff}}{\lambda} = 0.78 * \frac{2\pi n}{\lambda} * \Delta L \quad (4.18)$$

where ΔL_{eff} is the effective change in length when considering the effects of elongation and refractive index changes. The effective modeled change in length is therefore $0.78\varepsilon L$.

The following assumptions and restrictions are used for modeling the physical changes due to strain:

1) *All strains are expressed as axial strains* – Since proposed model for the spatial distribution of scatterers assumes that scatterers only exist along the optic axis, deformations that would cause the scatterers to move in a direction that does not lie on the optic axis will have no effect on the calculated backscatter.
2) *The applied strain must be* $0 \leq \varepsilon \leq 6.5*10^{-3}$ - Strains above this amount have been shown to exceed the break threshold of SMF [101].

3) *Strains will be evenly applied across the perturbed section* – This assumption is made for simplicity only.

4) *The length of a perturbed section is restricted to be* $0 < l_c \leq 100$ m - This is arbitrarily chosen as to be an adequately long area of perturbation.

Given that the effects of a strain applied to an optical fiber will be expressed as changes to L, within the area of perturbation, an event will be simulated using the following steps:

1) Determine the range for a perturbation applied at L_{pert} as $A = L_{pert} - L/2$ and $B = L_{pert} + L/2$
2) Calculate total displacement over length L as $\Delta L = 0.78 L\varepsilon$
3) Move each scatterer whose location, $A \leq z_i < L_{pert}$, by $-\Delta L/2$.
4) Move each scatterer whose location, $L_{pert} < z_i \leq B$, by $+\Delta L/2$.

More advanced methods could be implemented in the future to apply more realistic changes to the physical model. Applying different weighting windows to the area of perturbation would provide a more realistic distribution of strain. For instance, applying a Gaussian window, where the amount of strain tapers off as the distance from the applied strain is increased may improve results.

4.6 Simulation of Backscattered Light Generated By a Propagating Pulse in a Singlemode Optical Fiber

The simulator simulates a laser pulse propagating down the fiber under test and calculates the backscattered field detected by the coherent receiver. The acquiring of this backscattered intensity over the length of the fiber is referred to as a *fast-time* acquisition (as discussed in Chapter 2). To reduce the complexity of the simulation, the following simplifications are made to the model:

1) **Polarization effects are neglected** – As covered in section 3.2.2, it is known that polarization fading will occur in coherent phase-OTDR systems. These effects are neglected in the current model to reduce complexity.
2) **Effects of pulse shape are ignored** – As covered in section 3.3.3, changes in pulse shape will have an effect on system performance. The model assumes a rectangular pulse. This pulse shape is still the most commonly used shape due to the ease in which it can be produced.
3) **Ideal electronics are assumed** – Components such as the ADC are assumed to work in an ideal way.

The fast-time acquisition is performed as follows:

1) At time t, determine the location, z_L, of the leading edge of the pulse in the FUT.
2) Calculate the sum of the scattered fields from all scatterers located between z_L-$W_p/2 \leq z \leq z_L$ that contribute to the beat frequency.
3) Calculate the phase of the detected signal using phase unwrapping
4) Increment the time by the sampling rate, $t = t + t_s$
5) Repeat steps 1 through 4 until the entire length of the FUT is sampled.

Both laser phase noise and laser drift will be applied to the simulation if specified.

4.6.1 Calculation of Detected Intensity

The calculated intensity is determined by the number of scatterers, N, within the probe pulse at time t and is computed as,

$$I = \sum_{n=i}^{i+N} a[n] exp\left(-2\alpha_A z[n]\right) cos\left(2\pi\Delta v[n] t - 2\pi v[n]\tau_n + \vartheta_n[n]\right) \quad (4.19)$$

Here we have $a[n]$ and $z[n]$ as the amplitude and spatial location of the nth scatterer. $\Delta v[n]$ is the beat frequency between the scattering wave and the LO and $v[n]$ is the probe frequency, both terms include laser drift, thus,

$$\Delta v[n] = \Delta v_{fs} + \Delta v_d[n] \quad (4.20)$$

$$v[n] = v_0 + \Delta v_d[n] \quad (4.21)$$

The last phase term of Eq. 4.19, $\vartheta_n[n]$, represents the phase noise. In the simulation, we assume that the bandwidth of the photo-detectors is unlimited.

Once the intensity term is calculated, it can be *I/Q* demodulated back to baseband where it is filtered in the simulation. A digital low pass filter (LPF) is used, where the bandwidth is chosen to be 40% greater than the BW of the probe pulse. This has been shown to be an optimal BW setting for phase-OTDR systems using rectangular probes pulses [102].

4.6.2 Phase Extraction and Unwrapping

Once the *I* and *Q* signals are filtered, the phase of the signal is computed as,

$$\phi(t) = a\tan 2\left(\frac{I(t)}{Q(t)}\right) \quad (4.22)$$

Once all time steps in the fast-time acquisition are calculated, all of the calculated phases are then unwrapped using a phase unwrapping algorithm. The standard phase unwrapping algorithm works as follows:

if abs(ϕ[n+1] - ϕ[n]) > π then

    if (ϕ[n+1] < ϕ[n]) then

        ϕ[n+1] = ϕ[n+1] + 2π

    else

$\phi[n+1] = \phi[n+1] - 2\pi$

end

end

The major issue with this technique is that any noise included in the *I* and *Q* measurements can cause the phase unwrapping algorithm to erroneously assign the phase to be $\pm 2\pi$ if the true angle is close to π. This can cause the evolution of the phase terms over a fast time acquisition to differ substantially over time.

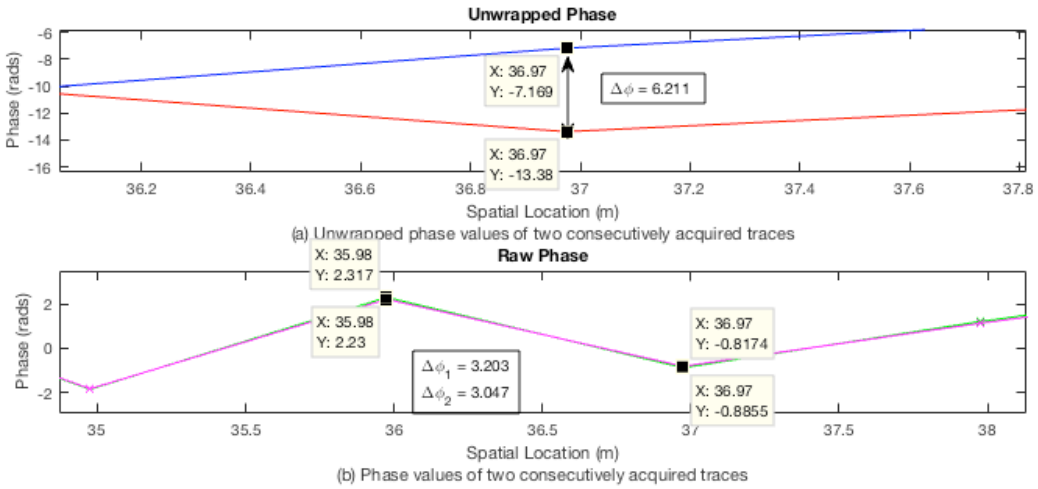


Fig. 31. Example of Phase Unwrapping Error Due to Phase Noise

Fig. 31 shows an example where the unwrapped phase causes a divergence in phase at a specific location between two consecutively acquired backscatter traces. The bottom figure of Fig. 31 shows the raw phase values of the two traces. The first trace shows a phase difference that is less than π between two data points, while the second trace shows a phase difference of greater than π. This small change between the two traces is caused by phase noise present in the simulation. Applying the unwrapping algorithm to this data results in the traces shown in the top figure of Fig. 31 where a phase difference of ~2π is now introduced.

4.7 Simulation of External Forces Applied to a Single Mode Optical Fiber

The time between fast-time acquisitions is referred to as *slow-time* (as described in Chapter 2). In the simulation, several fast-time acquisitions are performed. It is between these acquisitions where the perturbation events are applied to the fiber model. In a real-world situation, the perturbation would be continuously applied to the fiber; however, modeling the effects of the perturbation over every time step adds a significant amount of additional processing to the simulation. A short analysis is provided to validate the chosen approach, followed by an explanation of how the changes discussed in section 4.5.6 are applied to the physical model.

4.7.1 Effect of Perturbation Length and Pulse Width on Event Detection

It is only over the duration in which a light pulse travels through a perturbed section of fiber that any changes to the measured phase will be detected. The total amount of time that a perturbed section is illuminated is,

$$T_L = \frac{\left(L + W_p\right)}{v_g} \tag{4.23}$$

where L is the length of the perturbed section, and W_p is the width of the pulse. From Eq.4.15 we can say that that

$$\Delta\phi \propto \frac{2\pi}{\lambda} n\varepsilon L \tag{4.24}$$

If we assume that the strain applied is sinusoidal with a frequency, f, then

$$\Delta\phi \approx \frac{2\pi}{\lambda} n\varepsilon L \sin\left(2\pi f T_L\right) \tag{4.25}$$

Table 5 shows the estimated phase change based on various values of pulse width, perturbation length, and perturbation frequency.

Table 5. Calculation of Phase Change Induced by Perturbations of Various Lengths and Frequencies for Various Probe Pulse Widths

Pulse Width (m)	**Perturbation Length (m)**	**Frequency (Hz)**	**Phase Change (% max)**
1	10	20000	0.691%
2	10	20000	0.754%
3	10	20000	0.817%
4	10	20000	0.880%
5	10	20000	0.942%
6	10	20000	1.005%
7	10	20000	1.068%
8	10	20000	1.131%
9	10	20000	1.194%
10	10	20000	1.257%

The estimated changes are less than 2% of the total expected phase change. The approach taken therefore seems reasonable.

A slow-time acquisition is performed as shown by the flowchart in Fig. 32.

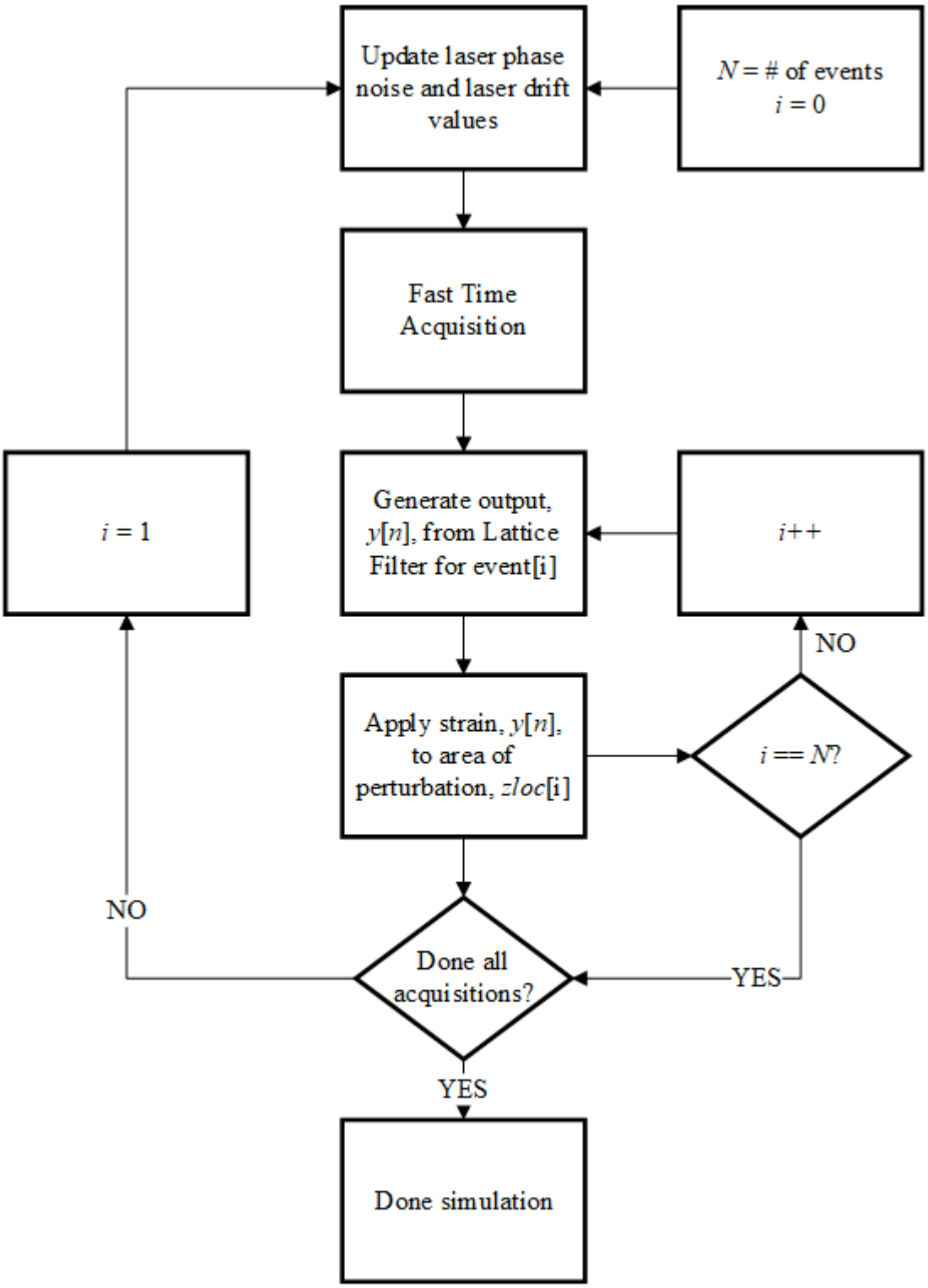


Fig. 32. Simulation Program Flow of Slow Time Acquisition

4.7.2 Verification of Simulator Operation By Applying Known Perturbation

To verify that the simulator operates as intended, based on the models and calculations presented, a simple simulation of a single known perturbation is performed and the results analyzed. For this test, the following steps are followed:

1) Fiber under test is randomly generated using parameters as shown in Table 6.
2) A fast time acquisition is performed with a pulse width equal to the sample rate (approximation of an impulse function), where the sampling resolution is $<< L$ (length of perturbation)
3) The phase information is extracted for the acquired trace.
4) A known strain, ε, is applied to scatterers between $(z_{loc}-\Delta L/2) \leq z \leq (z_{loc}+\Delta L/2)$.
5) A subsequent fast time acquisition is performed.
6) The phase information for the second trace is extracted.
7) The difference between the two acquired phase traces is calculated.
8) The phase change along the region of the disturbance is measured. The resulting phase change should be equivalent to Eq. 4.17 based on the applied strain.

The above test is performed 3 times with different perturbation settings as shown in Table 7.

Table 6. Simulation Parameters Used for Verification Tests

Parameter	Setting
Number of scatterers	100000
Length of Fiber	500 meters
Central Wavelength	1550 nm
Sampling Rate	1 GSPS

Pulse Width	0.5nS
Refractive index	1.47
Laser Linewidth	0 kHz
Laser Frequency Drift	0 MHz/s

Table 7. Perturbation Parameters Used for Verification Tests

Parameter	**Test 1 Settings**	**Test 2 Settings**	**Test 3 Settings**
Perturbation Location	300 meters	300 meters	300 meters
Perturbation Length	10 meters	10 meters	10 meters
Strain	1e-7	1e-8	1e-9
Expected Phase Change	4.6479 rad	0.46479 rad	0.04649 rad

The resulting phase backscatter trace of Test 1 is shown in Fig. 33.

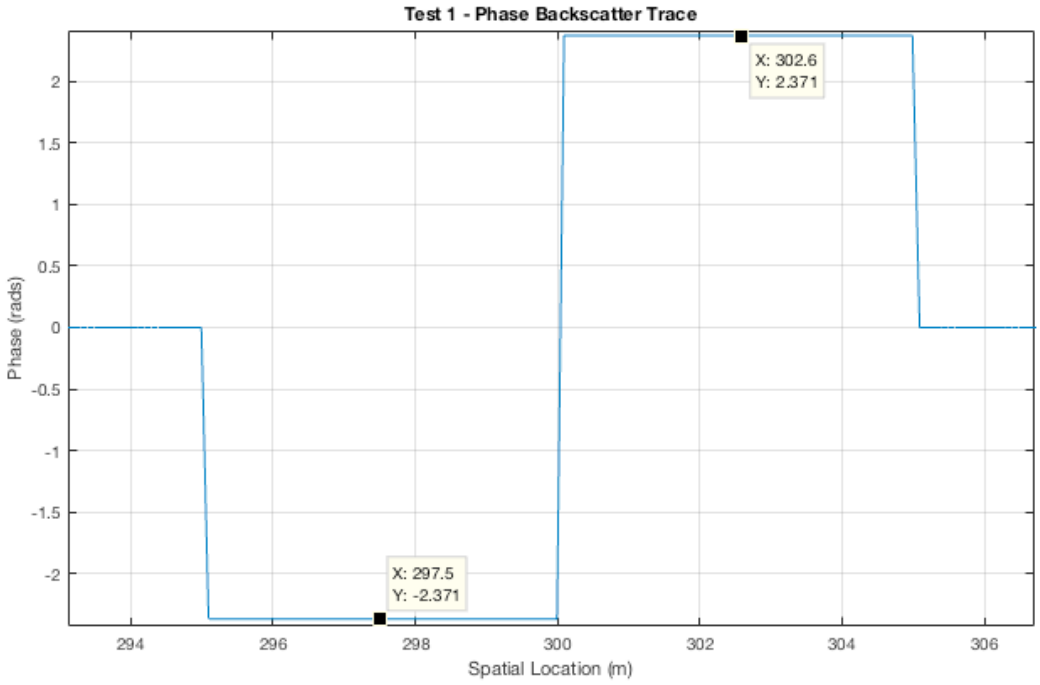


Fig. 33. Test 1 - Phase Backscatter Trace

The total measured phase change is 4.742 radians, which is within 2% of the expected change. The 10 m wide perturbation causes a phase change of -2.371 rads between 295 m and 300m, which is expected because in the simulation the scatterers are moved toward the source (causing a negative phase change). Between 300 m and 305 m the phase change is +2.371 rad, resulting from the scatterers on the right side of the center location of the perturbation being moved away from the source (causing a positive phase change).

The resulting phase backscatter trace of Test 2 is shown in Fig. 34.

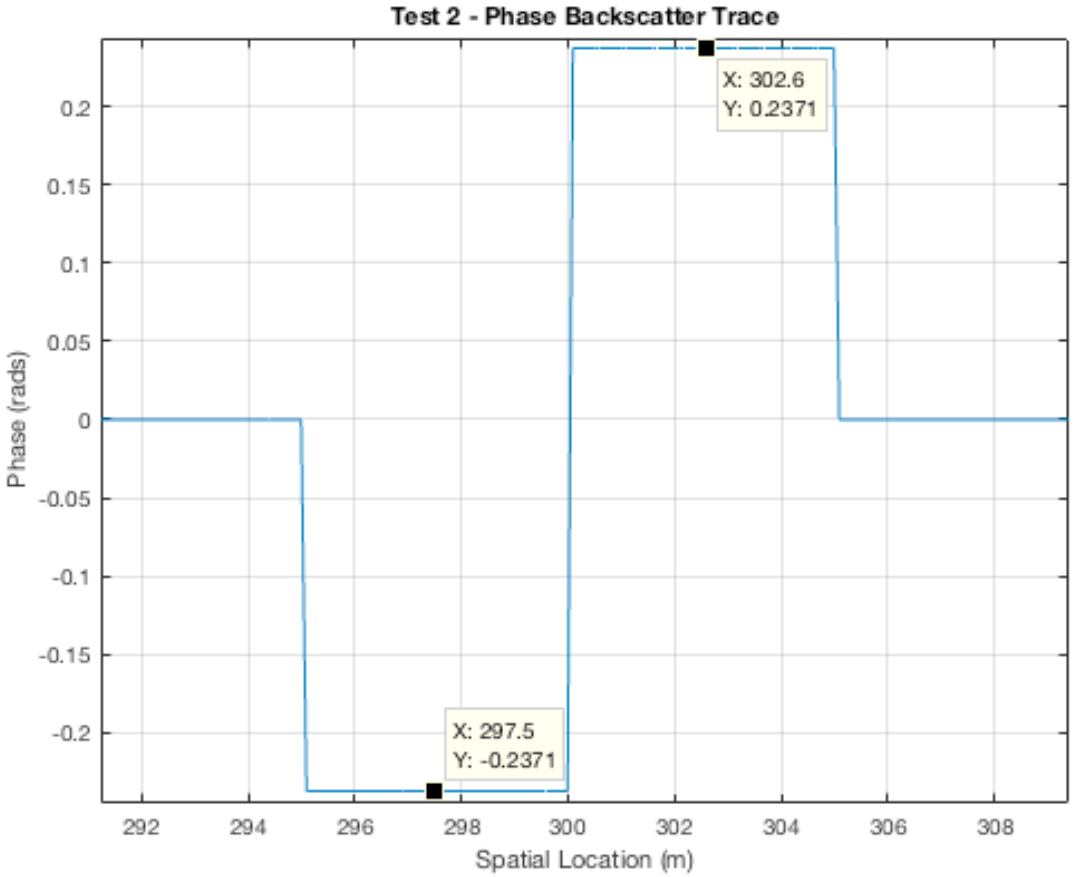


Fig. 34. Test 2 - Phase Backscatter Trace

The total phase change is 0.4742 radians, which is within 2% of the expected change. The resulting phase backscatter trace of Test 3 is shown in Fig. 35.

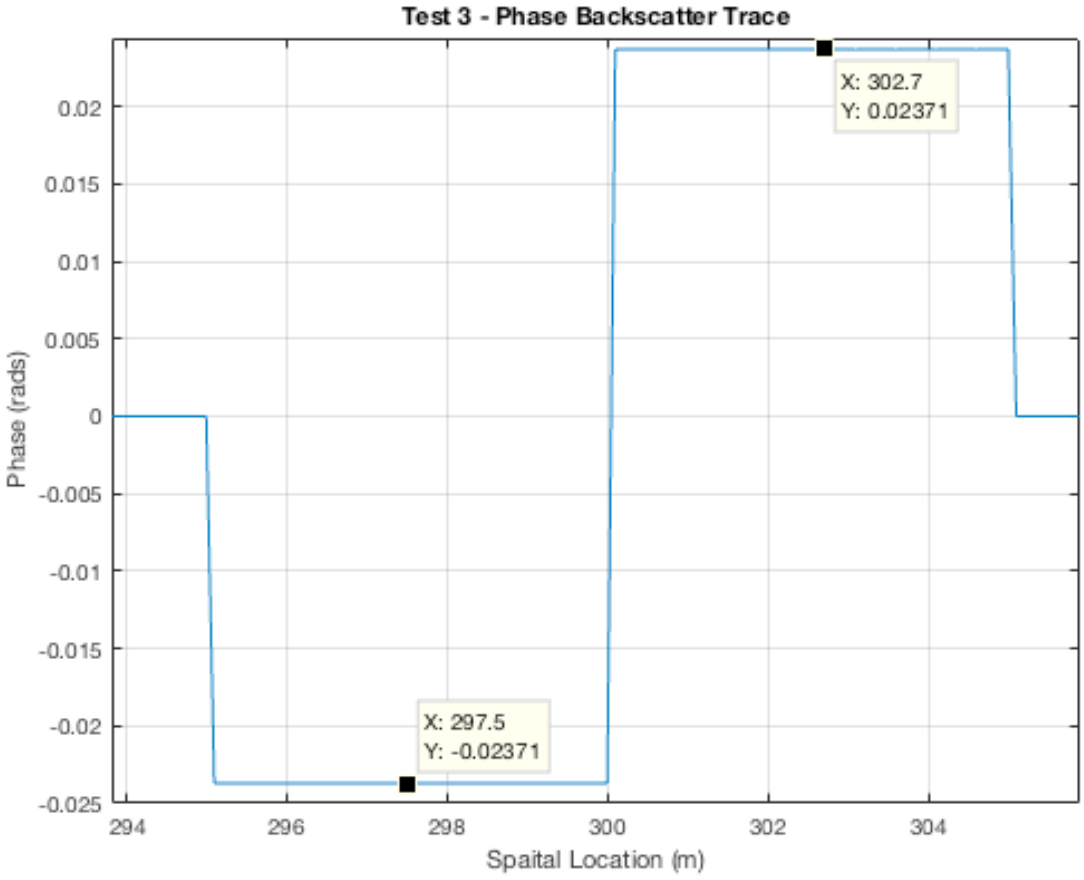


Fig. 35. Test 3 - Phase Backscatter Trace

The total phase change is 0.04742 radians, which is within 2% of the expected change.

These results show that the simulation software provides output that is consistent with the mathematical models described throughout this thesis.

4.8 Coherent Phase-OTDR Backscatter Trace Analysis

For any application using coherent phase-OTDR, the backscatter trace must be analyzed to extract the desired information. The methods used for analyzing phase-OTDR data are many and are not the main focus of this work. It is essential, however, that some type of trace analysis be implemented in the simulator in order for its results to be verified. Therefore, a few of the more

common methods have been implemented. This chapter will give a brief description of the methods implemented as well as some example results from their use for various simulations performed. The discussion of these analysis methods is preceded by a short analysis of a perturbed scatterer on the detected backscatter in the section that follows to help elucidate how these effects may be detected using different trace analysis methods.

4.8.1 Effects of Perturbed Scatterer on Detected Backscatter

Assume a force is applied to the fiber such that a scatterer at z_m experiences a phase change $\Delta\phi_m$. This change will affect the received intensity from scatterers between the regions $z_m \pm W_p/2$ as shown in Fig. 33.

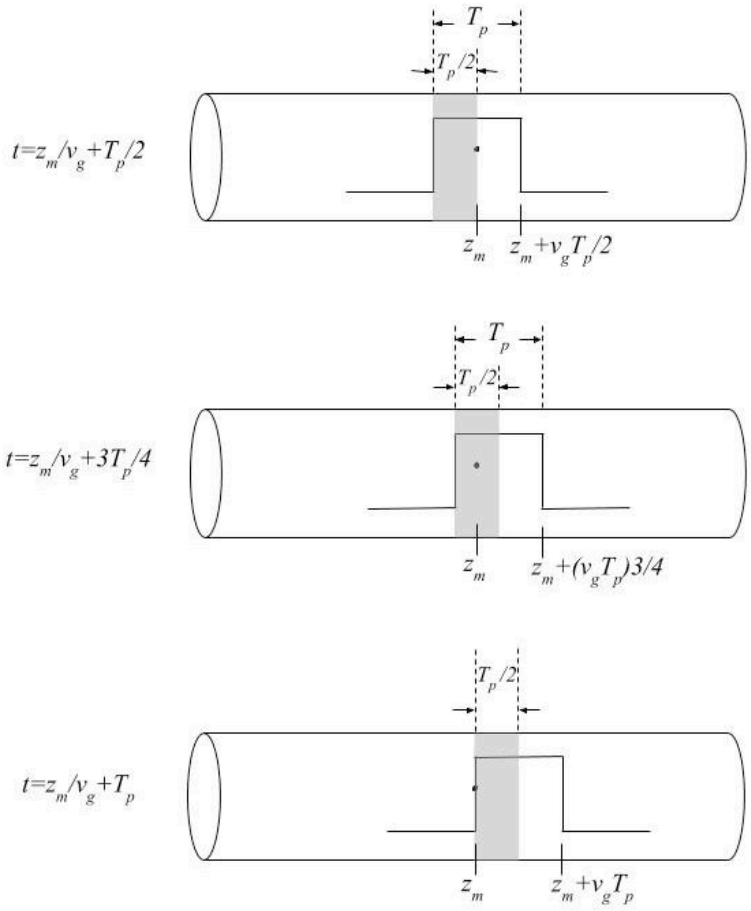


Fig. 36. Illustration of Spatial Area Contributing to Detected Rayleigh Backscatter for a given Scatterer and Probe Pulse Width

The time when the first backscatter including the perturbed scatterer at z_m is received by the photo-detector is at $t_{PD} = 2z_m/v_g$, shown in Fig. 36 (a). As the pulse continues to propagate, the perturbed scatterer will interfere with a forward propagating region of the fiber, half the width of the pulse, as shown by the grey shaded region in Fig. 36. The perturbed scatterer will no longer contribute to the received signal once it is no longer being excited by the pulse, as shown in Fig. 36 (c).

The difference between the acquired trace of the unperturbed fiber and that of the perturbed fiber is therefore isolated to the difference between the received intensity when t_{PD} satisfies Eq. 4.26.

$$2z_m/v_g \le t_{PD} \le 2\left(z_m/v_g + T_P\right) \tag{4.26}$$

Using Eq. 4.19, we can derive the following result,

$$\Delta I_{PD}(t) = \left|I_{PD}^{U}(t) - I_{PD}^{P}(t)\right| = \left|2a_m rect\left(\frac{t-\tau_m}{T_p}\right)\sum_{\substack{i=1 \\ i\neq m}}^{N} a_i rect\left(\frac{t-\tau_i}{T_p}\right)\left\{cos\left(\phi_i - \phi_m\right) - cos\left(\phi_i - \phi_m - \Delta\phi_m\right)\right\}\right| \tag{4.27}$$

Where $2z_m/v_g \le \tau_i$ where I^P_{PD} is the intensity measured from the perturbed fiber and I^U_{PD} is the intensity measured from the unperturbed fiber. Eq. 4.27 shows that the change in intensity due to a perturbed scatterer will follow a sinusoidal change as the propagating pulse illuminates the area of perturbation.

4.8.2 Backscatter Trace Intensity Analysis

A common method for analyzing the backscatter traces of a phase-OTDR is to plot the intensity of the received scatter over time. The intensity plot provides information on the attenuation of the fiber, which may be used to identify problems such as damage or breaks as is typical in telecom applications. It is common to use the intensity plot as a baseline measurement, which is then compared to future backscatter traces. By calculating the difference between two consecutive backscatter traces, external forces may be detected provided the SNR of the system compared to the disturbance is sufficiently large.

Fig. 37 shows the plotted traces of measured intensity from two different acquisition times, as well as the difference between the two traces. The simulation parameters were set as shown in Table 8.

