CROSS KERR & RAMAN EFFECTS ON THE SWITCHING EFFICIENCY OF SOLITON OPTICAL FIBER GATES

By

PATRICK CHILUFYA CHIMFWEMBE

Bachelor of Science (Hons.)
University College Cardiff
Wales, UK
1983.

Bachelor of Engineering
University of Zambia
Lusaka, Zambia
1988

Master of Science,
Boston University
Boston, Massachusetts
1990

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
May, 1999

CROSS KERR & RAMAN EFFECTS ON THE

SWITCHING EFFICIENCY OF SOLITON

OPTICAL FIBER GATES

Thesis Approved:

[illegible]

Thesis Adviser

[illegible]

[illegible]

James C West

Wayne B. Powell

Dean of the Graduate College

ACKNOWLEDGEMENTS

I extend my sincere thanks and gratitude to Dr. Jerzy Krasinski, my adviser and chairperson of my doctoral committee, for his guidance, dedication, patience, and encouragement during the term of my doctoral programme. His interest and insight into the problems encountered in my research work, greatly contributed towards the completion of this dissertation. My appreciation to the committee members Drs. Jim West, Jin-Joo Song, and Scott Shepard for their constructive comments and contribution of time during the course of the study.

Special thanks go to Dr. Czeslaw Radzewicz, a visiting Professor from University of Warsaw, Poland, for imparting to me essential optics laboratory skills. He also made it possible for me to know how to build a Ti-Sapphire laser and a parametric oscillator.

I am very grateful towards the Fulbright Programme, under the USA Government, and Oklahoma State University, through tuition waivers, for funding my Ph.D. programme at Oklahoma State University.

I would like to thank the University of Zambia for giving me the opportunity to pursue my Ph.D. studies, and supporting me during my programme of study.

I very much appreciate the cooperation, understanding and patience from my wife, Mary, during my programme of study. My two sons, Chilufya and Mwekwa were also sources of inspiration towards the completion of my programme of study.

TABLE OF CONTENTS

Chapter Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

LIST OF ACRONYMS AND SYMBOLS

c	Velocity of light in a vacuum
c.c.	complex conjugate
CFS	Cross Frequency Shifting
CHNLS	Coupled Higher-order Nonlinear Schrödinger
CPM	Cross-Phase Modulation
CS	Cross Steepening
CSI	Complete Shuffle Interaction
FWM	Four Wave Mixing
Gbps	Giga bits per second
GVD	Group Velocity Dispersion
HNLS	Higher-order Nonlinear Schrödinger
Mbps	Mega bits per second
$^{m}/_{o}$	mass percent
NLS	Nonlinear Schrödinger
NOLM	Nonlinear Optical Loop Mirror
PSI	Partial Shuffle Interaction
SDG	Soliton-Dragging Gate
SFS	Self Frequency Shifting
SOD	Second Order Dispersion
SPM	Self-Phase Modulation

SS	Self Steepening
STG	Soliton-Trapping Gate
TE	Transverse Electric
TM	Transverse Magnetic
TOD	Third Order Dispersion

CHAPTER I

INTRODUCTION

Optical transmission, amplification, switching, and processing have the potential of offering a large telecommunication and computing bandwidth. A suitable optical bit that has been used for optical transmission and switching is the soliton [1.1, 1.2]. Fiber based soliton switches and gates have been proposed and implemented for pulse widths of over 100 fs [1.2]. In order to achieve higher switching bit rates the soliton pulse width needs to be as small as is physically achievable, with respect to power constraints, higher order non-linear and dispersion effects. The main concern in this dissertation is the higher order nonlinear and dispersion effects, on the switching efficiency of four types of soliton optical fiber gates, namely the STG, SDG, STG-NOLM gate, and a newly discovered STG-PSI-NOLM gate, for sub-100 fs pulse operation. The switching efficiency of these gates inevitably impacts on their bandwidth.

New higher order nonlinear effects, affecting the switching efficiency of a STG, SDG, STG-NOLM, and STG-PSI-NOLM, have been investigated for soliton pulse widths below 100 fs, and are due to the cross Raman effects.

1.1 Objectives and Significance of the Study

The objective of the study was to investigate the effects of the self and cross Raman effects on the switching efficiency of STGs, SDGs, STG-NOLM gates, and STG-PSI-NOLM gates. The significance of the study was to develop an understanding, of how much the self and cross Raman effects affect the switching characteristics of STGs, SDGs, STG-NOLM gates, and STG-PSI-NOLM gates. These optical fiber gates have the potential of being used in optical telecommunication networks, for all-optical header processing in such networks as the Asynchronous Transfer Mode Broadband Integrated Services Digital Networks (ATM-BISDN).

Public telecommunication user bandwidth demands have been and are still on the increase, from the analogue telephone subscriber bandwidths, to the current ATM-BISDN subscriber bandwidths. This is because the user through user applications in the computer and communications fields invariably drives telecommunication. All-optical telecommunication networks will further increase subscriber bandwidths to meet future bandwidth needs. Demand and utilization of bandwidth, are user application elements. User applications can vary from a simple telephone conversation, to a virtual reality application, or even to the transmission of a holographic image. Figure 1.1 gives the types of applications and their respective bandwidth requirement [1.3].

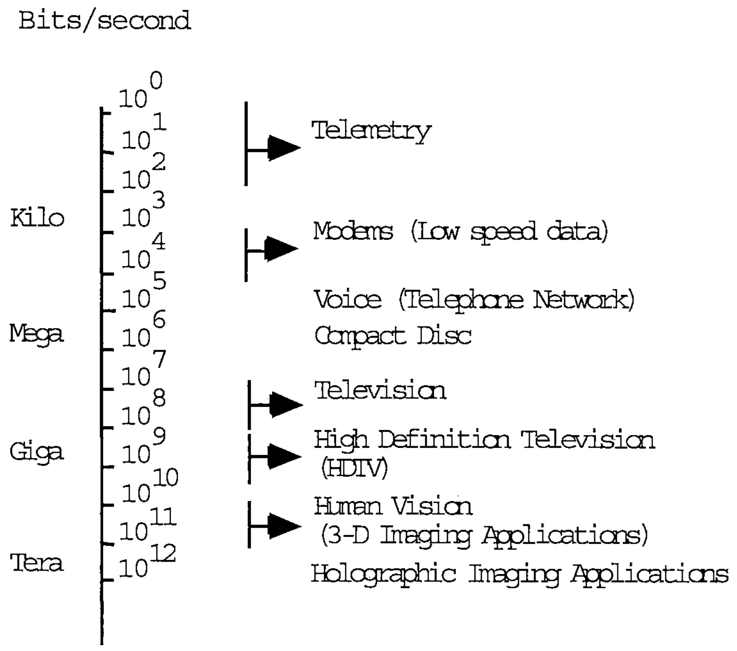


Figure 1.1 Application bandwidth requirements.

By nature, user application bandwidth demands can be conceptually thought of as being endless, but in reality are technology limited. Increase of user application bandwidth entails a corresponding increase in user access and aggregate trunk bandwidths. Currently, broadband network research is addressing some of the issues of increasing the user access and aggregate trunk bandwidths. The ATM-BISDN user access bandwidth of about 600 Mbps, will be followed by multi-gigabit user access rates, such as those envisaged in the Stockholm multi-gigabit network [1.4], and the IBC user optical network [1.5]. These are attempts by public and private network providers to increase user access and aggregate trunk bandwidths, so as to satisfy user application bandwidth demands. The ultimate goal of any telecommunication technology would be the provision of an inexpensive, quality, and large user bandwidth, making bandwidth the cheapest commodity in the telecommunication system. The only telecommunication technology currently, that has the potential capability of providing a cheap, quality and large bandwidth, is the all-optical telecommunication technology. The optical fiber has a potential low-loss window bandwidth of about 40 THz. All-optical switching devices have the potential capability of switching in the sub-picosecond region. These elements are near perfect building blocks for an all-optical terabit rate network, capable of delivering a cheap, quality and large bandwidth to the user.

All-optical switching devices have the potential of providing links that can support "ultrafast" bit rates. Ultrafast represents bit rates greater than 50 Gbps, or at least beyond bit rates achievable by future electronic systems. It has been predicted that Very Large Scale Integration (VLSI)-like electronics may be limited above 35 GHz, because of fundamental considerations such as transit, relaxation and thermal diffusion times [1.6]. Recent research in long-haul intensity modulation/direct detection (IM/DD) telecommunication systems [1.1, 1.7-1.9], have shown that the ideal pulse and medium for transmission, at ultrafast bit rates in long-haul IM/DD telecommunication systems, is the soliton and single-mode dispersion shifted (DS) optical fiber respectively. The single mode DS optical fiber is operated in the anomalous (negative) group velocity regime. Solitons are pulses that propagate nearly distortion-free for long distances in single-mode optical fibers under zero average power loss, and neglecting higher order non-linear and dispersion effects.

Soliton transmission and switching via all-optical switching devices such as the STGs, SDGs, STG-NOLM gates, and STG-PSI-NOLM gates, are capable of delivering ultrafast bit rates. However, in order to reach ultrafast bit rates, compact low power all-optical switching devices need to be developed. This is the motivation for studying the cross Kerr and Raman effects on the switching efficiency of these optical fiber gates. This is the first study to date, which has included the effect of cross steepening (CS) [1.77] that arises from the cross Raman effects. Included in this study is another cross Raman effect, cross frequency shifting (CFS), discovered by Menyuk *et al.* [1.10].

1.2 PROBLEM DEFINITION

This dissertation focuses on the cross Kerr and Raman effects on the switching efficiency of the STGs, SDGs, STG-NOLM gates, and STG-PSI-NOLM gates, for sub-100 fs pulse operation. This section provides the scope, limitations and logical assumptions upon which the chapters on analytical and numerical modeling are based on.

1.2.1 Basic Nonlinear Wave Propagation Theory

Nonlinear phase shift in optical fibers is due to the nonlinear response of the optical fiber under intense electromagnetic fields. The fundamental origin of the nonlinear response is due to the anharmonic motion of bound electrons under the influence of an intense applied field. This results in an induced polarization $\boldsymbol{P}$ from the electric dipoles, which is nonlinear in the electric field $\boldsymbol{E}$. The total polarization $\boldsymbol{P}$ can be written as the sum of the linear and nonlinear polarization components as $\boldsymbol{P}(\boldsymbol{r},t) = \boldsymbol{P}_L(\boldsymbol{r},t) + \boldsymbol{P}_{NL}(\boldsymbol{r},t)$, where $\boldsymbol{P}_{NL}$ and $\boldsymbol{P}_L$ are linear and nonlinear polarization components respectively. Using Maxwell's differential equations, the electromagnetic propagation in an optical fiber can be obtained as follows.

$$\nabla \times \boldsymbol{E} = -\frac{\partial \boldsymbol{B}}{\partial t} \tag{1.1}$$

$$\nabla \times \boldsymbol{H} = \frac{\partial \boldsymbol{D}}{\partial t} + \boldsymbol{J}_c \tag{1.2}$$

$$\nabla \cdot \boldsymbol{D} = q_{ev} \tag{1.3}$$

$$\nabla \cdot \boldsymbol{B} = q_{mv} \tag{1.4}$$

$$\boldsymbol{D} = \varepsilon_0 \boldsymbol{E} + \boldsymbol{P} \tag{1.5}$$

$$\boldsymbol{B} = \mu_0 \boldsymbol{H} + \boldsymbol{M} \tag{1.6}$$

E and ***H*** are the electric and magnetic field strengths respectively. ***B*** and ***D*** are the magnetic and electric flux densities respectively. ***P*** and ***M*** are the electric and magnetic polarizations respectively. $\boldsymbol{J_C}$ is the conduction current density. The electric and magnetic volume charge densities are q_{ev} and q_{mv} respectively. A nonmagnetic optical fiber in the absence of free charges gives $\boldsymbol{M}=\boldsymbol{J_C}=q_{ev}=q_{mv}=0$. This gives the propagation wave equation as

$$\nabla \times \nabla \times \boldsymbol{E} = -\frac{1}{c^2}\frac{\partial^2 \boldsymbol{E}}{\partial t^2} - \mu_0 \frac{\partial^2 \boldsymbol{P}}{\partial t^2}, \tag{1.7}$$

where $1/c^2=\mu_o\varepsilon_o$ and c is the velocity of light in a vacuum. The electric polarization ***P*** satisfies the general relation [1.11]

$$\boldsymbol{P} = \varepsilon_0\left(\chi^{(1)} \cdot \boldsymbol{E} + \chi^{(2)}\vdots\boldsymbol{EE} + \chi^{(3)}\vdots\boldsymbol{EEE}+\ldots\right), \tag{1.8}$$

where $\chi^{(1)}$ is the linear first order susceptibility (second-rank tensor), $\chi^{(2)}$ and $\chi^{(3)}$ are the second (third-rank tensor) and third (fourth-rank tensor) order susceptibilities respectively.

Optical fiber consists of silica, $[SiO_2]_n$, which has a centrosymmetric macromolecular crystalline structure that possesses inversion symmetry. Crystalline structures that possess inversion symmetry have the second-order nonlinear susceptibility $\chi^{(2)}$ vanishing identically [1.12]. Due to this optical fibers do not exhibit second-order nonlinear effects. The lowest order nonlinear effects in optical fibers originate from the third-order susceptibility $\chi^{(3)}$, which is responsible for third-harmonic generation, four-wave mixing, nonlinear refraction and nonlinear birefringence. The dominant nonlinear effect in optical fibers is the intensity dependence of the refractive index. The intensity dependence of the refractive index results from the contribution of $\chi^{(3)}$, such that the refractive index in an optical fiber supporting a single pulse becomes

$$n(\omega,|\boldsymbol{E}|^2) = n_o(\omega) + n_2|\boldsymbol{E}|^2 . \tag{1.9}$$

Where $n_o(\omega)$ is the linear frequency dependent refractive index, $|E|^2$ is the optical electrical field intensity inside the fiber, and n_2 is the nonlinear refractive index coefficient which is a function of $\chi^{(3)}$.

The intensity dependence of the refractive index gives rise to two most widely studied nonlinear effects, namely self-phase modulation (SPM) and cross-phase modulation (CPM). SPM refers to the self-induced phase shift experienced by an optical field during its propagation in optical fibers. The magnitude of the phase shift through an optical fiber of length L can be obtained from

$$\phi = n(\omega,|\boldsymbol{E}|^2)\beta_z L = \left(n_o(\omega) + n_2|\boldsymbol{E}|^2\right)\beta_z L , \tag{1.10}$$

where β_z=$2\pi/(\lambda/n_o(\omega))$ is the propagation constant in the direction parallel to the fiber length L. The optical wavelength is denoted as λ. The nonlinear phase shift due to SPM is determined from

$$\phi_{NL} = n_2|\boldsymbol{E}|^2 \beta_z L . \tag{1.11}$$

One form of CPM is the nonlinear phase shift on an optical field induced by co-propagating optical fields at different wavelengths [1.13-1.17]. This form of CPM is the effect of the average optical field, due to two optical fields at frequencies ω_1 and ω_2, on the nonlinear refractive index coefficient n_2. The total optical field polarized in the x-axis may be represent as

$$\boldsymbol{E} = \frac{1}{2}\hat{\boldsymbol{a}}_x\left[\boldsymbol{E}_1 e^{-j\omega_1 t} + \boldsymbol{E}_2 e^{-j\omega_2 t} + \boldsymbol{E}_1^* e^{j\omega_1 t} + \boldsymbol{E}_2^* e^{j\omega_2 t}\right]. \tag{1.12}$$

The nonlinear phase shift on the optical fields at ω_1 and ω_2, are given by [1.18]

$$\phi_{NL1} = n_2\left(\left|\boldsymbol{E}_1\right|^2 + 2\left|\boldsymbol{E}_2\right|^2\right)\beta_z L \tag{1.13}$$

and

$$\phi_{NL2} = n_2\left(\left|\boldsymbol{E}_2\right|^2 + 2\left|\boldsymbol{E}_1\right|^2\right)\beta_z L \tag{1.14}$$

respectively, neglecting all terms that generate polarization at frequencies other than ω_1 and ω_2, because of their non-phase matched character. The nonlinear phase shift arising from the first term and the second term on the right-hand side of (1.13) and (1.14) is due to SPM and CPM respectively. The second form of CPM is the nonlinear phase shift due to the co-propagation of two optical fields at the same wavelength, but different polarizations. In this form of CPM the nonlinear phase shift is due to the effect of the average optical field, of the two orthogonally polarized optical fields, on the nonlinear refractive index coefficient n_2 [1.18]. The total electric field of two orthogonally polarized optical pulses at frequency ω_1 can be represented by

$$\boldsymbol{E} = \frac{1}{2}\left[\left(\hat{\boldsymbol{a}}_x\boldsymbol{E}_x + \hat{\boldsymbol{a}}_y\boldsymbol{E}_y\right)e^{-j\omega_1 t} + \left(\hat{\boldsymbol{a}}_x\boldsymbol{E}_x^* + \hat{\boldsymbol{a}}_y\boldsymbol{E}_y^*\right)e^{j\omega_1 t}\right]. \tag{1.15}$$

The nonlinear phase shift on the optical fields in the x and y polarizations, are given by

$$\phi_x = n_2\left(\left|\boldsymbol{E}_x\right|^2 + \frac{2}{3}\left|\boldsymbol{E}_y\right|^2\right)\beta_z^x L \tag{1.16}$$

and

$$\phi_y = n_2\left(\left|\boldsymbol{E}_y\right|^2 + \frac{2}{3}\left|\boldsymbol{E}_x\right|^2\right)\beta_z^y L \tag{1.17}$$

respectively. β_z^x and β_z^y are the propagation constants in the direction parallel to the fiber length L in the x and y polarization axes.