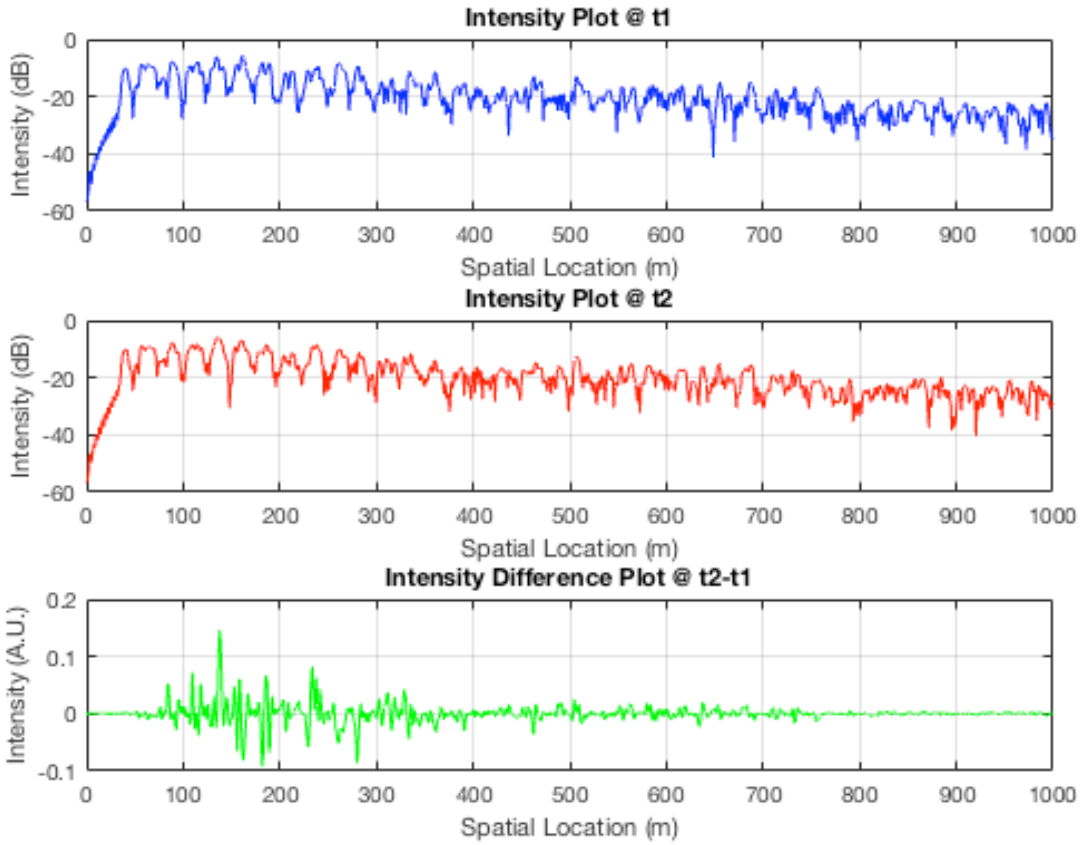


Fig. 37. Intensity Plots of Simulated Backscatter

Table 8. List of Parameters Used for Simulation of a Fast Time Acquisition

Parameter	**Setting**
Number of scatterers	100000
Length of Fiber	1000 meters
Central Wavelength	1550 nm
Pulse Width	0.1uS
Refractive index	1.47
Laser Linewidth	100 kHz

Number of acquisitions	16

The difference in measured intensity between the two times is due to the laser phase noise. Taking the average of many intensity traces can reduce the effect of this noise; however, this decreases the effective repetition rate of the system making it less responsive to high-frequency perturbations. Fig. 38 shows the results of averaging over 8 consecutive traces from the same data that was used to generate the traces shown in Fig. 37.

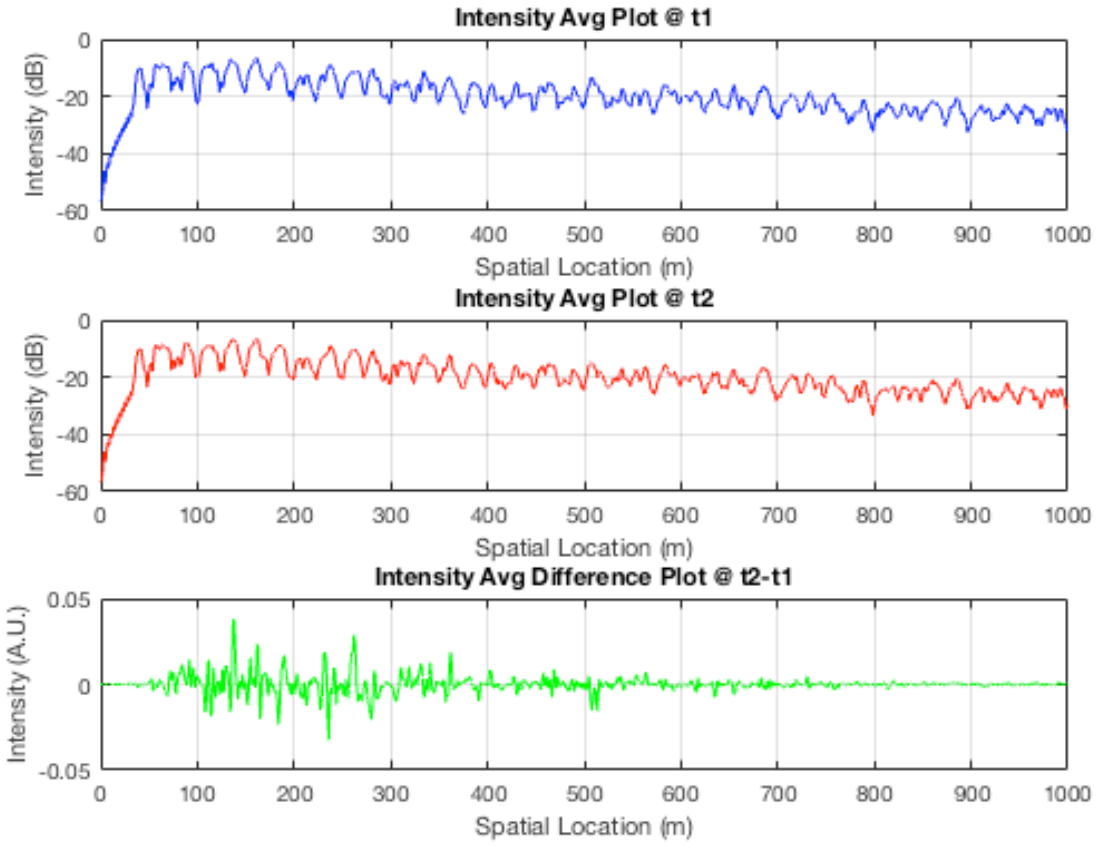


Fig. 38. Intensity Plots of Simulated Backscatter Averaged Over 8 Consecutive Acquisitions

The resulting difference in intensity traces shown in Fig. 38 is reduced compared to the difference shown in Fig. 37 as expected.

4.8.3 Backscatter Trace Phase Analysis

Similar to a backscatter intensity analysis one can plot the calculated phase angles over time to obtain a backscatter trace phase plot. The phase plot may be used to obtain baseline measurements to compare against future backscatter traces.

A phase difference plot shows the difference in phase at each spatial location between two slow-time acquisitions. A phase difference plot and the two original phase angle plots are shown in Fig. 39.

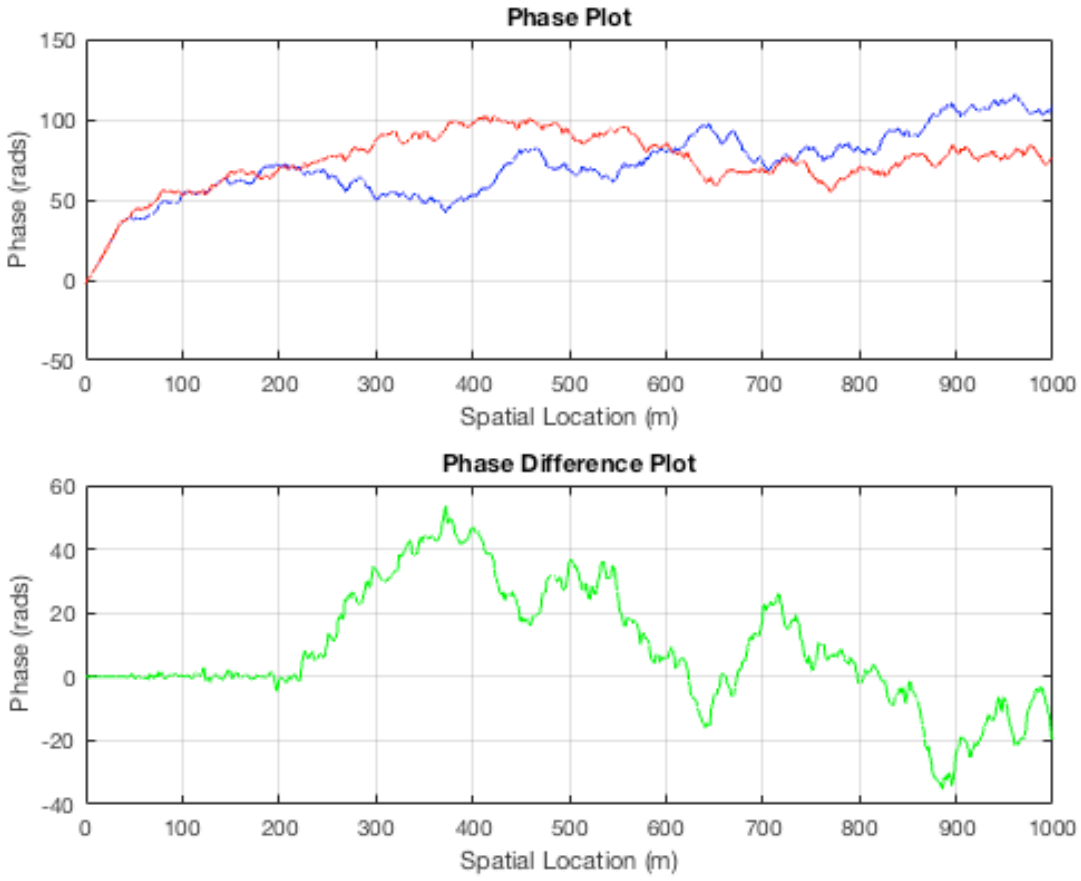


Fig. 39. Example of Phase Plots for Two Consecutive Acquisitions & Phase Difference Plot Between Two Consecutive Acquisitions

This is a useful method of identifying change; however, this method becomes problematic when dealing with phase plots where noise has caused the phase diverge over time. This divergence is a result of the phase unwrapping and is not easily solvable as discussed in 4.6.2.

4.8.4 Backscatter Trace Phase Differential Analysis

The largest source of noise in many systems is the laser phase noise. This is particularly true of systems that have long lengths of fiber since the phase noise of a system grows with time as explained in section 3.1.2. One useful method used to reduce the effect of phase noise is to use the phase differential method [103]. Since the laser phase noise is Gaussian white noise, it can be reduced through averaging. One commonly used method is to perform averaging over several backscatter traces, where each spatial location is averaged over some number of acquired traces N. Although this results in a reduction of noise, it also reduces the effective slow-time acquisition rate and thus the maximum detectable frequency of the system. The phase differential method achieves a reduction in noise through averaging while retaining the maximum detectable frequency at the cost of a reduced spatial resolution.

Referring to Fig. 40, the section of fiber between the locations A and B is further divided into sub-sections of length Δz.

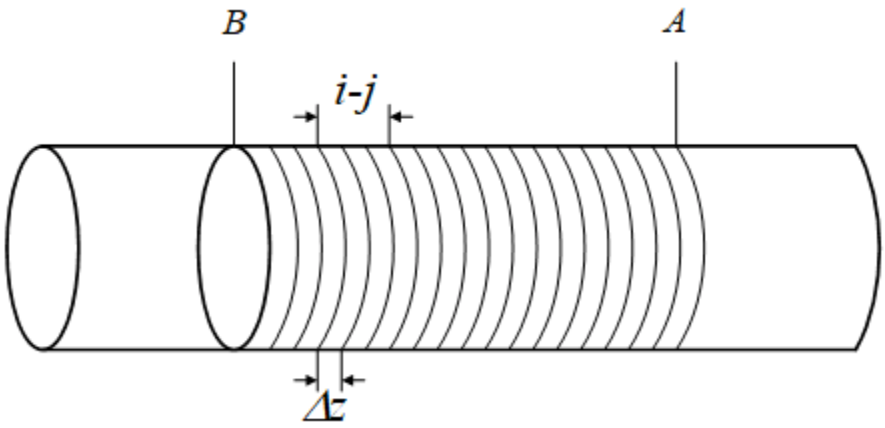


Fig. 40. Partitioning of Optical Fiber Sections for Differential Averaging Method

The differential phase between A and B, $\theta_{A,B}(t)$, is calculated as,

$$\vartheta_{A,B}(t) = \sum_{i=1+(A+B)/2}^{A} \sum_{j=B}^{i-1-(A-B)/2} \frac{\left(\vartheta_i(t) - \vartheta_j(t)\right)(A-B)}{(i-j)N} \qquad (4.28)$$

As noted by Tu, the effectiveness of this method depends on the choice of sub-separation distance i and j, with smaller separations yielding better accuracy but also more overlap in the computed sub-sections which degrades the effectiveness of the noise suppression since an overlap creates a partial dependence between computed sub-sections. This effect can be seen in Fig. 41, which shows the resulting effective variance of error for different A, B separations and different i, j sub-separations.

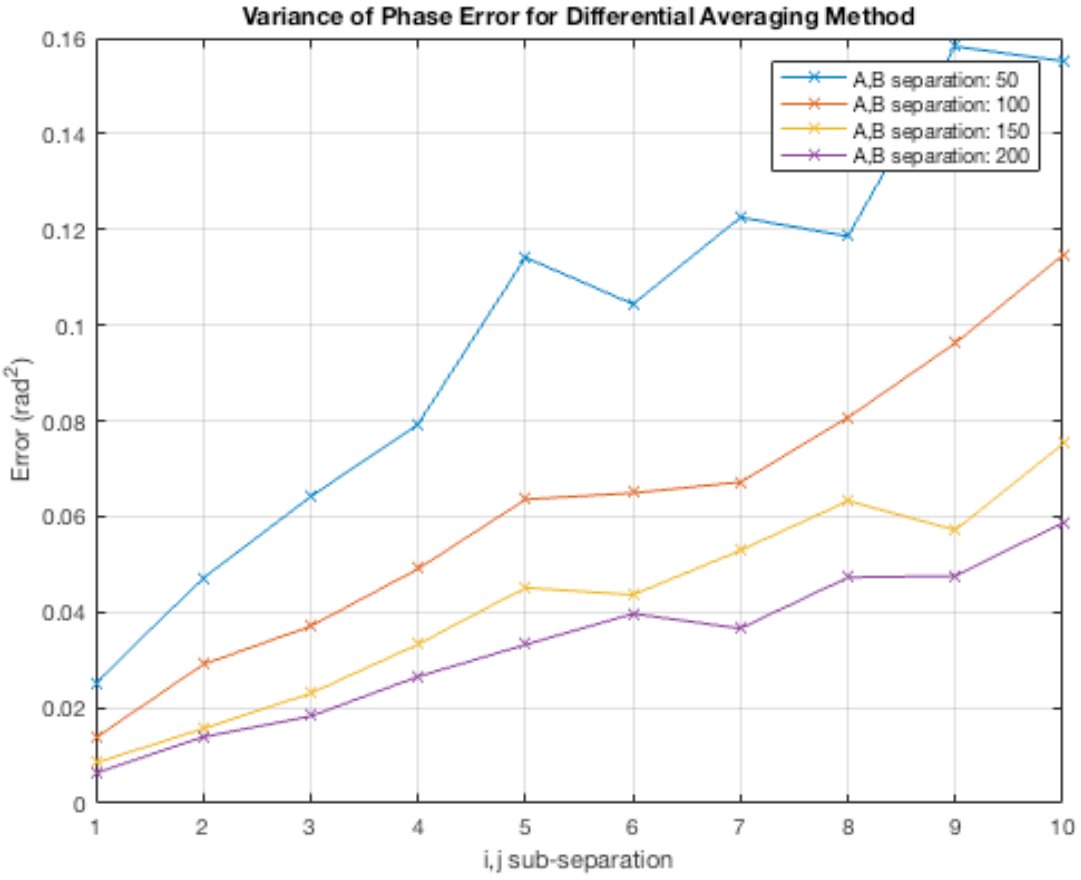


Fig. 41. Variance of Phase Error Using Differential Averaging Method

4.8.5 Frequency Domain Analysis of Backscatter Traces

The preceding sections focused on analysis methods that were performed in the time domain. In this section, frequency domain analysis is considered. In particular, the frequency domain analysis of a fiber being perturbed is considered.

As described in section 4.7, an external force occurring at a particular spatial location is typically too slow to observe using the backscattered signal from a single probe pulse. The frequency analysis is therefore performed over a slower time scale by calculating the periodogram at each spatial location using the data acquired over several probe pulses (slow-time). This results in an array of power spectral densities (PSD) for each spatial location along the fiber.

To illustrate, three simulations are run with different settings, and the results are reviewed here. In each of the simulations, different types of perturbations are applied to the fiber, specifically; a sinusoidal perturbation, a stationary perturbation and a non-stationary perturbation. Table 9 shows the configuration settings used for each simulation. These values were held constant throughout the tests.

Table 9. List of Parameters Used for Simulation of a Slow-Time Acquisition

Parameter	**Value**
Number of scatterers	50000
Length of Fiber	4000 meters
Central Wavelength	1550 nm
Pulse Width	2 nS
Refractive index	1.47
Laser Linewidth	5000 Hz
Laser Frequency Drift	1000 Hz/s
Frequency Shift	48 MHz
Sampling Rate	500 MSPS

4.8.5.1 Frequency Domain Analysis Results of Simulation with Sinusoidal Perturbations

The first simulation was done with pure sinusoidal perturbations. The details of the applied perturbations are listed in Table 10.

Table 10. Sinusoidal Event Parameters Used in Simulation

Parameter	**Value**

Frequencies	2000 Hz
	4200 Hz
	6400 Hz
Locations	2050 m
	1100 m
	817 m
Widths of perturbation	10 m
	10 m
	10 m
Strain	1e-7
	1e-7
	1e-7

As shown in Fig. 42, the difference in consecutively acquired intensity backscatter traces show peaks at the perturbation locations.

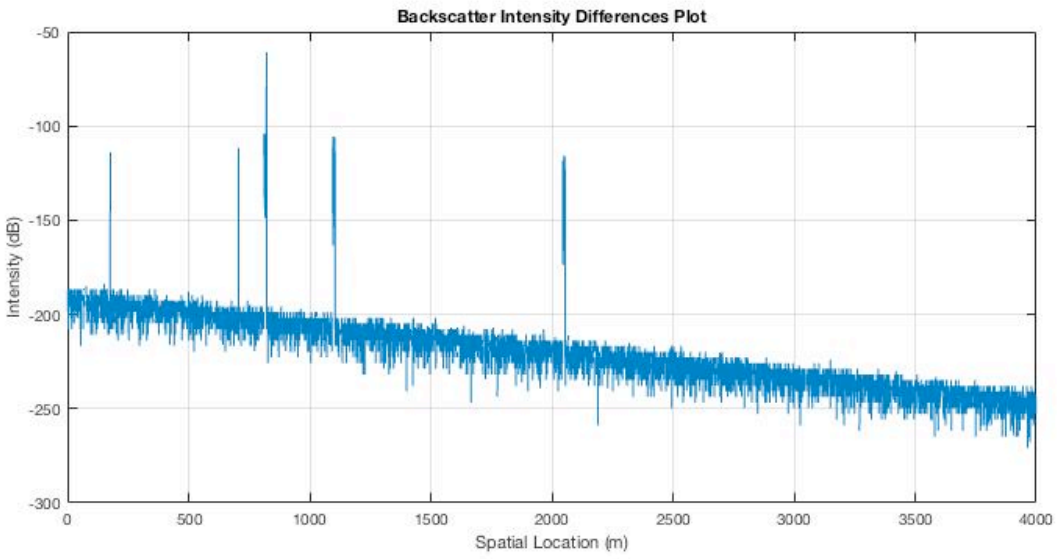


Fig. 42. Difference of Consecutively Acquired Intensity Backscatter Traces Under Sinusoidal Perturbation

A frequency analysis of the phase angle backscatter traces is shown in Fig. 43.

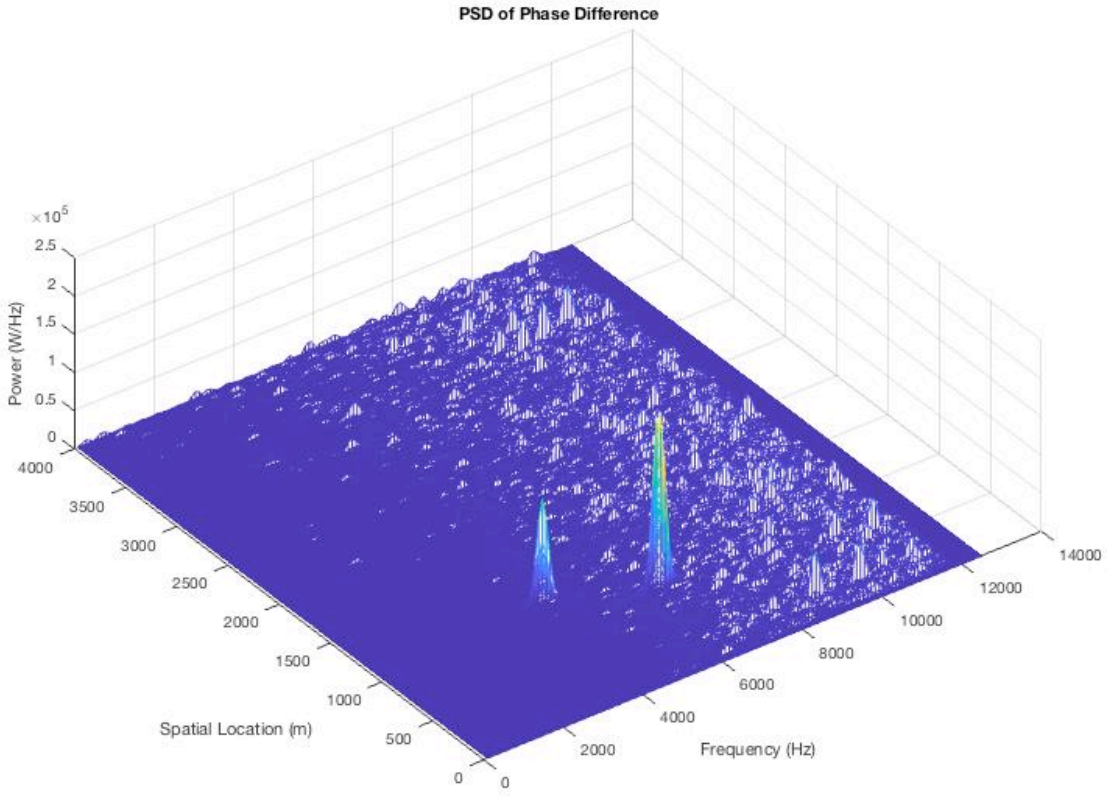


Fig. 43. PSD of Phase Angle Backscatter Trace Differences with Sinusoidal Perturbations

Two of the three perturbations are clearly visible, but the third perturbation at 2000 m is not easily observable. The third perturbation has a low SNR due to laser phase noise (which increases with distance from the source) and thus is not as easily resolved as the two closer perturbations. Using the phase differential averaging method, which reduces noise contributed to laser phase noise, the third perturbation becomes clearly observable as can be seen in Fig. 44.

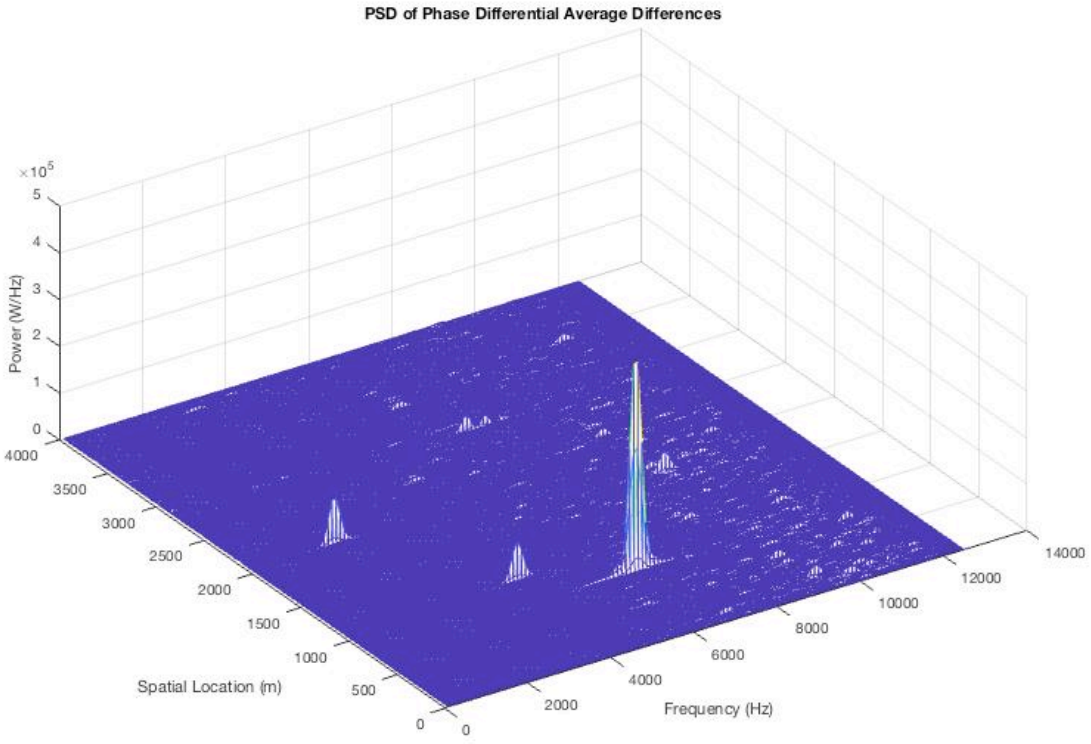


Fig. 44. PSD of Phase Differential Averaging Trace Differences with Sinusoidal Perturbations

4.8.5.2 Frequency Domain Analysis Results of Simulation with Stationary Perturbations

The second simulation was run with stationary perturbations. The details of the applied perturbations are listed in Table 11.

Table 11. Stationary Event Parameters Used in Simulation

Parameter	**Value**
Frequencies	4000 Hz
Locations	480 m
Widths of perturbation	10 m
Strain	1e-7

As shown in Fig. 45, the difference in consecutively acquired intensity backscatter traces clearly shows a peak at the perturbation location.

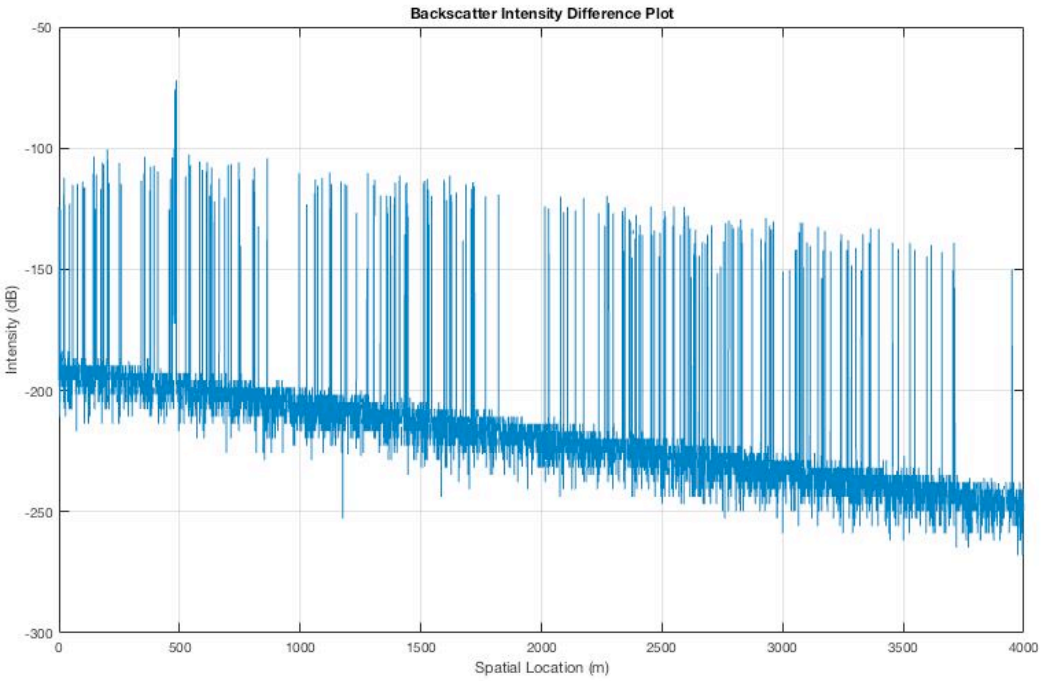


Fig. 45. Difference of Consecutively Acquired Intensity Backscatter Traces Under Stationary Perturbation

The PSD of the applied strain is shown in Fig. 46 which shows the spectral peak near 4000 Hz.

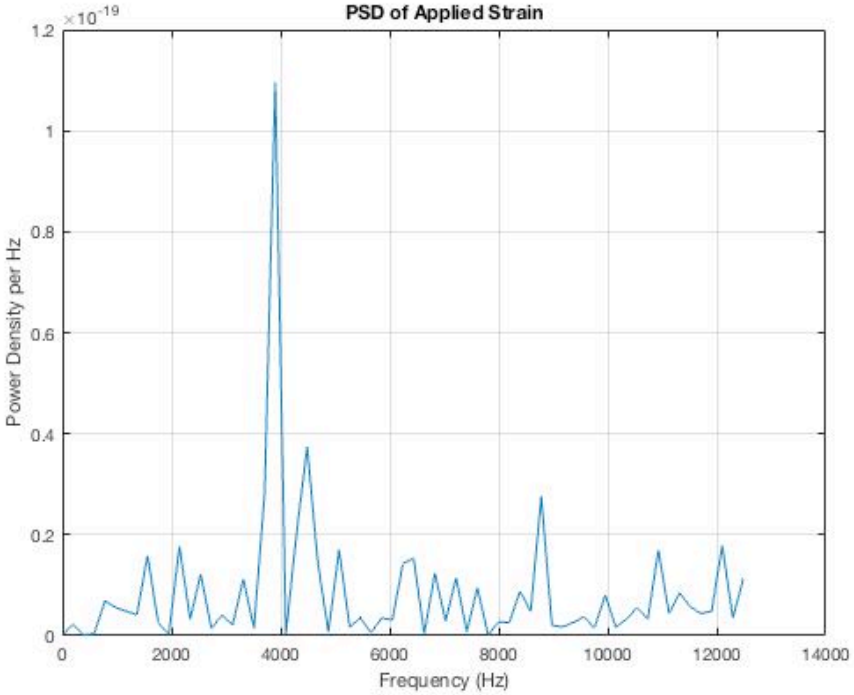


Fig. 46. Power Spectral Density of Stationary Strain

A frequency analysis of the phase angle backscatter trace differences is shown in Fig. 47.

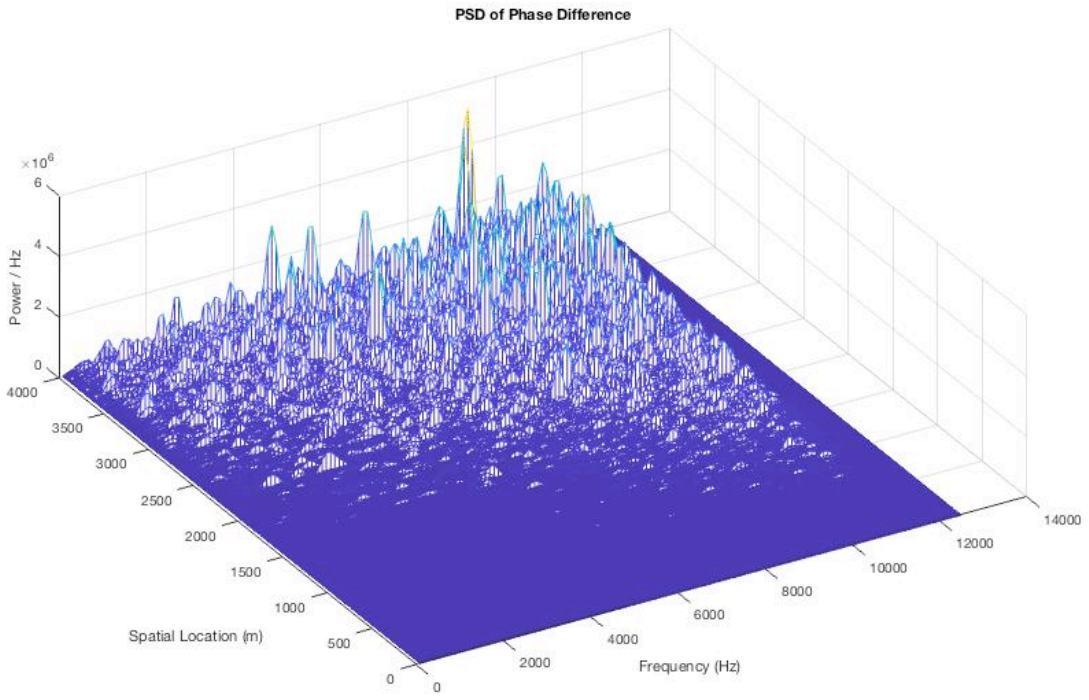


Fig. 47. PSD of Phase Angle Backscatter Trace Differences with Stationary Perturbation

From this plot a perturbation at 480 m is not immediately apparent when compared to the other spectral peaks. Using the phase differential averaging method, the stationary perturbation becomes clearly observable as shown in Fig. 48.

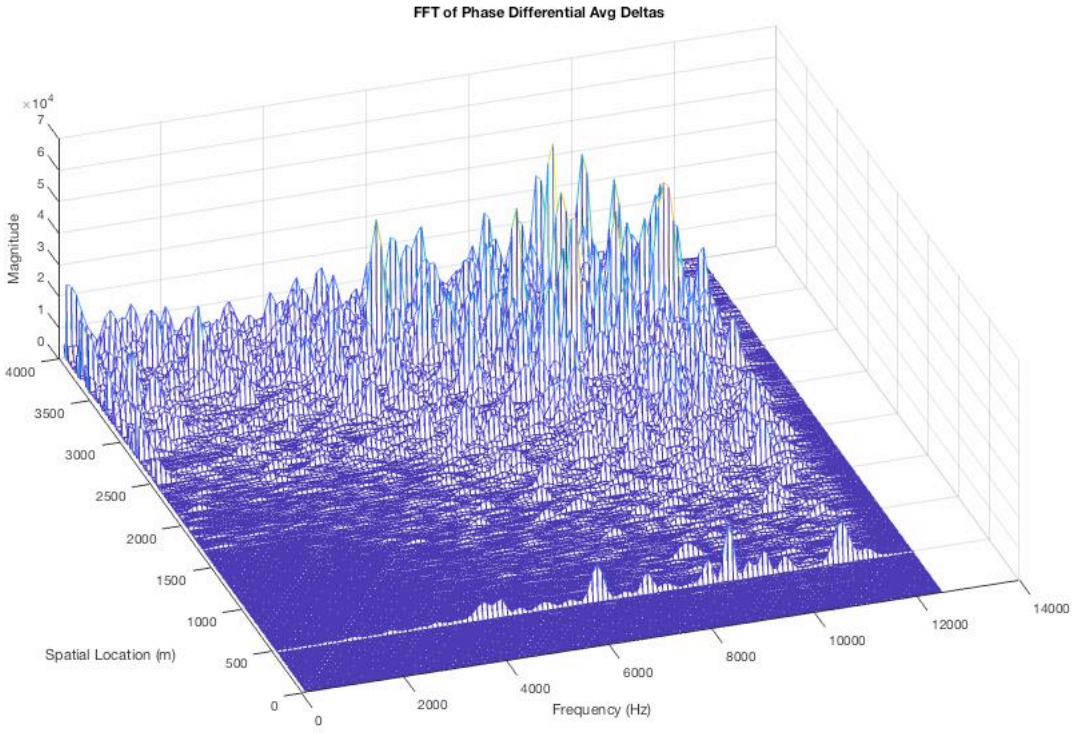


Fig. 48. PSD of Phase Differential Averaging Trace Differences with Stationary Perturbations

4.8.5.3 Frequency Domain Analysis Results of Simulation with Non-Stationary Perturbations

The final simulation was run with a non-stationary perturbation. The details of the applied perturbation are listed in Table 12.

Table 12. Non-Stationary Event Parameters Used in Simulation

Parameter	**Value**
Frequencies	4000 Hz
Location	1000 m
Widths of perturbation	10 m
Strain	1e-7

As shown in Fig. 49, the difference in consecutively acquired intensity backscatter traces clearly shows a peak at the perturbation location.

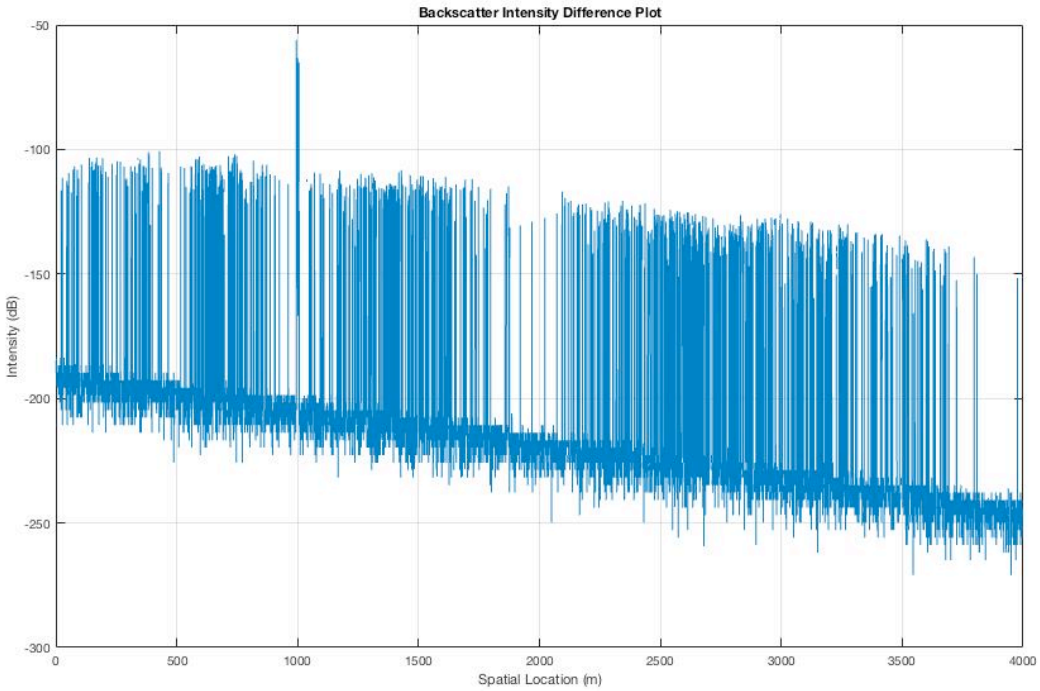


Fig. 49. Difference of Consecutively Acquired Intensity Backscatter Traces Under Non-Stationary Perturbation

The PSD of the applied strain is shown in Fig. 50 which shows two peaks of similar magnitude between 3000Hz-4000Hz.

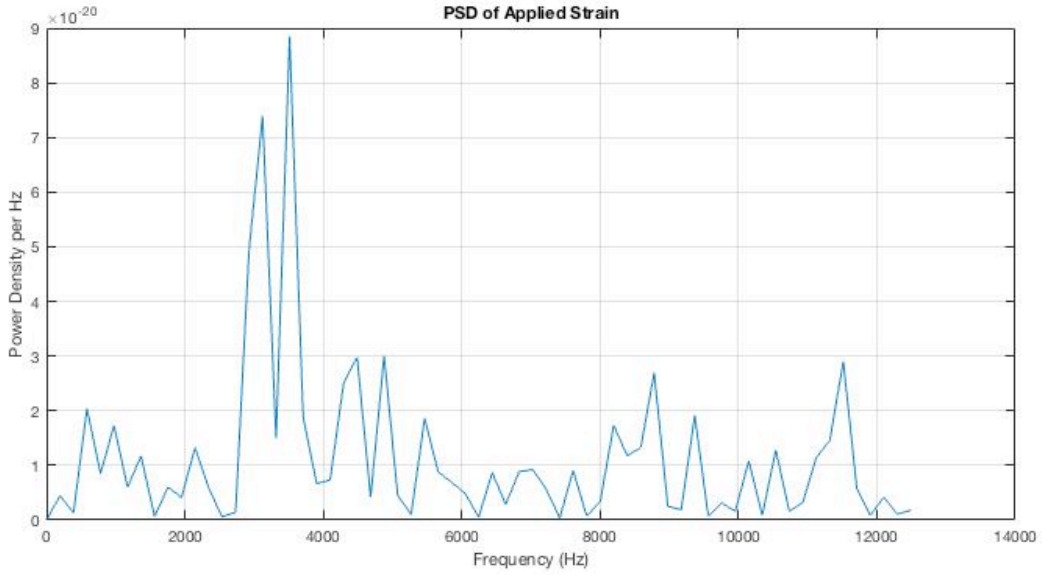


Fig. 50. Power Spectral Density of Non-Stationary Strain

A frequency analysis of the phase angle backscatter trace differences is shown in Fig. 51.

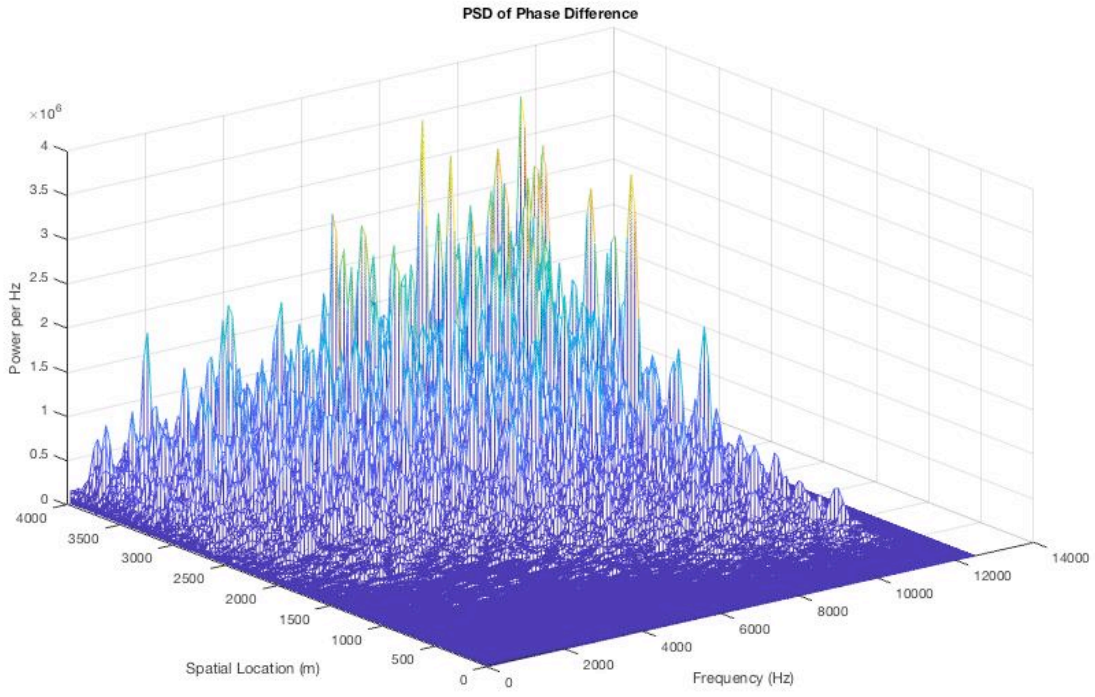


Fig. 51. PSD of Phase Angle Backscatter Trace Differences with Non-Stationary Perturbation

From this plot a perturbation at 1000 m is not immediately apparent when compared to the other spectral peaks. Using the phase differential averaging method, the non-stationary perturbation becomes observable as shown in Fig. 52.

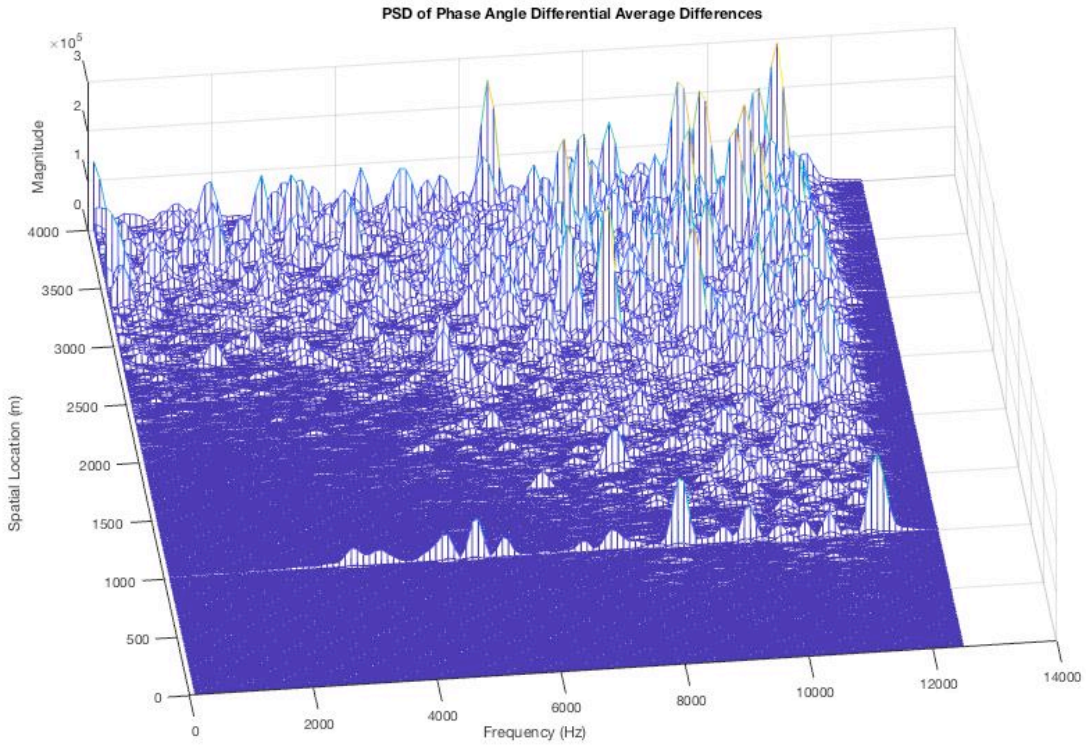


Fig. 52. PSD of Phase Differential Averaging Trace Differences with Non-Stationary Perturbations

Chapter 5

Conclusions and Future Work

In summary, this thesis provided a detailed description of the physics of Rayleigh backscattering in optical fibers as well a brief overview of optical reflectometry techniques, and a more detailed overview of phase-based detection systems.

A simulator for coherent phase-OTDR systems was also developed, with a particular emphasis on simulating external perturbations to the physical model. This contribution by the Author provides a means to conduct future work in this area of research. There are several different areas where future work could be conducted to expand on the work covered in this thesis.

5.1 Verification of Simulator by Experiment

Unfortunately, due to time and financial constraints, the simulations could not be verified by experiment. This is the most important work yet to be performed.

5.2 Improving Noise Model of Simulator

As described in this thesis, several sources of noise were ignored in the simulator for simplicity. There are some obvious noise sources that would useful to model such as:

- Laser relative intensity noise (RIN)
- Photo-Detector noise (e.g. shot noise)
- Electronics noise (e.g. $1/f$ noise)

5.3 Adding Polarization Effects to Simulation of Physical Model

As described in this thesis, polarization effects were ignored for simplicity. Polarization fading, as described in section 3.2.2, is a major cause of signal fading in systems that utilize coherent detection receiver topologies and would be a useful addition to the simulator.

This would require adding birefringence data to the model, which would be coupled to spatial locations. This information would allow for the calculation of state of polarization changes to be simulated as well as other potential polarization-related effects such as polarization mode dispersion (PMD).

5.4 Modeling of Additional Types of Forces Applied to Fiber

In this work, the types of forces applied to the fiber were limited to axial strains. The model developed could also readily be applied to axial compression with minimal work. However, if polarization information was also modeled the types of forces that could be modeled could include those of a higher dimensionality such as twisting and bending forces. These would impart different combinations of circular and linear birefringence to the fiber.

5.5 Improved Data Analysis & Visualization Methods

The simulator that has been developed was meant to provide basic analysis and plotting capabilities. There are several more analysis methods that would be interesting to include in this simulator tool. Some examples of analysis methods seen in the literature include:

- Time-Frequency analysis of backscatter traces, such as Wavelets [15], [104], [105], Hilbert-Huang Transform [106], or Fractional Fourier Transforms [107]
- Edge detection of backscatter traces [108]
- Unwrapping of phase using Kalman Filtering [109]

It would also be useful to investigate the implementation of any data signal processing in a real-time manner. The current simulator performs post-processing on all the acquired data. This is typically suitable for research purposes, but does not reflect the limitations often encountered with real-world systems, which have to be able to operate continuously and therefore require iterative processing methods.

5.6 Simulator Performance Enhancements

As more functionality is added to the simulator, several other performance related aspects of the simulator could be improved. Memory management could easily be improved as the memory requirements for the simulator increase. Currently the simulator creates data structures to hold the data for the fiber model, event generation information, as well results from any data analysis. Simulations require increasing amounts of memory resources as the number of scatterers, fiber length and number of acquired traces increase. This becomes limiting for most standard computer systems for more than a few hundred trace acquisitions even for short fiber lengths. This problem could be addressed by segmenting the simulation into sections allowing for intermediate results to be stored in non-volatile memory for later retrieval.

Other simple improvements such as limiting data types for variables and data could be performed as well. The current simulator uses the standard double types in Matlab. In a real system, sampled data is likely to be 12-bit data (or less) when sampled at high speed. Handling data with this resolution would be far less memory intensive than the current method. Ideally, one could set the desired resolution to simulate various systems.

APPENDIX A - Similarity of Rayleigh Scattering to Speckle

The scattering light within a plane of the fiber that is contained within the pulse can be expressed as,

$$U(\xi,\eta) = \sum_{i=1}^{N} a_i \exp(j\beta_i)\delta(\xi - \xi_i)\delta(\eta - \eta_i)$$

where a_i is the amplitude of the i^{th} contributing scatterer, β_i is the phase of the wave from the i^{th} scatterer and η and ξ are the coordinates in the plane of interest. Furthermore, we make the following assumptions:

a) Scatterers are randomly distributed over scattering area with uniform probability
b) The amplitudes of the scatterers are statistically independent
c) The phase, β_i, are statistically independent and uniformly distributed on the interval $-\pi$ to π

Taking the scattering light from the η, ξ plane to the far-field we apply the Fraunhofer diffraction formula [110],

$$U(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} U(\xi,\eta)\exp\left(-j\frac{2\pi}{\lambda R}(x\xi + y\eta)\right)d\xi\, d\eta$$

which yields,

$$U(x,y) = \sum_{i=1}^{N} a_i \exp(j\beta_i)\exp\left(-j\frac{2\pi}{\lambda R}(x\xi_i + y\eta_i)\right)$$

This expression shows that the resulting amplitude is due to the contribution of many scatterers with random amplitude and phase. As noted by many authors [45], [46] this is similar to the phenomena of speckle. Given that the number of scatterers is a very large number, $U(x, y)$ tends to a complex Gaussian distribution due to the central limit theorem and can be expressed as,

$$p_U\left(U_{Re},U_{Im}\right)=\frac{1}{2\pi\sigma^2}exp\left[-\frac{1}{2}\left(\frac{U_{Re}^2+U_{Im}^2}{\sigma^2}\right)\right]$$

where σ^2 is the variance of the field and U_{Re} and U_{Im} are the real and imaginary components of the field, respectively. This particular distribution is known as circularly symmetric complex Gaussian.

APPENDIX B – Source Code

FILE: @FiberSim/FiberSim.m

```
% This file contains the class definition for the fiber simulator

classdef (Sealed) FiberSim < matlab.mixin.SetGet
    properties
        N               % number of scatterers
        A               % maximum amplitude of scatterers (a.u)
        Bs              % Backscatter fraction
        Tp              % pulse time duration (s)
        lambda          % wavelength (nm) [for LO]
        fd              % frequency drift rate of laser (in Hz/s)
        fs              % frequency shift from AOM (Hz)
        flw             % linewidth of laser (in Hz)
        ng  = 1.5;      % group refractive index
        min_dz = 1000;  % minimum "macro" scatterer spacing (factor of
lambda)
        c = physconst('LightSpeed');        % speed of light (m/s)
        L               % length of fiber being simulated (meters)
        alpha           % general attenuation of fiber (dB/km)
        Ts              % sampling rate for system (SPS)
        seeds           % random number generator seeds (if specified)
        LogHandle       % handle to log display
    end

    methods
        % this is the constructor
        function obj = FiberSim(params, LogHandle)
            obj@matlab.mixin.SetGet;
            if nargin > 0
                obj.N = params(1);
                obj.A = params(2);
                obj.Tp = params(3);
                obj.fs = params(4);
                obj.lambda = params(5);
                obj.fd = params(6);
                obj.flw = params(7);
                obj.L = params(8);
                obj.alpha = params(9);
                obj.Ts = params(10);
                obj.LogHandle = LogHandle;
            end
        end

        % Function declarations

        % functions for calculating internal parameters
        dz = get_dz(obj);
        Wp = get_Wp(obj);
        f = get_freq(obj);
        T = get_acq_time(obj);
        ft_idx = get_event_fast_time_range(obj, eventObj);
```

```
        [z_i, a_i, n_i, new_a_seed, new_z_seed] = generate_scatterers(obj,
a_seed, z_seed);

        % logging functions
        sim_print(obj, s, varargin);

        % functions for generating noise
        [np, new_np_seed] = generate_phase_noise(obj, N, np_seed);
        [nd, new_nd_seed, nd_test] = generate_laser_drift(obj, N, M,
np_seed);

        % functions for generating and applying perturbation events
        vevents = generate_vibration_events(obj, vevents_params, z_i);
        nevents = generate_nonstationary_events(obj, nevents_params, z_i,
num_traces);
        tevent = generate_test_event(obj, tevent_params, z_i);
        [pz, png, dep] = pert_section(obj, varargin);

        % analysis functions
        [phi, fft_phi, I] = calculate_phase_angle(obj, s_i, n_i, np, nd);
        fix_phi_terms = fix_phase_angles(obj, phi_terms, d_phi_terms,
d_phi_max);
        [z, d_phi_ft] = calculate_phase_differentials(obj, phi,
dz_step_factor, sub_step_factor);
        [z, d_phi_AB] = calculate_phase_differential_avg(obj, phi, dAB, djk);
        y = calculate_trace_averages(obj, x, N);
        [f, y] = calculate_fft(obj, z, x, D);
        phi_terms = unwrap(obj, old_phi_terms, new_phi_terms);
        plot_scatterers(obj, z_i);
        plot_pulse(obj);

    end
end

FILE: @FiberSim/unwrap.m
% -------------------------------------------------------------------------
% Function: unwrap
% Description: This function performs a "smart" unwrap of the phase based
%           on the previous phase trace
% Parameters:
%       obj - The FiberSim object
%       old_phi_terms - array of the previous phase angles
%       new_phi_terms - array of the new phase angles to unwrap
%
% Returns:
%       phi_terms - array of unwrapped phase angles
%
% Author:   Michael Brown
% Date: Aug 8, 2017
% -------------------------------------------------------------------------
function phi_terms = unwrap(obj, old_phi_terms, new_phi_terms)

    N = length(old_phi_terms)
    phi_terms(1,1) = new_phi_terms(1,1);
    offset = 0;
    new_offset = 0;
```

```
    for i=1:N-1
        dphi = abs(new_phi_terms(1,i+1) - new_phi_terms(1,i));
        val = abs(old_phi_terms(1,i+1)-(new_phi_terms(1,i+1)+offset));
        phi_terms(1,i+1) = new_phi_terms(1,i+1) + offset;
        dmin = val;

        val = abs(old_phi_terms(1,i+1)-(new_phi_terms(1,i+1)+offset+2*pi));
        if (val < dmin)
            phi_terms(1,i+1) = new_phi_terms(1,i+1)+offset+2*pi;
            new_offset = offset + 2*pi;
        end

        val = abs(old_phi_terms(1,i+1)-(new_phi_terms(1,i+1)+offset-2*pi));
        if (val < dmin)
            phi_terms(1,i+1) = new_phi_terms(1,i+1)+offset-2*pi;
            dmin = val;
            new_offset = offset - 2*pi;
        end

        offset = new_offset;
    end
end
```

FILE: @FiberSim/sim_print.m

```
% ------------------------------------------------------------------------
% Function: sim_print
% Description: This function prints the string parameter to the console
% output as well as the text area for the GUI log.
% Parameters:
%       obj - The FiberSim object
%       s - string to print
%       varargin - any string arguments
%
% Author:   Michael Brown
% Date:     July 9, 2017
% ------------------------------------------------------------------------
function sim_print(obj, s, varargin)
    % create string
    str = sprintf(s, cell2mat(varargin));

    % print to console
    fprintf('%s\n', str);

    % print to GUI log
    if (obj.LogHandle ~= -1)
        obj.LogHandle.Value(length(obj.LogHandle.Value)+1) = {str};
    end
end
```

FILE: @FiberSim/pert_section.m

```
% ------------------------------------------------------------------------
% Function: pert_section
% Description: This function modifies the scatterers between two locations
%               in the fiber. The origin of the force is assumed to be in
%               the center of two locations and the phase change is assumed
%               to be applied outward from the center with uniform change
%               on each scatterer in the region.
```

```
% Parameters:
%       obj - The FiberSim object
%       varargin:
%           Required:
%           z - Array containing the region to be perturbed
%           A - starting location of force
%           B - ending location of force
%           dphi - The change in phase to be applied
%
%           Optional:
%           n - The acquisition number since the start of the perturbation
%
% Returns:
%       pz - Array containing perturbed region
%       dep - the applied strain
%
% Author:   Michael Brown
% Date: May 25, 2017
% ------------------------------------------------------------------------
function [pz, dep] = pert_section(obj, varargin)
    nargs = length(varargin);   % get the total number of arguments
    eventObj = varargin{3};

    if (isa(eventObj, 'NonStationaryEvent'))
        isVibration = false;
        isTest = false;
        ep = eventObj.ep;
        zloc = eventObj.zloc;
        z_idx = eventObj.z_idx;
        z_l = eventObj.l;
        fprintf('NonStationary Event!!!!');
    elseif (isa(eventObj, 'TestEvent'))   % we assume test if not others
        isTest = true;
        z_idx = eventObj.z_idx;
        zloc = eventObj.zloc;
        z_l = eventObj.l;
        ep = eventObj.ep;
        fprintf('pert test!!!');
    elseif (isa(eventObj, 'VibrationEvent'))
        isVibration = true;
        isTest = false;
        ep = eventObj.ep;
        zloc = eventObj.zloc;
        z_idx = eventObj.z_idx;
        z_l = eventObj.l;
        v_wn = eventObj.v_wn;
        fprintf('Stationary Event!!!!');
    else
        isTest = true;
        ep = eventObj.ep;
        zloc = eventObj.zloc;
        z_idx = eventObj.z_idx;
        z_l = eventObj.l;
        fprintf('pert test!!!');
    end

    z = varargin{1};
```

```
    if (isTest == false)
        if (isVibration == true)
            wn = 0.2*randn(1,1);

            % determine the change of strain
            dep = ep*(0.5+0.5*eventObj.AR_lat_filter(0, wn));
        else
            % deal with nonstationary events
            wn = 0.2*randn(1,1);

            % determine the change of strain
            dep = ep*(0.5+0.5*eventObj.AR_lat_filter(eventObj.d(1,n),wn));
        end
    else
        dep = ep*(0.5+0.5*(-
cos(2*pi*eventObj.freq*varargin{end}*obj.get_acq_time())));
        obj.sim_print('Test frequency: %d\n', eventObj.freq);
        obj.sim_print('Time: %d\n', varargin{end}*obj.get_acq_time());
    end

    % get number of bins in section of fiber
    num_bins = round(z_l/(obj.min_dz*obj.lambda));
    obj.sim_print('Minimum dz: %d\n', obj.min_dz*obj.lambda);
    obj.sim_print('Number of bins in perturbed section: %d\n', num_bins);

    % calculate displacement in each bin
    dz = abs(0.78*dep*z_l/2);
    obj.sim_print('Displacement of perturbed bins: %d\n', dz);
    obj.sim_print('Z location: %d\n', zloc);

    % distances between existing scatterers must be expanded/compressed by
    % dz.
    j=1;
    pz = zeros(1,z_idx(2)-z_idx(1));

    for i=z_idx(1):z_idx(2)
        if (z(i) < zloc)
            z(i) = z(i) - dz;
        elseif (z(i) > zloc)
            z(i) = z(i) + dz;
        end

        pz(1,j) = z(i);
        j = j + 1;
    end
end

FILE: @FiberSim/get_Wp.m
% ------------------------------------------------------------------
% Function: get_Wp
% Description: This function calculates the width of the light pulse in
% meters
% Parameters:
%       obj - The FiberSim object
%
% Returns:
%       Wp - pulse width in meters
```

```
%
% Author:   Michael Brown
% Date: Aug 10, 2016
% ------------------------------------------------------------------------

function Wp = get_Wp(obj)
    % calculate the width of the pulse
    Wp = obj.Tp*obj.c/obj.ng;
end

FILE: @FiberSim/get_Wp.m
% ------------------------------------------------------------------------
% Function: get_freq
% Description: This function calculates laser frequency
% Parameters:
%       obj - The FiberSim object
%
% Returns:
%       f - laser frequency (in Hz)
%
% Author:   Michael Brown
% Date: May 23, 2017
% ------------------------------------------------------------------------

function f = get_freq(obj)
    f = (obj.c/obj.lambda);
end

FILE: @FiberSim/get_event_fast_time_range.m
% ------------------------------------------------------------------------
% Function: get_event_fast_time_range
% Description: This function returns the start and end locations of an
% event based on fast time sample number.
%
% Parameters:
%       obj      - The FiberSim object
%       eventObj - Handle to the event object
%
% Returns:
%       ft_idx - array with start index and end index values of event
location.
%
% Author:   Michael Brown
% Date: Oct 14, 2017
% ------------------------------------------------------------------------

function ft_idx = get_event_fast_time_range(obj, eventObj)
    % calculate the distance traveled by the pulse during between samples
    dz = obj.get_dz();

    % create an array of spatial locations of where the detected scattering
originated from
    % in relation to time of detection
    z = [0:dz/2:obj.L];     % light must travel to z and back to be detected

    % find the indices that where the event is located
    A = eventObj.zloc - eventObj.l/2;
```

```
    B = eventObj.zloc + eventObj.l/2;
    ft_range = (z(:)>=A) & (z(:)<=B);
    ft_idx_all = find(ft_range);
    ft_idx = [ft_idx_all(1); ft_idx_all(end)];
end

FILE: @FiberSim/get_dz.m
% ------------------------------------------------------------------------
% Function: get_dz
% Description: This function calculates the spatial resolution of the
% system.
% Parameters:
%       obj - The FiberSim object
%
% Returns:
%       dz - spatial resolution in meters
%
% Author:   Michael Brown
% Date: Aug 10, 2016
% ------------------------------------------------------------------------

function dz = get_dz(obj)
    % calculate the distance traveled by the pulse during between samples
    dz = (obj.c/obj.ng)*obj.Ts;
end

FILE: @FiberSim/get_acq_time.m
% ------------------------------------------------------------------------
% Function: get_acq_time
% Description: This function calculates the total time required to acquire
% samples from the entire length of fiber
%
% Parameters:
%       obj - The FiberSim object
%
% Returns:
%       T - Total time in seconds
%
% Author:   Michael Brown
% Date: Aug 10, 2016
% ------------------------------------------------------------------------

function T = get_acq_time(obj)
    % calculate the total time required to capture the entire trace
    T = 2*obj.L/(obj.c/obj.ng);
end

FILE: @FiberSim/generate_vibration_events.m
% ------------------------------------------------------------------------
% Function: generate_vibration_events
% Description: Generates vibration events based on user input
% Parameters:
%       obj - The FiberSim object
%       vevents_params - the vibration events parameters array
%       z_i - array of scatterer locations
%
% Returns:
```

```
%       vevents - array of vevents objects
%
% Author:   Michael Brown
% Date: Nov 12, 2017
% -------------------------------------------------------------------------

function vevents = generate_vibration_events(obj, vevents_params, z_i)
    N_vibration_events = vevents_params(2);
    obj.sim_print('Generating vibration events ......');

    vevents_params(end+1) = 0.01; % set v_wn (not used)
    vevents_params(end+1) = obj.get_acq_time();

    for i=1:N_vibration_events
        vevents(i) = VibrationEvent(z_i, obj.L, vevents_params(2:end));
    end

    % save the events to the temporary file
    save('vevents_tmp', 'vevents');

end
```