The second form of CPM is the physical mechanism responsible for soliton-dragging in SDGs and soliton-trapping in STGs, by which time-shifting keying switching logic is effected. The effects of higher order nonlinear and dispersion effects affect the operation of STGs, SDGs, STG-NOLM gates, and STG-PSI-NOLM gates.

1.2.2 Higher-Order Nonlinear and Dispersion Effects

The soliton pulses that were studied were less than 100 fs (FWHM). For ultrashort pulse widths less than 100 fs, higher-order dispersion and nonlinear effects come into play. This is because the spectral width of the ultrashort pulses becomes comparable to the carrier frequency, thus requiring a higher accuracy analytical and numerical model. Also the spectrum of such pulses are wide enough (approximately greater than or equal to 5 THz) for the Raman gain to amplify low frequency components, by transferring energy from the high-frequency components of the same pulse.

Group velocity dispersion (GVD) effects are due to dependence of the linear refractive index on frequency, which can be approximated by the Sellmeier equation [1.19] for areas far from medium resonances, namely

$$n^2(\omega) = 1 + \sum_{j=1}^{m} \frac{B_j \omega_j^2}{\omega_j^2 - \omega^2}. \tag{1.18}$$

Where ω_j is the resonance frequency and B_j is the strength of the jth resonance. The effects of fiber dispersion can be accounted for by the Taylor series expansion of the propagation constant β, about the center frequency ω_o as follows

$$\beta(\omega) = \frac{n(\omega)\omega}{c} = \beta_0 + \frac{\beta_1(\omega-\omega_0)}{1!} + \frac{\beta_2(\omega-\omega_0)^2}{2!} + ... + \frac{\beta_m(\omega-\omega_0)^m}{m!} + ..., \tag{1.19}$$

where

$$\beta_m = \left(\frac{d^m\beta}{d\omega^m}\right)_{\omega=\omega_0} \quad \text{for } m=0,1,2,3,..., \tag{1.20}$$

$$\beta_1 = \frac{d\beta}{d\omega} = \frac{1}{c}\left(n + \omega\frac{dn}{d\omega}\right) = \frac{n_g}{c} = \frac{1}{v_g}, \tag{1.21}$$

$$\beta_2 = \frac{d\beta_1}{d\omega} = \frac{d}{d\omega}\left(\frac{1}{v_g}\right), \text{ and} \tag{1.22}$$

$$\beta_3 = \frac{d\beta_2}{d\omega} = \frac{d^2}{d\omega^2}\left(\frac{1}{v_g}\right). \tag{1.23}$$

Where v_g is the group velocity, n_g is the group refractive index and β_2 the GVD parameter. For ultrashort pulse widths less than 100 fs, higher-order dispersion effects is due to the third-order dispersion, namely β_3 [1.20, 1.21] must be included. The nonlinear effects that come into play for ultrashort pulses less than 100 fs are self-steepening (SS) [1.22-1.30], and self-frequency shifting (SFS) [1.31, 1.32]. SS and SFS are due to the Kerr and Raman effects respectively. SS of an optical pulse results from the intensity dependence of the group velocity [1.22, 1.23, 1.33, 1.34] and leads to asymmetry in the SPM-broaden spectra [1.35-1.39]. The amplification of low frequency components by the Raman gain, by transferring energy from the high-frequency components of the same pulse is known as SFS.

Kodama and Hasegawa [1.40] derived a higher-order nonlinear Schröndinger (HNLS) equation for vector fields, based on the asymptotic perturbation technique developed by Taniuti *et al.* [1.41]. Their analytical model of the nonlinear propagation of an optical pulse in a singlemode fiber includes the effects of SS and CFS, and is given by

$$\begin{aligned}\frac{\partial A}{\partial z}+\beta_1\frac{\partial A}{\partial t}+\frac{i}{2}\beta_2\frac{\partial^2 A}{\partial t^2}+\frac{\alpha}{2}A=i\gamma|A|^2A+\frac{1}{6}\beta_3\frac{\partial^3 A}{\partial t^3}\\ -a_1\frac{\partial(|A^2|A)}{\partial t^3}-a_2A\frac{\partial(|A|^2)}{\partial t^3},\end{aligned} \tag{1.24}$$

which can used to study the effects of SS and SFS. Equation (1.24) is applicable to for a single optical pulse propagating in a singlemode fiber. For two orthogonally polarized pulses at the same frequency, requires equation (1.24) to be modified to introduce the cross effects due to the Kerr and Raman effects.

1.2.3 Cross Kerr and Raman Effects

The cross Kerr and Raman effects, in the co-propagation of two orthogonally polarized optical pulses in a singlemode fiber, are due to the interaction of the electric field in one polarization axis with the electric field in the other axis through the cubic nonlinear susceptibility $\chi^{(3)}$. Using a quasi-monochromatic assumption, the Fourier transform of the cubic nonlinear susceptibility $\tilde{\chi}^{(3)}$ can be Taylor series expanded around the central frequencies of the three interacting electric field modes $\omega_i=\omega_{l_i}+\Delta\omega_i$, for i=1, 2, 3, as by [1.42]

$$\tilde{\chi}^{(3)}(\omega_1,\omega_2,\omega_3)=\tilde{\chi}^{(3)}(\omega_{l_1},\omega_{l_2},\omega_{l_3})+\sum_{j=1}^{3}\frac{\partial\tilde{\chi}^{(3)}}{\partial\omega_j}\Delta\omega_j+\dots \qquad (1.25)$$

The sum of the frequencies ω_1, ω_2 and ω_3 is equal to the center frequency ω_0 of the co-propagating orthogonally polarized pulses, and their respective magnitudes are equal to the magnitude of ω_0. The leading term in (1.25) represents the Kerr effect and the next order terms, the Raman effect, representing the scattered frequency. The cross Kerr effects arise from the interaction of the leading term in (1.25) with three electric field modes. Similarly the cross Raman effects arise from the interaction of the next order terms with three electric field modes. The three electric field modes originate from the two orthogonal polarizations.

The quantum view of the Raman effect in optical fibers [1.10], like the Kerr effect, is due to the nonlinear polarizability generated by absorption of two photons and the emission of one or the absorption of one photon and the emission of two. All photons in the optical pulse have frequencies close to the pulse carrier frequency, ω_0. Optical fibers have no quantum states at energies near $\hbar\omega_0$ or $2\hbar\omega_0$ in the anomalous dispersion regime in which solitons propagate. Energy levels due to nuclear interactions are far lower, and energy levels due to electronic interactions are far higher. There are three processes that contribute to the positive frequency portion of the polarizability. In the first type, shown in Fig. 1.2(a), two photons are absorbed and then one is emitted. The process in Fig. 1.2(a) is the fastest of the three processes, and is necessarily instantaneous in comparison with the other two processes. This can be observed from the large delay gap along the horizontal axis in Fig. 1.2(b) and Fig. 1.2(c). In the process in Fig. 1.2(a), the molecular system does not return to the ground state. In the second and third processes, shown in

Fig. 1.2(b) and Fig. 1.2(c), there is an absorption and emission in any order, followed by another absorption. In these processes, the molecular system returns near the ground state before the last absorption. There are also three accompanying conjugate processes, not shown here, with two emissions and one absorption, which contribute to the negative frequency portion of the polarizability. The quantum states in the processes in Fig. 1.2, whose energies differ from the ground state by multiplies of $\hbar\omega_0$, are virtual since the optical fiber only has quantum states that satisfy $\hbar\omega_j << \hbar\omega_0$ or $\hbar\omega_j >> \hbar\omega_0$. Their lifetimes in these virtual states are limited to ω_0^{-1} by the uncertainty principle. The Raman effect is associated with processes with a finite time delay, and the typical Raman response times for silica fibers is in the range of 2 to 4 fs.

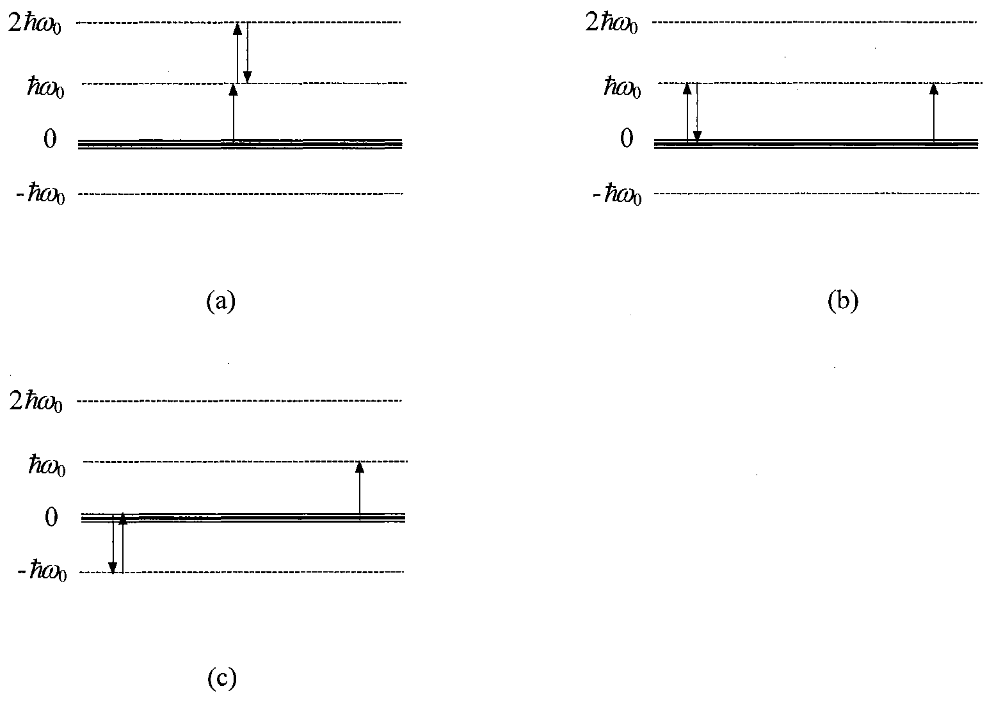


Figure 1.2 Molecular transitions for the polarizability positive frequency component in the Kerr and Raman effects. Solid and dashed lines represent the nuclear and virtual levels respectively.

The study of the effects arising from the cross Kerr and Raman effects, on the switching efficiency of STGs, SDGs, STG-NOLM gates, and STG-PSI-NOLM gates, is the basis of this dissertation.

1.2.4 Soliton Dragging and Trapping Gates

The physical mechanism of operation of a SDG and a STG is CPM [1.15,1.43]. The basic physical structure of the SDG and STG consists of a singlemode fiber with a moderate to strong birefringence.

In a SDG, two soliton pulses (unequal amplitudes) that are orthogonally polarized are injected simultaneously into the fiber to interact through CPM. This results in a carrier frequency chirp in the two pulses. The birefringence of the fiber generates polarization dispersion through propagation (dispersive delay line) of the pulses. This translates the frequency shift into a time shift. The pulse along the slow axis speeds up, while the pulse along the fast axis slows down. The control pulse amplitude is greater than the signal pulse amplitude in a SDG. Assuming that a signal is a pulse with guard bands surrounding its time slot. Then a logic "1" corresponds to a pulse that arrives within the clock window, and a logic "0" to no signal or an improperly timed pulse. For the inverter SDG/STG in Figure 1.3, the fiber length is trimmed so that in the absence of any signal the control pulse *C* arrives within the clock time window and corresponds to a logic "1" at the output as *F*. When the signal *S* is present, it interacts with the control pulse through soliton dragging/trapping and pulls *C* out of the clock time window to give a logic "0" at the output *F*.

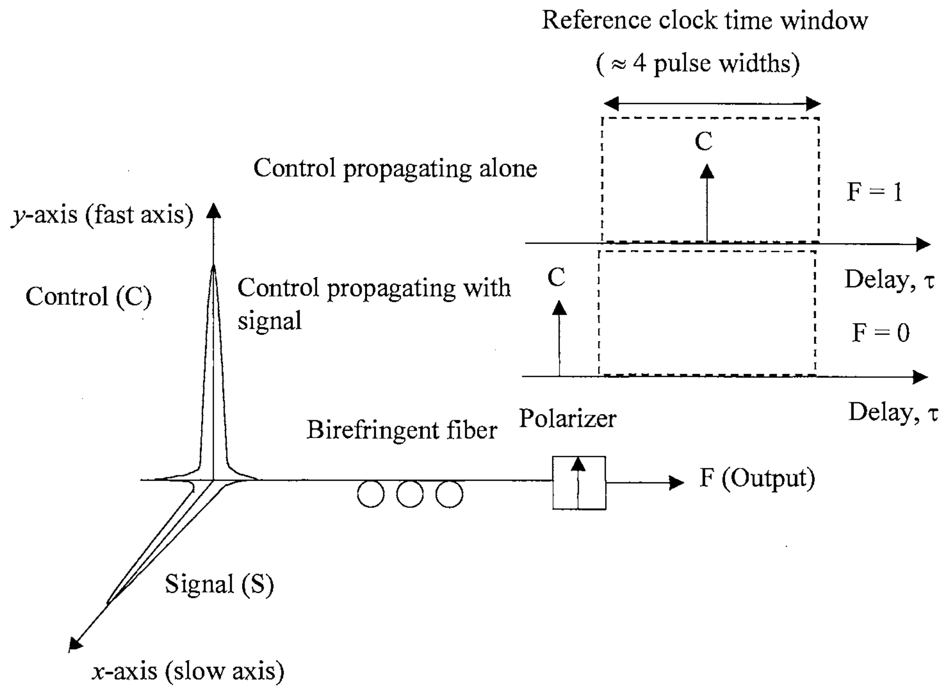


Figure 1.3 Physical structure of an inverter SDG/STG.

For the STG, the orthogonally polarized soliton pulses (equal amplitudes) trap one another through CPM, but here intensity dependent effects compensate the birefringence [1.44,1.45]. For equal pulse amplitudes the two solitons have their carrier frequency chirped equally but in opposite directions. Through GVD, which is a function of carrier frequency, the soliton along the fast axis slows down and the soliton along the slow axis speeds up, to attain an average speed at which they are both trapped. This also leads to an improper arrival time of the control pulse *C* at the output *F*. Thus implementing the same logic operation as in the SDG.

1.2.5 STG-NOLM and STG-PSI-NOLM Gates

The STG-NOLM combines the operation of STG and a NOLM in one device, and the SDG-NOLM also combines the operation of a SDG and a NOLM in one device. A NOLM [1.57], also known as a nonlinear sagnac interferometer, is an all-optical nonlinear Mach-Zehnder interferometer. An interferometer requires a π-phase shift between two arms to switch from constructive to destructive interference. In the NOLM, the π-phase shift is obtained by a nonlinear phase shift due SPM or CPM. The NOLM is a four-port directional coupler in which two ports on one side are connected by a continuous loop of fiber as shown in Fig. 1.4. The two arms of the interferometer correspond to the two counter-propagating directions around the loop, and this configuration is very stable since both arms involve exactly the same optical path. When the cross-coupler divides the input equally, the NOLM acts as a perfect mirror. For splitting ratios other than 50:50, the phase shift is different in the two directions, and the NOLM acts as an intensity dependent mirror.

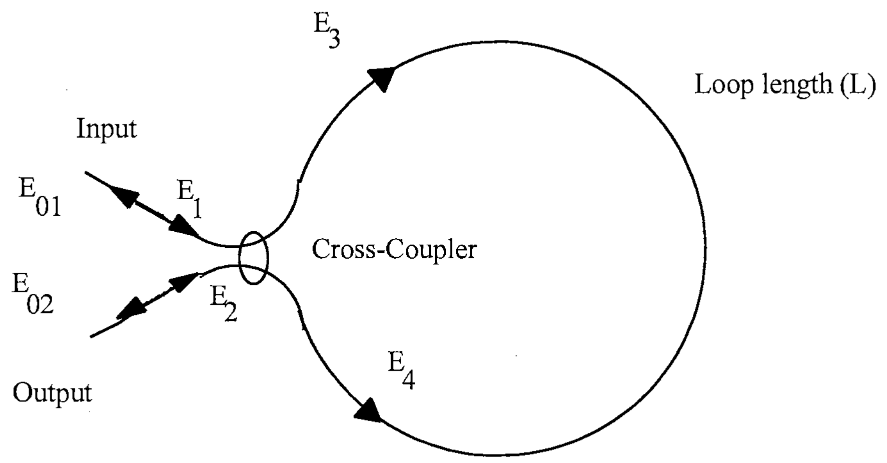


Figure 1.4 Physical structure of a NOLM.

The NOLM can be modified to produce a control dependent switch or gate. This can be done by the introduction of two polarizing beam splitters near the cross-coupler to introduce and remove a signal pulse, with the control pulse entered at the input port [1.74,1.76]. The cross-coupler splitting ratio is set at 50:50. Fig. 1.5 shows the physical structure of a STG-NOLM gate. The clockwise control pulse in the presence of a signal, from the inlet polarizing beam splitter, experiences CPM. If the clockwise control pulse accumulates a π-phase shift, then the control pulse will experience constructive inference into the output port, otherwise the control pulse will be reflected back into the input port. The structure of an SDG-NOLM gate is similar, except that it uses a SDG instead of a STG.

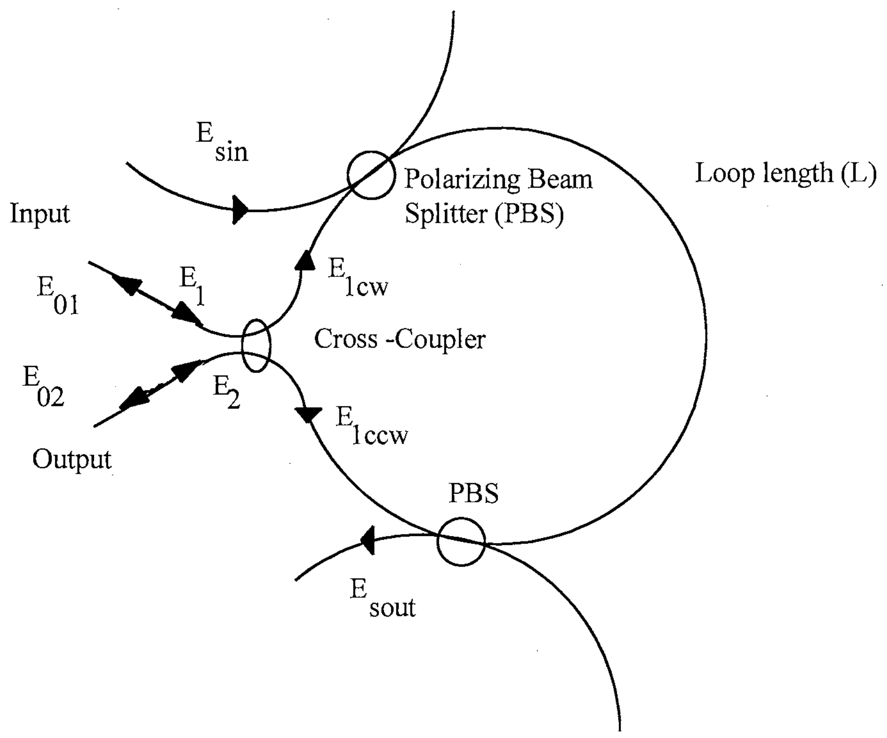


Figure 1.5 Physical structure of a STG-NOLM gate.

In this dissertation the SDG-NOLM is not studied, because it requires a much longer interaction length, in the NOLM, due to the signal amplitude being smaller than the control pulse. The smaller signal amplitude produces a smaller phase shift. To increase CPM interaction between the clockwise control and signal pulses, the length of fiber in the NOLM can be segmented with cross-splices. The cross-splices, produced by rotating a mandrel by 90° with respect to each axis, flips the fiber *x*-axis and *y*-axis. This forces the pulses to walk through each other, in each segment, in the absence of the trapping effect. If trapping is avoided in the STG-NOLM and SDG-NOLM, a complete shuffle interaction (CSI) technique can be used, where a series of complete slip through CPM interactions occur, between the clockwise control and signal pulses. However, to avoid trapping in the STG-NOLM and SDG-NOLM, for sub-100 fs pulses, a very large birefringence is required, resulting in very many and very short interaction segment lengths. The strong trapping effect is capable of inducing pulse distortions that are undesirable. To initially avoid the pulse distortions in the STG-NOLM, where the trapping effect is present, a partial shuffle interaction (PSI) technique can be used. The PSI technique involves as series of partial slip through CPM interactions between constantly overlapping clockwise control and signal pulses. The PSI technique is a newly discovered technique that has been studied here for the first time. The STG-PSI-NOLM, is a STG-NOLM featuring the PSI technique to avoid trapping in the initial 5 to 10 walk-off lengths, and thus provides better switching characteristics.

1.3 LITERATURE REVIEW

Ultrafast optical fiber switching devices can be categorized into two main classes namely, routing switches and logic switches. Control for routing switches is typically in a different physical format than the data. On the other hand, control for the logic switches is in the same physical format as the data. Another difference between routing and logic switches is that routing switches base their decision on position, where as logic switches base their decision on digital logic. A number of optical fiber based routing switches have been studied and developed in the laboratory for example Kerr gates, four-wave-mixing gates, nonlinear directional couplers, and Mach-Zehnder interferometer or nonlinear optical loop mirror (NOLM). All these switches depend upon the nonlinear refractive index n_2 for their operation. Kerr gates use the change in polarization-state due to the intensity dependent refractive index n_2 [1.46, 1.50]. Four-wave-mixing gates use two closely spaced frequencies from the control and signal beams interacting through the nonlinearity to generate new Stokes and anti-Stokes side-band frequencies [1.51-1.53]. Nonlinear directional couplers use the intensity dependent nonlinear refractive index n_2 change to block normal coupling between guides [1.54-1.56]. The NOLM uses the phase difference generated by the intensity dependent nonlinear refractive index n_2 between two channels [1.57-1.62]. Examples of logic switches are digital soliton gates, that depend on all-optical soliton interaction in fibers such as soliton-dragging gates (SDG), soliton-interaction gates (SIG), and soliton-trapping gates (STG) [1.2].

Research on the NOLM and the nonlinear amplifying loop mirror (NALM) based switches have used the phase difference induced by the nonlinear refractive index through SPM or CPM. A NALM is basically a NOLM with a gain medium inserted in the loop of the NOLM. The NOLM was first proposed by Doran and Wood [1.57], and was based on SPM. Mollenauer and Mitschke first discovered the soliton SFS in an optical fiber, for sub-picosecond pulses [1.63]. The effects of soliton SFS on a NOLM switch, were observed by Islam, *et al.* for sub-picosecond soliton pulses [1.59]. It was noted that the frequency shift due to SFS asymmetrizes the pulses and temporally separates the two counter propagating pulses in the NOLM, which in turn reduces the output coupling efficiency. The first studies that showed how the output coupling efficiency of a NALM, using ultrashort soliton-like pulses, are affected by third order dispersion (TOD), SS and SFS was conducted by Pearson, *et al.* [1.64]. Their studies show that the output coupling efficiency of a low gain and a high gain NALM are sensitive and insensitive respectively to the pulse width for a dispersion shifted, or dispersion flattened, or standard fiber. Pearson, *et al.*, results also show that the output coupling efficiency of a NALM is much lower at pulse widths less than 100 fs, due to TOD, SS, and SFS. Very recently research by Lee, *et al.* [1.65], have demonstrated a walk-off balanced NOLM switch of 35 ps pulses with 4 W peak power. In their configuration, two counter-propagating control beams are used to balance out the nonlinear phase shifts induced by each beam and compensate the walk-off effects occurring in each of the interferometric arms. This leads to an improved switching speed for the NOLM.

Islam *et al.* [1.66-1.68], who invented soliton gates, have conducted a series of experiments on soliton gates. They obtained switching energy as low as 1 pJ with a pulse duration of 500 fs, and an inter-pulse spacing of 10 times the pulse duration [1.69]. There are two types of soliton gates that operate on two orthogonally polarized soliton pulses, namely the soliton-dragging gate (SDG) and the soliton-trapping gate (STG). In the SDG, the velocity of the control pulse changes when interacting with the signal pulse, but ultimately the two pulses separate. In the STG, the velocity shift is sufficient to lock the signal pulse and the control pulse together. The detailed equations governing the operation of the SDG/STG were derived by Menyuk [1.44, 1.45, 1.70], without the inclusion of higher order linear and nonlinear effects due to TOD, SS, SFS, CS, and CFS. Theoretical and numerical work carried out by Islam, Menyuk *et. al.* [1.10, 1.71] and Headley III and Agrawal [1.72], show that the effects of SFS and CFS delay the soliton pulse arrival time, irrespective of polarization axis. Very recently another fiber soliton gate, operating through a large frequency shift caused by CPM, and birefringent walk-off (polarization dispersion) has been demonstrated [1.73]. No other work to date has studied the effects of TOD, SS, SFS, CS and CFS on the switching efficiency of the STGs and SDGs, with sub-100 fs pulses. For sub-100 fs soliton pulses the soliton period is much shorter. This implies a lower latency for the STG/SDG and higher bit rates. Unfortunately this also introduces higher linear and nonlinear effects due to TOD, SS, SFS, CS and CFS.

A SDG-NOLM gate has been numerically demonstrated by Williams, *et al.* [1.74]. They simulated a SDG with a polarization maintaining low-birefringence fiber in a NOLM gate. Their SDG-NOLM gate is similar to the NOLM gate by Moores, *et al.* [1.61], and Whitaker, *et al.* [1.75], except that Williams, *et al.* incorporate a long interaction length between orthogonally polarized solitons, through the use of a fiber with low-birefringence. The fiber with low-birefringence gives a long walk-off length, requiring fewer exchanges of the birefringent axes. The slow walk-off in conjunction with CPM leads to soliton dragging (frequency chirp due to CPM and walk-off). This configuration combines the advantages of a SDG and a NOLM gate, and results in a logic gate with a low switching energy and increased tolerance of timing fluctuations. A SDG-

NOLM has experimentally been demonstrated by Islam et. al. [1.76]. No work to date has studied the effects of TOD, SS, SFS, CS and CFS on the switching efficiency of the STG-NOLM or STG-NOLM, with sub-100 fs pulses. In this dissertation the SDG-NOLM is not studied, because it requires a much longer interaction length, in the NOLM, due to the signal amplitude being smaller than the control pulse. The smaller signal amplitude produces a smaller phase shift.

This dissertation studies the effect of higher order nonlinear and dispersion effects, on the switching efficiency of four types of soliton based optical fiber gates, namely STGs, SDGs, STG-NOLM gates, and STG-PSI-NOLM gates.

1.4 DISSERTATION ORGANIZATION

The dissertation is organized in a logical format, starting from the analytical modeling of the cross Kerr and Raman effects in Chapter II, through numerical modeling in Chapter III, to the computer simulation algorithms implemented in the study, in Chapter IV. Chapter V presents the results obtained in Phase I and Phase II. Results from Phase I are based on simulations of a single femtosecond soliton pulse propagating in a singlemode fiber SMF01. The singlemode fiber SMF01 is a simulated birefringent dispersion shifted optical fiber that can be physically manufactured. Chapter I, which presents standard results of a single femtosecond soliton pulse propagating in a singlemode fiber [1.18], was used as a simulation control model. In Chapter II, the analytical model of the singlemode fiber SMF01 is presented with its dispersion characteristics. The singlemode fiber SMF01 is also used in Phase II as the medium of propagation. Results from Phase II are based on the co-propagation of two orthogonally polarized femtosecond pulses in the singlemode fiber SMF01, operating as either a soliton-dragging gate (SDG) or soliton-trapping gate (STG).

In Chapter II the reductive perturbation method is presented and subsequently used to derive the cross Kerr effect terms and cross Raman effect terms. The physical meaning of the cross Kerr effect terms and cross Raman effect terms are then presented. Chapter II proceeds to give the analytical model of the singlemode fiber SMF01 with its dispersion characteristics. Also in Chapter II, the analytical models for the time-shift keying switching logic for SDGs and STGs, and the interferometric switch of a NOLM are given.

In Chapter III, the symmetrized split-step Fourier method is presented. Chapter III proceeds to give the numerical models used later to develop the computer algorithms, for the single soliton pulse propagation and the co-propagation of two orthogonally polarized solitons. The numerical models represent how the Kerr and Raman effects have been implemented. The accuracy of the numerical models is also presented here.

Chapter IV gives the computer simulation algorithms used to implement the numerical propagation models in Chapter III. The program modules are briefly described to give their functionality, and their comprehensive listing is given in Appendix C.

Chapter V presents the results for Phase I, Phase II, and Phase III. The effects due to the cross Kerr and Raman effects on the switching efficiency of SDGs, STGs, STG-NOLM gates and STG-PSI-NOLM gates are presented qualitatively and quantitatively.

Chapter VI gives a summary and conclusions on the results obtained from Phase I, Phase II and Phase III. Recommendations for further study of the cross Kerr and Raman effects are presented.

CHAPTER II

ANALYTICAL MODELLING

In order to study the propagation of solitons in a singlemode fiber, appropriate analytical models need to be developed. Two propagation models are derived for the single soliton propagation, and the propagation of two orthogonally polarized solitons. Both analytical models are based on the reductive perturbation method, developed by Asano and Taniuti [2.1-2.4] and as applied by Kodama and Hasegawa [2.5] to nonlinear pulse propagation in singlemode fiber, starting from a three dimensional vector Maxwell equation to a one dimensional scalar equation.

The analytical models of the soliton-dragging and soliton-trapping gates are presented, and followed by the singlemode fiber analytical model with its dispersion characteristics.

2.1 PERTURBATION METHODS

The reductive perturbation method originates from the method of strained parameters, also known as the Lindstedt-Poincaré method [2.6]. Lighthill later developed the Lindstedt-Poincaré method into the method of strained coordinates, also know as Lighthill's technique [2.7, 2.8]. Asano and Taniuti later extended Lighthill's technique to non-dispersive wave propagation in slightly inhomogeneous media to form the reductive perturbation method. Kodama and Hasegawa later applied the reductive perturbation method to the nonlinear pulse propagation in a singlemode optical fiber.

The Lindstedt-Poincaré method and Lighthill's are described by Nayfeh [2.9] and are briefly described in the following sections.

2.1.1 Lindstedt-Poincaré Method

To illustrate the Lindstedt-Poincaré method, consider the perturbation solutions of equations such as

$$\ddot{u} + \omega_o^2 u = \varepsilon f(u, \dot{u}), \qquad \varepsilon << 1 . \tag{2.1}$$

The fundamental idea in the Lindstedt-Poincaré method is based on the observation that the nonlinearities alter the frequency (perturbed parameter) of the system from the linear one ω_0 to the nonlinear one ω (ε). To account for this change in frequency, a new variable, $\tau = \omega t$, is introduced and ω and u are expanded in powers of ε as

$$\begin{aligned} u &= u_0(\tau) + \varepsilon u_1(\tau) + \varepsilon^2 u_2^2(\tau) + \ldots \\ \omega &= \omega_0 + \varepsilon\omega_1 + \varepsilon^2\omega_2^2 + \ldots \ . \end{aligned} \tag{2.2}$$

Equations in (2.2) are substituted into in equation (2.1), and the coefficients of like powers of ε are equated to give equations from the zeroth order of ε to the nth order of ε. The zeroth order solution of u, u_0, from the zeroth order of ε equation is substituted into the first order of ε equation, whose first order solution of u is u_1. The first order solution u_1 is substituted into the second order of ε equation, and so on up to the nth order of ε equation. At the nth order of ε, the parameter ω_n for $n\geq 1$ is selected to prevent the appearance of secular terms (singularities).

2.1.2 Lighthill's Technique

The parameter expansion in the Lindstedt-Poincaré method can be interpreted as a near-identity transformation, which is generalized in Lighthill's technique to render approximate solutions uniformly valid. In Lighthill's technique, a non-uniformity encountered in expanding a function such as $u(x_1, x_2, x_3,\ldots, x_n; \varepsilon)$ in powers of ε, requires the expansion of not only the dependent variable u but also the independent variable exhibiting the non-uniformity, say x_1, in powers of ε in terms of a new independent variable as follows.

$$\begin{aligned} u &= \sum_{m=0}^{N-1} \varepsilon^m u_m(s, x_2, x_3, \ldots, x_n) + O(\varepsilon^N) \\ x_1 &= s + \sum_{m=1}^{N} \varepsilon^m \xi_m(s, x_2, x_3, \ldots, x_n) + O(\varepsilon^{N+1}) \end{aligned} \qquad (2.3)$$

The expansion of x_1 can be viewed as a near-identity transformation from x_1 to s. The straining functions, ξ_m, are determined such that u is uniformly valid, that is $u_m/u_{m-1} \prec \infty \quad \forall x$. If $\xi_m = \omega_m s$ and ω_m are constants, Lighthill's technique becomes the

Lindstedt-Poincaré method. Since Lighthill's transformation strains a coordinate rather than a parameter, thus this technique is called the method of strained coordinates.

2.1.3 Reductive Perturbation Method

Asano and Taniuti extended Lighthill's technique to non-disperse wave propagation in slightly inhomogeneous media as the reductive perturbation method. Kodama and Hasegawa then applied the reductive perturbation method to nonlinear pulse propagation, in a singlemode fiber. A full description of the reductive perturbation method, applied to nonlinear pulse propagation by Kodama and Hasegawa follows [2.5, 2.10].