FILE: @FiberSim/generate_scatterers.m

```
% -------------------------------------------------------------------------
% Function: generate_scatterers
% Description: This program generates scatterers and distributes them
%                                 in space along a length of fiber.
% Author:   Michael Brown
% Date: June 30, 2016
%
% Parameters:
%       obj - The FiberSim object
%       a_seed - random number seed for amplitudes and refractive indices
%       z_seed - random number seed for locations
%
% Returns:
%       z_i - array of scatterer locations
%       a_i - array of scatterer amplitudes
%       n_i - array of scatterer refractive index
%       new_a_seed - random seeds used for generating amplitude information
%       new_z_seed - random seeds used for generating location information
% -------------------------------------------------------------------------

function [z_i, a_i, n_i, new_a_seed, new_z_seed] = generate_scatterers(obj,
a_seed, z_seed)

    obj.sim_print('Generating %i scatterers.....\n', obj.N);

    if (isstruct(a_seed))
        obj.sim_print('Re-creating random amplitude from seed file......\n');
        new_a_seed = rng(a_seed);
    else
        obj.sim_print('Creating new random amplitude seed......\n');
        new_a_seed = rng;
    end
```

```
    % generate the amplitudes
    a_i = rand(1,obj.N);    % generate amplitudes for the scatterers
    a_i = obj.A.*a_i;           % amplitudes are between 0 and A in amplitude

    % generate refractive indices
    n_i = obj.ng + obj.ng*0.00001*randn(1,obj.N); % two percent std dev
    %n_i = obj.ng*ones(1,obj.N);

    if (isstruct(z_seed))
        obj.sim_print('Re-creating random location from seed file......\n');
        new_z_seed = rng(z_seed);
    else
        obj.sim_print('Creating new random location seed.....\n');
        new_z_seed = rng;
    end

    % generate the locations
    min_dz = obj.min_dz*obj.lambda;    % set minimum spacing between "macro"
scatterers
    E = obj.L - (min_dz*obj.N);  % calculate the empty space in the FUT
    R = E*rand(1,obj.N);         % generate array of empty space with N
points
    Rn = E*R/sum(R);             % "normalize" so that expected value of sum
of elements in R is 2E/N
    z = min_dz*ones(1,obj.N)+Rn; % create array of scatterers with min
distance randomly spaced by Rn
    z_i = cumsum(z)-1;           % create array of scatterers in ascending
manner

    figure;
    plot(z_i);
    [z_i, idx] = sort(z_i); % sort the scatterers in ascending order
end
```

FILE: @FiberSim/generate_phase_noise.m

```
% ------------------------------------------------------------------------
% Function: generate_phase_noise
% Description: This function generates a matrix of values corresponding to
% the laser phase noise in the system at various sample times. The matrix
% contains both fast time [rows] (corresponding to time of detection) and
% slow time [columns] (corresponding to different acquired traces.
%
% Parameters:
%       obj - The FiberSim object
%       N   - number of trace acquisitions
%       np_seed - random seed for generating phase noise
%
% Returns:
%       np - matrix of phase noises
%       new_np_seed - seed used to generate the phase noise
%
% Author:   Michael Brown
% Date: June 2, 2016
% ------------------------------------------------------------------------

function [np, new_np_seed] = generate_phase_noise(obj, N, np_seed)
```

```
    % get the time step
    dt = obj.Ts;    % based on sampling frequency

    % calculate the total time required to capture the entire trace
    T = get_acq_time(obj);

    % create an array of sample times based on the sampling rate of the
system
    t = [0:dt:T];

    % create matrix
    np = zeros(length(t), N);
    new_np_seed = rng;
    [A,B] = size(np);

    % The phase noise from a laser is expressed as a Wiener process and
    % thus has a Gaussian variance such that v^2 = 2*pi*f*Ts
    if (obj.flw ~= 0)   % if the linewidth is zero then leave the array set
to zeros
        % create scalar vector for distance term
        obj.sim_print('Maximum phase noise variance: %d\n', 2*pi*obj.flw*T);
        a = [0:2*pi*obj.flw*dt:2*pi*obj.flw*T];
        a = sqrt(a);    % get std dev
        obj.sim_print('Maximum phase noise std dev: %d\n',
sqrt(2*pi*obj.flw*T));
        a = a';

        % determine random number seeding
        if (isstruct(np_seed))
            new_np_seed = rng(np_seed);
        end

        % create for each trace
        np = randn(A,B);

        for i=1:N
            np(:,i) = a.*np(:,i);
        end
    end
end
```

```
FILE: @FiberSim/generate_nonstationary_events.m
% ------------------------------------------------------------------------
% Function: generate_nonstationary_events
% Description: Generates nonstationary events based on user input
% Parameters:
%       obj - The FiberSim object
%       nevents_params - the nonstationary events parameters array
%       z_i - the array of scatterer locations
%       num_traces - the number of slow time acquisitions
%
% Returns:
%       nevents - array of vevents objects
%
% Author:   Michael Brown
% Date: Nov 12, 2017
% ------------------------------------------------------------------------
```

```
function nevents = generate_nonstationary_events(obj, nevents_params, z_i,
num_traces)
    N_nonstationary_events = nevents_params(2);
    obj.sim_print('Generating nonstationary events ......');

    nevents_params(end+1) = 0.01; % set v_wn (not used)
    nevents_params(end+1) = obj.get_acq_time();

    for i=1:N_nonstationary_events
        nevents(i) = NonStationaryEvent(z_i, obj.L, num_traces,
nevents_params(2:end));
    end
    nevents(1)
    % save the events to the temporary file
    save('nevents_tmp', 'nevents');
end
```

```
FILE: @FiberSim/generate_laser_drift.m
 % ------------------------------------------------------------------------
% Function: generate_laser_drift
% Description: This function generates a matrix of values corresponding to
% the laser phase noise in the system at various sample times. The matrix
% contains both fast time [rows] (corresponding to time of detection) and
% slow time [columns] (corresponding to different acquired traces.
%
% When generating laser drift values, we use the following approach:
% - random values are generated over a short time period vs
% - random values are generated over a long time period vl
% - we use cubic spline interpolation to "smooth" the long period noise so
% that we obtain the same number of points as the short time period
% - this creates a type of "average" drift with a "noise" component at a
% shorter time scale
%
% Parameters:
%       obj - The FiberSim object
%       N   - number of trace acquisitions
%       M   - number of fast time samples
%       np_seed - random seed for generating laser drift noise
%
% Returns:
%       nd - matrix of drift values (from 0)
%       new_nd_seed - seed used to generate the laser drift
%
% Author:   Michael Brown
% Date: Oct 5, 2017
% ------------------------------------------------------------------------

function [nd, new_nd_seed, nd_test] = generate_laser_drift(obj, N, M,
nd_seed)
    Tr = obj.get_acq_time();
    obj.fd*Tr;

    % determine random number seeding
    if (isstruct(nd_seed))
        new_nd_seed = rng(nd_seed);
    else
```

```
        new_nd_seed = rng;
    end

    nd = zeros(M,N);                % actual drift value

    if (obj.fd ~= 0)   % if the drift is zero then leave the array set to
zeros
        % the short time variation we allow to change at the smallest time
        % scale for now
        vs = (obj.fd*obj.Ts*1000).*randn(M,N);     % short time values

        % the long time variation is more likely to happen on the order of
        % seconds. We therefore determine the number of random points to
        % generate on the scale of seconds and then interpolate between the
        % point to create a smoother transition.
        Tt = Tr*N;  % the total time of the simulation

        % if we're simulating for less than a second then just generate the
        % start and stop points. The maximum variance proportional to the
total
        % time
        if (Tt < 1)
            N_slow_time = 2;
        else
            N_slow_time = floor(Tt)+1;
        end

        vl = obj.fd*Tr*N.*randn(1,N_slow_time);     % long time values

        Ttl = [0:1:1*(N_slow_time-1)];               % total time, at long
time step
        Tts = [0:obj.Ts:obj.Ts*(N*M)];     % total time, at short time step

        vli = spline(Ttl,vl,Tts);        % cubic spline interpolation of the
slow time noise
        vli = vli';

        nd_test = vli;

        % create actual drift matrix
        for i=1:N
            start_idx = (i-1)*M+1;
            end_idx   = (i*M);
            nd(:,i) = vs(:,i) + vli(start_idx:end_idx,1);   % add short and
long time noise
            nd_test(start_idx:end_idx,1) = vs(:,i) +
vli(start_idx:end_idx,1);   % add short and long time noise
        end
    end
end

FILE: @FiberSim/calculate_trace_average.m
% ------------------------------------------------------------------------
% Function: calculate_trace_average
% Description: This function takes an array of trace data and performs a
% moving average filter over the entire set of data.
```

```
%
% Parameters:
%       obj - The FiberSim object
%       x   - data array
%       N   - number of averages to perform
%
% Returns:
%       y   - filtered array
%
% Author:   Michael Brown
% Date: Nov 21, 2017
% ------------------------------------------------------------------------

function y = calculate_trace_averages(obj, x, N)

    [M,K] = size(x); % x is a multi dimensional array, we need average across
slow times

    % check to see if the number of averages is too big
    if (N >= M)
        y = -1;
        obj.sim_print('ERROR: Number of averages is greater than the number
of traces\n');
    else
        P = M-N+1; % length of resulting array
        y = zeros(P,K);
        obj.sim_print('Averaged array to be of size: %d x %d\n', P, K);

        % create indices for slow time indexing
        start_idx = 1;
        end_idx = N;

        for i=1:P % move through slow time
            subx = x(start_idx:end_idx,:);
            for j=1:K % average each fast time section through slow time
                y(i,j) = mean(subx(:,j));
            end
            start_idx = start_idx+1;
            end_idx = end_idx+1;
        end
    end
end
```

FILE: @FiberSim/calculate_phase_differentials.m

```
% ------------------------------------------------------------------------
% Function: calculate_phase_differentials
% Description: This function calculates the phase differences between
% locations at a separation distance dz, throughout the length of the fiber
% Parameters:
%       obj - The FiberSim object
%       phi - The phase terms
%       dz_step - the spatial resolution step size
%
% Returns:
%       d_phi_ft - array of differential phase angles for a single fast
%       time acquisition
%
```

```
% Author:   Michael Brown
% Date: Nov 12, 2017
% ------------------------------------------------------------------------

function [z, d_phi_ft] = calculate_phase_differentials(obj, phi,
dz_step_factor, sub_step_factor)

    % get the actual sampling resolution
    dz = obj.get_dz();

    % calculate the number of diff samples that will be measured
    %N = floor(2*obj.L/dz);   % get maximum number of samples
    %N = length(phi);    % get maximum number of samples
    % p = floor(dz_step_factor/dz); % get spacing of samples
    % q = floor(sub_step_factor/dz);  % get number of samples to increase
between values
    p = dz_step_factor;
    q = sub_step_factor;
    N = floor((length(phi)/q)-(p/q)+1);

    fprintf('length of phi: %d\n', length(phi));
    fprintf('N: %d\n', N);

    % protect against 0 size
    if (p == 0)
        p = 1;
    end

    % calculate the differentials
    j = 1;
    for i=1:N
        d_phi_ft(1,i) = phi(1, p+j-1) - phi(1,j);
        j = j + q;
    end

    z = [dz/2:(q*dz/2):(q*dz/2)*length(d_phi_ft)];

end

FILE: @FiberSim/calculate_phase_differential_avg.m
% ------------------------------------------------------------------------
% Function: calculate_phase_differential_avg
% Description: This function uses the phase difference avg method between
% locations at a separation distance dAB, with sub-separation, djk
% throughout the length of the fiber
% Parameters:
%       obj - The FiberSim object
%       phi - The phase terms
%       dAB - spatial separation
%       djk - spatial sub-separation
%
% Returns:
%       d_phi_ft - array of differential phase angles for a single fast
%       time acquisition
%
% Author:   Michael Brown
% Date: Nov 12, 2017
```

```
% ------------------------------------------------------------------

function [z, d_phi_AB] = calculate_phase_differential_avg(obj, phi, dAB, djk)

    N = floor(length(phi)/dAB);  % get number of new spatial locations
    dz = obj.get_dz(); % get sampling resolution
    ABi = dAB; % index jump parameter
    jki = djk; % index jump for sub separation

    fprintf('Spatial Resolution: %d\n', dz/2);
    fprintf('New Spatial Resolution: %d\n', dz*dAB/2);
    fprintf('Sub-separation Resolution: %d\n', dz*jki/2);
    fprintf('Number of avgs to compute: %d\n', N);
    fprintf('Total number of samples: %d\n', length(phi));

    d_phi_AB = zeros(1,N);  % results array

    for i=1:N   % loop for each spatial location
        start_idx = (i-1)*ABi+1;
        end_idx = i*ABi;

        fprintf('Starting idx %d\n', start_idx);
        fprintf('End idx %d\n', end_idx);
        phi_AB = phi(1,start_idx:end_idx); % get section of fiber to perform
differential avg

        % perform summations of sub-separations
        K = ceil(1+ABi/2);

        n = 1;
        for k=K:jki:ABi
            J = floor(k-1-ABi/2);
            for j=1:jki:(J+1)
                if (i==1)
                    d_phi_AB(1,i) = (phi_AB(1,k)-phi_AB(1,j))/(k-j);
                else
                    d_phi_AB(1,i) = d_phi_AB(1,i) + (phi_AB(1,k)-
phi_AB(1,j))/(k-j);
                end
                n = n + 1;
            end
        end

        d_phi_AB(1,i) = dAB*d_phi_AB(1,i)/n;

    end

    z = [ABi*(dz/2):ABi*(dz/2):(dz/2)*ABi*N];  % create new z-axis
end

FILE: @FiberSim/calculate_phase_angle.m
% ------------------------------------------------------------------
% Function: calculate_phase_angle
% Description: This function calculates the phase angle from the given
%              scatterers
% Parameters:
```

```
%       obj - The FiberSim object
%       s_i - 2d array containing the location of the scatterers and their
associated amplitude
%       n_i - array of refractive indices
%       np - phase noise array
%       nd - laser drift noise array
%
% Returns:
%       phi - array of calculated phase angles
%       I_terms - array of calculated intensities
%
% Author:   Michael Brown
% Date: May 23, 2017
% -------------------------------------------------------------------------

function [phi,fft_phi, I_terms] = calculate_phase_angle(obj, s_i, n_i, np,
nd)
    % In this function we assume that the received signal is mixed with the
    % LO and that therefore we have a beat frequency that is created. Only
    % the intermediate frequency (IF) portion of the signal is retained.
    % The DC and high frequency content will be assumed to be filtered.

    % calculate the width of the pulse
    Wp = get_Wp(obj);

    % calculate the total time required to capture the entire trace
    T = get_acq_time(obj);

    % create an array of sample times based on the sampling rate of the
system
    t = [0:obj.Ts:T];
    obj.sim_print('FAST TIME SAMPLES:  %d', length(t));

    % calculate the distance traveled by the pulse during between samples
    dz = get_dz(obj);

    % create an array of spatial locations of where the detected scattering
originated from
    % in relation to time of detection
    z = [0:dz/2:obj.L];     % light must travel to z and back to be detected
    obj.sim_print('FAST TIME SAMPLES:  %d', length(z));

    % define the array of photo-detector intensity over time
    I_pd = zeros(1, length(t)); % 2d array. Intensity vs. time
    Q_pd = I_pd;  % quadrature component

    % we use this variable to update our current progress of iterating
    % through the scatterers

    percent = round(0.1*length(t));

    if (percent < 10)
        percent = 10;
    end

    % take all samples to the end of the fiber
```

```
    for i=1:length(t)
        if (mod(i,percent) == 0) % print update every 10%
            fprintf('%i%% complete\r', round(100*i/length(t)));
        end

        % determine which scatterers are within the pulse
        if (z(i) < Wp/2)
            z_range = s_i(1,:)<=z(i);
        else
            % contributing scatterers are located between the leading edge
            % and the middle of the pulse
            z_range = s_i(1,:)>=(z(i)-Wp/2) & s_i(1,:)<=z(i);
        end

        % get number of scatterers in pulse
        num_scatterers = sum(z_range);

        % get the non-zero indices (scatterers) in the pulse
        s_idx = find(z_range);

        I = zeros(1,num_scatterers); % temporary
        Q = I;
        at = (10^obj.alpha)/1000;   % convert attenuation coeff to linear
scale per meter
        v0 = obj.c/(obj.ng*obj.lambda); % convert to frequency
        v = v0 + nd(i);             % add laser drift noise

        % iterate through all scatterers in the pulse
        for j=1:num_scatterers
            zj = s_i(1,s_idx(j));   % location of scatterer
            nj = n_i(1,s_idx(j));   % get the corresponding index of
refraction
            tau = zj*obj.ng/obj.c;    % time to scatterer
            v = (obj.c/(obj.lambda)) + obj.fs + nd(i); % optical frequency
            theta = 2*pi*tau*v;     % phase component from probe frequency
            I(1,j) = s_i(2,s_idx(j))*exp(-2*at*zj)*cos(2*pi*obj.fs*t(i)-
theta+np(i));
            Q(1,j) = s_i(2,s_idx(j))*exp(-2*at*zj)*sin(2*pi*obj.fs*t(i)-
theta+np(i));
        end

        % sum the contributions of the scatterering of all cross terms
        I_pd(1,i) = sum(I(1,:));
        Q_pd(1,i) = sum(Q(1,:));

        % demodulate down to baseband
        I_bb(1,i) = I_pd(1,i)*cos(2*pi*obj.fs*t(i));
        Q_bb(1,i) = Q_pd(1,i)*cos(2*pi*obj.fs*t(i));
    end

    % low pass filter the demodulated signals to remove unwanted high
    % frequency noise
    passband_f = (1/obj.Tp)*1.4;
    stopband_f = (1/obj.Tp)*2;

    if (stopband_f > 0.5*(1/obj.Ts))
```

```
        I_bb_lpf(1,:) = I_bb(1,:);
        Q_bb_lpf(1,:) = Q_bb(1,:);
    else
        lpFilt = designfilt('lowpassfir', 'PassbandFrequency', passband_f,
...
             'StopbandFrequency', stopband_f, 'PassbandRipple', 0.1, ...
             'StopbandAttenuation', 65, 'SampleRate', 1/obj.Ts,
'DesignMethod', 'kaiserwin');
        % apply lpf
        I_bb_lpf(1,:) = filter(lpFilt, I_bb(1,:));
        Q_bb_lpf(1,:) = filter(lpFilt, Q_bb(1,:));
    end

    kz = obj.c/(obj.fs*obj.ng); % distance of one rad
    I_terms = zeros(1, length(I_bb));

    % if I_pd is non zero we need to calculate arctan, otherwise assign 0
angle
    for i=1:length(I_bb)
        num_cycles = floor(i*dz/(2*kz));

        if (Q_bb(1,i) == 0)
            phi(1,i) = 0;
        else
            phi(1,i) = atan2(I_bb_lpf(1,i), Q_bb_lpf(1,i));% -
2*pi*obj.fs*t(i);% + 2*pi*num_cycles;
        end

        I_terms(1,i) = sqrt(I_bb_lpf(1,i)^2 + Q_bb_lpf(1,i)^2);
    end
end
```

FILE: @FiberSim/calculate_fft.m

```
% ------------------------------------------------------------------------
% Function: calculate_fft
% Description: This function takes a multidimensional array and performs
% an FFT across the specified dimension (either slow or fast time)
%
% Parameters:
%       obj - The FiberSim object
%       z   - array of spatial info (for plotting)
%       x   - data array
%       D   - dimension to analyze
%
% Returns:
%       f   - array of frequencies
%       y   - FFT info
%
% Author:   Michael Brown
% Date: Nov 21, 2017
% ------------------------------------------------------------------------

function [f, y] = calculate_fft(obj, z, x, D)

    switch(D)
        case 1
            x1 = x;
```

```
            [N,M] = size(x1);    % get the dimensional sizes of the data
        case 2
            x1 = x';
            [N,M] = size(x1);    % get the dimensional sizes of the data
    end

    N_fft = M;
    K = N;

    n = nextpow2(N_fft);

    N_fft = 2^n;
    y = zeros(K,N_fft/2);

    obj.sim_print('FFT samples: %d\n', N_fft);
    obj.sim_print('Dimensions are %d x %d \n', N, M);

    if (0.5/obj.get_acq_time() < 22000)
        passband_f = (0.9)*(0.5/obj.get_acq_time());
        stopband_f = 1.1*passband_f;
    else
        passband_f = 0.9*22000;  % audio
        stopband_f = 1.1*passband_f;
    end

    obj.sim_print('Passband freq: %d\n', passband_f);
    obj.sim_print('Stopband freq: %d\n', stopband_f);
    obj.sim_print('Sampling rate: %d\n', 1/obj.get_acq_time());

    % low pass data based on sampling rate
    lpFilt = designfilt('lowpassfir', 'PassbandFrequency', passband_f, ...
     'StopbandFrequency', stopband_f, 'PassbandRipple', 0.1, ...
    'StopbandAttenuation', 65, 'SampleRate', 1/obj.get_acq_time(),
'DesignMethod', 'kaiserwin');

    fmax = 1/obj.get_acq_time();
    wn = hanning(length(x1(1,:)));

    for k=1:K
       x_filt = filter(lpFilt, x1(k,:));
       % data includes random noise, so get auto-correlation and do FFT of
       % that
       acorr = xcorr(x_filt);
       acorr = acorr(length(wn):end); % remove negative lags
       y_fft = fft(acorr, N_fft); % get FFT
       y_fft = y_fft.*conj(y_fft); % get magnitude
       y(k,:) = (y_fft(1,1:N_fft/2)); % get real part
    end

    f = [fmax/(N_fft):fmax/(N_fft):fmax/2];
end

FILE: @VibrationEvent/VibrationEvent.m
% -------------------------------------------------------------------------
% Class: VibrationEvent
% Description: This class contains the properties needed to describe a
```

```
%               vibration event.
%
%               Vibration events are defined by their filter coefficients.
%               The simulation uses an Autoregressive (AR) process to
%               simulate vibration events. This method allows for the
%               compact representation of a more complex signal and it
%               allows for easy control over the noise variance for a
%               particular event.
%
%               The method of creating events is as follows:
%
%               1) Generate the desired signal, which will be composed of
%               several frequencies and noise x(t) =
%               sin(w1*t)+sin(w2*t)+..+w(t). The number of samples used
%               will be enough for 1 cycle of lowest frequency.
%
%               2) Use Levison-Durbin method to calculate the AR
%               coefficients & lattice reflection coefficients
%
%               3) The coefficients are used for iterative generation of
%               events.
%
% Author:   Michael Brown
% Date: June 23, 2017
% ------------------------------------------------------------------------

% The VibrationEvent class is defined as a handle class so that we
% modify its contents from within a function (pass by reference)
classdef VibrationEvent < matlab.mixin.SetGet
    properties
        freq    % array of frequencies of vibration
        ep      % strain of event
        zloc    % location of vibration on fiber
        l       % length of fiber undergoing strain
        z_idx   % start and stop indices in scatterer location array
        An      % AR coefficients (order is p=5)
        k       % reflection coefficients
        v_wn    % variance of AGWN
        wn      % array of innovations (length p+1)
        Lat_Filt_Enable     % flag to enable use of lattice filter
        Discrete_Filt_Enable % flag to enable use of direct filter
        gp      % backwards predictors of current event
        gp_hat  % backwards predictor estimations of current event
        fp      % forward preductors of current event
    end

    methods
        % this is the constructor
        function obj=VibrationEvent(z, L, events_arg)
            obj@matlab.mixin.SetGet;
            if (length(events_arg) > 0)
                obj.generate_params(L, events_arg);
                v_wn = events_arg(9);
                obj.v_wn = v_wn;
                Tr = events_arg(end);     % repetition rate of system

                obj.get_zrange(z);  % get the scatterers that are in the
perturbed region
```

```
                x = obj.generate_AR_K_coef(Tr); % calculate the AR model
coeffcients & lattice reflection coefs

                % initialize the lattice filter predictors

                % create array of backwards prediction estimates
                % need to create 2xK array since we will need to keep track
of
                % current and last predictors for each level
                obj.gp_hat = zeros(2,length(obj.k)+1);

                obj.gp = obj.gp_hat; % create array of backwards predictions
                obj.fp = obj.gp_hat; % create array of forward predictions
            end
        end

        % list class functions
        y = AR_lat_filter(obj, d, w, x);
        get_zrange(obj, z);
        dz = get_drange(obj);
        generate_params(obj, L, events_arg);
    end
end
```

FILE: @VibrationEvent/get_zrange.m

```
% ------------------------------------------------------------------------
% Function: get_zrange
% Description: This function searches for all the scatterers that are in
%               the area of perturbation and returns start and end indices
% Parameters:
%       obj - The VibrationEvent object
%       z   - array of scatterer locations
%
% Author:   Michael Brown
% Date: Aug 9, 2017
% ------------------------------------------------------------------------

function get_zrange(obj, z)
    % find the start and stop indices for the scatterer locations
    A = obj.zloc - obj.l/2;
    B = obj.zloc + obj.l/2;
    z_range = (z(:)>=A) & (z(:)<=B);
    z_indices = find(z_range)
    obj.z_idx = [z_indices(1); z_indices(end)];
end
```

FILE: @VibrationEvent/generate_params.m

```
% ------------------------------------------------------------------------
% Function: generate_params
% Description: This function generates the parameters for the vibration
%               event
% Parameters:
%       obj - The VibrationEvent object
%       events - event parameters passed from GUI
%
% Returns:
%
```

```
% Author:   Michael Brown
% Date: Aug 8, 2017
% ------------------------------------------------------------------------

function generate_params(obj, L, events_arg)
    % extract the event parameters
    f_max = events_arg(2);
    f_min = events_arg(3);
    f_num = events_arg(4);
    v_ep_max = events_arg(5);
    v_ep_min = events_arg(6);
    v_length_max = events_arg(7);
    v_length_min = events_arg(8);

    fprintf('Generating vibration parameters\n');
    % generate the vibration event parameters randomly
    obj.freq = rand(1,f_num)*(f_max-f_min)+f_min;
    obj.ep = rand(1)*(v_ep_max-v_ep_min)+v_ep_min;
    obj.l = rand(1)*(v_length_max - v_length_min)+v_length_min;
    obj.zloc = obj.l/2+rand(1)*(L-obj.l);
end
```

FILE: @VibrationEvent/generate_AR_K_coef.m

```
% ------------------------------------------------------------------------
% Function: generate_AR_K_coef
% Description: This function generates the desired signal (sinusoids plus
% noise) & uses the Levison-Durbin method to calculate the AR coefficients
% for an AR model as well as the reflection coefficients for a lattice
filter.
%
% Parameters:
%       obj - The VibrationEvent object
%       Tr  - Repetition rate
%
% Returns:
%       x - target signal
%
% Author:   Michael Brown
% Date: Aug 9, 2017
% ------------------------------------------------------------------------

function x = generate_AR_K_coef(obj, Tr)
    t = [0:Tr:Tr*(20000-1)]; % create time array
    x = zeros(1,20000);  % create result array

    % loop through all the frequencies in the target signal
    for i=1:length(obj.freq)
        x = x + sin(2*pi*obj.freq(i).*t);
    end

    % add AWGN
    wn = randn(1,20000);
    x = x + wn;
    x = x./max(x); % normalize

    % get correlation of x
    r = xcorr(x,'biased');
```

```
    r(1:length(x)-1) = [];  % remove negative lags

    % calculate the AR coefficients & lattice filter reflection
    % coefficients
    [obj.An, er, obj.k] = levinson(r,10);     % use 10th order AR by default
end

FILE: @VibrationEvent/AR_lat_filter.m
% ------------------------------------------------------------------------
% Function: AR_lat_filter
% Description: This function implements a Warped IIR (WIIR) lattice filter.
%
% Parameters:
%       obj - The VibrationEvent object
%       d - warping coefficient
%       w - Array of innovations
% Optional:
%       x - Array of previous outputs
%
% Returns:
%       y - Array of filtered output
%
% Author:   Michael Brown
% Date: Aug 8, 2017
% ------------------------------------------------------------------------

function y = AR_lat_filter(obj, d, w, x)

    % if we don't supply the previous samples we set them all to zeros
    y_prev = zeros(1,length(obj.An));

    if (nargin == 4)
        y_prev(1:length(x)) = x; % set the previous samples to x
    end

    Discrete_Filter = obj.Discrete_Filt_Enable;
    Lattice_Filter = obj.Lat_Filt_Enable;

    if (Discrete_Filter == true)
        % do the filtering in an explicit way, instead of using "filter"
        % function
        for i=1:length(w)
            y(i) = w(i);   % start with the current innovation value

            % iterate through all the coefficients and previous samples
            for j=1:length(obj.An)-1
                % calculate the value of the all pass filter
                % used for creating a frequency warped version of the signal
                dn = (-d+y_prev(j))/(1-d*y_prev(j));
                % add to current value
                y(i) = y(i) - obj.An(j+1)*dn;
            end

            % update the array of previous outputs
            y_prev(2:end) = y_prev(1:end-1);
            y_prev(1) = y(i);
        end
```

```
    end

    if (Lattice_Filter == true)
        % NOTE:
        % We let f(0) be the output of the filter and f(p) be the
        % input to the filter.

        % iterate through all innovations
        for i=1:length(w)
            N = length(obj.k)+1;
            k = obj.k'; % we read this in reverse order

            % move samples back in time
            obj.gp(2,:) = obj.gp(1,:); % let gp(n) = gp(n-1), etc
            obj.gp_hat(2,:) = obj.gp_hat(1,:);
            obj.fp(2,:) = obj.fp(1,:);

            for j = N:-1:2   % iterate through all levels
                if (j == N) % at the input to the filter
                    % set f(p) to current input
                    obj.fp(1,j) = w(i);
                end

                obj.gp_hat(1,j) = obj.gp(2,j-1) - d*(obj.gp(1,j-1) -
obj.gp_hat(2,j));
                obj.fp(1,j-1) = -obj.gp_hat(1,j)*k(j-1) +obj.fp(1,j); %
estimate forward predictor
                obj.gp(1,j) = obj.gp_hat(1,j) + k(j-1)*obj.fp(1,j-1); %
estimate backward predictor

                if (j == 2)
                    obj.gp(1,j-1) = obj.fp(1,j-1);  % set predictors equal at
output
                end
            end
            y(i) = obj.fp(1,1); % save the output of the filter over time
        end
    end
end
```

FILE: @NonStationaryEvent/NonStationaryEvent.m

```
% -------------------------------------------------------------------
% Class: NonStationaryEvent
% Description: This class contains the properties needed to describe a
%               nonstationary vibration event.
%
%               Nonstationary events are sub classes of vibration events
%               with the difference that the warping factor for the WIIR
%               filter is time varying.
%
%               Generation of the time varying warping factor is as
%               follows:
%
%               1) A random series of N values between -1 and +1 are
%               generated, where N is the number of traces acquired by the
%               system
%
```

```
%               2) The generated array of values is then smoothed via low
%               pass filtering.
%
%               3) The resulting array of coefficients is used to change
%               the values on the all-pass filter section of the lattice
%               filter
%
% Author:   Michael Brown
% Date: Aug 22, 2017
% ------------------------------------------------------------------------

% The NonStationaryEvent class is defined as a VibrationEvent sub-class

classdef NonStationaryEvent < VibrationEvent

    properties
        d           % array of warping coefficients
    end

    methods
        % this is the constructor
        function obj=NonStationaryEvent(z, L, N, events_arg)
            if (nargin > 0)
                z_arg = z;
                L_arg = L;
                events_arg_arg = events_arg;
            else
                z_arg = [];
                L_arg = [];
                events_arg_arg = {};
            end

            obj@VibrationEvent(z_arg, L_arg, events_arg_arg);

            if (nargin > 0)
                obj.d = obj.generate_TV_warping(N);
            end
        end

        d = generate_TV_warping(obj, N);
    end
end

FILE: @NonStationaryEvent/generate_TV_warping.m
% ------------------------------------------------------------------------
% Function: generate_TV_warping
% Description: This function generates a random array of warping
%               coefficents used for generating a nonstationary AR signal
% Parameters:
%       obj - The NonStationaryEvent object
%       N - the size of the target array
%
% Returns:
%       d - the TV coefficient array
%
% Author:   Michael Brown
% Date: Aug 22, 2017
```

```
% ------------------------------------------------------------------

function d = generate_TV_warping(obj, N)
    d = rand(1,N);  % create array of random values
    a = ones(1,N)/N; % create array of filter coefficients (evenly weighted,
for moving average)
    d = filter(a, 1, d); % run the data through the filter
end

FILE: @run_fibersim.m
% ------------------------------------------------------------------------
% Function: run_fibersim
% Description: This program runs a simulation of the phase otdr simulator.
% Parameters:
% varargin - variable parameters which include:
%   [seed filename]         - Name of the file which contains the seed values
for random number generator
%   [parameter filename]    - Name of the file which contain the parameter
values
%   N                       - # of scatterers to simulate
%   A                       - amplitude of scatterers
%   Tp                      - Duration of pulse
%   fs                      - frequency shift from AOM (in Hz)
%   lamba                   - wavelength of pulse
%   flw                     - linewidth of laser (in Hz)
%   fd                      - frequency drift rate of laser (in Hz/s)
%   L                       - Length of fiber (in meters)
%   alpha                   - attentuation coefficient (dB/km)
%   Ts                      - sampling rate of system
%   M                       - number of acquisitions to obtain
%   savefile                - whether to save the seed and parameters to a
%                             file
%
% Returns:
%   d_phi_terms         - the differential phase values (slow time)
%   phi_terms           - the absolute phase terms (slow time)
%   I_terms             - the intensity values (slow time)
%   np                  - phase noise values (slow time)
%   nd                  - laser drift noise
%   z                   - array of spatial positions
%   vevents             - array of vibration events
%   nevents             - array of nonstationary events
%
% Author:   Michael Brown
% Date: July 1, 2016
% ------------------------------------------------------------------------

function [fiberObj, phi_terms, I_terms, dep, np, nd, z, vevents, nevents] =
...
run_fibersim(num_traces, vevents_en, nevents_en, vevents_params,
nevents_params, params, seeds, LogHandle)

    % extract the different seeds from the seed parameter
    a_seed = seeds(1);
    z_seed = seeds(2);
    np_seed = seeds(3);
    nd_seed = seeds(4);
```

```
    % make sure that parameters have been created
    if (params ~= -1)
        % call the constructor
        fiberObj = FiberSim(params, LogHandle);
        fiberObj.LogHandle.Value = {''};
        fiberObj.sim_print('Starting fiber sim....');
        fiberObj.sim_print('FUT LENGTH:              %d m', fiberObj.L);
        fiberObj.sim_print('PULSE LENGTH:            %d m',
fiberObj.get_Wp());
        fiberObj.sim_print('SAMPLE RATE:             %d MHz',
1/(1e6*fiberObj.Ts));
        fiberObj.sim_print('SPATIAL RESOLUTION:      %d m',
0.5*fiberObj.Ts*fiberObj.c/fiberObj.ng);
        fiberObj.sim_print('REPETITON RATE:          %d Hz',
1/fiberObj.get_acq_time());
        fiberObj.sim_print('NUMBER OF ACQUISITIONS:  %d\n', num_traces);

        % calculate the distance traveled by the pulse during between samples
        dz = fiberObj.get_dz();

        % create an array of spatial locations of where the detected
scattering originated from
        % in relation to time of detection
        z = [0:dz/2:fiberObj.L];        % light must travel to z and back to
be detected

        fiberObj.sim_print('FAST TIME SAMPLES:  %d', length(z));

        Tr = fiberObj.get_acq_time(); % round trip time

        % create an array of sample times based on the sampling rate of the
system
        t = [0:fiberObj.Ts:Tr];

        fiberObj.sim_print('FAST TIME SAMPLES:  %d', length(t));

        % generate scatterers
        [z_i, a_i, n_i, new_a_seed, new_z_seed] =
fiberObj.generate_scatterers(a_seed, z_seed);

        % generate laser phase noise
        fiberObj.sim_print('Generating random phase noise .....\n');
        [np, new_np_seed] = fiberObj.generate_phase_noise(num_traces,
np_seed);
        np_no_noise = zeros(length(np), num_traces);

        % generate laser drift
        fiberObj.sim_print('Generating laser frequency drift noise .....\n');
        num_samples = length(z);
        [nd, new_nd_seed] = fiberObj.generate_laser_drift(num_traces,
num_samples, nd_seed);
        nd_no_noise = zeros(length(nd), num_traces);

        % 2) create structure of locations and amplitudes
        s_i = [z_i; a_i];
```

```
        % Generate vibration events
        if (vevents_en == 1)
            vevents = fiberObj.generate_vibration_events(vevents_params,
z_i);
        elseif (vevents_en == 2)
            load(vevents_params(1));
        else
            vevents = 0;
        end

        % deal with nonstationary events
        if (nevents_en == 1)
            nevents = fiberObj.generate_nonstationary_events(nevents_params,
z_i, num_traces);
        elseif(nevents_en == 2)
            load(nevents_params(1));
        else
            nevents = 0;
        end

        %TEST_EVENT_EN = true;
        TEST_EVENT_EN = false;

        % generate a simple test event if enabled
        if (TEST_EVENT_EN)
            for i=1:3
                tevent_params(1) = 0;
                tevent_params(2) = 1;
                tevent_params(3) = 2000*i+(i-1)*200;
                tevent_params(4) = 2000*i+(i-1)*200;
                tevent_params(5) = 1;
                tevent_params(6) = 1e-7;
                tevent_params(7) = 1e-7;
                tevent_params(8) = 10;
                tevent_params(9) = 10;
                tevent_params(10) = fiberObj.L/(2*i)+i*50;
                tevent(i) = fiberObj.generate_test_event(tevent_params, z_i);
            end
        end

        % save seeds to temporary file
        new_seeds = [new_a_seed; new_z_seed; new_np_seed; new_nd_seed];
        save('seeds_tmp', 'new_seeds');

        % Create status bar
        wb_update_str = sprintf('Simulating acquisition %i', 1);
        wb = waitbar(1/(num_traces*10), wb_update_str);

        % 4) Obtain the baseline trace
        [phi_terms_new, tmp, I] = fiberObj.calculate_phase_angle(s_i, n_i,
np(:,1), nd(:,1));
        phi_terms = zeros(num_traces, length(phi_terms_new));
        phi_terms_raw = phi_terms;
        I_terms = zeros(num_traces, length(phi_terms_new));

        fiberObj.sim_print('FAST TIME SAMPLES:  %d', length(phi_terms_new));
```

```
        % calculate the unwrapped phases
        phi_terms_raw(1,:) = phi_terms_new;
        phi_terms(1,:) = unwrap(phi_terms_new);
        phi_terms_new = unwrap(phi_terms_new);

        I_terms(1,:) = I;

        % debug (save strain values)
        dep = zeros( (length(vevents)+length(nevents)), (num_traces-1) );

        % define maximum expected phase change in slow time (20kHz event)
        d_phi_max = 20000*fiberObj.get_acq_time()*2*pi;

        z_init = z_i;   % save original locations

        if (TEST_EVENT_EN)
            z_hist(1,1) = z_i(tevent(1).z_idx(1));
        end

        if (vevents_en == 1)
            for i=1:length(vevents)
                z_hist(i,1) = z_i(vevents(i).z_idx(1));
            end
        end

        % 5) obtain the rest of the traces and calculate the deltas
        for i=1:num_traces-1
            fiberObj.sim_print('Acquiring trace %d of %d.....', i,
num_traces);
            phi_terms_old = phi_terms_new;

            % perturb a section of the fiber
            if (vevents_en == 1)
                for j=1:length(vevents)
                    fiberObj.sim_print('Applying vibration: %d\n', j);
                    [pz, dep(j,i)] = fiberObj.pert_section(z_init,
vevents(j), i);
                    z_hist(j,i+1) = z_i(vevents(j).z_idx(1));
                    z_i(1,vevents(j).z_idx(1):vevents(j).z_idx(2)) = pz;
                    s_i = [z_i; a_i];
                end

                s_i = [z_i; a_i];
            end

            if (nevents_en == 1)
                for j=1:length(nevents)
                    fiberObj.sim_print('Apply nonstationary event: %d\n', j);
                    [pz, dep(j,i)] = fiberObj.pert_section(z_init,
nevents(j), i);
                    z_hist(j,i+1) = z_i(nevents(j).z_idx(1));
                    z_i(1,nevents(j).z_idx(1):nevents(j).z_idx(2)) = pz;
                    s_i = [z_i; a_i];
                end
```

```
                s_i = [z_i; a_i];
            end

            if (TEST_EVENT_EN)
                for j=1:length(tevent)
                    fiberObj.sim_print('Applying test event\n');
                    [pz, test_dep(i)] = fiberObj.pert_section(z_init,
tevent(j), i);
                    z_i(1,tevent(j).z_idx(1):tevent(j).z_idx(2)) = pz;
                    z_hist(1,i+1) = z_i(tevent(j).z_idx(1));
                    s_i = [z_i; a_i];
                end
            end

            wb_update_str = sprintf('Simulating acquisition %i', i+1);
            waitbar((i+1)*10/(num_traces*10), wb, wb_update_str);

            % get the new trace
            [phi_terms_new, tmp, I] = fiberObj.calculate_phase_angle(s_i,
n_i, np(:,i+1), nd(:,i+1));
            phi_terms(i+1,:) = fiberObj.unwrap(phi_terms(i,:),
phi_terms_new(1,:));
            phi_terms_raw(i+1,:) = phi_terms_new;
            phi_terms_new = unwrap(phi_terms_new);
            I_terms(i+1,:) = I;
        end

        close(wb);
    else
        fiberObj.sim_print('Terminating program...');
    end
end
```