Starting from Maxwell's equation for the electric field $\boldsymbol{E}$ in the singlemode fiber,

$$\nabla \times \nabla \times \boldsymbol{E} = -\frac{1}{c^2}\frac{\partial^2}{\partial t^2}\left(\chi * \boldsymbol{E}\right). \qquad (2.4)$$

Here χ is the real-space susceptibility tensor of the fiber, that includes the nonlinear response, and $\chi * \boldsymbol{E}$ denotes the correlation integral [2.11]

$$\begin{aligned} \chi * \boldsymbol{E} \equiv & \int_{-\infty}^{t} dt_1 \chi^{(1)}(t-t_1)\cdot \boldsymbol{E}(t_1) \\ & + \int_{-\infty}^{t} dt_1 \int_{-\infty}^{t} dt_2 \int_{-\infty}^{t} dt_3 \chi^{(3)}(t-t_1, t-t_2, t-t_3) \vdots \boldsymbol{E}(t_1)\boldsymbol{E}(t_2)\boldsymbol{E}(t_3). \end{aligned} \qquad (2.5)$$

Here $\chi^{(1)}$ and $\chi^{(3)}$ represent the linear and cubic nonlinear susceptibility tensors, respectively. The susceptibility tensors depend on the spatial coordinates in the transverse direction of the fiber axis. It is assumed that the fiber is isotropic, such that the linear susceptibility $\chi^{(1)}$ can be considered as a scalar valued function. The nonlinear term $\chi^{(3)}$ includes the Kerr and Raman effect with the retardation. The second order susceptibility $\chi^{(2)}$ is negligible since the optical fiber consists of silica, $[SiO_2]_n$, which has a centrosymmetric macromolecular crystalline structure that possesses inversion symmetry. Equation (2.4) can rewritten as

$$\nabla^2 \boldsymbol{E} - \frac{1}{c^2}\frac{\partial^2}{\partial t^2}\left(\chi * \boldsymbol{E}\right) - \nabla(\nabla \cdot \boldsymbol{E}) = 0 . \qquad (2.6)$$

The fibers have an inhomogeneous refractive index, which implies $\nabla\cdot\boldsymbol{E}\neq 0$. This further implies, neither a transverse electric (TE) or a transverse magnetic (TM) field can describe the electric field in an optical fiber. This also means that the Maxwell equation cannot be exactly reduced to a scalar equation, except for a weakly guided optical fiber such a singlemode fiber.

To obtain an approximate one dimensional scalar envelope equation from the three dimensional vector Maxwell's equation, the following assumptions are made:

(1) Quasi-monochromatic mode approximation, with pulse width $\tau = 1/\Delta\omega$, much greater than $1/\omega_1$, where $\Delta\omega$ is the spectral width and ω_1 is the carrier frequency. It is assumed that $\Delta\omega$ is much larger than the phonon spectrum of the fiber such that stimulated Brillouin scattering is sufficiently suppressed.

(2) Singlemode fiber with only one polarization mode present.

(3) Negligible fiber linear and nonlinear birefringence.

The quasi-monochromatic assumption allows the electric field to be written in the form,

$$\begin{aligned} \boldsymbol{E}(t,\, z,\, \boldsymbol{r}) &= \frac{1}{2\pi}\int_{-\infty}^{\infty} \boldsymbol{E}(\omega,\, \boldsymbol{r}) e^{i[k(\omega)z-\omega t]} d\omega \\ &= \boldsymbol{E}_1(\tau,\, \xi,\, \boldsymbol{r};\, \varepsilon) e^{i(k_1 z-\omega_1 t)} + \text{c.c.}, \end{aligned} \qquad (2.7)$$

with the slow varying amplitude expressed as

$$\boldsymbol{E}_1(\tau,\ \xi,\ \boldsymbol{r};\ \varepsilon) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \widetilde{\boldsymbol{E}}(\omega_1 + \Delta\omega,\ \boldsymbol{r}) e^{i\{[k(\omega)-k_1]-(\Delta\omega)t\}} d\omega . \tag{2.8}$$

The vector $\boldsymbol{r}$ is the transverse electric field position vector, and the variation of $\boldsymbol{E}$ is of the first order. Here $\omega = \omega_1 + \Delta\omega$, due to the quasi-monochromatic assumption. The reduced (slow varying) coordinates τ and ξ can be represented in terms of their strained coordinates t (time) and z (distance along fiber axis) respectively, with a perturbation parameter ε included, as follows

$$\tau = \varepsilon(t - k_1' z), \tag{2.9}$$

$$\xi = \varepsilon^2 z . \tag{2.10}$$

The small perturbation ε satisfies the condition $|\varepsilon| = \Delta\omega/\omega_1 << 1$. The first order dispersion parameter at the carrier frequency ω_1 is represented by k_1'. The mode propagation constant $k(\omega)$ (wave number) can be Taylor series expanded as follows.

$$\begin{aligned} k(\omega) = k(\omega)\big|_{\omega_1} + \frac{\partial k}{\partial\omega}\bigg|_{\omega_1}(\omega-\omega_1) + \frac{1}{2!}\frac{\partial^2 k}{\partial\omega^2}\bigg|_{\omega_1}(\omega-\omega_1)^2 \\ + \frac{1}{3!}\frac{\partial^3 k}{\partial\omega^3}\bigg|_{\omega_1}(\omega-\omega_1)^3 + \ldots \\ = k_1 + k_1'\Delta\omega + \frac{1}{2!}k_1''(\Delta\omega)^2 + \frac{1}{3!}k_1'''(\Delta\omega)^3 + \ldots . \end{aligned} \tag{2.11}$$

The electric field $\boldsymbol{E}(z,\ t)$ (inverse Fourier transform) and its Fourier transform $\widetilde{\boldsymbol{E}}(\Delta k,\ \Delta\omega)$ are given by

$$\boldsymbol{E}(z,t) = \frac{1}{(2\pi)^2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \widetilde{\boldsymbol{E}}(\Delta k,\ \Delta\omega) e^{-i(\Delta\omega t - \Delta k z)} d(\Delta k) d(\Delta\omega), \tag{2.12}$$

$$\widetilde{\boldsymbol{E}}(\Delta k,\ \Delta\omega) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \boldsymbol{E}(z,\ t) e^{i(\Delta\omega t - \Delta k z)} dz dt . \tag{2.13}$$

From equations (2.12) and (2.13) it can be seen that $\partial \boldsymbol{E}/\partial t$ and $\partial \boldsymbol{E}/\partial z$ are Fourier transform of $-i\Delta\omega\tilde{\boldsymbol{E}}$ and $i\Delta k\tilde{\boldsymbol{E}}$ respectively. Therefore $\Delta\omega$ and Δk can be written in terms of the partial differential operators as $i\partial/\partial t$ and $-i\partial/\partial z$ respectively.

The total electric field, polarized say in the x axis, can be written as

$$\boldsymbol{E}^x(z,\ t) = \sum_{-\infty}^{\infty} \boldsymbol{E}_l^x(\tau,\xi,r;\varepsilon)e^{i(k_l z-\omega_l t)}, \tag{2.14}$$

with $k_l = lk_1$ and $\omega_l = l\omega_1$. The quasi-monochromatic approximation for the linear response is given by

$$\begin{aligned}
&\int_{-\infty}^{t} dt\chi^{(1)}(t-t_1)\boldsymbol{E}^x(t_1) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\tilde{\chi}^{(1)}(\omega)\tilde{\boldsymbol{E}}^x(\omega)e^{i\{k(\omega)z-\omega t\}}d\omega \\
&= \sum_l \tilde{\chi}^{(1)}(\omega_l+\Delta\omega)\left[\frac{1}{2\pi}\int_{-\infty}^{\infty}\tilde{\boldsymbol{E}}^x(\omega_l+\Delta\omega)e^{i\{[k(\omega)-k_l]z-(\Delta\omega)t\}}d(\Delta\omega)\right]e^{i(k_l z-\omega_l t)} \\
&= \sum_l \tilde{\chi}^{(1)}\left(\omega_l + i\varepsilon\frac{\partial}{\partial\tau}\right)\boldsymbol{E}_l^x(\tau,\ \xi,\ \boldsymbol{r};\ \varepsilon)e^{i(k_l z-\omega_l t)}.
\end{aligned} \tag{2.15}$$

The correspondence $\Delta\omega \leftrightarrow i\varepsilon(\partial/\partial\tau)$ has been used in the integrand. The nonlinear response may be similarly found as follows.

$$\begin{aligned}
&\int_{-\infty}^{t}dt_l\int_{-\infty}^{t}dt_2\int_{-\infty}^{t}dt_3\chi^{(3)}(t-t_1,\ t-t_2,\ t-t_3)\vdots\boldsymbol{E}^x(t_1)\boldsymbol{E}^x(t_2)\boldsymbol{E}^x(t_3) \\
&= \sum_{l,m,n}\tilde{\chi}^{(3)}\left(\omega_l + i\varepsilon\frac{\partial}{\partial\tau_l},\ \omega_m + i\varepsilon\frac{\partial}{\partial\tau_m},\ \omega_n + i\varepsilon\frac{\partial}{\partial\tau}\right)\vdots\boldsymbol{E}_l^x\boldsymbol{E}_m^x\boldsymbol{E}_n^x e^{i(l+m+n)(k_1 z-\omega_1 t)}
\end{aligned} \tag{2.16}$$

The derivatives $\partial/\partial\tau_l$ are defined as $\partial/\partial\tau_l(\boldsymbol{E}_l^x\boldsymbol{E}_m^x\boldsymbol{E}_n^x) = (\partial\boldsymbol{E}_l^x/\partial\tau)\boldsymbol{E}_m^x\boldsymbol{E}_n^x$ for all *l*, *m* and *n*.

In order to construct the one-dimensional scalar wave equation for the electric field $\boldsymbol{E}_l^x(\tau,\ \xi,\ \boldsymbol{r};\ \varepsilon)$, (2.15) and (2.16) are substituted into Maxwell's equation (2.6). Using the quasi-monochromatic assumption, the operators $\partial / \partial\tau$ and $\partial/\partial z$ are replaced as follows.

$$\omega = \omega_l + \Delta\omega \leftrightarrow i\frac{\partial}{\partial t} = \omega_l + i\varepsilon\frac{\partial}{\partial t}, \tag{2.17}$$

$$k(\omega) - k_l = \Delta k_l(\tau) + \Delta k_l(\xi) =\leftrightarrow -i\frac{\partial}{\partial z} = k_l + i\varepsilon k_l^{'}\frac{\partial}{\partial \tau} - i\varepsilon^2\frac{\partial}{\partial \xi}. \tag{2.18}$$

In the sense of the reductive perturbation method, Maxwell's equation is conveniently rewritten for the lth mode as follows.

$$\begin{aligned} &\hat{\boldsymbol{K}}^x\left(k_l + i\varepsilon k_l^{'}\frac{\partial}{\partial\tau} - i\varepsilon^2\frac{\partial}{\partial\xi},\ \omega_l + i\varepsilon\frac{\partial}{\partial\tau},\ \nabla_{x\perp}\right)\boldsymbol{E}_l^x \\ &+\sum_{l_1+l_2+l_3=l}\hat{\boldsymbol{N}}^x\left(\omega_{l_1} + i\varepsilon\frac{\partial}{\partial\tau_{l_1}},\ \omega_{l_2} + i\varepsilon\frac{\partial}{\partial\tau_{l_2}},\ \omega_{l_3} + i\varepsilon\frac{\partial}{\partial\tau_{l_3}}\right)\vdots\boldsymbol{E}_{l_1}\boldsymbol{E}_{l_2}\boldsymbol{E}_{l_3} = 0 \end{aligned} \tag{2.19}$$

The linear part of the operator $\hat{\boldsymbol{K}}^x$ is defined as

$$\begin{aligned} \hat{\boldsymbol{L}}^x(k,\ \omega,\ \nabla_{x\perp}) &= \hat{\boldsymbol{K}}^x\left(k + i\varepsilon k^{'}\frac{\partial}{\partial\tau} - i\varepsilon^2\frac{\partial}{\partial\xi},\ \omega + i\varepsilon\frac{\partial}{\partial\tau},\ \nabla_{x\perp}\right)\Bigg|_{\varepsilon=0} \\ &= \nabla_{x\perp}^2 - k^2 + \frac{\omega^2}{c^2}\tilde{\chi}^{(1)} - \nabla_x(\nabla_x\cdot)\big|_{\partial/\partial z=ik}. \end{aligned} \tag{2.20}$$

Here $\nabla_{x\perp}^2 = \frac{\partial^2}{\partial x^2}$, $\nabla_{x\perp} = \frac{\partial}{\partial x}\hat{\boldsymbol{x}}$ and $\nabla_x = \nabla_{x\perp} + \frac{\partial}{\partial z}\hat{\boldsymbol{z}}$.

The nonlinear operator is defined as

$$\hat{\boldsymbol{N}}^x(\omega_1,\ \omega_2,\ \omega_3) = \frac{(\omega_1+\omega_2+\omega_3)^2}{c^2}\tilde{\chi}^{(3)}(\omega_1,\ \omega_2,\ \omega_3). \tag{2.21}$$

Using the quasi-monochromatic assumption, an expansion of $\tilde{\chi}^{(3)}$ around the central frequencies of the three modes, $\omega_i = \omega_{l_i} + \Delta\omega_i$ for $i = 1, 2$, and 3,

$$\tilde{\chi}^{(3)}(\omega_1,\ \omega_2,\ \omega_3) = \tilde{\chi}^{(3)}(\omega_{l_1},\ \omega_{l_2},\ \omega_{l_3}) + \sum_{j=1}^{3}\frac{\partial\tilde{\chi}^{(3)}}{\partial\omega_j}\Delta\omega_j + \ldots \tag{2.22}$$

The leading term in (2.22) represents the Kerr effect and the next order terms, the Raman effect at scattered frequency $\omega + \Delta\omega_j$. Substituting (2.20) to (2.22) into (2.19) and expanding in powers of ε gives the lth mode envelope equation as follows.

$$\begin{aligned}
&\hat{\boldsymbol{L}}_l^x \boldsymbol{E}_l^x + i\varepsilon \frac{d\hat{\boldsymbol{L}}_l^x}{d\omega_l}\frac{\partial \boldsymbol{E}_l^x}{\partial \tau} - \frac{1}{2}\varepsilon^2 \frac{d^2\hat{\boldsymbol{L}}_l^x}{d\omega_l^2}\frac{\partial^2 \boldsymbol{E}_l^x}{\partial \tau^2} - \frac{i\varepsilon^3}{6}\frac{d^3\hat{\boldsymbol{L}}_l^x}{d\omega_l^3}\frac{\partial^3 \boldsymbol{E}_l^x}{\partial \tau^3} \\
&-\varepsilon^2 \frac{\partial \hat{\boldsymbol{L}}_l^x}{\partial k_l}\left\{ i\frac{\partial \boldsymbol{E}_l^x}{\partial \xi} - \frac{1}{2}k_l''\frac{\partial^2 \boldsymbol{E}_l^x}{\partial \tau^2} - \frac{i\varepsilon k_l'''}{6}\frac{\partial^3 \boldsymbol{E}_l^x}{\partial \tau^3}\right\} \\
&-\varepsilon^3 \frac{d}{d\omega_l}\left(\frac{\partial \hat{\boldsymbol{L}}_l^x}{\partial k_l}\right) i\frac{\partial}{\partial \tau}\left\{ i\frac{\partial \boldsymbol{E}_l^x}{\partial \xi} - \frac{1}{2}k_l''\frac{\partial^2 \boldsymbol{E}_l^x}{\partial \tau^2}\right\} \\
&+\sum_{l_1+l_2+l_3=l}\left\{ \hat{\boldsymbol{N}}_{l_1l_2l_3}^x \vdots \boldsymbol{E}_{l_1}\boldsymbol{E}_{l_2}\boldsymbol{E}_{l_3} + i\varepsilon\sum_{i=1}^{3}\frac{\partial \hat{\boldsymbol{N}}_{l_1l_2l_3}^x}{\partial \omega_{l_i}} \vdots \frac{\partial}{\partial \tau_i}\left(\boldsymbol{E}_{l_1}\boldsymbol{E}_{l_2}\boldsymbol{E}_{l_3}\right)\right\} + \mathbf{O}(\varepsilon^4) = 0
\end{aligned} \tag{2.23}$$

Here the linear operator $\hat{\boldsymbol{L}}_l^x$ and the nonlinear response tensor operator $\hat{\boldsymbol{N}}_{l_1l_2l_3}^x$ are defined as

$$\hat{\boldsymbol{L}}_l^x\left(k_l,\ \omega_l,\ \nabla_{x\perp}\right) = \nabla_{x\perp}^2 - k_l^2 + \frac{\omega_l^2}{c^2}\tilde{\chi}^{(1)} - \nabla_x(\nabla_x \cdot)\Big|_{\partial/\partial z = ik_l}, \tag{2.24}$$

$$\hat{\boldsymbol{N}}_{l_1l_2l_3}^x\left(\omega_{l_1},\ \omega_{l_2},\ \omega_{l_3}\right) = \frac{\omega_l^2}{c^2}\tilde{\chi}^{(3)}(\omega_{l_1},\ \omega_{l_2},\ \omega_{l_3})\,. \tag{2.25}$$

The mode frequency $\omega_l = \omega_1 + \omega_2 + \omega_3$, with $|\omega_i| = \omega_l$ for $i = 1, 2,$ and 3. In the expansion of (2.23) the following identities were employed.

$$\begin{aligned}
&k_l' = \frac{dk_l}{d\omega_l}, \\
&\frac{d\hat{\boldsymbol{L}}_l^x}{d\omega_l} = \frac{\partial \hat{\boldsymbol{L}}_l^x}{\partial \omega_l} + k_l'\frac{\partial \hat{\boldsymbol{L}}_l^x}{\partial k_l}, \\
&\frac{d^2\hat{\boldsymbol{L}}_l^x}{d\omega_l^2} = \frac{\partial^2 \hat{\boldsymbol{L}}_l^x}{\partial \omega_l^2} + 2k_l'\frac{\partial^2 \hat{\boldsymbol{L}}_l^x}{\partial \omega_l \partial k_l} + k_l''\frac{\partial \hat{\boldsymbol{L}}_l^x}{\partial k_l} + (k_l')^2\frac{\partial^2 \hat{\boldsymbol{L}}_l^x}{\partial k_l^2},
\end{aligned} \tag{2.26}$$

and so on. Using the methodology of the reductive perturbation method, $\boldsymbol{E}_l^x(\tau,\ \xi,\ \boldsymbol{r};\ \varepsilon)$ can be expanded in terms of ε as

$$\boldsymbol{E}_l^x(\tau,\ \xi,\ \boldsymbol{r};\ \varepsilon) = \sum_{n=1}^{\infty} \varepsilon^n \boldsymbol{E}_l^{x(n)}(\tau,\ \xi,\ \boldsymbol{r}). \tag{2.27}$$

At each order of ε in equation (2.23), the coefficients are equated and Fredholm's alternative theorem (solvability condition) applied to render the complete expansion uniformly valid, as given in the manner below.

At order ε from (2.23) with l=1

$$\boldsymbol{L}_1^x \boldsymbol{E}_1^{x(1)} = 0. \tag{2.28}$$

The solution to (2.28) is of the form

$$\boldsymbol{E}_1^{x(1)}(\tau,\ \xi,\ \boldsymbol{r};\ \varepsilon) = q_1^{(1)}(\tau,\ \xi)\mathbf{U}_x(\ \boldsymbol{r}). \tag{2.29}$$

It is assumed here that the fiber is a singlemode polarization-maintaining fiber, carrying only one polarization. Let the eigenfunction $\mathbf{U}_x(\boldsymbol{r})$ and $\mathbf{U}_y(\boldsymbol{r})$ describes only the confinement of the pulse in the x and y polarization directions respectively, and the total confinement of the pulse be given by $\mathbf{U}(\boldsymbol{r})$= $\mathbf{U}_x(\boldsymbol{r})$+$\mathbf{U}_y(\boldsymbol{r})$. The coefficient $q_1^{(1)}(\tau,\ \xi)$ is a complex envelope scalar function satisfying a higher-order equation of (2.23). The linear operator $\hat{\boldsymbol{L}}_l^x$ is self-adjoint as in the following dot product

$$(\mathbf{U},\hat{\boldsymbol{L}}_l^x\mathbf{V}) \equiv \int_D \mathbf{U}^* \cdot \hat{\boldsymbol{L}}_l^x \mathbf{V}\, dS = \int_D \mathbf{V}\cdot \hat{\boldsymbol{L}}_l^x \mathbf{U}^*\, dS = (\mathbf{V}^*,\hat{\boldsymbol{L}}_l^x\mathbf{U}^*) = (\hat{\boldsymbol{L}}_l^x\mathbf{U},\mathbf{V}). \tag{2.30}$$

Here D is the total cross-section of the fiber and $\mathbf{V}$ is the inhomogeneity orthogonal to the adjoint of $\mathbf{U}$. $\mathbf{V}$ and $\mathbf{U}$ vanish to zero at the boundary of D. Using Fredholm's alternative

$$(\mathbf{U},\hat{\boldsymbol{L}}_1^x\boldsymbol{E}_1^{x(1)}) = (\mathbf{U}_x,\hat{\boldsymbol{L}}_1^x\mathbf{U}_x) = 0, \tag{2.31}$$

gives the value of k_1 as

$$\begin{aligned} k_1^2(\mathbf{U}_{x\perp},\mathbf{U}_{x\perp}) &= \frac{\omega_1^2}{c^2}(\mathbf{U}_x,\tilde{\chi}_0^{(1)}\mathbf{U}_x) + (\mathbf{U}_x,\nabla_{x\perp}^2\mathbf{U}_x) - (\nabla_{x\perp}\cdot\mathbf{U}_x,\nabla_{x\perp}\cdot\mathbf{U}_x) \\ &\quad - ik_1\left[(U_z,\nabla_{x\perp}\cdot\mathbf{U}_x) + (\nabla_{x\perp}\cdot\mathbf{U}_x,U_z)\right]. \end{aligned} \tag{2.32}$$

Here $\tilde{\chi}_0^{(1)} = \mathrm{Re}(\tilde{\chi}^{(1)})$, and $\mathbf{U}_x = \mathbf{U}_{x\perp} + U_z\hat{\mathbf{z}}$. The last two terms in (2.32) corresponds to the fact that $\nabla \boldsymbol{E}^x \neq 0$, implying that $E_z \neq 0$. For a weakly guided singlemode fiber the relative refractive index Δ is much less than unity, as given by

$$\Delta = \frac{n_1^2 - n_2^2}{2n_1^2} \approx \frac{n_1 - n_2}{n_1} \text{ for } \Delta \prec\prec 1. \qquad (2.33)$$

The core refractive index and cladding refractive index are n_1 and n_2 respectively. This leads to $E_z \approx 0$ and $U_z \approx 0$. Thus the weakly guided mode may be approximated by $\mathbf{U}_x = \mathbf{U}_{x\perp} = \nabla_y \phi(y) \times \hat{\mathbf{z}}$. Here $\phi(y)$ is the scalar eigenfunction dependent only on y, since the bound mode field function $\mathbf{U}_x$ is polarized in the x-axis. Substituting $\mathbf{U}_x$ and $U_z = 0$ in (2.32) gives the mode propagation constant k_1 as

$$k_1 = \sqrt{\frac{\left(\frac{\omega_1}{c}\right)^2 \int_D \left|\nabla_{y\perp}\phi_1\right|^2 \tilde{\chi}_0^{(1)} dS - \int_D \left|\nabla_{y\perp}\phi_1\right|^2 dS}{\int_D \left|\nabla_{y\perp}\phi_1\right|^2 dS}}. \qquad (2.34)$$

At order ε^2 from (2.23) with l=1

$$\boldsymbol{L}_1^x \boldsymbol{E}_1^{x(2)} + i\frac{d\boldsymbol{L}_1^x}{d\omega_1}\frac{\partial \boldsymbol{E}_1^{x(1)}}{\partial \tau} = 0. \qquad (2.35)$$

Applying Fredholm's alternative to (2.35) gives

$$\left(\mathbf{U}_x, \boldsymbol{L}_1^x \boldsymbol{E}_1^{x(2)}\right) = -i\int_D \mathbf{U}_x^* \cdot \frac{d\boldsymbol{L}_1^x}{d\omega_1}\frac{\partial\left(q_1^{(1)}\mathbf{U}_x\right)}{\partial \tau} dS = 0. \qquad (2.36)$$

The solution to (2.36) gives the group velocity as

$$v_g = \frac{1}{\frac{dk_1}{d\omega_1}} = \frac{c}{\sqrt{\chi_0^{(1)}}}. \qquad (2.37)$$

Integrating (2.35) with respect to $\hat{\boldsymbol{L}}_1^x$ gives

$$E_1^{(2)} = -i\frac{\partial q_1^{(1)}}{\partial \tau}\frac{\partial \mathbf{U}_x}{\partial \omega_1} + q_1^{(2)}\mathbf{U}_x. \tag{2.38}$$

At order ε^3 from (2.23) with l=1

$$\begin{aligned} &L_1^x E_1^{x(3)} + i\frac{dL_1^x}{d\omega_1}\frac{\partial E_1^{x(2)}}{\partial \tau} - \frac{1}{2}\frac{d^2 L_1^x}{d\omega_1^2}\frac{\partial^2 E_1^{x(1)}}{\partial \tau^2} \\ &- \frac{\partial L_1^x}{\partial k_1}\left\{ i\frac{\partial E_1^{x(1)}}{\partial \xi} - \frac{1}{2}k_1''\frac{\partial^2 E_1^{x(1)}}{\partial \tau^2} \right\} \\ &- \sum_{l_1+l_2+l_3=1} \hat{N}_{l_1 l_2 l_3}^x \vdots E_{l_1}^{(1)} E_{l_2}^{(1)} E_{l_3}^{(1)} = 0. \end{aligned} \tag{2.39}$$

The nonlinear component (last term) in (2.39) can be simplified as [2.11]

$$\begin{aligned} &\sum_{l_1+l_2+l_3=1} \hat{N}_{l_1 l_2 l_3}^x \vdots E_{l_1}^{(1)} E_{l_2}^{(1)} E_{l_3}^{(1)} = \frac{\omega_1^2}{c^2} \sum_{l_1+l_2+l_3=1} \tilde{\chi}_{l_1 l_2 l_3}^{(3)} \vdots E_{l_1}^{(1)} E_{l_2}^{(1)} E_{l_3}^{(1)} \\ &= \frac{\omega_1^2}{c^2} \sum_{l_1+l_2+l_3=1} \frac{3}{2} \sum_j \left\{ \begin{array}{l} \tilde{\chi}_{xxyy}^{(3)} E_{l_1}^{x(1)} E_{l_2}^{j(1)} E_{l_3}^{j(1)*} + \tilde{\chi}_{xyxy}^{(3)} E_{l_1}^{j(1)} E_{l_2}^{x(1)} E_{l_3}^{j(1)*} \\ + \tilde{\chi}_{xyyx}^{(3)} E_{l_1}^{j(1)} E_{l_2}^{j(1)} E_{l_3}^{x(1)} \end{array} \right\}. \end{aligned} \tag{2.40}$$

The summation over j is summed over x and y polarization axes. Here l_1, l_2, and l_3 can only take on values of either +1 or −1, and their sum must equal to +1. A weakly guided singlemode fiber can be assumed to be isotropic, and this leads to

$$\tilde{\chi}_{xxxx}^{(3)} = \tilde{\chi}_{yyyy}^{(3)} = \tilde{\chi}_{zzzz}^{(3)} = \tilde{\chi}_{xxyy}^{(3)} + \tilde{\chi}_{xyxy}^{(3)} + \tilde{\chi}_{xyyx}^{(3)}. \tag{2.41}$$

In silica fiber the dominant contribution to the third order susceptibility $\chi^{(3)}$ is of electronic origin, and thus its three components are nearly of the same magnitude [2.12], such that

$$\tilde{\chi}_{xxyy}^{(3)} \approx \tilde{\chi}_{xyxy}^{(3)} \approx \tilde{\chi}_{xyyx}^{(3)}. \tag{2.42}$$

But $E_1^{x(1)} = q_1^{(1)}\mathbf{U}_x$, $E_{-1}^{x(1)} = \left(q_1^{(1)}\mathbf{U}_x\right)_{-1} = q_1^{(1)*}\mathbf{U}_x^*$, $E_1^{y(j)} = 0\ \forall j \geq 1$, and letting $\frac{3}{4}\tilde{\chi}_{xxxx}^{(3)} = \tilde{\chi}_e^{(3)}$ and substituting into (2.40) gives

$$\sum_{l_1+l_2+l_3=1} \hat{\boldsymbol{N}}^x_{l_1 l_2 l_3} \vdots \boldsymbol{E}^{(1)}_{l_1} \boldsymbol{E}^{(1)}_{l_2} \boldsymbol{E}^{(1)}_{l_3} = \frac{\omega_1^2}{c^2} \tilde{\chi}_e^{(3)} |q_1^{(1)}|^2 q_1^{(1)} (\mathbf{U}_x \cdot \mathbf{U}_x^*) \mathbf{U}_x. \tag{2.43}$$

Substituting $\boldsymbol{E}_1^{x(1)} = q_1^{(1)} \mathbf{U}_x$ and (2.43) into (2.39) and applying $\left(\mathbf{U}_x, \hat{\boldsymbol{L}}_1^x \boldsymbol{E}_1^{x(3)}\right) = 0$ (Fredholm's alternative), and using $\left(\mathbf{U}_x, \hat{\boldsymbol{L}}_1^x \boldsymbol{E}_1^{x(1)}\right) = \left(\mathbf{U}_x, \hat{\boldsymbol{L}}_1^x \boldsymbol{E}_1^{x(2)}\right) = 0$ gives

$$\begin{aligned} &\left\{ i \frac{\partial q_1^{(1)}}{\partial \xi} - \frac{1}{2} k_1'' \frac{\partial^2 q_1^{(1)}}{\partial \tau^2} \right\} \left(\mathbf{U}_x, \frac{\partial \hat{\boldsymbol{L}}_1^x}{\partial k_1} \mathbf{U}_x \right) - \frac{\omega^2}{c^2} \left(\mathbf{U}_x, \tilde{\chi}_e^{(3)} (\mathbf{U}_x \cdot \mathbf{U}_x^*) \mathbf{U}_x \right) |q_1^{(1)}|^2 q_1^{(1)} \\ &= i \frac{\partial q_1^{(1)}}{\partial \xi} - \frac{1}{2} k_1'' \frac{\partial^2 q_1^{(1)}}{\partial \tau^2} - a_1 |q_1^{(1)}|^2 q_1^{(1)} = 0. \end{aligned} \tag{2.44a}$$

The coefficient a_1 is defined as

$$a_1 = \left(\frac{\frac{\omega^2}{c^2} \left(\mathbf{U}_x, \tilde{\chi}_e^{(3)} (\mathbf{U}_x \cdot \mathbf{U}_x^*) \mathbf{U}_x \right)}{\left(\mathbf{U}_x, \frac{\partial \hat{\boldsymbol{L}}_1^x}{\partial k_1} \mathbf{U}_x \right)} \right). \tag{2.44b}$$

At order ε^4 from (2.23) with l=1

$$\begin{aligned} &\hat{\boldsymbol{L}}_1^x \boldsymbol{E}_1^{x(4)} + i \frac{d \hat{\boldsymbol{L}}_1^x}{d\omega_1} \frac{\partial \boldsymbol{E}_1^{x(3)}}{\partial \tau} - \frac{1}{2} \frac{d^2 \hat{\boldsymbol{L}}_1^x}{d\omega_1^2} \frac{\partial^2 \boldsymbol{E}_1^{x(2)}}{\partial \tau^2} - \frac{i}{6} \frac{d^3 \hat{\boldsymbol{L}}_1^x}{d\omega_1^3} \frac{\partial^3 \boldsymbol{E}_1^{x(1)}}{\partial \tau^3} \\ &- \frac{\partial \hat{\boldsymbol{L}}_1^x}{\partial k_1} \left\{ i \frac{\partial \boldsymbol{E}_1^{x(2)}}{\partial \xi} - \frac{1}{2} k_1'' \frac{\partial^2 \boldsymbol{E}_1^{x(2)}}{\partial \tau^2} - \frac{i k_1'''}{6} \frac{\partial^3 \boldsymbol{E}_1^{x(1)}}{\partial \tau^3} \right\} \\ &- \frac{d}{d\omega_1} \left(\frac{\partial \hat{\boldsymbol{L}}_1^x}{\partial k_1} \right) i \frac{\partial}{\partial \tau} \left\{ i \frac{\partial \boldsymbol{E}_1^{x(1)}}{\partial \xi} - \frac{1}{2} k_1'' \frac{\partial^2 \boldsymbol{E}_1^{x(1)}}{\partial \tau^2} \right\} \\ &+ \sum_{l_1+l_2+l_3=1} \left\{ \sum_{i+j+k=4} \hat{\boldsymbol{N}}^x_{l_1 l_2 l_3} \vdots \boldsymbol{E}^{(i)}_{l_1} \boldsymbol{E}^{(j)}_{l_2} \boldsymbol{E}^{(k)}_{l_3} + i \sum_{i=1}^{3} \frac{\partial \hat{\boldsymbol{N}}^x_{l_1 l_2 l_3}}{\partial \omega_{l_i}} \vdots \frac{\partial}{\partial \tau_i} \left(\boldsymbol{E}^{(1)}_{l_1} \boldsymbol{E}^{(1)}_{l_2} \boldsymbol{E}^{(1)}_{l_3} \right) \right\} = 0. \end{aligned} \tag{2.45}$$