FILE: FiberSimGUI_2017a

```
classdef FiberSimGUI_2017a < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        PhaseOTDRSimulatorUIFigure      matlab.ui.Figure
        TabGroup                        matlab.ui.container.TabGroup
        CfgTab                          matlab.ui.container.Tab
        LabelDropDown                   matlab.ui.control.Label
        DetectionTypeDropDown           matlab.ui.control.DropDown
        Label                           matlab.ui.control.Label
        ParameterSelection              matlab.ui.container.ButtonGroup
        FileButton                      matlab.ui.control.RadioButton
        ManualButton                    matlab.ui.control.RadioButton
        LabelEditField                  matlab.ui.control.Label
        FileNameStr                     matlab.ui.control.EditField
        FileBrowseButton                matlab.ui.control.Button
        Label2                          matlab.ui.control.Label
        LabelNumericEditField           matlab.ui.control.Label
        FiberLengthField                matlab.ui.control.NumericEditField
        Label3                          matlab.ui.control.Label
        LabelNumericEditField2          matlab.ui.control.Label
        NumScatterersField              matlab.ui.control.NumericEditField
```

```
        LabelNumericEditField7          matlab.ui.control.Label
        FiberAttCoefField               matlab.ui.control.NumericEditField
        Label9                          matlab.ui.control.Label
        LabelNumericEditField8          matlab.ui.control.Label
        IORField                        matlab.ui.control.NumericEditField
        Label4                          matlab.ui.control.Label
        LabelNumericEditField3          matlab.ui.control.Label
        PulseDurationField              matlab.ui.control.NumericEditField
        Label5                          matlab.ui.control.Label
        LabelNumericEditField4          matlab.ui.control.Label
        SamplingRateField               matlab.ui.control.NumericEditField
        Label6                          matlab.ui.control.Label
        LabelNumericEditField6          matlab.ui.control.Label
        LaserLambdaField                matlab.ui.control.NumericEditField
        Label8                          matlab.ui.control.Label
        LabelNumericEditField5          matlab.ui.control.Label
        LaserLinewidthField             matlab.ui.control.NumericEditField
        Label7                          matlab.ui.control.Label
        LabelNumericEditField9          matlab.ui.control.Label
        LaserDriftRateField             matlab.ui.control.NumericEditField
        Label10                         matlab.ui.control.Label
        LabelNumericEditField10         matlab.ui.control.Label
        FrequencyShiftField             matlab.ui.control.NumericEditField
        Label11                         matlab.ui.control.Label
        LabelNumericEditField11         matlab.ui.control.Label
        MaxScattererMagField            matlab.ui.control.NumericEditField
        RandomNumberGroup               matlab.ui.container.ButtonGroup
        SeedFileButton                  matlab.ui.control.RadioButton
        GenerateSeedButton              matlab.ui.control.RadioButton
        LabelEditField2                 matlab.ui.control.Label
        SeedFileStr                     matlab.ui.control.EditField
        SeedFileBrowseButton            matlab.ui.control.Button
        EventSetupTab                   matlab.ui.container.Tab
        Panel                           matlab.ui.container.Panel
        VibrationEventCheckBox          matlab.ui.control.CheckBox
        NonstationaryEventCheckBox      matlab.ui.control.CheckBox
        VibrationButtonGroup            matlab.ui.container.ButtonGroup
        VibrationFileButton             matlab.ui.control.RadioButton
        VibrationManualButton           matlab.ui.control.RadioButton
        LabelNumericEditField13         matlab.ui.control.Label
        NumVibrationEventsField         matlab.ui.control.NumericEditField
        LabelNumericEditField14         matlab.ui.control.Label
        VibrationMaxFreqField           matlab.ui.control.NumericEditField
        LabelNumericEditField16         matlab.ui.control.Label
        VibrationMaxStrainField         matlab.ui.control.NumericEditField
        LabelNumericEditField20         matlab.ui.control.Label
        VibrationMaxLengthField         matlab.ui.control.NumericEditField
        Label14                         matlab.ui.control.Label
        Label16                         matlab.ui.control.Label
        LabelNumericEditField15         matlab.ui.control.Label
        VibrationMinFreqField           matlab.ui.control.NumericEditField
        LabelNumericEditField17         matlab.ui.control.Label
        VibrationMinStrainField         matlab.ui.control.NumericEditField
        Label17                         matlab.ui.control.Label
        VibrationMinLengthField         matlab.ui.control.NumericEditField
```

```
        Label18                          matlab.ui.control.Label
        Label15                          matlab.ui.control.Label
        LabelEditField4                  matlab.ui.control.Label
        VibrationFileStrField            matlab.ui.control.EditField
        VibrationBrowseButton            matlab.ui.control.Button
        LabelNumericEditField21          matlab.ui.control.Label
        VibrationNumFreqField            matlab.ui.control.NumericEditField
        NonstationaryButtonGroup         matlab.ui.container.ButtonGroup
        NonstationaryFileButton          matlab.ui.control.RadioButton
        NonstationaryManualButton        matlab.ui.control.RadioButton
        LabelNumericEditField18          matlab.ui.control.Label
        NumNonstationaryEventsField      matlab.ui.control.NumericEditField
        Label13                          matlab.ui.control.Label
        NonstationaryMaxStrainField      matlab.ui.control.NumericEditField
        Label20                          matlab.ui.control.Label
        NonstationaryMaxLengthField      matlab.ui.control.NumericEditField
        LabelNumericEditField19          matlab.ui.control.Label
        NonstationaryMinStrainField      matlab.ui.control.NumericEditField
        Label21                          matlab.ui.control.Label
        NonstationaryMinLengthField      matlab.ui.control.NumericEditField
        Label19                          matlab.ui.control.Label
        LabelEditField5                  matlab.ui.control.Label
        NonstationaryFileStrField        matlab.ui.control.EditField
        NonstationaryBrowseButton        matlab.ui.control.Button
        Label27                          matlab.ui.control.Label
        NonstationaryNumFreqField        matlab.ui.control.NumericEditField
        Label28                          matlab.ui.control.Label
        NonstationaryMaxFreqField        matlab.ui.control.NumericEditField
        Label29                          matlab.ui.control.Label
        NonstationaryMinFreqField        matlab.ui.control.NumericEditField
        SimulationTab                    matlab.ui.container.Tab
        Label12                          matlab.ui.control.Label
        LabelNumericEditField12          matlab.ui.control.Label
        NumTracesField                   matlab.ui.control.NumericEditField
        LabelTextArea                    matlab.ui.control.Label
        SimulationLog                    matlab.ui.control.TextArea
        SimulationStartButton            matlab.ui.control.Button
        SaveParamsButton                 matlab.ui.control.Button
        SaveEventsButton                 matlab.ui.control.Button
        SaveSeedButton                   matlab.ui.control.Button
        Label24                          matlab.ui.control.Label
        Label26                          matlab.ui.control.Label
        Label25                          matlab.ui.control.Label
        ViewEventTableButton             matlab.ui.control.Button
        ClearButton                      matlab.ui.control.Button
        TraceAnalysisTab                 matlab.ui.container.Tab
        AnalysisTypeLabel                matlab.ui.control.Label
        PlotSelectionMenu                matlab.ui.control.DropDown
        AnalysisPlot                     matlab.ui.control.UIAxes
        UndockFigureButton               matlab.ui.control.Button
        TraceAveragesEditFieldLabel      matlab.ui.control.Label
        TraceAveragesEditField           matlab.ui.control.NumericEditField
        ABseparationfactorEditFieldLabel  matlab.ui.control.Label
        ABseparationfactorEditField      matlab.ui.control.NumericEditField
        SubseparationfactorEditFieldLabel  matlab.ui.control.Label
```

```
        SubseparationfactorEditField    matlab.ui.control.NumericEditField
        ShowTraceDeltasCheckBox         matlab.ui.control.CheckBox
        DifferentialSeparationFactorEditFieldLabel
matlab.ui.control.Label
        DifferentialSeparationFactorEditField
matlab.ui.control.NumericEditField
        SubseparationfactorEditField_2Label  matlab.ui.control.Label
        SubseparationfactorEditField_2  matlab.ui.control.NumericEditField
        DifferentialAnalysisParametersLabel  matlab.ui.control.Label
        DifferentialAvgAnalysisParametersLabel  matlab.ui.control.Label
        SlowTimeFFTButton               matlab.ui.control.Button
        AcquisitionNumberSpinnerLabel   matlab.ui.control.Label
        AcquisitionSpinner              matlab.ui.control.Spinner
        EventLogTab                     matlab.ui.container.Tab
        StationaryEventsSummary         matlab.ui.container.Panel
        StationaryEventsSummaryLog      matlab.ui.control.TextArea
        EventNumberSpinnerLabel         matlab.ui.control.Label
        StationaryEventsSpinner         matlab.ui.control.Spinner
        StationaryEventsFreqPlot        matlab.ui.control.UIAxes
        StationaryEventsUndockFigure    matlab.ui.control.Button
        NonStationaryEventsSummary      matlab.ui.container.Panel
        LabelTextArea2                  matlab.ui.control.Label
        NonStationaryEventsSummaryLog   matlab.ui.control.TextArea
        EventNumberSpinner_2Label       matlab.ui.control.Label
        NonStationaryEventsSpinner      matlab.ui.control.Spinner
        NonStationaryEventsFreqPlot     matlab.ui.control.UIAxes
        NonStationaryEventsUndockFigure  matlab.ui.control.Button
    end


    properties (Access = private)
        fiberObj       % fiber object
        last_x         % last saved x-axis
        last_y         % last saved y-axis
        last_z         % last saved z-axis
        phi_terms      % phase angle terms
        I_terms        % intensity terms
        np             % phase noise component
        nd             % laser drift component
        z_axis         % spatial axis
        vevents        % generated stationary events
        nevents        % generated nonstationary events
        dep            % array of applied strain
    end

    methods (Access = private)

        % handle the plotting of data
        function results = handlePlot(app)
            % get plot type and acquisition number
            plotType = app.PlotSelectionMenu.Value;
            acqNum = app.AcquisitionSpinner.Value;
            numAvgs = app.TraceAveragesEditField.Value;
            plotScale = 0; % linear
```

```
            str = sprintf('Calculating average....\n');
            app.SimulationLog.Value(length(app.SimulationLog.Value)+1) =
{str};

            % check to see if we are performing trace level averaging
            if (numAvgs > 1)
                phi_avg =
app.fiberObj.calculate_trace_averages(app.phi_terms, numAvgs);
                I_avg = app.fiberObj.calculate_trace_averages(app.I_terms,
numAvgs);
            else
                phi_avg = app.phi_terms;
                I_avg = app.I_terms;
            end

            % make sure we don't exceed the maximum index
            if (acqNum >= app.NumTracesField.Value)
                acqNum = app.NumTracesField.Value-1;
            end

            % Calculate selected plot type
            switch plotType
                case 'Phase Angle Plot'
                    % check to see if we're calculated trace deltas
                    if (app.ShowTraceDeltasCheckBox.Value)
                        %phi_avg(acqNum+1,:) =
app.fiberObj.fix_phase_angles(phi_avg(acqNum:acqNum+1,:), 2*pi);
                        y = phi_avg(acqNum+1,:) - phi_avg(acqNum,:);
                        app.AnalysisPlot.Title.String = 'Backscatter Phase
Angles Difference Plot';
                    else
                        y = phi_avg(acqNum,:);
                        app.AnalysisPlot.Title.String = 'Backscatter Phase
Angles Plot';
                    end

                    x = app.z_axis;
                    ylabel(app.AnalysisPlot, 'Phase (rads)');

                case 'Phase Differential Plot'
                    dz_step =
app.DifferentialSeparationFactorEditField.Value;
                    inc_step = app.SubseparationfactorEditField_2.Value;

                    str = sprintf('Calculating phase
differentials....\n');

app.SimulationLog.Value(length(app.SimulationLog.Value)+1) = {str};

                    if (app.ShowTraceDeltasCheckBox.Value)
                        [x,y1] =
app.fiberObj.calculate_phase_differentials(phi_avg(acqNum, :), dz_step,
inc_step);
                        [x,y2] =
app.fiberObj.calculate_phase_differentials(phi_avg(acqNum+1, :), dz_step,
```

```
inc_step);
                        y = y2-y1;
                        app.AnalysisPlot.Title.String = 'Backscatter Phase
Angles Differential Deltas Method Plot';
                    else
                        [x,y] =
app.fiberObj.calculate_phase_differentials(phi_avg(acqNum, :), dz_step,
inc_step);
                        app.AnalysisPlot.Title.String = 'Backscatter Phase
Angles Differential Method Plot';
                    end

                    ylabel(app.AnalysisPlot, 'Phase (rads)');

                case 'Phase Differential Avg Plot'
                    dAB = app.ABseparationfactorEditField.Value;
                    djk = app.SubseparationfactorEditField.Value;

                    str = sprintf('Calculating phi differential
avgs....\n');

app.SimulationLog.Value(length(app.SimulationLog.Value)+1) = {str};

                    if (app.ShowTraceDeltasCheckBox.Value)
                        [x,y1] =
app.fiberObj.calculate_phase_differential_avg(phi_avg(acqNum,:), dAB,
djk);
                        [x,y2] =
app.fiberObj.calculate_phase_differential_avg(phi_avg(acqNum+1,:), dAB,
djk);
                        y = y2-y1;
                        app.AnalysisPlot.Title.String = 'Backscatter Phase
Angles Differential Avg Deltas Method Plot';
                    else
                        [x,y] =
app.fiberObj.calculate_phase_differential_avg(phi_avg(acqNum,:), dAB,
djk);
                        app.AnalysisPlot.Title.String = 'Backscatter Phase
Angles Differential Avg Method Plot';
                    end

                    ylabel(app.AnalysisPlot, 'Phase (rads)');

                case 'Intensity Plot'
                    if (app.ShowTraceDeltasCheckBox.Value)
                        I_diff = abs(I_avg(acqNum+1,:) - I_avg(acqNum,:));
                        if (I_diff == 0)
                            I_diff = 1e-40;
                        end
                        y = 10*log10(I_diff);
                        plotScale = 1; % log scale
                        app.AnalysisPlot.Title.String = 'Backscatter
Intensity Differences Plot';
                    else
                        y = 10*log10(app.I_terms(acqNum,:));
```

```
                        plotScale = 1; % log scale
                        app.AnalysisPlot.Title.String = 'Backscatter
Intensity Plot';
                    end

                    x = app.z_axis;

                case 'STFT'
                case 'Laser Phase Noise'
                    y = app.np(:, acqNum);
                    x = app.z_axis;
                    app.AnalysisPlot.Title.String = 'Laser Phase Noise';

                otherwise
            end

            str = sprintf('Plotting values....\n');
            app.SimulationLog.Value(length(app.SimulationLog.Value)+1) =
{str};

            if (plotScale == 1)
                app.AnalysisPlot.YScale = 'log';
                plot(app.AnalysisPlot, x, y);
                ylabel(app.AnalysisPlot, 'Magnitude (dB)');
            else
                % plot the requested data
                app.AnalysisPlot.YScale = 'linear';
                plot(app.AnalysisPlot, x, y);
            end

            xlabel(app.AnalysisPlot, 'Spatial Location (m)');
            app.last_x = x;
            app.last_y = y;
        end

        function results = handle3DPlot(app)
            % get plot type
            plotType = app.PlotSelectionMenu.Value;
            numAvgs = app.TraceAveragesEditField.Value;
            plotScale = 0; % linear

            str = sprintf('Calculating average....\n');
            app.SimulationLog.Value(length(app.SimulationLog.Value)+1) =
{str};

            % check to see if we are performing trace level averaging
            if (numAvgs > 1)
                phi_avg =
app.fiberObj.calculate_trace_averages(app.phi_terms, numAvgs);
                I_avg = app.fiberObj.calculate_trace_averages(app.I_terms,
numAvgs);
            else
                phi_avg = app.phi_terms;
                I_avg = app.I_terms;
            end
```

```
            % Calculate selected plot type
            switch plotType
                case 'Phase Angle Plot'
                    [N,M] = size(phi_avg);
                    y = zeros(N,M);

                    % check to see if we're calculated trace deltas
                    if (app.ShowTraceDeltasCheckBox.Value)
                        for i=1:N-1
                            %phi_avg(i+1,:) =
app.fiberObj.fix_phase_angles(phi_avg(i:i+1,:), 2*pi);
                            y(i,:) = phi_avg(i+1,:) - phi_avg(i,:);
                            plot_title_str = 'FFT of Phase Difference';
                        end
                    else
                        y = phi_avg;
                        plot_title_str = 'FFT of Phase Angles';
                    end

                    x = app.z_axis;

                case 'Phase Differential Plot'
                    dz_step =
app.DifferentialSeparationFactorEditField.Value;
                    inc_step = app.SubseparationfactorEditField_2.Value;

                    [N,M] = size(phi_avg);

                    % calculate number of samples in each calculation
                    p = dz_step; % get spacing of samples
                    q = inc_step;  % get number of samples to increase
between values
                    K = floor((M/q)-(p/q)+1);

                    y = zeros(N,K);    % initialize array

                    str = sprintf('Calculating phase
differentials....\n');

app.SimulationLog.Value(length(app.SimulationLog.Value)+1) = {str};

                    if (app.ShowTraceDeltasCheckBox.Value)
                        for i=1:N-1
                            [x,y1] =
app.fiberObj.calculate_phase_differentials(phi_avg(i, :), dz_step,
inc_step);
                            [x,y2] =
app.fiberObj.calculate_phase_differentials(phi_avg(i+1, :), dz_step,
inc_step);
                            y(i,:) = y2-y1;
                        end
                        plot_title_str = 'FFT of Phase Differential
Deltas';
                    else
```

```
                        for i=1:N
                            [x,y(i,:)] =
app.fiberObj.calculate_phase_differentials(phi_avg(i, :), dz_step,
inc_step);
                        end
                        plot_title_str = 'FFT of Phase Differential';
                    end

                case 'Phase Differential Avg Plot'
                    dAB = app.ABseparationfactorEditField.Value;
                    djk = app.SubseparationfactorEditField.Value;

                    [N,M] = size(phi_avg);
                    M = floor(length(phi_avg)/dAB);  % get number of new
spatial locations
                    y = zeros(N,M);

                    str = sprintf('Calculating phase differentials
average....\n');

app.SimulationLog.Value(length(app.SimulationLog.Value)+1) = {str};

                    if (app.ShowTraceDeltasCheckBox.Value)
                        for i=1:N-1
                            [x,y1] =
app.fiberObj.calculate_phase_differential_avg(phi_avg(i,:), dAB, djk);
                            [x,y2] =
app.fiberObj.calculate_phase_differential_avg(phi_avg(i+1,:), dAB, djk);
                            y(i,:) = y2-y1;
                        end
                        plot_title_str = 'FFT of Phase Differential Avg
Deltas';
                    else
                        for i=1:N
                            [x,y(i,:)] =
app.fiberObj.calculate_phase_differential_avg(phi_avg(i,:), dAB, djk);
                        end
                        plot_title_str = 'FFT of Phase Differential Avg
Plot';
                    end

                case 'Intensity Plot'
                    [N,M] = size(I_avg);
                    y = zeros(N,M);
                    plotScale = 1; % log

                    if (app.ShowTraceDeltasCheckBox.Value)
                        for i=1:N-1
                            I_diff = abs(I_avg(i+1,:) - I_avg(i,:));
                            if (I_diff == 0)
                                I_diff = 1e-40;
                            end
                            y(i,:) = I_diff;
                        end
                        plot_title_str = 'FFT of Intensity Differences';
```

```
                    else
                        y = I_avg;
                        plot_title_str = 'FFT of Intensity Plot';
                    end

                    x = app.z_axis;

                case 'STFT'
                case 'Laser Phase Noise'
                    y = app.np(:, acqNum);
                    x = app.z_axis;
                    app.AnalysisPlot.Title.String = 'Laser Phase Noise';

                otherwise
            end

            str = sprintf('Calculating FFT....\n');
            app.SimulationLog.Value(length(app.SimulationLog.Value)+1) =
{str};

            z = x;
            [f, x] = app.fiberObj.calculate_fft(x, y, 2);
            figure;

            if (plotScale == 1)
                x = 10*log10(x);
                set(gca, 'ZScale', 'log');
            end
            mesh(f,z,x);

            title(plot_title_str);
            zlabel('Magnitude');
            xlabel('Frequency (Hz)');
            ylabel('Spatial Location (m)');
        end
    end


    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app)
            % reorganize the tab order.
            % there is no other way to do this.
            TabOrder = app.TabGroup.Children;
            cfgTab = TabOrder(1);
            eventSetupTab = TabOrder(2);
            simulationTab = TabOrder(3);
            traceAnalysisTab = TabOrder(4);
            eventLogTab = TabOrder(5);
            app.TabGroup.Children = [cfgTab; eventSetupTab; eventLogTab;
simulationTab; traceAnalysisTab];
        end

        % Value changed function: AcquisitionSpinner
```

```
        function AcquisitionSpinnerValueChanged(app, event)
            app.handlePlot();
        end

        % Button pushed function: ClearButton
        function ClearButtonPushed(app, event)
            clear all; close all;
        end

        % Button pushed function: FileBrowseButton
        function FileBrowseButtonButtonPushed(app, event)
            [filename,pathname,filtidx] = uigetfile('../*.mat');
            str = [pathname,filename];
            app.FileNameStr.Value = str;
        end

        % Value changed function: NonStationaryEventsSpinner
        function NonStationaryEventsSpinnerValueChanged(app, event)
            str = sprintf('Getting Event Info....\n');
            app.SimulationLog.Value(length(app.SimulationLog.Value)+1) =
{str};

            value = app.NonStationaryEventsSpinner.Value;

            if (app.nevents(1) ~= -1)
                if (value > length(app.nevents))
                    value = length(app.nevents);
                end

                strnames = {'Frequencies:    ', 'Strain:    ', 'Location:
', 'Length:    '};
                pnames = {'freq', 'ep', 'zloc', 'l'};
                N = length(pnames);

                for i=1:N
                    getval = get(app.nevents(value), pnames{i});
                    strval = mat2str(getval);
                    app.NonStationaryEventsSummaryLog.Value(i) =
{[strnames{i}, strval]};
                end
            end

            str = sprintf('Calculating PSD....\n');
            app.SimulationLog.Value(length(app.SimulationLog.Value)+1) =
{str};

            str = sprintf('Acq Time: %d\n', app.fiberObj.get_acq_time());
            app.SimulationLog.Value(length(app.SimulationLog.Value)+1) =
{str};

            N = length(app.dep(value,:));
            u = mean(app.dep(value,:));
            y = app.dep(value,:)-u; % remove DC value

            fmax = 1/app.fiberObj.get_acq_time();
```

```
            [psdx, f] = periodogram(y, [], N, fmax, 'one-sided');

            plot(app.NonStationaryEventsFreqPlot, f, psdx);
            xlabel(app.NonStationaryEventsFreqPlot, 'Frequency (Hz)');
            ylabel(app.NonStationaryEventsFreqPlot, 'PSD (W/Hz)');
            title(app.NonStationaryEventsFreqPlot, 'PSD of Event');

            app.last_x = f;
            app.last_y = psdx;
        end

        % Button pushed function: NonStationaryEventsUndockFigure
        function NonStationaryEventsUndockFigureButtonPushed(app, event)
            figure;
            plot(app.last_x, app.last_y);
            title('PSD of Applied Strain');
            ylabel('Power Density (W/Hz)');
            xlabel('Frequency (Hz)');
        end

        % Button pushed function: NonstationaryBrowseButton
        function NonstationaryBrowseButtonButtonPushed(app, event)
            [filename, pathname, filt] = uigetfile('../*.mat');
            str = [pathname, filename];
            app.NonstationaryFileStrField.Value = str;
        end

        % Selection changed function: NonstationaryButtonGroup
        function NonstationaryButtonGroupSelectionChanged(app, event)
            selectedButton = app.NonstationaryButtonGroup.SelectedObject;
            if (selectedButton == app.NonstationaryManualButton)
                app.NumNonstationaryEventsField.Enable = 'on';
                app.NonstationaryNumFreqField.Enable = 'on';
                app.NonstationaryMaxFreqField.Enable = 'on';
                app.NonstationaryMinFreqField.Enable = 'on';
                app.NonstationaryMaxStrainField.Enable = 'on';
                app.NonstationaryMinStrainField.Enable = 'on';
                app.NonstationaryMaxLengthField.Enable = 'on';
                app.NonstationaryMinLengthField.Enable = 'on';
            else
                app.NumNonstationaryEventsField.Enable = 'off';
                app.NonstationaryNumFreqField.Enable = 'off';
                app.NonstationaryMaxFreqField.Enable = 'off';
                app.NonstationaryMinFreqField.Enable = 'off';
                app.NonstationaryMaxStrainField.Enable = 'off';
                app.NonstationaryMinStrainField.Enable = 'off';
                app.NonstationaryMaxLengthField.Enable = 'off';
                app.NonstationaryMinLengthField.Enable = 'off';
            end
        end

        % Value changed function: NonstationaryEventCheckBox
        function NonstationaryEventCheckBoxValueChanged(app, event)
            value = app.NonstationaryEventCheckBox.Value;
            if (value == 1)
```

```
            app.NonstationaryFileButton.Enable = 'on';
            app.NonstationaryManualButton.Enable = 'on';
        else
            app.NonstationaryFileButton.Value = 1;
            app.NonstationaryFileButton.Enable = 'off';
            app.NonstationaryManualButton.Enable = 'off';
            app.NumNonstationaryEventsField.Enable = 'off';
            app.NonstationaryNumFreqField.Enable = 'off';
            app.NonstationaryMaxFreqField.Enable = 'off';
            app.NonstationaryMinFreqfield.Enable = 'off';
            app.NonstationaryMaxStrainField.Enable = 'off';
            app.NonstationaryMinStrainField.Enable = 'off';
            app.NonstationaryMaxLengthField.Enable = 'off';
            app.NonstationaryMinLengthField.Enable = 'off';
        end
    end

    % Selection changed function: ParameterSelection
    function ParameterSelectionSelectionChanged(app, event)
        selectedButton = app.ParameterSelection.SelectedObject;
        if (selectedButton == app.FileButton)
            app.FiberLengthField.Enable = 'off';
            app.NumScatterersField.Enable = 'off';
            app.FiberAttCoefField.Enable = 'off';
            app.IORField.Enable = 'off';
            app.PulseDurationField.Enable = 'off';
            app.SamplingRateField.Enable = 'off';
            app.LaserLambdaField.Enable = 'off';
            app.LaserLinewidthField.Enable = 'off';
            app.LaserDriftRateField.Enable = 'off';
            app.FrequencyShiftField.Enable = 'off';
            app.MaxScattererMagField.Enable = 'off';
            app.FileNameStr.Enable = 'on';
        else
            app.FiberLengthField.Enable = 'on';
            app.NumScatterersField.Enable = 'on';
            app.FiberAttCoefField.Enable = 'on';
            app.IORField.Enable = 'on';
            app.PulseDurationField.Enable = 'on';
            app.SamplingRateField.Enable = 'on';
            app.LaserLambdaField.Enable = 'on';
            app.LaserLinewidthField.Enable = 'on';
            app.LaserDriftRateField.Enable = 'on';
            app.FrequencyShiftField.Enable = 'on';
            app.MaxScattererMagField.Enable = 'on';
            app.FileNameStr.Enable = 'off';
        end
    end

    % Value changed function: PlotSelectionMenu
    function PlotSelectionMenuValueChanged(app, event)
        app.handlePlot();
    end

    % Button pushed function: SaveEventsButton
```

```
        function SaveEventsButtonButtonPushed(app, event)
            [filename, filepath] = uiputfile('../*.mat');
            filestr = strcat(filepath, filename);
            copyfile('vevents_tmp.mat', filestr);
        end

        % Button pushed function: SaveParamsButton
        function SaveParamsButtonButtonPushed(app, event)
            [filename, filepath] = uiputfile('../*.mat');
            filestr = strcat(filepath, filename);
            copyfile('params_tmp.mat', filestr);
        end

        % Button pushed function: SaveSeedButton
        function SaveSeedButtonButtonPushed(app, event)
            [filename, filepath] = uiputfile('../*.mat');
            filestr = strcat(filepath, filename);
            copyfile('seeds_tmp.mat', filestr);
        end

        % Button pushed function: SeedFileBrowseButton
        function SeedFileBrowseButtonButtonPushed(app, event)
            [filename, pathname, filt] = uigetfile('../*.mat');
            str = [pathname, filename];
            app.SeedFileStr.Value = str;
        end

        % Button pushed function: SimulationStartButton
        function SimulationStartButtonButtonPushed(app, event)
            % Step 1. Save the parameters to temporary files to be loaded
            % by the simulation function
            selectedButton = app.ParameterSelection.SelectedObject;
            if (selectedButton == app.ManualButton)
                params(1) = app.NumScatterersField.Value;
                params(2) = app.MaxScattererMagField.Value;
                params(3) = app.PulseDurationField.Value;
                params(4) = app.FrequencyShiftField.Value;
                params(5) = app.LaserLambdaField.Value;
                params(6) = app.LaserDriftRateField.Value;
                params(7) = app.LaserLinewidthField.Value;
                params(8) = app.FiberLengthField.Value;
                params(9) = app.FiberAttCoefField.Value;
                params(10) = app.SamplingRateField.Value;
                save('params_tmp', 'params');
            else
                % load the specified file
                load(app.FileNameStr.Value);
            end

            % check to see if the seeds are to be generated or loaded
            % from a file
            selectedButton = app.RandomNumberGroup.SelectedObject;
            if (selectedButton == app.GenerateSeedButton)
                seeds = ones(1,4);
                seeds = seeds.*-1;
```

```
            else     % load from file
                load(app.SeedFileStr.Value);
                seeds = new_seeds;
            end

            if (app.VibrationEventCheckBox.Value == 1)
                selectedButton = app.VibrationButtonGroup.SelectedObject;
                if (selectedButton == app.VibrationManualButton)
                    vevent_params(1) = 0;
                    vevent_params(2) = app.NumVibrationEventsField.Value;
                    vevent_params(3) = app.VibrationMaxFreqField.Value;
                    vevent_params(4) = app.VibrationMinFreqField.Value;
                    vevent_params(5) = app.VibrationNumFreqField.Value;
                    vevent_params(6) = app.VibrationMaxStrainField.Value;
                    vevent_params(7) = app.VibrationMinStrainField.Value;
                    vevent_params(8) = app.VibrationMaxLengthField.Value;
                    vevent_params(9) = app.VibrationMinLengthField.Value;
                    vevents_en = 1;        % set to 1, manual parameters,
new seed
                else
                    vevent_params(1) = app.VibrationFileStrField.Value;
                    vevents_en = 2;        % set to 2, load existing
events
                end
            else
                vevents_en = 0;
                vevent_params = -1;
            end

            if (app.NonstationaryEventCheckBox.Value == 1)
                selectedButton =
app.NonstationaryButtonGroup.SelectedObject;
                if (selectedButton == app.NonstationaryManualButton)
                    nevent_params(1) = 0;
                    nevent_params(2) =
app.NumNonstationaryEventsField.Value;
                    nevent_params(3) =
app.NonstationaryMaxFreqField.Value;
                    nevent_params(4) =
app.NonstationaryMinFreqField.Value;
                    nevent_params(5) =
app.NonstationaryNumFreqField.Value;
                    nevent_params(6) =
app.NonstationaryMaxStrainField.Value;
                    nevent_params(7) =
app.NonstationaryMinStrainField.Value;
                    nevent_params(8) =
app.NonstationaryMaxLengthField.Value;
                    nevent_params(9) =
app.NonstationaryMinLengthField.Value;
                    nevents_en = 1;
                else
                    nevent_params(1) = app.NonstaionaryFileStrField.Value;
                    nevents_en = 2;
                end
```

```
            else
                nevents_en = 0;
                nevent_params = -1;
            end

            % when this button is pushed we start the simulation
            [app.fiberObj, app.phi_terms, app.I_terms, app.dep, app.np,
app.nd, app.z_axis, app.vevents, app.nevents] = ...
                run_fibersim(app.NumTracesField.Value, vevents_en,
nevents_en, vevent_params, nevent_params, params, seeds,
app.SimulationLog);

            % after the simulatin is complete we allow the user to save
the parameters, events and seeds
            app.SaveParamsButton.Enable = 'on';
            app.SaveEventsButton.Enable = 'on';
            app.SaveSeedButton.Enable = 'on';
        end

        % Button pushed function: SlowTimeFFTButton
        function SlowTimeFFTButtonPushed(app, event)
            app.handle3DPlot();
        end

        % Value changed function: StationaryEventsSpinner
        function StationaryEventsSpinnerValueChanged(app, event)
            str = sprintf('Getting Event Info....\n');
            app.SimulationLog.Value(length(app.SimulationLog.Value)+1) =
{str};

            value = app.StationaryEventsSpinner.Value;

            if (app.vevents(1) ~= -1)
                if (value > length(app.vevents))
                    value = length(app.vevents);
                end

                strnames = {'Frequencies:    ', 'Strain:    ', 'Location:
', 'Length:    '};
                pnames = {'freq', 'ep', 'zloc', 'l'};
                N = length(pnames);

                for i=1:N
                    getval = get(app.vevents(value), pnames{i});
                    strval = mat2str(getval);
                    app.StationaryEventsSummaryLog.Value(i) =
{[strnames{i}, strval]};
                end
            end

            str = sprintf('Calculating PSD....\n');
            app.SimulationLog.Value(length(app.SimulationLog.Value)+1) =
{str};

            str = sprintf('Acq Time: %d\n', app.fiberObj.get_acq_time());
```

```
            app.SimulationLog.Value(length(app.SimulationLog.Value)+1) =
{str};

            N = length(app.dep(value,:));
            u = mean(app.dep(value,:));
            y = app.dep(value,:)-u; % remove DC value

            fmax = 1/app.fiberObj.get_acq_time();
            [psdx, f] = periodogram(y, [], N, fmax, 'one-sided');

            plot(app.StationaryEventsFreqPlot, f, psdx);
            xlabel(app.StationaryEventsFreqPlot, 'Frequency (Hz)');
            ylabel(app.StationaryEventsFreqPlot, 'PSD (W/Hz)');
            title(app.StationaryEventsFreqPlot, 'PSD of Event');

            app.last_x = f;
            app.last_y = psdx;
        end

        % Button pushed function: StationaryEventsUndockFigure
        function StationaryEventsUndockFigureButtonPushed(app, event)
            figure;
            plot(app.last_x, app.last_y); grid on;
            title('PSD of Applied Strain');
            ylabel('Power Density (W/Hz)');
            xlabel('Frequency (Hz)');
        end

        % Button pushed function: UndockFigureButton
        function UndockFigureButtonPushed(app, event)
            figure;
            plot(app.last_x, app.last_y); grid on;
            title('undocked figure');
        end

        % Button pushed function: VibrationBrowseButton
        function VibrationBrowseButtonButtonPushed(app, event)
            [filename, pathname, filt] = uigetfile('../*.mat');
            str = [pathname, filename];
            app.VibrationFileStrField.Value = str;
        end

        % Selection changed function: VibrationButtonGroup
        function VibrationButtonGroupSelectionChanged(app, event)
            selectedButton = app.VibrationButtonGroup.SelectedObject;
            if (selectedButton == app.VibrationManualButton)
                app.NumVibrationEventsField.Enable = 'on';
                app.VibrationMaxFreqField.Enable = 'on';
                app.VibrationMinFreqField.Enable = 'on';
                app.VibrationNumFreqField.Enable = 'on';
                app.VibrationMaxStrainField.Enable = 'on';
                app.VibrationMinStrainField.Enable = 'on';
                app.VibrationMaxLengthField.Enable = 'on';
                app.VibrationMinLengthField.Enable = 'on';
            else
```

```
                app.NumVibrationEventsField.Enable = 'off';
                app.VibrationMaxFreqField.Enable = 'off';
                app.VibrationMinFreqField.Enable = 'off';
                app.VibrationNumFreqField.Enable = 'off';
                app.VibrationMaxStrainField.Enable = 'off';
                app.VibrationMinStrainField.Enable = 'off';
                app.VibrationMaxLengthField.Enable = 'off';
                app.VibrationMinLengthField.Enable = 'off';
            end
        end

        % Value changed function: VibrationEventCheckBox
        function VibrationEventCheckBoxValueChanged(app, event)
            value = app.VibrationEventCheckBox.Value;
            if (value == 1)
                app.VibrationFileButton.Enable = 'on';
                app.VibrationManualButton.Enable = 'on';
            else
                app.VibrationFileButton.Value = 1;
                app.VibrationFileButton.Enable = 'off';
                app.VibrationManualButton.Enable = 'off';
                app.NumVibrationEventsField.Enable = 'off';
                app.VibrationMaxFreqField.Enable = 'off';
                app.VibrationMinFreqField.Enable = 'off';
                app.VibrationNumFreqField.Enable = 'off';
                app.VibrationMaxStrainField.Enable = 'off';
                app.VibrationMinStrainField.Enable = 'off';
                app.VibrationMaxLengthField.Enable = 'off';
                app.VibrationMinLengthField.Enable = 'off';
            end
        end

        % Button pushed function: ViewEventTableButton
        function ViewEventTableButtonButtonPushed(app, event)
            f = figure;
            load('vevents_tmp.mat', 'vevents');
            t = uitable(f, 'Data', randi(100,10,3), 'Position', [20 20 262
204]);
        end
    end

    % App initialization and construction
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create PhaseOTDRSimulatorUIFigure
            app.PhaseOTDRSimulatorUIFigure = uifigure;
            app.PhaseOTDRSimulatorUIFigure.Position = [101 101 722 562];
            app.PhaseOTDRSimulatorUIFigure.Name = 'Phase-OTDR Simulator';

            % Create TabGroup
            app.TabGroup = uitabgroup(app.PhaseOTDRSimulatorUIFigure);
            app.TabGroup.Position = [1 1 720 562];
```

```
            % Create CfgTab
            app.CfgTab = uitab(app.TabGroup);
            app.CfgTab.Title = 'Configuration';

            % Create LabelDropDown
            app.LabelDropDown = uilabel(app.CfgTab);
            app.LabelDropDown.HorizontalAlignment = 'right';
            app.LabelDropDown.Position = [17.03125 478 82 15];
            app.LabelDropDown.Text = 'Detection Type';

            % Create DetectionTypeDropDown
            app.DetectionTypeDropDown = uidropdown(app.CfgTab);
            app.DetectionTypeDropDown.Items = {'Coherent I/Q', 'Coherent
Heterodyne', 'Non-coherent Direct Detection'};
            app.DetectionTypeDropDown.Position = [114.03125 474 188 22];
            app.DetectionTypeDropDown.Value = 'Coherent I/Q';

            % Create Label
            app.Label = uilabel(app.CfgTab);
            app.Label.FontSize = 16;
            app.Label.FontWeight = 'bold';
            app.Label.FontColor = [1 0 0];
            app.Label.Position = [17 505 99 21];
            app.Label.Text = 'System Type';

            % Create ParameterSelection
            app.ParameterSelection = uibuttongroup(app.CfgTab);
            app.ParameterSelection.SelectionChangedFcn =
createCallbackFcn(app, @ParameterSelectionSelectionChanged, true);
            app.ParameterSelection.Title = 'Parameter Selection';
            app.ParameterSelection.FontWeight = 'bold';
            app.ParameterSelection.FontSize = 18;
            app.ParameterSelection.Position = [17 44 681 303];

            % Create FileButton
            app.FileButton = uiradiobutton(app.ParameterSelection);
            app.FileButton.Text = 'File';
            app.FileButton.Position = [11 249 37.671875 16];
            app.FileButton.Value = true;

            % Create ManualButton
            app.ManualButton = uiradiobutton(app.ParameterSelection);
            app.ManualButton.Text = 'Manual';
            app.ManualButton.Position = [11 227 58.359375 16];

            % Create LabelEditField
            app.LabelEditField = uilabel(app.ParameterSelection);
            app.LabelEditField.HorizontalAlignment = 'right';
            app.LabelEditField.Position = [238.03125 250 20 15];
            app.LabelEditField.Text = '';

            % Create FileNameStr
            app.FileNameStr = uieditfield(app.ParameterSelection, 'text');
            app.FileNameStr.Position = [273.03125 246 156 22];
```

```
            % Create FileBrowseButton
            app.FileBrowseButton = uibutton(app.ParameterSelection,
'push');
            app.FileBrowseButton.ButtonPushedFcn = createCallbackFcn(app,
@FileBrowseButtonButtonPushed, true);
            app.FileBrowseButton.Position = [161 246 60 22];
            app.FileBrowseButton.Text = 'Browse';

            % Create Label2
            app.Label2 = uilabel(app.ParameterSelection);
            app.Label2.FontSize = 16;
            app.Label2.FontWeight = 'bold';
            app.Label2.FontColor = [1 0 0];
            app.Label2.Position = [11 202 233 21];
            app.Label2.Text = 'Simulation Parameter Settings';

            % Create LabelNumericEditField
            app.LabelNumericEditField = uilabel(app.ParameterSelection);
            app.LabelNumericEditField.HorizontalAlignment = 'right';