In the nonlinear component (last term) in (2.45) the variables *i, j, k* represent the perturbation order of the electric field components that must sum up to 4. The last term in (2.45) can be simplified as

$$\sum_{l_1+l_2+l_3=1}\left\{\sum_{i+j+k=4}\hat{\boldsymbol{N}}^x_{l_1l_2l_3}:\boldsymbol{E}^{(i)}_{l_1}\boldsymbol{E}^{(j)}_{l_2}\boldsymbol{E}^{(k)}_{l_3}+i\sum_{i=1}^{3}\frac{\partial\hat{\boldsymbol{N}}^x_{l_1l_2l_3}}{\partial\omega_{l_i}}:\frac{\partial}{\partial\tau_i}\left(\boldsymbol{E}^{(1)}_{l_1}\boldsymbol{E}^{(1)}_{l_2}\boldsymbol{E}^{(1)}_{l_3}\right)\right\}$$
$$=\frac{\omega_1^2}{c^2}\sum_{l_1+l_2+l_3=1}\left\{\sum_{i+j+k=4}\tilde{\chi}^{(3)}:\boldsymbol{E}^{(i)}_{l_1}\boldsymbol{E}^{(j)}_{l_2}\boldsymbol{E}^{(k)}_{l_3}+i\sum_{i=1}^{3}\frac{\partial\tilde{\chi}^{(3)}}{\partial\omega_{l_i}}:\frac{\partial\left(\boldsymbol{E}^{(1)}_{l_1}\boldsymbol{E}^{(1)}_{l_2}\boldsymbol{E}^{(1)}_{l_3}\right)}{\partial\tau_i}\right\}. \tag{2.46}$$

The two components of (2.46) can be further simplified individually as

$$\frac{\omega_1^2}{c^2}\sum_{l_1+l_2+l_3=1}\sum_{i+j+k=4}\tilde{\chi}^{(3)}:\boldsymbol{E}^{(i)}_{l_1}\boldsymbol{E}^{(j)}_{l_2}\boldsymbol{E}^{(k)}_{l_3}$$
$$=\frac{\omega_1^2}{c^2}\tilde{\chi}^{(3)}_e\left\{2\left|\boldsymbol{E}^{x(1)}_1\right|^2\boldsymbol{E}^{x(2)}_1+\left(\boldsymbol{E}^{x(1)}_1\right)^2\boldsymbol{E}^{x(2)*}_1\right\}, \tag{2.47}$$

and

$$\frac{\omega_1^2}{c^2}\sum_{l_1+l_2+l_3=1}\sum_{i=1}^{3}\frac{\partial\tilde{\chi}^{(3)}}{\partial\omega_{l_i}}:\frac{\partial\left(\boldsymbol{E}^{(1)}_{l_1}\boldsymbol{E}^{(1)}_{l_2}\boldsymbol{E}^{(1)}_{l_3}\right)}{\partial\tau_i}$$
$$=\frac{\omega_1^2}{c^2}\left\{\frac{\partial}{\partial\omega_1}\left(\tilde{\chi}^{(3)}_e\left[\left|\boldsymbol{E}^{x(1)}_1\right|^2\frac{\partial\boldsymbol{E}^{x(1)}_1}{\partial\tau}\right]\right)+\frac{\partial}{\partial(-\omega_1)}\left(\tilde{\chi}^{(3)}_e\left[\boldsymbol{E}^{x(1)}_1\frac{\partial\left|\boldsymbol{E}^{x(1)}_1\right|^2}{\partial\tau}\right]\right)\right\}. \tag{2.48}$$

Substituting $\boldsymbol{E}^{x(1)}_1=q^{(1)}_1\mathbf{U}_x$, $\boldsymbol{E}^{x(2)}_1=q^{(2)}_1\mathbf{U}_x$, (2.47), (2.48) into (2.45) and applying $\left(\mathbf{U}_x,\hat{\boldsymbol{L}}^x_1\boldsymbol{E}^{x(4)}_1\right)=0$, and using $\left(\mathbf{U}_x,\hat{\boldsymbol{L}}^x_1\boldsymbol{E}^{x(1)}_1\right)=\left(\mathbf{U}_x,\hat{\boldsymbol{L}}^x_1\boldsymbol{E}^{x(2)}_1\right)=\left(\mathbf{U}_x,\hat{\boldsymbol{L}}^x_1\boldsymbol{E}^{x(3)}_1\right)=0$ gives

$$\left(i\frac{\partial q^{(2)}_1}{\partial\xi}-\frac{1}{2}k''_1\frac{\partial^2q^{(2)}_1}{\partial\tau^2}-\frac{i}{6}k'''_1\frac{\partial^3q^{(1)}_1}{\partial\tau^3}\right)\left(\mathbf{U}_x,\frac{\partial\hat{\boldsymbol{L}}^x_1}{\partial k_1}\mathbf{U}_x\right)$$
$$-\frac{\omega_1^2}{c^2}\left\{\begin{array}{l}2\left|q^{(1)}_1\right|^2q^{(2)}_1\left(\mathbf{U}_x,\tilde{\chi}^{(3)}_e(\mathbf{U}_x\cdot\mathbf{U}^*_x)\mathbf{U}_x\right)+q^{(1)2}_1q^{(2)*}_1\left(\mathbf{U}_x,\tilde{\chi}^{(3)}_e(\mathbf{U}_x\cdot\mathbf{U}^*_x)\mathbf{U}_x\right)\\ +i\left[\begin{array}{l}\left|q^{(1)}_1\right|^2\frac{\partial q^{(1)}_1}{\partial\tau}\left(\mathbf{U}_x,\frac{\partial}{\partial\omega_1}\left(\tilde{\chi}^{(3)}_e(\mathbf{U}_x\cdot\mathbf{U}^*_x)\mathbf{U}_x\right)\right)\\ +q^{(1)}_1\frac{\partial\left|q^{(1)}_1\right|^2}{\partial\tau}\left(\mathbf{U}_x,\frac{\partial}{\partial(-\omega_1)}\left(\tilde{\chi}^{(3)}_e(\mathbf{U}_x\cdot\mathbf{U}^*_x)\mathbf{U}_x\right)\right)\end{array}\right]\end{array}\right\} \tag{2.49}$$
$$=0$$

Let

$$a=\left(\mathbf{U}_x,\frac{\partial\hat{\mathbf{L}}_1^x}{\partial k_1}\mathbf{U}_x\right);\ b_1=\frac{2\omega_1^2}{ac^2}\left(\mathbf{U}_x,\tilde{\chi}_e^{(3)}(\mathbf{U}_x\cdot\mathbf{U}_x^*)\mathbf{U}_x\right)$$

$$b_2=\frac{\omega_1^2}{ac^2}\left(\mathbf{U}_x,\tilde{\chi}_e^{(3)}(\mathbf{U}_x\cdot\mathbf{U}_x^*)\mathbf{U}_x\right);\ c_1=\frac{\omega_1^2}{ac^2}\left(\mathbf{U}_x,\frac{\partial}{\partial\omega_1}\left(\tilde{\chi}_e^{(3)}(\mathbf{U}_x\cdot\mathbf{U}_x^*)\mathbf{U}_x\right)\right) \qquad (2.50)$$

$$c_2=\frac{\omega_1^2}{ac^2}\left(\mathbf{U}_x,\frac{\partial}{\partial(-\omega_1)}\left(\tilde{\chi}_e^{(3)}(\mathbf{U}_x\cdot\mathbf{U}_x^*)\mathbf{U}_x\right)\right).$$

Substituting into (2.49) gives

$$\left(i\frac{\partial q_1^{(2)}}{\partial\xi}-\frac{1}{2}k_1''\frac{\partial^2 q_1^{(2)}}{\partial\tau^2}-\frac{i}{6}k_1'''\frac{\partial^3 q_1^{(1)}}{\partial\tau^3}\right)$$
$$-\left\{b_1\left|q_1^{(1)}\right|^2 q_1^{(2)}+b_2 q_1^{(1)2}q_1^{(2)*}+i\left[c_1\frac{\partial\left(\left|q_1^{(1)}\right|^2 q_1^{(1)}\right)}{\partial\tau}+d_1 q_1^{(1)}\frac{\partial\left|q_1^{(1)}\right|^2}{\partial\tau}\right]\right\} \qquad (2.51)$$
$$=0.$$

Here $d_1 = c_2 - c_1$. But $q_1\mathbf{U}_x=\left(\varepsilon q_1^{(1)}+\varepsilon^2 q_1^{(2)}\right)\mathbf{U}_x$. This implies that at order ε, $q_1^{(1)}\approx\frac{q_1}{\varepsilon}$,

and at order ε^2, $q_1^{(2)}\approx\frac{q_1}{\varepsilon^2}$. Substituting for $q_1^{(1)}$ and $q_1^{(2)}$ in (2.51) and multiplying by ε^2

gives

$$\left(i\frac{\partial q_1}{\partial\xi}-\frac{1}{2}k_1''\frac{\partial^2 q_1}{\partial\tau^2}-\frac{i\varepsilon}{6}k_1'''\frac{\partial^3 q_1}{\partial\tau^3}\right) \qquad (2.52)$$
$$-\left\{\tilde{e}_1\left|q_1\right|^2 q_1+i\varepsilon\left[\tilde{c}_1\frac{\partial\left(\left|q_1\right|^2 q_1\right)}{\partial\tau}+\tilde{d}_1 q_1\frac{\partial\left|q_1\right|^2}{\partial\tau}\right]\right\}=\mathbf{O}(\varepsilon^3)$$

Here $\tilde{e}_1=\left(b_1+b_2\right)/\varepsilon^2$, $\tilde{c}_1=c_1/\varepsilon^2$, and $\tilde{d}_1=d_1/\varepsilon^2$. In equation (2.52) the second, third, fourth, fifth and sixth terms represent the second order dispersion (SOD), third order dispersion (TOD), self phase modulation (SPM), self-steepening (SS), and self-frequency shifting (SFS).

2.2 DERIVATION OF CROSS KERR AND RAMAN EFFECTS

Following the reductive perturbation method, applied to nonlinear pulse propagation by Kodama and Hasegawa in section (2.1.3), and extending it to two orthogonally polarized monochromatic soliton pulses, the cross Kerr and Raman effects can be derived. The assumptions are: (i) Quasi-monochromatic mode approximation, (ii) The singlemode fiber supports two orthogonally polarized modes, and (iii) The fiber linear and nonlinear birefringence is initially negligible. To practically maintain two orthogonally polarized pulses in a singlemode fiber, the birefringence must be finite. The negligible fiber linear and nonlinear birefringence assumption here is later corrected in the numerical propagation model, to reflect this practical requirement.

The extension of the theory in (2.1.3) to handle two orthogonally polarized monochromatic soliton pulses, requires the use of two linear operators, namely $\hat{L}_l^x$ and $\hat{L}_l^y$ for the x and y polarizations respectively. It also requires the use of two non-linear response tensor operators, namely $\hat{N}_{l_1 l_2 l_3}^x$ and $\hat{N}_{l_1 l_2 l_3}^y$ for the x and y polarizations respectively. These operators are defined as follows

$$\hat{L}_l^x\left(k_{xl},\, \omega_l,\, \nabla_{x\perp}\right) = \nabla_{x\perp}^2 - k_{xl}^2 + \frac{\omega_l^2}{c^2}\tilde{\chi}^{(1)} - \nabla_x(\nabla_x \cdot)\Big|_{\partial/\partial z = ik_{xl}}, \tag{2.53}$$

$$\hat{L}_l^y\left(k_{yl},\, \omega_l,\, \nabla_{x\perp}\right) = \nabla_{y\perp}^2 - k_{yl}^2 + \frac{\omega_l^2}{c^2}\tilde{\chi}^{(1)} - \nabla_y(\nabla_y \cdot)\Big|_{\partial/\partial z = ik_{yl}}, \tag{2.54}$$

$$\hat{N}_{l_1 l_2 l_3}^x\left(\omega_{l_1},\, \omega_{l_2},\, \omega_{l_3}\right) = \frac{\omega_l^2}{c^2}\tilde{\chi}^{x(3)}(\omega_{l_1},\, \omega_{l_2},\, \omega_{l_3}), \tag{2.55}$$

$$\hat{N}_{l_1 l_2 l_3}^y\left(\omega_{l_1},\, \omega_{l_2},\, \omega_{l_3}\right) = \frac{\omega_l^2}{c^2}\tilde{\chi}^{y(3)}(\omega_{l_1},\, \omega_{l_2},\, \omega_{l_3}). \tag{2.56}$$

The total electric field can be represented as

$$\begin{aligned}\boldsymbol{E}(z,\ t) &= \boldsymbol{E}^x(z,\ t)+\boldsymbol{E}^y(z,\ t)\\ &=\sum_{-\infty}^{\infty}\boldsymbol{E}_l^x(\tau,\xi,r;\varepsilon)e^{i(k_{xl}z-\omega_l t)}+\sum_{-\infty}^{\infty}\boldsymbol{E}_l^y(\tau,\xi,r;\varepsilon)e^{i(k_{yl}z-\omega_l t)}.\end{aligned} \tag{2.57}$$

Here $k_{xl} = lk_{x1}$, $k_{yl} = lk_{y1}$, and $\omega_l = l\omega_1$. Using the methodology of the reductive perturbation method, $\boldsymbol{E}_l^x(\tau,\ \xi,\ \boldsymbol{r};\ \varepsilon)$ and $\boldsymbol{E}_l^y(\tau,\ \xi,\ \boldsymbol{r};\ \varepsilon)$ can be expanded in terms of ε as

$$\boldsymbol{E}_l^x(\tau,\ \xi,\ \boldsymbol{r};\ \varepsilon)=\sum_{n=1}^{\infty}\varepsilon^n\boldsymbol{E}_l^{x(n)}(\tau,\ \xi,\ \boldsymbol{r}), \tag{2.58}$$

$$\boldsymbol{E}_l^y(\tau,\ \xi,\ \boldsymbol{r};\ \varepsilon)=\sum_{n=1}^{\infty}\varepsilon^n\boldsymbol{E}_l^{y(n)}(\tau,\ \xi,\ \boldsymbol{r}). \tag{2.59}$$

To derive the coupled NLSE (CNLSE) with the cross Kerr and Raman effects, the *x*-polarization NLSE is first derived and thereafter the *y* polarization NSLE is inferred by similar mathematical manipulations as in the *x* polarization axis.

For the *x*-polarization axis equation (2.23) becomes

$$\begin{aligned}&\hat{\boldsymbol{L}}_l^x\boldsymbol{E}_l^x+i\varepsilon\frac{d\hat{\boldsymbol{L}}_l^x}{d\omega_l}\frac{\partial\boldsymbol{E}_l^x}{\partial\tau}-\frac{1}{2}\varepsilon^2\frac{d^2\hat{\boldsymbol{L}}_l^x}{d\omega_l^2}\frac{\partial^2\boldsymbol{E}_l^x}{\partial\tau^2}-\frac{i\varepsilon^3}{6}\frac{d^3\hat{\boldsymbol{L}}_l^x}{d\omega_l^3}\frac{\partial^3\boldsymbol{E}_l^x}{\partial\tau^3}\\ &-\varepsilon^2\frac{\partial\hat{\boldsymbol{L}}_l^x}{\partial k_{xl}}\left\{i\frac{\partial\boldsymbol{E}_l^x}{\partial\xi}-\frac{1}{2}k_{xl}''\frac{\partial^2\boldsymbol{E}_l^x}{\partial\tau^2}-\frac{i\varepsilon k_{xl}'''}{6}\frac{\partial^3\boldsymbol{E}_l^x}{\partial\tau^3}\right\}\\ &-\varepsilon^3\frac{d}{d\omega_l}\left(\frac{\partial\hat{\boldsymbol{L}}_l^x}{\partial k_{xl}}\right)i\frac{\partial}{\partial\tau}\left\{i\frac{\partial\boldsymbol{E}_l^x}{\partial\xi}-\frac{1}{2}k_{xl}''\frac{\partial^2\boldsymbol{E}_l^x}{\partial\tau^2}\right\}\\ &+\sum_{l_1+l_2+l_3=l}\left\{\hat{\boldsymbol{N}}_{l_1l_2l_3}^x\vdots\boldsymbol{E}_{l_1}\boldsymbol{E}_{l_2}\boldsymbol{E}_{l_3}+i\varepsilon\sum_{i=1}^{3}\frac{\partial\hat{\boldsymbol{N}}_{l_1l_2l_3}^x}{\partial\omega_{l_i}}\vdots\frac{\partial}{\partial\tau_i}\left(\boldsymbol{E}_{l_1}\boldsymbol{E}_{l_2}\boldsymbol{E}_{l_3}\right)\right\}+\mathbf{O}(\varepsilon^4)=0.\end{aligned} \tag{2.60}$$

At order ε from (2.60) with l=1

$$\boldsymbol{L}_1^x\boldsymbol{E}_1^{x(1)}=0. \tag{2.61}$$

The solution to (2.61) is of the form

$$\boldsymbol{E}_1^{x(1)}(\tau,\ \xi,\ \boldsymbol{r};\ \varepsilon) = q_1^{(1)}(\tau,\ \xi)\mathbf{U}_x(\ \boldsymbol{r}) . \qquad (2.62)$$

Here the fiber is a singlemode polarization-maintaining fiber, carrying two polarizations. Using Fredholm's alternative

$$(\mathbf{U}, \hat{\boldsymbol{L}}_1^x \boldsymbol{E}_1^{x(1)}) = (\mathbf{U}_x, \hat{\boldsymbol{L}}_1^x \mathbf{U}_x) = 0 , \qquad (2.63)$$

gives the value of k_{x1} as

$$k_{x1}(\mathbf{U}_{x\perp}, \mathbf{U}_{x\perp}) = \sqrt{\begin{matrix}\frac{\omega^2}{c^2}(\mathbf{U}_x, \tilde{\chi}_0^{(1)}\mathbf{U}_x) + (\mathbf{U}_x, \nabla_{x\perp}^2 \mathbf{U}_x) - (\nabla_{x\perp}\cdot\mathbf{U}_x, \nabla_{x\perp}\cdot\mathbf{U}_x) \\ -ik_{x1}\left[(U_z, \nabla_{x\perp}\cdot\mathbf{U}_x) + (\nabla_{x\perp}\cdot\mathbf{U}_x, U_z)\right]\end{matrix}} \qquad (2.64)$$

$$\approx \sqrt{\frac{\omega^2}{c^2}(\mathbf{U}_x, \tilde{\chi}_0^{(1)}\mathbf{U}_x) + (\mathbf{U}_x, \nabla_{x\perp}^2 \mathbf{U}_x) - (\nabla_{x\perp}\cdot\mathbf{U}_x, \nabla_{x\perp}\cdot\mathbf{U}_x)}$$

At order ε^2 from (2.60) with l=1

$$\boldsymbol{L}_1^x \boldsymbol{E}_1^{x(2)} + i\frac{d\boldsymbol{L}_1^x}{d\omega_1}\frac{\partial \boldsymbol{E}_1^{x(1)}}{\partial \tau} = 0 . \qquad (2.65)$$

Applying Fredholm's alternative to (2.65) gives

$$\left(\mathbf{U}_x,\ \boldsymbol{L}_1^x \boldsymbol{E}_1^{x(2)}\right) = -i\int_D \mathbf{U}_x^* \cdot \frac{d\boldsymbol{L}_1^x}{d\omega_1}\frac{\partial\left(q_1^{(1)}\mathbf{U}_x\right)}{\partial \tau} dS = 0 . \qquad (2.66)$$

The solution to (2.66) gives the group velocity as

$$v_{xg} = \frac{1}{\frac{dk_{x1}}{d\omega_1}} = \frac{c}{\sqrt{\chi_0^{x(1)}}} . \qquad (2.67)$$

Here $\tilde{\chi}_0^{x(1)}$ is the x-axis component of $\tilde{\chi}_0^{(1)}$.

At order ε^3 from (2.60) with l=1

$$\begin{aligned} &\boldsymbol{L}_1^x \boldsymbol{E}_1^{x(3)} + i\frac{d\boldsymbol{L}_1^x}{d\omega_1}\frac{\partial \boldsymbol{E}_1^{x(2)}}{\partial\tau} - \frac{1}{2}\frac{d^2\boldsymbol{L}_1^x}{d\omega_1^2}\frac{\partial^2 \boldsymbol{E}_1^{x(1)}}{\partial\tau^2} \\ &- \frac{\partial \boldsymbol{L}_1^x}{\partial k_{x1}}\left\{ i\frac{\partial \boldsymbol{E}_1^{x(1)}}{\partial\xi} - \frac{1}{2}k''_{x1}\frac{\partial^2 \boldsymbol{E}_1^{x(1)}}{\partial\tau^2}\right\} \\ &- \sum_{l_1+l_2+l_3=1} \hat{\boldsymbol{N}}^x_{l_1 l_2 l_3} \vdots \boldsymbol{E}_{l_1}^{(1)}\boldsymbol{E}_{l_2}^{(1)}\boldsymbol{E}_{l_3}^{(1)} = 0. \end{aligned} \tag{2.68}$$

The nonlinear component (last term) in (2.68) can be simplified as

$$\begin{aligned} &\sum_{l_1+l_2+l_3=1} \hat{\boldsymbol{N}}^x_{l_1 l_2 l_3} \vdots \boldsymbol{E}_{l_1}^{(1)}\boldsymbol{E}_{l_2}^{(1)}\boldsymbol{E}_{l_3}^{(1)} = \frac{\omega_1^2}{c^2}\sum_{l_1+l_2+l_3=1} \tilde{\chi}^{x(3)}_{l_1 l_2 l_3} \vdots \boldsymbol{E}_{l_1}^{(1)}\boldsymbol{E}_{l_2}^{(1)}\boldsymbol{E}_{l_3}^{(1)} \\ &= \frac{\omega_1^2}{c^2}\sum_{l_1+l_2+l_3=1}\frac{3}{2}\sum_j \left\{ \begin{aligned} &\tilde{\chi}^{(3)}_{xxyy}\boldsymbol{E}_{l_1}^{x(1)}\boldsymbol{E}_{l_2}^{j(1)}\boldsymbol{E}_{l_3}^{j(1)*} + \tilde{\chi}^{(3)}_{xyxy}\boldsymbol{E}_{l_1}^{j(1)}\boldsymbol{E}_{l_2}^{x(1)}\boldsymbol{E}_{l_3}^{j(1)*} \\ &+ \tilde{\chi}^{(3)}_{xyyx}\boldsymbol{E}_{l_1}^{j(1)}\boldsymbol{E}_{l_2}^{j(1)}\boldsymbol{E}_{l_3}^{x(1)} \end{aligned} \right\}. \end{aligned} \tag{2.69}$$

But $\boldsymbol{E}_1^{x(1)} = q_1^{(1)}\mathbf{U}_x$, $\boldsymbol{E}_{-1}^{x(1)} = \left(q_1^{(1)}\mathbf{U}_x\right)_{-1} = q_1^{(1)*}\mathbf{U}_x^*$, $\boldsymbol{E}_1^{y(1)} = q_2^{(1)}\mathbf{U}_y$,

$\boldsymbol{E}_{-1}^{y(1)} = \left(q_2^{(1)}\right)_{-1}\mathbf{U}_y = q_2^{(1)*}\mathbf{U}_y^*$, and substituting into (2.69) gives

$$\begin{aligned} &\sum_{l_1+l_2+l_3=1} \hat{\boldsymbol{N}}^x_{l_1 l_2 l_3} \vdots \boldsymbol{E}_{l_1}^{(1)}\boldsymbol{E}_{l_2}^{(1)}\boldsymbol{E}_{l_3}^{(1)} \\ &= \frac{\omega_1^2}{c^2}\tilde{\chi}_e^{(3)}\left(\begin{aligned} &|q_1^{(1)}|^2 q_1^{(1)}(\mathbf{U}_x\cdot\mathbf{U}_x^*)\mathbf{U}_x + |q_2^{(1)}|^2 q_1^{(1)}(\mathbf{U}_y\cdot\mathbf{U}_y^*)\mathbf{U}_x \\ &+ q_2^{(1)2} q_1^{(1)*}(\mathbf{U}_y\cdot\mathbf{U}_x^*)\mathbf{U}_y \end{aligned} \right). \end{aligned} \tag{2.70}$$

Substituting $\boldsymbol{E}_1^{x(1)} = q_1^{(1)}\mathbf{U}_x$, and (2.70) into (2.68) and applying $\left(\mathbf{U}_x, \hat{\boldsymbol{L}}_1^x\boldsymbol{E}_1^{x(3)}\right) = 0$, and using $\left(\mathbf{U}_x, \hat{\boldsymbol{L}}_1^x\boldsymbol{E}_1^{x(1)}\right) = \left(\mathbf{U}_x, \hat{\boldsymbol{L}}_1^x\boldsymbol{E}_1^{x(2)}\right) = 0$ gives

$$\begin{aligned} &\left\{ i\frac{\partial q_1^{(1)}}{\partial\xi} - \frac{1}{2}k''_{x1}\frac{\partial^2 q_1^{(1)}}{\partial\tau^2}\right\}\left(\mathbf{U}_x, \frac{\partial\hat{\boldsymbol{L}}_1^x}{\partial k_{x1}}\mathbf{U}_x\right) \\ &- \frac{\omega^2}{c^2}\left\{ \begin{aligned} &|q_1^{(1)}|^2 q_1^{(1)}\left(\mathbf{U}_x, \chi_e^{(3)}(\mathbf{U}_x\cdot\mathbf{U}_x^*)\mathbf{U}_x\right) + |q_2^{(1)}|^2 q_1^{(1)}\left(\mathbf{U}_x, \chi_e^{(3)}(\mathbf{U}_y\cdot\mathbf{U}_y^*)\mathbf{U}_x\right) \\ &+ q_2^{(1)2} q_1^{(1)}\left(\mathbf{U}_x, \chi_e^{(3)}(\mathbf{U}_y\cdot\mathbf{U}_x^*)\mathbf{U}_y\right) \end{aligned} \right\} \\ &= i\frac{\partial q_1^{(1)}}{\partial\xi} - \frac{1}{2}k''_{x1}\frac{\partial^2 q_1^{(1)}}{\partial\tau^2} - \left\{ F_1|q_1^{(1)}|^2 q_1^{(1)} + F_2|q_2^{(1)}|^2 q_1^{(1)} + F_3 q_2^{(1)2} q_1^{(1)*}\right\} = 0. \end{aligned} \tag{2.71}$$

The coefficients in (2.71) are defined as

$$F_1=\left(\mathbf{U}_x,\tilde{\chi}_e^{(3)}\left(\mathbf{U}_x\cdot\mathbf{U}_x^*\right)\mathbf{U}_x\right)/a_1;\ F_2=\left(\mathbf{U}_x,\tilde{\chi}_e^{(3)}\left(\mathbf{U}_y\cdot\mathbf{U}_y^*\right)\mathbf{U}_x\right)/a_1$$
$$F_3=\left(\mathbf{U}_x,\tilde{\chi}_e^{(3)}\left(\mathbf{U}_y\cdot\mathbf{U}_x^*\right)\mathbf{U}_y\right)/a_1;\ a_1=\left(\mathbf{U}_x,\frac{\partial\hat{\boldsymbol{L}}_1^x}{\partial k_{x1}}\mathbf{U}_x\right) \tag{2.72}$$

At order ε^4 from (2.60) with l=1

$$\begin{aligned}
&\hat{\boldsymbol{L}}_1^x\boldsymbol{E}_1^{x(4)}+i\frac{d\hat{\boldsymbol{L}}_1^x}{d\omega_1}\frac{\partial\boldsymbol{E}_1^{x(3)}}{\partial\tau}-\frac{1}{2}\frac{d^2\hat{\boldsymbol{L}}_1^x}{d\omega_1^2}\frac{\partial^2\boldsymbol{E}_1^{x(2)}}{\partial\tau^2}-\frac{i}{6}\frac{d^3\hat{\boldsymbol{L}}_1^x}{d\omega_1^3}\frac{\partial^3\boldsymbol{E}_1^{x(1)}}{\partial\tau^3}\\
&-\frac{\partial\hat{\boldsymbol{L}}_1^x}{\partial k_{x1}}\left\{i\frac{\partial\boldsymbol{E}_1^{(2)}}{\partial\xi}-\frac{1}{2}k_{x1}^{''}\frac{\partial^2\boldsymbol{E}_1^{x(2)}}{\partial\tau^2}-\frac{ik_{x1}^{'''}}{6}\frac{\partial^3\boldsymbol{E}_1^{x(1)}}{\partial\tau^3}\right\}\\
&-\frac{d}{d\omega_1}\left(\frac{\partial\hat{\boldsymbol{L}}_1^x}{\partial k_{x1}}\right)i\frac{\partial}{\partial\tau}\left\{i\frac{\partial\boldsymbol{E}_1^{x(1)}}{\partial\xi}-\frac{1}{2}k_{x1}^{''}\frac{\partial^2\boldsymbol{E}_1^{x(1)}}{\partial\tau^2}\right\}\\
&+\sum_{l_1+l_2+l_3=1}\left\{\sum_{i+j+k=4}\hat{\boldsymbol{N}}_{l_1l_2l_3}^x\vdots\boldsymbol{E}_{l_1}^{(i)}\boldsymbol{E}_{l_2}^{(j)}\boldsymbol{E}_{l_3}^{(k)}+i\sum_{i=1}^{3}\frac{\partial\hat{\boldsymbol{N}}_{l_1l_2l_3}^x}{\partial\omega_{l_i}}\vdots\frac{\partial}{\partial\tau_i}\left(\boldsymbol{E}_{l_1}^{(1)}\boldsymbol{E}_{l_2}^{(1)}\boldsymbol{E}_{l_3}^{(1)}\right)\right\}=0.
\end{aligned}\tag{2.73}$$

The last term in (2.73) can be simplified as

$$\begin{aligned}
&\sum_{l_1+l_2+l_3=1}\left\{\sum_{i+j+k=4}\hat{\boldsymbol{N}}_{l_1l_2l_3}^x\vdots\boldsymbol{E}_{l_1}^{(i)}\boldsymbol{E}_{l_2}^{(j)}\boldsymbol{E}_{l_3}^{(k)}+i\sum_{i=1}^{3}\frac{\partial\hat{\boldsymbol{N}}_{l_1l_2l_3}^x}{\partial\omega_{l_i}}\vdots\frac{\partial}{\partial\tau_i}\left(\boldsymbol{E}_{l_1}\boldsymbol{E}_{l_2}\boldsymbol{E}_{l_3}\right)\right\}\\
&=\frac{\omega_1^2}{c^2}\sum_{l_1+l_2+l_3=1}\left\{\sum_{i+j+k=4}\tilde{\chi}^{x(3)}\vdots\boldsymbol{E}_{l_1}^{(i)}\boldsymbol{E}_{l_2}^{(j)}\boldsymbol{E}_{l_3}^{(k)}+i\sum_{i=1}^{3}\frac{\partial\tilde{\chi}^{x(3)}}{\partial\omega_{l_i}}\vdots\frac{\partial\left(\boldsymbol{E}_{l_1}^{(1)}\boldsymbol{E}_{l_2}^{(1)}\boldsymbol{E}_{l_3}^{(1)}\right)}{\partial\tau_i}\right\}.
\end{aligned}\tag{2.74}$$

The two components of (2.74) can be further simplified individually as

$$\begin{aligned}
&\frac{\omega_1^2}{c^2}\sum_{l_1+l_2+l_3=1}\sum_{i+j+k=4}\tilde{\chi}^{x(3)}\vdots\boldsymbol{E}_{l_1}^{(i)}\boldsymbol{E}_{l_2}^{(j)}\boldsymbol{E}_{l_3}^{(k)}\\
&=\frac{\omega_1^2}{c^2}\chi_e^{(3)}\left\{\begin{aligned}&2\left|\boldsymbol{E}_1^{x(1)}\right|^2\boldsymbol{E}_1^{x(2)}+\left(\boldsymbol{E}_1^{x(1)}\right)^2\boldsymbol{E}_1^{x(2)*}+\frac{2}{3}\left|\boldsymbol{E}_1^{y(1)}\right|^2\boldsymbol{E}_1^{x(2)}\\&+\frac{1}{3}\left(\boldsymbol{E}_1^{y(1)}\right)^2\boldsymbol{E}_1^{x(2)*}+\frac{2}{3}\boldsymbol{E}_1^{y(1)}\boldsymbol{E}_1^{y(2)*}\boldsymbol{E}_1^{x(1)}+\frac{2}{3}\boldsymbol{E}_1^{y(2)}\boldsymbol{E}_1^{y(1)*}\boldsymbol{E}_1^{x(1)}\\&+\frac{2}{3}\boldsymbol{E}_1^{y(1)}\boldsymbol{E}_1^{y(2)}\boldsymbol{E}_1^{x(1)*}\end{aligned}\right\},
\end{aligned}\tag{2.75}$$

and

$$\frac{\omega_1^2}{c^2}\sum_{l_1+l_2+l_3=1}\sum_{i=1}^{3}\frac{\partial\tilde{\chi}^{x(3)}}{\partial\omega_{l_i}}\vdots\frac{\partial\left(\boldsymbol{E}_{l_1}^{(1)}\boldsymbol{E}_{l_2}^{(1)}\boldsymbol{E}_{l_3}^{(1)}\right)}{\partial\tau_i}$$

$$=\frac{\omega_1^2}{c^2}\left\{\begin{array}{l}\frac{\partial}{\partial\omega_1}\left(\tilde{\chi}_e^{(3)}\left[\left|\boldsymbol{E}_1^{x(1)}\right|^2\frac{\partial\boldsymbol{E}_1^{x(1)}}{\partial\tau}\right]\right)+\frac{\partial}{\partial(-\omega_1)}\left(\tilde{\chi}_e^{(3)}\left[\boldsymbol{E}_1^{x(1)}\frac{\partial\left|\boldsymbol{E}_1^{x(1)}\right|^2}{\partial\tau}\right]\right)\\ +\frac{\partial}{\partial\omega_1}\left(\tilde{\chi}_e^{(3)}\left[\frac{2}{3}\left|\boldsymbol{E}_1^{y(1)}\right|^2\frac{\partial\boldsymbol{E}_1^{x(1)}}{\partial\tau}+\frac{1}{3}\boldsymbol{E}_1^{y(1)}\boldsymbol{E}_1^{x(1)*}\frac{\partial\boldsymbol{E}_1^{y(1)}}{\partial\tau}\right]\right)\\ +\frac{\partial}{\partial(-\omega_1)}\left(\tilde{\chi}_e^{(3)}\left[\frac{2}{3}\boldsymbol{E}_1^{x(1)}\frac{\partial\left|\boldsymbol{E}_1^{y(1)}\right|^2}{\partial\tau}+\frac{1}{3}\boldsymbol{E}_1^{y(1)}\frac{\partial\left(\boldsymbol{E}_1^{y(1)}\boldsymbol{E}_1^{x(1)*}\right)}{\partial\tau}\right]\right)\end{array}\right\}. \tag{2.76}$$