            app.LabelNumericEditField.Position = [15.03125 171 68 15];
            app.LabelNumericEditField.Text = 'Fiber Length';

            % Create FiberLengthField
            app.FiberLengthField = uieditfield(app.ParameterSelection,
'numeric');
            app.FiberLengthField.Limits = [1 Inf];
            app.FiberLengthField.Enable = 'off';
            app.FiberLengthField.Position = [161.03125 167 60 22];
            app.FiberLengthField.Value = 1000;

            % Create Label3
            app.Label3 = uilabel(app.ParameterSelection);
            app.Label3.FontAngle = 'italic';
            app.Label3.Position = [229 171 37 15];
            app.Label3.Text = 'meters';

            % Create LabelNumericEditField2
            app.LabelNumericEditField2 = uilabel(app.ParameterSelection);
            app.LabelNumericEditField2.HorizontalAlignment = 'right';
            app.LabelNumericEditField2.Position = [15 138 114 15];
            app.LabelNumericEditField2.Text = 'Number of scatterers';

            % Create NumScatterersField
            app.NumScatterersField = uieditfield(app.ParameterSelection,
'numeric');
            app.NumScatterersField.Limits = [1 Inf];
            app.NumScatterersField.ValueDisplayFormat = '%.0f';
            app.NumScatterersField.Enable = 'off';
            app.NumScatterersField.Position = [161.03125 134 60 22];
            app.NumScatterersField.Value = 10000;

            % Create LabelNumericEditField7
            app.LabelNumericEditField7 = uilabel(app.ParameterSelection);
            app.LabelNumericEditField7.HorizontalAlignment = 'right';
```

```
            app.LabelNumericEditField7.Position = [15.03125 105 91 15];
            app.LabelNumericEditField7.Text = 'Attenuation Coef';

            % Create FiberAttCoefField
            app.FiberAttCoefField = uieditfield(app.ParameterSelection,
'numeric');
            app.FiberAttCoefField.Limits = [0 Inf];
            app.FiberAttCoefField.Enable = 'off';
            app.FiberAttCoefField.Position = [161.03125 101 60 22];
            app.FiberAttCoefField.Value = 0.2;

            % Create Label9
            app.Label9 = uilabel(app.ParameterSelection);
            app.Label9.FontAngle = 'italic';
            app.Label9.Position = [229 105 36 15];
            app.Label9.Text = 'dB/km';

            % Create LabelNumericEditField8
            app.LabelNumericEditField8 = uilabel(app.ParameterSelection);
            app.LabelNumericEditField8.HorizontalAlignment = 'right';
            app.LabelNumericEditField8.Position = [15.03125 72 87 15];
            app.LabelNumericEditField8.Text = 'Refractive Index';

            % Create IORField
            app.IORField = uieditfield(app.ParameterSelection, 'numeric');
            app.IORField.Limits = [1 Inf];
            app.IORField.Enable = 'off';
            app.IORField.Position = [161.03125 68 60 22];
            app.IORField.Value = 1.47;

            % Create Label4
            app.Label4 = uilabel(app.ParameterSelection);
            app.Label4.FontSize = 16;
            app.Label4.FontWeight = 'bold';
            app.Label4.FontColor = [1 0 0];
            app.Label4.Position = [329 202 124 21];
            app.Label4.Text = 'System Settings';

            % Create LabelNumericEditField3
            app.LabelNumericEditField3 = uilabel(app.ParameterSelection);
            app.LabelNumericEditField3.HorizontalAlignment = 'right';
            app.LabelNumericEditField3.Position = [330.03125 171 79 15];
            app.LabelNumericEditField3.Text = 'Pulse Duration';

            % Create PulseDurationField
            app.PulseDurationField = uieditfield(app.ParameterSelection,
'numeric');
            app.PulseDurationField.Limits = [0 Inf];
            app.PulseDurationField.Enable = 'off';
            app.PulseDurationField.Position = [440.03125 167 63 22];
            app.PulseDurationField.Value = 1e-08;

            % Create Label5
            app.Label5 = uilabel(app.ParameterSelection);
            app.Label5.FontAngle = 'italic';
```

```
            app.Label5.Position = [516 171 46 15];
            app.Label5.Text = 'seconds';

            % Create LabelNumericEditField4
            app.LabelNumericEditField4 = uilabel(app.ParameterSelection);
            app.LabelNumericEditField4.HorizontalAlignment = 'right';
            app.LabelNumericEditField4.Position = [330.03125 139 79 15];
            app.LabelNumericEditField4.Text = 'Sampling Rate';

            % Create SamplingRateField
            app.SamplingRateField = uieditfield(app.ParameterSelection,
'numeric');
            app.SamplingRateField.Limits = [0 Inf];
            app.SamplingRateField.Enable = 'off';
            app.SamplingRateField.Position = [440.03125 135 63 22];
            app.SamplingRateField.Value = 1e-08;

            % Create Label6
            app.Label6 = uilabel(app.ParameterSelection);
            app.Label6.FontAngle = 'italic';
            app.Label6.Position = [516 138 24 15];
            app.Label6.Text = 'SPS';

            % Create LabelNumericEditField6
            app.LabelNumericEditField6 = uilabel(app.ParameterSelection);
            app.LabelNumericEditField6.HorizontalAlignment = 'right';
            app.LabelNumericEditField6.Position = [330.03125 107 97 15];
            app.LabelNumericEditField6.Text = 'Laser Wavelength';

            % Create LaserLambdaField
            app.LaserLambdaField = uieditfield(app.ParameterSelection,
'numeric');
            app.LaserLambdaField.Limits = [0 Inf];
            app.LaserLambdaField.Enable = 'off';
            app.LaserLambdaField.Position = [440.03125 103 63 22];
            app.LaserLambdaField.Value = 1.55e-06;

            % Create Label8
            app.Label8 = uilabel(app.ParameterSelection);
            app.Label8.FontAngle = 'italic';
            app.Label8.Position = [516 107 37 15];
            app.Label8.Text = 'meters';

            % Create LabelNumericEditField5
            app.LabelNumericEditField5 = uilabel(app.ParameterSelection);
            app.LabelNumericEditField5.HorizontalAlignment = 'right';
            app.LabelNumericEditField5.Position = [330.03125 75 85 15];
            app.LabelNumericEditField5.Text = 'Laser Linewidth';

            % Create LaserLinewidthField
            app.LaserLinewidthField = uieditfield(app.ParameterSelection,
'numeric');
            app.LaserLinewidthField.Limits = [0 Inf];
            app.LaserLinewidthField.Enable = 'off';
            app.LaserLinewidthField.Position = [440.03125 71 63 22];
```

```
            app.LaserLinewidthField.Value = 1000;

            % Create Label7
            app.Label7 = uilabel(app.ParameterSelection);
            app.Label7.FontAngle = 'italic';
            app.Label7.Position = [516 75 20 15];
            app.Label7.Text = 'Hz';

            % Create LabelNumericEditField9
            app.LabelNumericEditField9 = uilabel(app.ParameterSelection);
            app.LabelNumericEditField9.HorizontalAlignment = 'right';
            app.LabelNumericEditField9.Position = [330.03125 43 84 15];
            app.LabelNumericEditField9.Text = 'Laser Drift Rate';

            % Create LaserDriftRateField
            app.LaserDriftRateField = uieditfield(app.ParameterSelection,
'numeric');
            app.LaserDriftRateField.Enable = 'off';
            app.LaserDriftRateField.Position = [440.03125 39 63 22];

            % Create Label10
            app.Label10 = uilabel(app.ParameterSelection);
            app.Label10.FontAngle = 'italic';
            app.Label10.Position = [516 43 24 15];
            app.Label10.Text = 'Hz/s';

            % Create LabelNumericEditField10
            app.LabelNumericEditField10 = uilabel(app.ParameterSelection);
            app.LabelNumericEditField10.HorizontalAlignment = 'right';
            app.LabelNumericEditField10.Position = [330.03125 11 85 15];
            app.LabelNumericEditField10.Text = 'Frequency Shift';

            % Create FrequencyShiftField
            app.FrequencyShiftField = uieditfield(app.ParameterSelection,
'numeric');
            app.FrequencyShiftField.Limits = [0 Inf];
            app.FrequencyShiftField.Enable = 'off';
            app.FrequencyShiftField.Position = [440.03125 7 63 22];
            app.FrequencyShiftField.Value = 48000000;

            % Create Label11
            app.Label11 = uilabel(app.ParameterSelection);
            app.Label11.FontAngle = 'italic';
            app.Label11.Position = [516 11 20 15];
            app.Label11.Text = 'Hz';

            % Create LabelNumericEditField11
            app.LabelNumericEditField11 = uilabel(app.ParameterSelection);
            app.LabelNumericEditField11.HorizontalAlignment = 'right';
            app.LabelNumericEditField11.Position = [15.03125 40 131 15];
            app.LabelNumericEditField11.Text = 'Max scatterer amplitude';

            % Create MaxScattererMagField
            app.MaxScattererMagField = uieditfield(app.ParameterSelection,
'numeric');
```

```
            app.MaxScattererMagField.Limits = [0 1];
            app.MaxScattererMagField.Enable = 'off';
            app.MaxScattererMagField.Position = [161.03125 36 60 22];
            app.MaxScattererMagField.Value = 0.001;

            % Create RandomNumberGroup
            app.RandomNumberGroup = uibuttongroup(app.CfgTab);
            app.RandomNumberGroup.Title = 'Random Number Settings';
            app.RandomNumberGroup.Position = [17 357 681 106];

            % Create SeedFileButton
            app.SeedFileButton = uiradiobutton(app.RandomNumberGroup);
            app.SeedFileButton.Text = 'Seed File';
            app.SeedFileButton.Position = [11 60 68.78125 16];
            app.SeedFileButton.Value = true;

            % Create GenerateSeedButton
            app.GenerateSeedButton = uiradiobutton(app.RandomNumberGroup);
            app.GenerateSeedButton.Text = 'Generate New Seed';
            app.GenerateSeedButton.Position = [11 22 127 16];

            % Create LabelEditField2
            app.LabelEditField2 = uilabel(app.RandomNumberGroup);
            app.LabelEditField2.HorizontalAlignment = 'right';
            app.LabelEditField2.Position = [238.03125 61 20 15];
            app.LabelEditField2.Text = '';

            % Create SeedFileStr
            app.SeedFileStr = uieditfield(app.RandomNumberGroup, 'text');
            app.SeedFileStr.Position = [273.03125 57 156 22];

            % Create SeedFileBrowseButton
            app.SeedFileBrowseButton = uibutton(app.RandomNumberGroup,
'push');
            app.SeedFileBrowseButton.ButtonPushedFcn =
createCallbackFcn(app, @SeedFileBrowseButtonButtonPushed, true);
            app.SeedFileBrowseButton.Position = [161 57 60 22];
            app.SeedFileBrowseButton.Text = 'Browse';

            % Create EventSetupTab
            app.EventSetupTab = uitab(app.TabGroup);
            app.EventSetupTab.Title = 'Event Setup';

            % Create Panel
            app.Panel = uipanel(app.EventSetupTab);
            app.Panel.ForegroundColor = [1 0 0];
            app.Panel.Title = 'Event Selection';
            app.Panel.FontWeight = 'bold';
            app.Panel.FontSize = 16;
            app.Panel.Position = [1 20 721 521];

            % Create VibrationEventCheckBox
            app.VibrationEventCheckBox = uicheckbox(app.Panel);
            app.VibrationEventCheckBox.ValueChangedFcn =
createCallbackFcn(app, @VibrationEventCheckBoxValueChanged, true);
```

```
            app.VibrationEventCheckBox.Text = 'Vibration Events';
            app.VibrationEventCheckBox.FontWeight = 'bold';
            app.VibrationEventCheckBox.Position = [25 472 112.34375 16];

            % Create NonstationaryEventCheckBox
            app.NonstationaryEventCheckBox = uicheckbox(app.Panel);
            app.NonstationaryEventCheckBox.ValueChangedFcn =
createCallbackFcn(app, @NonstationaryEventCheckBoxValueChanged, true);
            app.NonstationaryEventCheckBox.Text = 'Nonstationary Events';
            app.NonstationaryEventCheckBox.FontWeight = 'bold';
            app.NonstationaryEventCheckBox.Position = [25 239 141.484375
16];

            % Create VibrationButtonGroup
            app.VibrationButtonGroup = uibuttongroup(app.Panel);
            app.VibrationButtonGroup.SelectionChangedFcn =
createCallbackFcn(app, @VibrationButtonGroupSelectionChanged, true);
            app.VibrationButtonGroup.Title = 'Vibration Configuration';
            app.VibrationButtonGroup.Position = [25 272 539 190];

            % Create VibrationFileButton
            app.VibrationFileButton =
uiradiobutton(app.VibrationButtonGroup);
            app.VibrationFileButton.Enable = 'off';
            app.VibrationFileButton.Text = 'Event File';
            app.VibrationFileButton.Position = [11 143 71.234375 21];
            app.VibrationFileButton.Value = true;

            % Create VibrationManualButton
            app.VibrationManualButton =
uiradiobutton(app.VibrationButtonGroup);
            app.VibrationManualButton.Enable = 'off';
            app.VibrationManualButton.Text = 'Manual';
            app.VibrationManualButton.Position = [11 123 58.359375 15];

            % Create LabelNumericEditField13
            app.LabelNumericEditField13 =
uilabel(app.VibrationButtonGroup);
            app.LabelNumericEditField13.HorizontalAlignment = 'right';
            app.LabelNumericEditField13.Position = [11.03125 96 97 15];
            app.LabelNumericEditField13.Text = 'Number of Events';

            % Create NumVibrationEventsField
            app.NumVibrationEventsField =
uieditfield(app.VibrationButtonGroup, 'numeric');
            app.NumVibrationEventsField.Limits = [1 Inf];
            app.NumVibrationEventsField.Enable = 'off';
            app.NumVibrationEventsField.Position = [123.03125 92 40 22];
            app.NumVibrationEventsField.Value = 1;

            % Create LabelNumericEditField14
            app.LabelNumericEditField14 =
uilabel(app.VibrationButtonGroup);
            app.LabelNumericEditField14.HorizontalAlignment = 'right';
            app.LabelNumericEditField14.Position = [11.03125 67 84 15];
```

```
            app.LabelNumericEditField14.Text = 'Max Frequency';

            % Create VibrationMaxFreqField
            app.VibrationMaxFreqField =
uieditfield(app.VibrationButtonGroup, 'numeric');
            app.VibrationMaxFreqField.Limits = [0 Inf];
            app.VibrationMaxFreqField.Enable = 'off';
            app.VibrationMaxFreqField.Position = [123.03125 63 87 22];
            app.VibrationMaxFreqField.Value = 10000;

            % Create LabelNumericEditField16
            app.LabelNumericEditField16 =
uilabel(app.VibrationButtonGroup);
            app.LabelNumericEditField16.HorizontalAlignment = 'right';
            app.LabelNumericEditField16.Position = [11.03125 38 58 15];
            app.LabelNumericEditField16.Text = 'Max Strain';

            % Create VibrationMaxStrainField
            app.VibrationMaxStrainField =
uieditfield(app.VibrationButtonGroup, 'numeric');
            app.VibrationMaxStrainField.Limits = [0 0.0065];
            app.VibrationMaxStrainField.Enable = 'off';
            app.VibrationMaxStrainField.Position = [123.03125 34 87 22];
            app.VibrationMaxStrainField.Value = 1e-06;

            % Create LabelNumericEditField20
            app.LabelNumericEditField20 =
uilabel(app.VibrationButtonGroup);
            app.LabelNumericEditField20.HorizontalAlignment = 'right';
            app.LabelNumericEditField20.Position = [11.03125 9 64 15];
            app.LabelNumericEditField20.Text = 'Max Length';

            % Create VibrationMaxLengthField
            app.VibrationMaxLengthField =
uieditfield(app.VibrationButtonGroup, 'numeric');
            app.VibrationMaxLengthField.Limits = [0 Inf];
            app.VibrationMaxLengthField.Enable = 'off';
            app.VibrationMaxLengthField.Position = [123.03125 5 87 22];
            app.VibrationMaxLengthField.Value = 1;

            % Create Label14
            app.Label14 = uilabel(app.VibrationButtonGroup);
            app.Label14.Position = [225 67 20 15];
            app.Label14.Text = 'Hz';

            % Create Label16
            app.Label16 = uilabel(app.VibrationButtonGroup);
            app.Label16.Position = [225 5 37 15];
            app.Label16.Text = 'meters';

            % Create LabelNumericEditField15
            app.LabelNumericEditField15 =
uilabel(app.VibrationButtonGroup);
            app.LabelNumericEditField15.HorizontalAlignment = 'right';
            app.LabelNumericEditField15.Position = [288.03125 67 80 15];
```

```
            app.LabelNumericEditField15.Text = 'Min Frequency';

            % Create VibrationMinFreqField
            app.VibrationMinFreqField =
uieditfield(app.VibrationButtonGroup, 'numeric');
            app.VibrationMinFreqField.Limits = [0 Inf];
            app.VibrationMinFreqField.Enable = 'off';
            app.VibrationMinFreqField.Position = [394.03125 63 89 22];
            app.VibrationMinFreqField.Value = 100;

            % Create LabelNumericEditField17
            app.LabelNumericEditField17 =
uilabel(app.VibrationButtonGroup);
            app.LabelNumericEditField17.HorizontalAlignment = 'right';
            app.LabelNumericEditField17.Position = [288.03125 40 55 15];
            app.LabelNumericEditField17.Text = 'Min Strain';

            % Create VibrationMinStrainField
            app.VibrationMinStrainField =
uieditfield(app.VibrationButtonGroup, 'numeric');
            app.VibrationMinStrainField.Limits = [0 1e-05];
            app.VibrationMinStrainField.Enable = 'off';
            app.VibrationMinStrainField.Position = [394.03125 36 89 22];
            app.VibrationMinStrainField.Value = 1e-08;

            % Create Label17
            app.Label17 = uilabel(app.VibrationButtonGroup);
            app.Label17.HorizontalAlignment = 'right';
            app.Label17.Position = [288.03125 9 61 15];
            app.Label17.Text = 'Min Length';

            % Create VibrationMinLengthField
            app.VibrationMinLengthField =
uieditfield(app.VibrationButtonGroup, 'numeric');
            app.VibrationMinLengthField.Limits = [0 Inf];
            app.VibrationMinLengthField.Enable = 'off';
            app.VibrationMinLengthField.Position = [394.03125 5 89 22];
            app.VibrationMinLengthField.Value = 0.5;

            % Create Label18
            app.Label18 = uilabel(app.VibrationButtonGroup);
            app.Label18.Position = [494 9 37 15];
            app.Label18.Text = 'meters';

            % Create Label15
            app.Label15 = uilabel(app.VibrationButtonGroup);
            app.Label15.Position = [494 67 20 15];
            app.Label15.Text = 'Hz';

            % Create LabelEditField4
            app.LabelEditField4 = uilabel(app.VibrationButtonGroup);
            app.LabelEditField4.HorizontalAlignment = 'right';
            app.LabelEditField4.Position = [234.03125 142 20 15];
            app.LabelEditField4.Text = '';
```

```
            % Create VibrationFileStrField
            app.VibrationFileStrField =
uieditfield(app.VibrationButtonGroup, 'text');
            app.VibrationFileStrField.Position = [209.03125 142 160 22];

            % Create VibrationBrowseButton
            app.VibrationBrowseButton = uibutton(app.VibrationButtonGroup,
'push');
            app.VibrationBrowseButton.ButtonPushedFcn =
createCallbackFcn(app, @VibrationBrowseButtonButtonPushed, true);
            app.VibrationBrowseButton.Position = [115 142 56 22];
            app.VibrationBrowseButton.Text = 'Browse';

            % Create LabelNumericEditField21
            app.LabelNumericEditField21 =
uilabel(app.VibrationButtonGroup);
            app.LabelNumericEditField21.HorizontalAlignment = 'right';
            app.LabelNumericEditField21.Position = [288.03125 96 127 15];
            app.LabelNumericEditField21.Text = 'Number of Frequencies';

            % Create VibrationNumFreqField
            app.VibrationNumFreqField =
uieditfield(app.VibrationButtonGroup, 'numeric');
            app.VibrationNumFreqField.Enable = 'off';
            app.VibrationNumFreqField.Position = [425.03125 92 58 22];

            % Create NonstationaryButtonGroup
            app.NonstationaryButtonGroup = uibuttongroup(app.Panel);
            app.NonstationaryButtonGroup.SelectionChangedFcn =
createCallbackFcn(app, @NonstationaryButtonGroupSelectionChanged, true);
            app.NonstationaryButtonGroup.Title = 'Nonstationary
Configuration';
            app.NonstationaryButtonGroup.Position = [25 21 539 202];

            % Create NonstationaryFileButton
            app.NonstationaryFileButton =
uiradiobutton(app.NonstationaryButtonGroup);
            app.NonstationaryFileButton.Enable = 'off';
            app.NonstationaryFileButton.Text = 'Event File';
            app.NonstationaryFileButton.Position = [11 159 71.234375 16];
            app.NonstationaryFileButton.Value = true;

            % Create NonstationaryManualButton
            app.NonstationaryManualButton =
uiradiobutton(app.NonstationaryButtonGroup);
            app.NonstationaryManualButton.Enable = 'off';
            app.NonstationaryManualButton.Text = 'Manual';
            app.NonstationaryManualButton.Position = [11 132 58.359375
16];

            % Create LabelNumericEditField18
            app.LabelNumericEditField18 =
uilabel(app.NonstationaryButtonGroup);
            app.LabelNumericEditField18.HorizontalAlignment = 'right';
            app.LabelNumericEditField18.Position = [11.03125 96 97 15];
```

```
            app.LabelNumericEditField18.Text = 'Number of Events';

            % Create NumNonstationaryEventsField
            app.NumNonstationaryEventsField =
uieditfield(app.NonstationaryButtonGroup, 'numeric');
            app.NumNonstationaryEventsField.Enable = 'off';
            app.NumNonstationaryEventsField.Position = [123.03125 92 40
22];

            % Create Label13
            app.Label13 = uilabel(app.NonstationaryButtonGroup);
            app.Label13.HorizontalAlignment = 'right';
            app.Label13.Position = [11.03125 38 58 15];
            app.Label13.Text = 'Max Strain';

            % Create NonstationaryMaxStrainField
            app.NonstationaryMaxStrainField =
uieditfield(app.NonstationaryButtonGroup, 'numeric');
            app.NonstationaryMaxStrainField.Limits = [0 Inf];
            app.NonstationaryMaxStrainField.Enable = 'off';
            app.NonstationaryMaxStrainField.Position = [123.03125 34 87
22];
            app.NonstationaryMaxStrainField.Value = 1e-06;

            % Create Label20
            app.Label20 = uilabel(app.NonstationaryButtonGroup);
            app.Label20.HorizontalAlignment = 'right';
            app.Label20.Position = [11.03125 10 64 15];
            app.Label20.Text = 'Max Length';

            % Create NonstationaryMaxLengthField
            app.NonstationaryMaxLengthField =
uieditfield(app.NonstationaryButtonGroup, 'numeric');
            app.NonstationaryMaxLengthField.Limits = [0 Inf];
            app.NonstationaryMaxLengthField.Enable = 'off';
            app.NonstationaryMaxLengthField.Position = [123.03125 6 87
22];
            app.NonstationaryMaxLengthField.Value = 10;

            % Create LabelNumericEditField19
            app.LabelNumericEditField19 =
uilabel(app.NonstationaryButtonGroup);
            app.LabelNumericEditField19.HorizontalAlignment = 'right';
            app.LabelNumericEditField19.Position = [288.03125 38 55 15];
            app.LabelNumericEditField19.Text = 'Min Strain';

            % Create NonstationaryMinStrainField
            app.NonstationaryMinStrainField =
uieditfield(app.NonstationaryButtonGroup, 'numeric');
            app.NonstationaryMinStrainField.Limits = [0 Inf];
            app.NonstationaryMinStrainField.Enable = 'off';
            app.NonstationaryMinStrainField.Position = [394.03125 34 89
22];
            app.NonstationaryMinStrainField.Value = 1e-08;
```

```
            % Create Label21
            app.Label21 = uilabel(app.NonstationaryButtonGroup);
            app.Label21.HorizontalAlignment = 'right';
            app.Label21.Position = [288.03125 10 61 15];
            app.Label21.Text = 'Min Length';

            % Create NonstationaryMinLengthField
            app.NonstationaryMinLengthField =
uieditfield(app.NonstationaryButtonGroup, 'numeric');
            app.NonstationaryMinLengthField.Limits = [0 Inf];
            app.NonstationaryMinLengthField.Enable = 'off';
            app.NonstationaryMinLengthField.Position = [394.03125 6 89
22];
            app.NonstationaryMinLengthField.Value = 1;

            % Create Label19
            app.Label19 = uilabel(app.NonstationaryButtonGroup);
            app.Label19.Position = [494 57 37 15];
            app.Label19.Text = 'meters';

            % Create LabelEditField5
            app.LabelEditField5 = uilabel(app.NonstationaryButtonGroup);
            app.LabelEditField5.HorizontalAlignment = 'right';
            app.LabelEditField5.Position = [220.03125 159 20 15];
            app.LabelEditField5.Text = '';

            % Create NonstationaryFileStrField
            app.NonstationaryFileStrField =
uieditfield(app.NonstationaryButtonGroup, 'text');
            app.NonstationaryFileStrField.Position = [189.03125 155 160
22];

            % Create NonstationaryBrowseButton
            app.NonstationaryBrowseButton =
uibutton(app.NonstationaryButtonGroup, 'push');
            app.NonstationaryBrowseButton.ButtonPushedFcn =
createCallbackFcn(app, @NonstationaryBrowseButtonButtonPushed, true);
            app.NonstationaryBrowseButton.Position = [126 155 53 22];
            app.NonstationaryBrowseButton.Text = 'Browse';

            % Create Label27
            app.Label27 = uilabel(app.NonstationaryButtonGroup);
            app.Label27.HorizontalAlignment = 'right';
            app.Label27.Position = [288 96 127 15];
            app.Label27.Text = 'Number of Frequencies';

            % Create NonstationaryNumFreqField
            app.NonstationaryNumFreqField =
uieditfield(app.NonstationaryButtonGroup, 'numeric');
            app.NonstationaryNumFreqField.Enable = 'off';
            app.NonstationaryNumFreqField.Position = [425 92 58 22];

            % Create Label28
            app.Label28 = uilabel(app.NonstationaryButtonGroup);
            app.Label28.HorizontalAlignment = 'right';
```

```
            app.Label28.Position = [11.03125 67 84 15];
            app.Label28.Text = 'Max Frequency';

            % Create NonstationaryMaxFreqField
            app.NonstationaryMaxFreqField =
uieditfield(app.NonstationaryButtonGroup, 'numeric');
            app.NonstationaryMaxFreqField.Limits = [0 Inf];
            app.NonstationaryMaxFreqField.Enable = 'off';
            app.NonstationaryMaxFreqField.Position = [123.03125 63 87 22];
            app.NonstationaryMaxFreqField.Value = 10000;

            % Create Label29
            app.Label29 = uilabel(app.NonstationaryButtonGroup);
            app.Label29.HorizontalAlignment = 'right';
            app.Label29.Position = [288.03125 67 80 15];
            app.Label29.Text = 'Min Frequency';

            % Create NonstationaryMinFreqField
            app.NonstationaryMinFreqField =
uieditfield(app.NonstationaryButtonGroup, 'numeric');
            app.NonstationaryMinFreqField.Limits = [0 Inf];
            app.NonstationaryMinFreqField.Enable = 'off';
            app.NonstationaryMinFreqField.Position = [394.03125 63 89 22];
            app.NonstationaryMinFreqField.Value = 100;

            % Create SimulationTab
            app.SimulationTab = uitab(app.TabGroup);
            app.SimulationTab.Title = 'Simulation';

            % Create Label12
            app.Label12 = uilabel(app.SimulationTab);
            app.Label12.FontSize = 22;
            app.Label12.FontWeight = 'bold';
            app.Label12.Position = [27 496 101 29];
            app.Label12.Text = 'Simulator';

            % Create LabelNumericEditField12
            app.LabelNumericEditField12 = uilabel(app.SimulationTab);
            app.LabelNumericEditField12.HorizontalAlignment = 'right';
            app.LabelNumericEditField12.Position = [27.03125 463 126 15];
            app.LabelNumericEditField12.Text = 'Number of Acquisitions';

            % Create NumTracesField
            app.NumTracesField = uieditfield(app.SimulationTab,
'numeric');
            app.NumTracesField.Limits = [2 Inf];
            app.NumTracesField.Position = [168.03125 459 100 22];
            app.NumTracesField.Value = 65;

            % Create LabelTextArea
            app.LabelTextArea = uilabel(app.SimulationTab);
            app.LabelTextArea.HorizontalAlignment = 'right';
            app.LabelTextArea.FontWeight = 'bold';
            app.LabelTextArea.Position = [35.03125 275 86 15];
            app.LabelTextArea.Text = 'Simulation Log';
```

```
            % Create SimulationLog
            app.SimulationLog = uitextarea(app.SimulationTab);
            app.SimulationLog.Editable = 'off';
            app.SimulationLog.Position = [131.03125 59 531 231];

            % Create SimulationStartButton
            app.SimulationStartButton = uibutton(app.SimulationTab,
'push');
            app.SimulationStartButton.ButtonPushedFcn =
createCallbackFcn(app, @SimulationStartButtonButtonPushed, true);
            app.SimulationStartButton.Position = [311 459 58 22];
            app.SimulationStartButton.Text = 'Start';

            % Create SaveParamsButton
            app.SaveParamsButton = uibutton(app.SimulationTab, 'push');
            app.SaveParamsButton.ButtonPushedFcn = createCallbackFcn(app,
@SaveParamsButtonButtonPushed, true);
            app.SaveParamsButton.Enable = 'off';
            app.SaveParamsButton.Position = [127 404 58 22];
            app.SaveParamsButton.Text = 'Save';

            % Create SaveEventsButton
            app.SaveEventsButton = uibutton(app.SimulationTab, 'push');
            app.SaveEventsButton.ButtonPushedFcn = createCallbackFcn(app,
@SaveEventsButtonButtonPushed, true);
            app.SaveEventsButton.Enable = 'off';
            app.SaveEventsButton.Position = [127 371 58 22];
            app.SaveEventsButton.Text = 'Save';

            % Create SaveSeedButton
            app.SaveSeedButton = uibutton(app.SimulationTab, 'push');
            app.SaveSeedButton.ButtonPushedFcn = createCallbackFcn(app,
@SaveSeedButtonButtonPushed, true);
            app.SaveSeedButton.Enable = 'off';
            app.SaveSeedButton.Position = [127 334 58 22];
            app.SaveSeedButton.Text = 'Save';

            % Create Label24
            app.Label24 = uilabel(app.SimulationTab);
            app.Label24.Position = [27 408 84 15];
            app.Label24.Text = 'Parameters File';

            % Create Label26
            app.Label26 = uilabel(app.SimulationTab);
            app.Label26.Position = [27 338 56 15];
            app.Label26.Text = 'Seeds File';

            % Create Label25
            app.Label25 = uilabel(app.SimulationTab);
            app.Label25.Position = [27 375 59 15];
            app.Label25.Text = 'Events File';

            % Create ViewEventTableButton
            app.ViewEventTableButton = uibutton(app.SimulationTab,
```

```
'push');
            app.ViewEventTableButton.ButtonPushedFcn =
createCallbackFcn(app, @ViewEventTableButtonButtonPushed, true);
            app.ViewEventTableButton.Position = [311 404 100 22];
            app.ViewEventTableButton.Text = 'View Events';

            % Create ClearButton
            app.ClearButton = uibutton(app.SimulationTab, 'push');
            app.ClearButton.ButtonPushedFcn = createCallbackFcn(app,
@ClearButtonPushed, true);
            app.ClearButton.Position = [389 459 66 22];
            app.ClearButton.Text = 'Clear';

            % Create TraceAnalysisTab
            app.TraceAnalysisTab = uitab(app.TabGroup);
            app.TraceAnalysisTab.Title = 'Trace Analysis';

            % Create AnalysisTypeLabel
            app.AnalysisTypeLabel = uilabel(app.TraceAnalysisTab);
            app.AnalysisTypeLabel.HorizontalAlignment = 'right';
            app.AnalysisTypeLabel.Position = [21 502 78 15];
            app.AnalysisTypeLabel.Text = 'Analysis Type';

            % Create PlotSelectionMenu
            app.PlotSelectionMenu = uidropdown(app.TraceAnalysisTab);
            app.PlotSelectionMenu.Items = {'Intensity Plot', 'Phase
Differential Avg Plot', 'Phase Differential Plot', 'Phase Angle Plot',
'Laser Phase Noise'};
            app.PlotSelectionMenu.ValueChangedFcn = createCallbackFcn(app,
@PlotSelectionMenuValueChanged, true);
            app.PlotSelectionMenu.Position = [126 500 161 20];
            app.PlotSelectionMenu.Value = 'Intensity Plot';

            % Create AnalysisPlot
            app.AnalysisPlot = uiaxes(app.TraceAnalysisTab);
            title(app.AnalysisPlot, 'Title')
            xlabel(app.AnalysisPlot, 'X')
            ylabel(app.AnalysisPlot, 'Y')
            app.AnalysisPlot.Box = 'on';
            app.AnalysisPlot.XGrid = 'on';
            app.AnalysisPlot.YGrid = 'on';
            app.AnalysisPlot.Position = [11 12 690 386];

            % Create UndockFigureButton
            app.UndockFigureButton = uibutton(app.TraceAnalysisTab,
'push');
            app.UndockFigureButton.ButtonPushedFcn =
createCallbackFcn(app, @UndockFigureButtonPushed, true);
            app.UndockFigureButton.Position = [521 442 100 22];
            app.UndockFigureButton.Text = 'Undock Figure';

            % Create TraceAveragesEditFieldLabel
            app.TraceAveragesEditFieldLabel =
uilabel(app.TraceAnalysisTab);
            app.TraceAveragesEditFieldLabel.HorizontalAlignment = 'right';
```

```
            app.TraceAveragesEditFieldLabel.Position = [516 502 88 15];
            app.TraceAveragesEditFieldLabel.Text = 'Trace Averages';

            % Create TraceAveragesEditField
            app.TraceAveragesEditField = uieditfield(app.TraceAnalysisTab,
'numeric');
            app.TraceAveragesEditField.Limits = [1 Inf];
            app.TraceAveragesEditField.ValueDisplayFormat = '%.0f';
            app.TraceAveragesEditField.Position = [619 498 29 22];
            app.TraceAveragesEditField.Value = 1;

            % Create ABseparationfactorEditFieldLabel
            app.ABseparationfactorEditFieldLabel =
uilabel(app.TraceAnalysisTab);
            app.ABseparationfactorEditFieldLabel.HorizontalAlignment =
'right';
            app.ABseparationfactorEditFieldLabel.Position = [21 449 116
15];
            app.ABseparationfactorEditFieldLabel.Text = 'AB separation
factor';

            % Create ABseparationfactorEditField
            app.ABseparationfactorEditField =
uieditfield(app.TraceAnalysisTab, 'numeric');
            app.ABseparationfactorEditField.Limits = [1 Inf];
            app.ABseparationfactorEditField.ValueDisplayFormat = '%.0f';
            app.ABseparationfactorEditField.Position = [157 445 27 22];
            app.ABseparationfactorEditField.Value = 1;

            % Create SubseparationfactorEditFieldLabel
            app.SubseparationfactorEditFieldLabel =
uilabel(app.TraceAnalysisTab);
            app.SubseparationfactorEditFieldLabel.HorizontalAlignment =
'right';
            app.SubseparationfactorEditFieldLabel.Position = [21.03125 417
121 15];
            app.SubseparationfactorEditFieldLabel.Text = 'Sub separation
factor';

            % Create SubseparationfactorEditField
            app.SubseparationfactorEditField =
uieditfield(app.TraceAnalysisTab, 'numeric');
            app.SubseparationfactorEditField.Limits = [1 Inf];
            app.SubseparationfactorEditField.ValueDisplayFormat = '%.0f';
            app.SubseparationfactorEditField.Position = [157.03125 413
27.984375 22];
            app.SubseparationfactorEditField.Value = 1;

            % Create ShowTraceDeltasCheckBox
            app.ShowTraceDeltasCheckBox =
uicheckbox(app.TraceAnalysisTab);
            app.ShowTraceDeltasCheckBox.Text = 'Show Trace Deltas';
            app.ShowTraceDeltasCheckBox.Position = [521 474 122 15];

            % Create DifferentialSeparationFactorEditFieldLabel
```

```
            app.DifferentialSeparationFactorEditFieldLabel =
uilabel(app.TraceAnalysisTab);

app.DifferentialSeparationFactorEditFieldLabel.HorizontalAlignment =
'right';
            app.DifferentialSeparationFactorEditFieldLabel.Position = [258
449 162 15];
            app.DifferentialSeparationFactorEditFieldLabel.Text =
'Differential Separation Factor';

            % Create DifferentialSeparationFactorEditField
            app.DifferentialSeparationFactorEditField =
uieditfield(app.TraceAnalysisTab, 'numeric');
            app.DifferentialSeparationFactorEditField.Limits = [1 Inf];
            app.DifferentialSeparationFactorEditField.ValueDisplayFormat =
'%.0f';
            app.DifferentialSeparationFactorEditField.Position = [440 445
27 22];
            app.DifferentialSeparationFactorEditField.Value = 1;

            % Create SubseparationfactorEditField_2Label
            app.SubseparationfactorEditField_2Label =
uilabel(app.TraceAnalysisTab);
            app.SubseparationfactorEditField_2Label.HorizontalAlignment =
'right';
            app.SubseparationfactorEditField_2Label.Position = [262 417
121 15];
            app.SubseparationfactorEditField_2Label.Text = 'Sub separation
factor';

            % Create SubseparationfactorEditField_2
            app.SubseparationfactorEditField_2 =
uieditfield(app.TraceAnalysisTab, 'numeric');
            app.SubseparationfactorEditField_2.Limits = [1 Inf];
            app.SubseparationfactorEditField_2.ValueDisplayFormat =
'%.0f';
            app.SubseparationfactorEditField_2.Position = [441 413 26 22];
            app.SubseparationfactorEditField_2.Value = 1;

            % Create DifferentialAnalysisParametersLabel
            app.DifferentialAnalysisParametersLabel =
uilabel(app.TraceAnalysisTab);
            app.DifferentialAnalysisParametersLabel.FontWeight = 'bold';
            app.DifferentialAnalysisParametersLabel.FontColor = [1 0 0];
            app.DifferentialAnalysisParametersLabel.Position = [266 474
188 15];
            app.DifferentialAnalysisParametersLabel.Text = 'Differential
Analysis Parameters';

            % Create DifferentialAvgAnalysisParametersLabel
            app.DifferentialAvgAnalysisParametersLabel =
uilabel(app.TraceAnalysisTab);
            app.DifferentialAvgAnalysisParametersLabel.FontWeight =
'bold';
            app.DifferentialAvgAnalysisParametersLabel.FontColor = [1 0
```

```
0];
            app.DifferentialAvgAnalysisParametersLabel.Position = [24 474
214 15];
            app.DifferentialAvgAnalysisParametersLabel.Text =
'Differential Avg Analysis Parameters';

            % Create SlowTimeFFTButton
            app.SlowTimeFFTButton = uibutton(app.TraceAnalysisTab,
'push');
            app.SlowTimeFFTButton.ButtonPushedFcn = createCallbackFcn(app,
@SlowTimeFFTButtonPushed, true);
            app.SlowTimeFFTButton.Position = [520 413 100 22];
            app.SlowTimeFFTButton.Text = 'Slow Time FFT';

            % Create AcquisitionNumberSpinnerLabel
            app.AcquisitionNumberSpinnerLabel =
uilabel(app.TraceAnalysisTab);
            app.AcquisitionNumberSpinnerLabel.HorizontalAlignment =
'right';
            app.AcquisitionNumberSpinnerLabel.Position = [300 502 111 15];
            app.AcquisitionNumberSpinnerLabel.Text = 'Acquisition Number';

            % Create AcquisitionSpinner
            app.AcquisitionSpinner = uispinner(app.TraceAnalysisTab);
            app.AcquisitionSpinner.ValueChangedFcn =
createCallbackFcn(app, @AcquisitionSpinnerValueChanged, true);
            app.AcquisitionSpinner.Limits = [1 Inf];
            app.AcquisitionSpinner.Position = [428 499 44 22];
            app.AcquisitionSpinner.Value = 1;

            % Create EventLogTab
            app.EventLogTab = uitab(app.TabGroup);
            app.EventLogTab.Title = 'Event Log';

            % Create StationaryEventsSummary
            app.StationaryEventsSummary = uipanel(app.EventLogTab);
            app.StationaryEventsSummary.Title = 'Stationary Events
Summary';
            app.StationaryEventsSummary.Position = [1 272 721 269];

            % Create StationaryEventsSummaryLog
            app.StationaryEventsSummaryLog =
uitextarea(app.StationaryEventsSummary);
            app.StationaryEventsSummaryLog.Position = [23.03125 18
149.953125 142];

            % Create EventNumberSpinnerLabel
            app.EventNumberSpinnerLabel =
uilabel(app.StationaryEventsSummary);
            app.EventNumberSpinnerLabel.HorizontalAlignment = 'right';
            app.EventNumberSpinnerLabel.Position = [23 214 82 15];
            app.EventNumberSpinnerLabel.Text = 'Event Number';

            % Create StationaryEventsSpinner
            app.StationaryEventsSpinner =
```

```
uispinner(app.StationaryEventsSummary);
            app.StationaryEventsSpinner.ValueChangedFcn =
createCallbackFcn(app, @StationaryEventsSpinnerValueChanged, true);
            app.StationaryEventsSpinner.Limits = [1 Inf];
            app.StationaryEventsSpinner.Position = [117 210 47 22];
            app.StationaryEventsSpinner.Value = 1;

            % Create StationaryEventsFreqPlot
            app.StationaryEventsFreqPlot =
uiaxes(app.StationaryEventsSummary);
            title(app.StationaryEventsFreqPlot, 'Title')
            xlabel(app.StationaryEventsFreqPlot, 'X')
            ylabel(app.StationaryEventsFreqPlot, 'Y')
            app.StationaryEventsFreqPlot.Position = [212 18 474 211];

            % Create StationaryEventsUndockFigure
            app.StationaryEventsUndockFigure =
uibutton(app.StationaryEventsSummary, 'push');
            app.StationaryEventsUndockFigure.ButtonPushedFcn =
createCallbackFcn(app, @StationaryEventsUndockFigureButtonPushed, true);
            app.StationaryEventsUndockFigure.Position = [23 177 150 22];
            app.StationaryEventsUndockFigure.Text = 'Undock Figure';

            % Create NonStationaryEventsSummary
            app.NonStationaryEventsSummary = uipanel(app.EventLogTab);
            app.NonStationaryEventsSummary.Title = 'NonStationary Events
Summary';
            app.NonStationaryEventsSummary.Position = [1 2 721 271];

            % Create LabelTextArea2
            app.LabelTextArea2 = uilabel(app.NonStationaryEventsSummary);
            app.LabelTextArea2.HorizontalAlignment = 'right';
            app.LabelTextArea2.Position = [172 217 20 15];
            app.LabelTextArea2.Text = '';

            % Create NonStationaryEventsSummaryLog
            app.NonStationaryEventsSummaryLog =
uitextarea(app.NonStationaryEventsSummary);
            app.NonStationaryEventsSummaryLog.Position = [23 22 150 131];

            % Create EventNumberSpinner_2Label
            app.EventNumberSpinner_2Label =
uilabel(app.NonStationaryEventsSummary);
            app.EventNumberSpinner_2Label.HorizontalAlignment = 'right';
            app.EventNumberSpinner_2Label.Position = [21 214 82 15];
            app.EventNumberSpinner_2Label.Text = 'Event Number';

            % Create NonStationaryEventsSpinner
            app.NonStationaryEventsSpinner =
uispinner(app.NonStationaryEventsSummary);
            app.NonStationaryEventsSpinner.ValueChangedFcn =
createCallbackFcn(app, @NonStationaryEventsSpinnerValueChanged, true);
            app.NonStationaryEventsSpinner.Limits = [1 Inf];
            app.NonStationaryEventsSpinner.Position = [117 210 47 22];
            app.NonStationaryEventsSpinner.Value = 1;
```

```
            % Create NonStationaryEventsFreqPlot
            app.NonStationaryEventsFreqPlot =
uiaxes(app.NonStationaryEventsSummary);
            title(app.NonStationaryEventsFreqPlot, 'Title')
            xlabel(app.NonStationaryEventsFreqPlot, 'X')
            ylabel(app.NonStationaryEventsFreqPlot, 'Y')
            app.NonStationaryEventsFreqPlot.Position = [210 22 476 210];

            % Create NonStationaryEventsUndockFigure
            app.NonStationaryEventsUndockFigure =
uibutton(app.NonStationaryEventsSummary, 'push');
            app.NonStationaryEventsUndockFigure.ButtonPushedFcn =
createCallbackFcn(app, @NonStationaryEventsUndockFigureButtonPushed,
true);
            app.NonStationaryEventsUndockFigure.Position = [23 175 150
22];
            app.NonStationaryEventsUndockFigure.Text = 'Undock Figure';
        end
    end

    methods (Access = public)

        % Construct app
        function app = FiberSimGUI_2017a

            % Create and configure components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.PhaseOTDRSimulatorUIFigure)

            % Execute the startup function
            runStartupFcn(app, @startupFcn)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.PhaseOTDRSimulatorUIFigure)
        end
    end
end
```

Bibliography

[1] A. Rogers, "Distributed optical-fibre sensing," *Meas. Sci. Technol.*, vol. 10, no. 8, pp. R75–R99, 1999.

[2] L. Palmieri and L. Schenato, "Distributed Optical Fiber Sensing Based on Rayleigh Scattering," *Open Opt. J.*, vol. 7, no. 24, pp. 104–127, 2013.

[3] C. K. Kirkendall and A. Dandridge, "Overview of high performance fibre-optic sensing," *J. Phys. D. Appl. Phys.*, vol. 37, pp. R197-R197–R216, 2004.

[4] X. Bao and L. Chen, "Recent progress in distributed fiber optic sensors," *Sensors (Basel)*, vol. 12, no. 7, pp. 8601–8639, 2012.

[5] J. C. Juarez and H. F. Taylor, "Polarization discrimination in a phase-sensitive optical time-domain reflectometer intrusion-sensor system.," *Opt. Lett.*, vol. 30, no. 24, pp. 3284–3286, 2005.

[6] C. K. Madsen, T. Bae, and R. A. Atkins, "Long fiber-optic perimeter sensor: Signature analysis," *Conf. Lasers Electro-Optics, 2007, CLEO 2007*, vol. 23, no. 5, p. 77843, 2007.

[7] H. N. Li, D. S. Li, and G. B. Song, "Recent applications of fiber optic sensors to health monitoring in civil engineering," *Eng. Struct.*, vol. 26, no. 11, pp. 1647–1657, 2004.

[8] M. Bravo, J. Saenz, M. Bravo-Navas, and M. Lopez-Amo, "Concrete Beam Bending Test Monitorization Using a High Strain Fiber Optic Sensor," *J. Light. Technol.*, vol. 30, no. 8, pp. 1085–1089, 2012.

[9] X. Guangping, S. L. Keey, and A. Asundi, "Optical time-domain re # ectometry for distributed sensing of the structural strain and deformation," vol. 32, pp. 437–447, 2000.

[10] S. Villalba and J. R. Casas, "Application of optical fiber distributed sensing to health monitoring of concrete structures," *Mech. Syst. Signal Process.*, vol. 39, no. 1–2, pp. 441–451, 2013.

[11] L. Zeni, "Optical fiber distributed sensors : a tool for in-situ structural and environmental monitoring," in *Italian Workshop on Landslides*, 2009.

[12] Z. W. Zhu, D. Y. Liu, Q. Y. Yuan, B. Liu, and J. C. Liu, "A novel distributed optic fiber

transduser for landslides monitoring," *Opt. Lasers Eng.*, vol. 49, no. 7, pp. 1019–1024, 2011.

[13] J. Chai, S. M. Wei, X. T. Chang, and J. X. Liu, "Monitoring deformation and damage on rock structures with distributed fiber optical sensing," *Int. J. Rock Mech. Min. Sci.*, vol. 41, no. SUPPL. 1, p. 2004, 2004.

[14] J. Tejedor, J. Macias-Guarasa, H. Martins, D. Piote, J. Pastor-Graells, S. Martin-Lopez, P. Corredera, and M. Gonzalez-Herraez, "A Novel Fiber Optic Based Surveillance System for Prevention of Pipeline Integrity Threats," *Sensors*, vol. 17, no. 2, p. 355, 2017.

[15] F. Peng, H. Wu, X. H. Jia, Y. J. Rao, Z. N. Wang, and Z. P. Peng, "Ultra-long high-sensitivity Phi-OTDR for high spatial resolution intrusion detection of pipelines," *Opt. Express*, vol. 22, no. 11, pp. 13804–13810, 2014.

[16] F. Peng, N. Duan, Y. J. Rao, and J. Li, "Real-time position and speed monitoring of trains using phase-sensitive OTDR," *IEEE Photonics Technol. Lett.*, vol. 26, no. 20, pp. 2055–2057, 2014.

[17] D. B. Keck, "Optical fiber spans 30 years," *Lightwave*, vol. 17, no. 8, pp. 78–82, 2000.

[18] C. Incorporated, "Corning ® SMF- 28 e +® LL Optical Fiber," no. July. pp. 2010–2011, 2011.

[19] R. Olshansky, "Propagation in glass optical waveguides," *Rev. Mod. Phys.*, vol. 51, no. 2, pp. 341–367, 1979.

[20] D. Gloge, "Propagation effects in optical fibers," ... *Theory Tech. IEEE Trans.*, vol. MTT-23, no. 1, pp. 106–120, 1975.

[21] D. Gloge, "The optical fibre as a transmission medium," *Reports Prog. Phys.*, vol. 42, pp. 1778–1824, 1979.

[22] S. Miller, A. J. Marcatili, and T. Li, "Part I: The Transmission Medium," *Proc. IEEE*, vol. 61, no. 12, pp. 1704–1726, 1973.

[23] R. M. Atkins, "Measurement of the ultraviolet absorption spectrum of optical fibers," *Opt. Lett.*, vol. 17, no. 7, pp. 469–471, 1992.

[24] T. Izawa, N. Shibata, and A. Takeda, "Optical attenuation in pure and doped fused silica in the ir wavelength region," *Appl. Phys. Lett.*, vol. 31, no. 1, pp. 33–35, 1977.

[25] O. Humbach, H. Fabian, U. Grzesik, U. Haken, and W. Heitmann, “Analysis of OH absorption bands in synthetic silica,” *J. Non. Cryst. Solids*, vol. 203, pp. 19–26, 1996.

[26] G. Thomas, B. Shraiman, P. Glodis, and M. Stephens, “Towards the clarity limit in optical fibre,” *Nature*, vol. 404, no. 6775, pp. 262–4, 2000.

[27] J. M. Senior and M. Y. Jamro, *Optical Fiber Communications: Principles and Practice*. 2009.

[28] M. Lancry, E. Régnier, and B. Poumellec, “Fictive temperature in silica-based glasses and its application to optical fiber manufacturing,” *Prog. Mater. Sci.*, vol. 57, no. 1, pp. 63–94, 2012.

[29] S. R. Nagel, J. B. MacChesney, and K. L. Walker, “An Overview of the Modified Chemical Vapor Deposition (MCVD) Process and Performance,” *IEEE Trans. Microw. Theory Tech.*, vol. 30, no. 4, pp. 305–322, 1982.

[30] H. Lydtin, “PCVD: A Technique Suitable for Large-Scale Fabrication of Optical Fibers,” *J. Light. Technol.*, vol. 4, no. 8, pp. 1034–1038, 1986.

[31] H. Murata, “Recent Developments in Vapor Phase Axial Deposition,” *J. Light. Technol.*, vol. 4, no. 8, pp. 1026–1033, 1986.

[32] O. W. Fibers, “The Outside Vapor Deposition Method of Fabricating,” pp. 1418–1423, 1982.

[33] L. L. Blyler and F. V. Dimarcello, “Fiber Drawing, Coating, and Jacketing,” *Proc. IEEE*, vol. 68, no. 10, pp. 1194–1198, 1980.

[34] D. P. Jablonowski, U. C. Paek, and L. S. Watkins, “Optical Fiber Manufacturing Techniques,” *AT&T Tech. J.*, vol. 66, no. 1, pp. 33–44, 1987.

[35] K. Saito, H. Kakiuchida, and A. J. Ikushima, “Light-scattering study of the glass transition in silica, with practical implications,” *J. Appl. Phys.*, vol. 84, no. 6, p. 3107, 1998.

[36] K. Saito, A. J. Ikushima, T. Ito, and A. Itoh, “A new method of developing ultralow-loss glasses,” *J. Appl. Phys.*, vol. 81, no. 11, p. 7129, 1997.

[37] K. Saito, “Approach for reducing the Rayleigh scattering loss in optical fibers,” *J. Appl. Phys.*, vol. 95, no. 4, p. 1733, 2004.

[38] K. Saito, M. Yamaguchi, H. Kakiuchida, A. J. Ikushima, K. Ohsono, and Y. Kurosawa, “Limit of

the Rayleigh scattering loss in silica fiber," *Appl. Phys. Lett.*, vol. 83, no. 25, p. 5175, 2003.

[39] B. Chu, *Laser Light Scattering*, 1st ed. New York: Academic Press, 1974.

[40] R. W. Boyd, *Non-Linear Optics, Third Edition*. 2008.

[41] I. L. Fabelinskii, *Molecular Scattering of Light*. New York: Plenum Press, 1968.

[42] L. D. Landau and E. M. Lifshitz, *Statistical Physics*, Second Rev. Pergamon Press, 1970.

[43] A. Einstein, "Theory of opalescence of homogeneous fluids and fluid mixtures in the neighbourhood of the critical state," *Ann. Phys.*, vol. 338, no. 16, pp. 1275–1298, 1910.

[44] J. SCHROEDER, R. MOHR, P. B. MACEDO, and C. J. MONTROSE, "Rayleigh and Brillouin Scattering in K2O-SiO2 Glasses," *J. Am. Ceram. Soc.*, vol. 56, no. 10, pp. 510–514, 1973.

[45] a. Hartog and M. Gold, "On the theory of backscattering in single-mode optical fibers," *J. Light. Technol.*, vol. 2, no. 2, pp. 76–82, 1984.

[46] P. Healey, "Statistics of Rayleigh Backscatter From a Single-Mode Fiber," *IEEE Trans. Commun.*, vol. 35, no. 2, pp. 210–214, 1987.

[47] R. K. Staubli and P. Gysel, "Statistical properties of single-mode fiber Rayleigh backscattered intensity and resulting detector current," *IEEE Trans. Commun.*, vol. 40, no. 6, pp. 1091–1097, 1992.

[48] D. Marcuse, *Principles of Optical Fiber Measurements*. Academic Press, 1981.

[49] Z. Chen, Y. Zeng, G. Hefferman, Y. Sun, and T. Wei, "FiberID: Molecular-level secret for identification of things," *2014 IEEE Int. Work. Inf. Forensics Secur. WIFS 2014*, pp. 84–88, 2015.

[50] H. F. Taylor and C. E. Lee, "Apparatus And Method For Fiber Optic Instrusion Sensing," 5,194,847, 1993.

[51] H. Izumita, Y. Koyamada, and S. Furukawa, "The performance limit of coherent OTDR enhanced with optical fiber amplifiers due to optical nonline - Lightwave Technology, Journal of," *Lightwave*, vol. 12, no. 7, pp. 1230–1238, 1994.

[52] W. Zou, S. Yang, X. Long, and J. Chen, "Optical pulse compression reflectometry: proposal and proof-of-concept experiment," *Opt. Express*, vol. 23, no. 1, p. 512, 2015.

[53] B. L. U. In, Z. H. P. An, Z. H. W. Ang, H. A. Z. Heng, Q. Y. E. Ing, R. Q. U. Onghui, and H. A. C. Ai, “High spatial resolution phase-sensitive optical time domain reflectometer with a frequency-swept pulse,” vol. 42, no. 3, pp. 1–4, 2017.

[54] K. Yuksel, M. Wuilpart, V. Moeyaert, and P. Mégret, “Optical frequency domain reflectometry: A review,” *2009 11th Int. Conf. Transparent Opt. Networks*, pp. 1–5, 2009.

[55] I. S. Reed, “On a Moment Theorem for Complex Gaussian Processes,” *IRE Trans. Inf. Theory*, vol. 8, no. 3, pp. 194–195, 1962.

[56] Z. Sha, H. Feng, and Z. Zeng, “Phase demodulation method in phase-sensitive OTDR without coherent detection,” *Opt. Express*, vol. 25, no. 5, p. 4831, 2017.

[57] E. Ip, A. Pak, T. Lau, D. J. F. Barros, and J. M. Kahn, “Coherent detection in optical fiber systems,” *Electronics*, vol. 16, no. 2, pp. 861–866, 2008.

[58] K. Kikuchi, *High Spectral Density Optical Communication Technologies*. 2010.

[59] P. Healey, R. C. Booth, B. E. Daymond-John, and B. K. Nayar, “OTDR in Single-Mode Fiber at 1.5 micro-meter using Homodyne Detection,” *Electron. Lett.*, vol. 20, no. 9, pp. 360–362, 1984.

[60] P. Healey and D. J. Malyon, “OTDR IN SINGLE-MODE FIBRE AT 1-5 um USING HETERODYNE DETECTION,” *Electron. Lett.*, vol. 18, no. 20, pp. 862–863, 1982.

[61] C. Wang, C. Wang, Y. Shang, X. Liu, and G. Peng, “Distributed acoustic mapping based on interferometry of phase optical time-domain reflectometry,” *Opt. Commun.*, vol. 346, pp. 172–177, 2015.

[62] G. Fang, T. Xu, S. Feng, and F. Li, “Phase-sensitive optical time domain reflectometer based on phase-generated carrier algorithm,” *J. Light. Technol.*, vol. 33, no. 13, pp. 2811–2816, 2015.

[63] Z. Wang, L. Zhang, S. Wang, N. Xue, F. Peng, M. Fan, W. Sun, X. Qian, J. Rao, and Y. Rao, “Coherent Φ-OTDR based on I/Q demodulation and homodyne detection,” *Opt. Express*, vol. 24, no. 2, p. 853, 2016.

[64] Z. Pan, K. Liang, Q. Ye, H. Cai, R. Qu, and Z. Fang, “Phase-sensitive OTDR system based on digital coherent detection,” vol. 8311, no. 8, p. 83110S, 2011.

[65] M. G. Taylor, “Phase estimation methods for optical coherent detection using digital signal processing,” *J. Light. Technol.*, vol. 27, no. 7, pp. 901–914, 2009.

[66] A. Zhirnov, A. Fedorov, K. Stepanov, E. Nesterov, V. Karasik, C. Svelto, and A. Pnev, “Effects of laser frequency drift in phase-sensitive optical time-domain reflectometry fiber sensors,” pp. 1–6, 2016.

[67] X. Zhong, C. X. Zhang, L. J. Li, S. Liang, Q. Li, Q. Y. Lu, X. X. Ding, and Q. Y. Cao, “Influences of laser source on phase-sensitivity optical time-domain reflectometer-based distributed intrusion sensor,” *Appl. Opt.*, vol. 53, no. 21, pp. 4645–4650, 2014.

[68] R. Tkach and A. Chraplyvy, “Phase noise and linewidth in an InGaAsP DFB laser,” *Light. Technol. J.*, vol. 4, no. 11, pp. 1711–1716, 1986.

[69] P. Healey, “Fading in Heterodyne OTDR,” *Electron. Lett.*, vol. 20, no. 7, pp. 3–5, 1984.

[70] R. E. Schuh, J. G. Ellison, A. S. Siddiqui, and D. H. O. Bebbington, “Polarisation OTDR measurements and theoretical analysis on fibres with twist and their implications for estimation of PMD,” *Electron. Lett.*, vol. 32, no. 4, pp. 387–388, 1996.

[71] H. F. Martins, S. Martin-Lopez, P. Corredera, P. Salgado, O. Frazão, and M. González-Herráez, “Modulation instability-induced fading in phase-sensitive optical time-domain reflectometry,” vol. 59004370, no. 6, pp. 872–874, 2640.

[72] H. Izumita, S. Furukawa, Y. Koyamada, and I. Sankawa, “Fading Noise-Reduction in Coherent Otdr,” *Ieee Photonics Technol. Lett.*, vol. 4, no. 2, pp. 201–203, 1992.

[73] A. K. W´ Ojcik, “Signal Statistics of Phase Dependent Optical Time Domain Reflectometry,” no. December, p. 97, 2006.

[74] a E. Alekseev, V. S. Vdovenko, B. G. Gorshkov, V. T. Potapov, and D. E. Simikin, “A phase-sensitive optical time-domain reflectometer with dual-pulse diverse frequency probe signal,” *Laser Phys.*, vol. 25, no. 6, p. 65101, 2015.

[75] Y. Shi, H. Feng, and Z. Zeng, “Phase-Sensitive Optical Time Domain Reflectometer with Dual-Wavelength Probe Pulse,” vol. 2015, 2015.

[76] J. Zhou, Z. Pan, Q. Ye, H. Cai, R. Qu, and Z. Fang, “Characteristics and explanations of interference fading of a φ-OTDR with a multi-frequency source,” *J. Light. Technol.*, vol. 31, no. 17, pp. 2947–2954, 2013.

[77] a E. Alekseev, V. S. Vdovenko, B. G. Gorshkov, V. T. Potapov, and D. E. Simikin, “A phase-sensitive optical time-domain reflectometer with dual-pulse phase modulated probe signal,” *Laser Phys.*, vol. 25, no. 6, p. 65101, 2015.

[78] M. A. R. F. E. Uiz, H. U. G. O. F. M. Artins, J. U. A. N. P. A. Raells, S. O. M. A. Opez, and M. I. G. O. Erraez, “Phase-sensitive OTDR probe pulse shapes robust against modulation-instability fading,” vol. 41, no. 24, pp. 5756–5759, 2016.

[79] L. B. Liokumovich, N. A. Ushakov, S. Member, O. I. Kotov, M. A. Bisyarin, A. H. Hartog, and S. Member, “Fundamentals of optical fiber sensing schemes based on coherent optical time domain reflectometry : Signal model under static fiber conditions,” 2015.

[80] Y. Nakano, G. Motosugi, Y. Yoshikuni, and T. Ikegami, “Aging characteristics of InGaAsP/InP DFB lasers,” *Electron. Lett.*, vol. 19, p. 437, 1983.

[81] Y. Chung, J. Jeong, and L. Cheng, “Aging-induced wavelength shifts in 1.5-μm DFB lasers,” *IEEE Photonics Technol. Lett.*, vol. 6, no. 7, pp. 792–795, 1994.

[82] M. Ito and T. Kimura, “Stationary and transient thermal properties of semiconductor laser diodes,” *IEEE J. Quantum Electron.*, vol. 17, no. 5, pp. 787–795, 1981.

[83] J. A. Barnes, A. R. Chi, L. S. Cutler, D. J. Healey, D. B. Leeson, E. T. McGunigal, J. A. Mullen, W. L. Smith, R. L. Sydnor, R. F. C. Vessot, and G. M. R. Winkler, “Characterization of Frequency Stability,” *IEEE Trans. Instrum. Meas.*, vol. IM-20, no. 2, pp. 105–120, 1971.

[84] F. L. Walls and D. W. Allan, “Measurements of Frequency Stability,” *Proc. IEEE*, vol. 74, no. 1, pp. 162–168, 1986.

[85] G. Tu, B. Yu, S. Zhen, K. Qian, and X. Zhang, “Enhancement of Signal Identification and Extraction in a Φ-OTDR Vibration Sensor,” *IEEE Photonics J.*, vol. 9, no. 1, 2017.

[86] M. Zhang, S. Wang, Y. Zheng, Y. Yang, X. Sa, and L. Zhang, “Enhancement for ??-OTDR

performance by using narrow linewidth light source and signal processing," *Photonic Sensors*, vol. 6, no. 1, pp. 58–62, 2016.

[87] K. Shimizu, T. Horiguchi, and Y. Koyamada, "Characteristics and Reduction of Coherent Fading Noise in Rayleigh Backscattering Measurement for Optical Fibers and Components," *J. Light. Technol.*, vol. 10, no. 7, pp. 982–987, 1992.

[88] Y. Shi, H. Feng, and Z. Zeng, "A long distance phase-sensitive optical time domain reflectometer with simple structure and high locating accuracy," *Sensors (Switzerland)*, vol. 15, no. 9, pp. 21957–21970, 2015.

[89] J. Makhoul, "Linear Prediction: A Tutorial Review," *Proc. IEEE*, vol. 63, no. 4, pp. 561–580, 1975.

[90] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing Principles, Algorithms, and Applications*, 4th ed. Prentice Hall, 2007.

[91] H. W. Strube, "Linear prediction on a warped frequency scale," *J. Acoust. Soc. Am.*, vol. 68, no. 4, pp. 1071–1076, 1980.

[92] M. Karjalainen and A. Harma, "Warped filters and their audio applications," *IEEE ASSP Work. Appl. Signal Process. to Audio Acoust. 1997*, pp. 0–3, 1997.

[93] A. Harma, M. Juntunen, and J. P. Kaipio, "Time-Varying Autoregressive Modeling of Audio and Speech Signals," in *Signal Processing Conference, 2000 10th European*, 2000.

[94] Y. Muanenda, C. J. Oton, S. Faralli, and F. Di Pasquale, "A Cost-Effective Distributed Acoustic Sensor Using a Commercial Off-the-Shelf DFB Laser and Direct Detection Phase-OTDR," *IEEE Photonics J.*, vol. 8, no. 1, 2016.

[95] A. Harma, "Implementation of frequency-warped recursive filters," *Signal Processing*, vol. 80, pp. 543–548, 2000.

[96] A. Bertholds and R. Dandliker, "Deformation of Single-Mode Optical Fibers Under Static Longitudinal Stress," *J. Light. Technol.*, vol. 5, no. 7, pp. 895–900, 1987.

[97] R. Ulrich and A. Simon, "Polarization optics of twisted single-mode fibers," *Appl. Opt.*, vol. 18,

no. 13, p. 2241, 1979.

[98] A. Galtarossa, D. Grosso, and L. Palmieri, “Accurate characterization of twist-induced optical activity in single-mode fibers by means of polarization-sensitive reflectometry,” *IEEE Photonics Technol. Lett.*, vol. 21, no. 22, pp. 1713–1715, 2009.

[99] C. D. Butter and G. B. Hocker, “Fiber Optic Strain Gauge,” *Appl. Opt.*, vol. 17, no. 18, pp. 2867–2869, 1978.

[100] A. Bertholds and R. Dändliker, “Determination of the Individual Strain-Optic Coefficients in Single-mode Optical Fibers,” *J. Light. Technol.*, vol. 6, no. 1, pp. 17–20, 1988.

[101] P. Antunes, H. Lima, J. Monteiro, and P. S. Andre, “Elastic Constant Measurement for Standard and Photosensitive Single Mode Optical Fibers,” *Microw. Opt. Technol. Lett.*, vol. 50, no. 9, pp. 2467–2469, 2008.

[102] X. Lu, M. A. Soto, and L. Thévenaz, “Optimal detection bandwidth for phase-sensitive optical time-domain reflectometry,” vol. 9916, p. 99162N, 2016.

[103] G. J. Tu, X. P. Zhang, Y. X. Zhang, F. Zhu, L. Xia, and B. Nakarmi, “The Development of an Phi-OTDR System for Quantitative Vibration Measurement,” *Ieee Photonics Technol. Lett.*, vol. 27, no. 12, pp. 1349–1352, 2015.

[104] H. Wu, L. Zhang, Y. Qian, H. Li, W. Zhang, and Y. Rao, “Multi-scale wavelet decomposition and its application in distributed optical fiber fences,” vol. 9655, p. 96553U, 2015.

[105] Z. Qin, L. Chen, and X. Bao, “Continuous wavelet transform for non-stationary vibration detection with phase-OTDR,” *Opt. Express*, vol. 20, no. 18, p. 20459, 2012.

[106] X. N. Hui, S. L. Zheng, J. H. Zhou, H. Chi, X. F. Jin, and X. M. Zhang, “Hilbert-Huang Transform Time-Frequency Analysis in phi-OTDR Distributed Sensor,” *Ieee Photonics Technol. Lett.*, vol. 26, no. 23, pp. 2403–2406, 2014.

[107] L. Shiloh and A. Eyal, “Distributed acoustic and vibration sensing via optical fractional Fourier transform reflectometry,” *Opt. Express*, vol. 25, no. 3, pp. 167–173, 2014.

[108] T. Zhu, X. Xiao, Q. He, and D. Diao, “Enhancement of SNR and spatial resolution in ℓ-OTDR

system by using two-dimensional edge detection method," *J. Light. Technol.*, vol. 31, no. 17, pp. 2851–2856, 2013.

[109] K. Nishiguchi, "Phase unwrapping for fiber-optic distributed acoustic sensing," *Proc. 47th Isc. Int. Symp. Stoch. Syst. Theory Its Appl.*, pp. 81–87, 2015.

[110] J. C. Dainty, "The Statistics of Speckle Patterns," in *Progress in Optics XIV*, 1976, pp. 3–44.