Substituting $\boldsymbol{E}_1^{x(1)}=q_1^{(1)}\mathbf{U}_x$, $\boldsymbol{E}_1^{x(2)}=q_1^{(2)}\mathbf{U}_x$, $\boldsymbol{E}_1^{y(1)}=q_2^{(1)}\mathbf{U}_y$, $\boldsymbol{E}_1^{y(2)}=q_2^{(2)}\mathbf{U}_y$, (2.75) and (2.76) into (2.73) and applying $\left(\mathbf{U}_x,\hat{\boldsymbol{L}}_1^x\boldsymbol{E}_1^{x(4)}\right)=0$, and using $\left(\mathbf{U}_x,\hat{\boldsymbol{L}}_1^x\boldsymbol{E}_1^{x(1)}\right)=\left(\mathbf{U}_x,\hat{\boldsymbol{L}}_1^x\boldsymbol{E}_1^{x(2)}\right)=\left(\mathbf{U}_x,\hat{\boldsymbol{L}}_1^x\boldsymbol{E}_1^{x(3)}\right)=0$ gives the following.

$$
\begin{aligned}
&\left(i\frac{\partial q_1^{(2)}}{\partial \xi}-\frac{1}{2}k_{x1}^{''}\frac{\partial^2 q_1^{(2)}}{\partial \tau^2}-\frac{i}{6}k_{x1}^{'''}\frac{\partial^3 q_1^{(1)}}{\partial \tau^3}\right)\left(\mathbf{U}_x,\frac{\partial \hat{\mathbf{L}}_1^x}{\partial k_{x1}}\mathbf{U}_x\right)\\
&-\frac{\omega_1^2}{c^2}\begin{Bmatrix}
2\left|q_1^{(1)}\right|^2 q_1^{(2)}\left(\mathbf{U}_x,\tilde{\chi}_e^{(3)}(\mathbf{U}_x\cdot\mathbf{U}_x^*)\mathbf{U}_x\right)+q_1^{(1)2}q_1^{(2)*}\left(\mathbf{U}_x,\tilde{\chi}_e^{(3)}(\mathbf{U}_x\cdot\mathbf{U}_x^*)\mathbf{U}_x\right)\\
+\frac{1}{3}\left|q_2^{(1)}\right|^2 q_1^{(2)}\left(\mathbf{U}_x,\tilde{\chi}_e^{(3)}(\mathbf{U}_y\cdot\mathbf{U}_y^*)\mathbf{U}_x\right)+\frac{1}{3}q_2^{(1)2}q_1^{(2)*}\left(\mathbf{U}_x,\tilde{\chi}_e^{(3)}(\mathbf{U}_y\cdot\mathbf{U}_x^*)\mathbf{U}_y\right)\\
+\frac{2}{3}q_2^{(1)}q_2^{(2)*}q_1^{(1)}\left(\mathbf{U}_x,\tilde{\chi}_e^{(3)}(\mathbf{U}_y\cdot\mathbf{U}_y^*)\mathbf{U}_x\right)+\frac{2}{3}q_2^{(2)}q_2^{(1)*}q_1^{(1)}\left(\mathbf{U}_x,\tilde{\chi}_e^{(3)}(\mathbf{U}_y\cdot\mathbf{U}_y^*)\mathbf{U}_x\right)\\
+\frac{2}{3}q_2^{(2)}q_2^{(1)}q_1^{(1)*}\left(\mathbf{U}_x,\tilde{\chi}_e^{(3)}(\mathbf{U}_y\cdot\mathbf{U}_x^*)\mathbf{U}_y\right)
\end{Bmatrix}\\
&-i\frac{\omega_1^2}{c^2}\begin{Bmatrix}
\left|q_1^{(1)}\right|^2\frac{\partial q_1^{(1)}}{\partial \tau}\left(\mathbf{U}_x,\frac{\partial}{\partial \omega_1}\left(\chi_e^{(3)}(\mathbf{U}_x\cdot\mathbf{U}_x^*)\mathbf{U}_x\right)\right)\\
+q_1^{(1)}\frac{\partial\left|q_1^{(1)}\right|^2}{\partial \tau}\left(\mathbf{U}_x,\frac{\partial}{\partial(-\omega_1)}\left(\chi_e^{(3)}(\mathbf{U}_x\cdot\mathbf{U}_x^*)\mathbf{U}_x\right)\right)\\
+\frac{2}{3}\left|q_2^{(1)}\right|^2\frac{\partial q_1^{(1)}}{\partial \tau}\left(\mathbf{U}_x,\frac{\partial}{\partial \omega_1}\left(\chi_e^{(3)}(\mathbf{U}_y\cdot\mathbf{U}_y^*)\mathbf{U}_x\right)\right)\\
+\frac{1}{3}q_2^{(1)}q_1^{(1)*}\frac{\partial q_2^{(1)}}{\partial \tau}\left(\mathbf{U}_x,\frac{\partial}{\partial \omega_1}\left(\chi_e^{(3)}(\mathbf{U}_y\cdot\mathbf{U}_x^*)\mathbf{U}_y\right)\right)\\
+\frac{2}{3}q_1^{(1)}\frac{\partial\left|q_2^{(1)}\right|^2}{\partial \tau}\left(\mathbf{U}_x,\frac{\partial}{\partial(-\omega_1)}\left(\chi_e^{(3)}(\mathbf{U}_y\cdot\mathbf{U}_y^*)\mathbf{U}_x\right)\right)\\
+\frac{1}{3}q_2^{(1)}\frac{\partial\left(q_2^{(1)}q_1^{(1)*}\right)}{\partial \tau}\left(\mathbf{U}_x,\frac{\partial}{\partial(-\omega_1)}\left(\chi_e^{(3)}(\mathbf{U}_y\cdot\mathbf{U}_x^*)\mathbf{U}_y\right)\right)
\end{Bmatrix}\\
&=0.
\end{aligned}
\tag{2.77}
$$

Let

$$
\begin{aligned}
&a=\left(\mathbf{U}_x,\frac{\partial \hat{\mathbf{L}}_1^x}{\partial k_{x1}}\mathbf{U}_x\right);\ b_1=\frac{2\omega_1^2}{ac^2}\left(\mathbf{U}_x,\tilde{\chi}_e^{(3)}(\mathbf{U}_x\cdot\mathbf{U}_x^*)\mathbf{U}_x\right)\\
&b_2=\frac{\omega_1^2}{ac^2}\left(\mathbf{U}_x,\tilde{\chi}_e^{(3)}(\mathbf{U}_x\cdot\mathbf{U}_x^*)\mathbf{U}_x\right);\ b_3=\frac{2}{3}\frac{\omega_1^2}{ac^2}\left(\mathbf{U}_x,\tilde{\chi}_e^{(3)}(\mathbf{U}_y\cdot\mathbf{U}_y^*)\mathbf{U}_x\right)\\
&b_4=\frac{1}{3}\frac{\omega_1^2}{ac^2}\left(\mathbf{U}_x,\tilde{\chi}_e^{(3)}(\mathbf{U}_y\cdot\mathbf{U}_x^*)\mathbf{U}_y\right);\ b_5=b_6=b_3\\
&b_7=\frac{2}{3}\frac{\omega_1^2}{ac^2}\left(\mathbf{U}_x,\tilde{\chi}_e^{(3)}(\mathbf{U}_y\cdot\mathbf{U}_x^*)\mathbf{U}_y\right);\ c_1=\frac{\omega_1^2}{ac^2}\left(\mathbf{U}_x,\frac{\partial}{\partial \omega_1}\left(\tilde{\chi}_e^{(3)}(\mathbf{U}_x\cdot\mathbf{U}_x^*)\mathbf{U}_x\right)\right)
\end{aligned}
\tag{2.78a}
$$

$$c_2 = \frac{\omega_1^2}{ac^2}\left(\mathbf{U}_x, \frac{\partial}{\partial)(-\omega_1)}\left(\tilde{\chi}_e^{(3)}(\mathbf{U}_x \cdot \mathbf{U}_x^*)\mathbf{U}_x\right)\right);$$

$$d_1 = \frac{2}{3}\frac{\omega_1^2}{ac^2}\left(\mathbf{U}_x, \frac{\partial}{\partial\omega_1}\left(\tilde{\chi}_e^{(3)}(\mathbf{U}_y \cdot \mathbf{U}_y^*)\mathbf{U}_x\right)\right);$$

$$d_2 = \frac{2}{3}\frac{\omega_1^2}{ac^2}\left(\mathbf{U}_x, \frac{\partial}{\partial(-\omega_1)}\left(\tilde{\chi}_e^{(3)}(\mathbf{U}_y \cdot \mathbf{U}_y^*)\mathbf{U}_x\right)\right); \tag{2.78b}$$

$$e_1 = \frac{1}{3}\frac{\omega_1^2}{ac^2}\left(\mathbf{U}_x, \frac{\partial}{\partial\omega_1}\left(\tilde{\chi}_e^{(3)}(\mathbf{U}_y \cdot \mathbf{U}_x^*)\mathbf{U}_y\right)\right);$$

$$e_2 = \frac{1}{3}\frac{\omega_1^2}{ac^2}\left(\mathbf{U}_x, \frac{\partial}{\partial(-\omega_1)}\left(\tilde{\chi}_e^{(3)}(\mathbf{U}_y \cdot \mathbf{U}_x^*)\mathbf{U}_y\right)\right).$$

Substituting (2.78a) and (2.78b) into (2.77) gives

$$\begin{aligned}
&\left(i\frac{\partial q_1^{(2)}}{\partial\xi} - \frac{1}{2}k_{x1}''\frac{\partial^2 q_1^{(2)}}{\partial\tau^2} - \frac{i}{6}k_{x1}'''\frac{\partial^3 q_1^{(1)}}{\partial\tau^3}\right) \\
&-\left\{\begin{array}{l}
b_1\left|q_1^{(1)}\right|^2 q_1^{(2)} + b_2 q_1^{(1)2} q_1^{(2)*} + b_3\left|q_2^{(1)}\right|^2 q_1^{(2)} \\
+ b_4 q_2^{(1)2} q_1^{(2)*} + b_5 q_2^{(1)} q_2^{(2)*} q_1^{(1)} \\
+ b_6 q_2^{(2)} q_2^{(1)*} q_1^{(1)} + b_7 q_2^{(2)} q_2^{(1)} q_1^{(1)*} \\
+ i\left[\begin{array}{l}
c_1\dfrac{\partial\left(\left|q_1^{(1)}\right|^2 q_1^{(1)}\right)}{\partial\tau} + \left(c_2 - c_1\right)q_1^{(1)}\dfrac{\partial\left|q_1^{(1)}\right|^2}{\partial\tau} \\
+ d_1\dfrac{\partial\left(\left|q_2^{(1)}\right|^2 q_1^{(1)}\right)}{\partial\tau} + \left(d_2 - d_1\right)q_1^{(1)}\dfrac{\partial\left|q_2^{(1)}\right|^2}{\partial\tau} \\
+ e_1\dfrac{\partial\left(q_2^{(1)} q_1^{(1)*} q_2^{(1)}\right)}{\partial\tau} + \left(e_2 - e_1\right)q_1^{(1)}\dfrac{\partial\left|q_1^{(1)}\right|^2}{\partial\tau}
\end{array}\right]
\end{array}\right\} \\
&= 0.
\end{aligned} \tag{2.79}$$

Let $c_2 - c_1 = f_1$, $d_2 - d_1 = g_1$, and $e_2 - e_1 = h_1$. Also note that $q_1\mathbf{U}_x = \left(\varepsilon q_1^{(1)} + \varepsilon^2 q_1^{(2)}\right)\mathbf{U}_x$ and $q_2\mathbf{U}_x = \left(\varepsilon q_2^{(1)} + \varepsilon^2 q_2^{(2)}\right)\mathbf{U}_x$. This implies that at order ε, $q_1^{(1)} \approx \frac{q_1}{\varepsilon}$ and $q_2^{(1)} \approx \frac{q_2}{\varepsilon}$, and at order ε^2, $q_1^{(2)} \approx \frac{q_1}{\varepsilon^2}$ and $q_2^{(2)} \approx \frac{q_2}{\varepsilon^2}$. Substituting for $q_1^{(1)}$, $q_1^{(2)}$, $q_2^{(1)}$ and $q_2^{(2)}$ in (2.79), setting $\widetilde{b}_1 = (b_1 + b_2)/\varepsilon^2$, $\widetilde{b}_2 = (b_3 + b_5 + b_6)/\varepsilon^2$, and $\widetilde{b}_3 = (b_4 + b_7)/\varepsilon^2$, and multiplying by ε^2 gives

$$\left(i\frac{\partial q_1}{\partial \xi} - \frac{1}{2}k_{x1}'' \frac{\partial^2 q_1}{\partial \tau^2} - \frac{i\varepsilon}{6}k_{x1}''' \frac{\partial^3 q_1}{\partial \tau^3}\right) - \left\{\begin{array}{l} \widetilde{b}_1 |q_1|^2 q_1 + \widetilde{b}_2 |q_2|^2 q_1 + \widetilde{b}_3 q_2{}^2 q_1^* \\ + i\varepsilon \left[\begin{array}{l} \widetilde{c}_1 \dfrac{\partial(|q_1|^2 q_1)}{\partial \tau} + \widetilde{f}_1 q_1 \dfrac{\partial |q_1|^2}{\partial \tau} \\ + \widetilde{d}_1 \dfrac{\partial(|q_2|^2 q_1)}{\partial \tau} + \widetilde{g}_1 q_1 \dfrac{\partial |q_2|^2}{\partial \tau} \\ + \widetilde{e}_1 \dfrac{\partial(q_2 q_1^* q_2)}{\partial \tau} + \widetilde{h}_1 q_2 \dfrac{\partial(q_2 q_1^*)}{\partial \tau} \end{array}\right] \end{array}\right\} = \mathbf{O}(\varepsilon^3). \tag{2.80}$$

Here $\widetilde{c}_1 = c_1/\varepsilon^2$, $\widetilde{d}_1 = d_1/\varepsilon^2$, $\widetilde{e}_1 = e_1/\varepsilon^2$, $\widetilde{f}_1 = f_1/\varepsilon^2$, $\widetilde{g}_1 = g_1/\varepsilon^2$, and $\widetilde{h}_1 = h_1/\varepsilon^2$. Similarly for $\boldsymbol{E}_1^y = q_2 \mathbf{U}_y = (\varepsilon q_2^{(1)} + \varepsilon^2 q_2^{(2)})\mathbf{U}_y$, using operators and $\hat{\boldsymbol{L}}_l^y$ and $\hat{\boldsymbol{N}}_{l_1 l_2 l_3}^y$ for the y-polarization in the reductive perturbation method, gives for $l=1$

$$\left(i\frac{\partial q_2}{\partial \xi} - \frac{1}{2}k_{y1}'' \frac{\partial^2 q_2}{\partial \tau^2} - \frac{i\varepsilon}{6}k_{y1}''' \frac{\partial^3 q_2}{\partial \tau^3}\right) - \left\{\begin{array}{l} \widetilde{b}_1^* |q_2|^2 q_2 + \widetilde{b}_2^* |q_1|^2 q_2 + \widetilde{b}_3^* q_1{}^2 q_2^* \\ + i\varepsilon \left[\begin{array}{l} \widetilde{c}_1^* \dfrac{\partial(|q_2|^2 q_2)}{\partial \tau} + \widetilde{f}_1^* q_2 \dfrac{\partial |q_2|^2}{\partial \tau} \\ + \widetilde{d}_1^* \dfrac{\partial(|q_1|^2 q_2)}{\partial \tau} + \widetilde{g}_1^* q_2 \dfrac{\partial |q_1|^2}{\partial \tau} \\ + \widetilde{e}_1^* \dfrac{\partial(q_1 q_2^* q_1)}{\partial \tau} + \widetilde{h}_1^* q_1 \dfrac{\partial(q_1 q_2^*)}{\partial \tau} \end{array}\right] \end{array}\right\} = \mathbf{O}(\varepsilon^3). \tag{2.81}$$

Here

$$\begin{aligned}
&k_{1y} = k_{1x}(\mathbf{U}_x \leftrightarrow \mathbf{U}_y);\ \tilde{b}_1^* = \tilde{b}_1(\mathbf{U}_x \leftrightarrow \mathbf{U}_y);\ \tilde{b}_2^* = \tilde{b}_2(\mathbf{U}_x \leftrightarrow \mathbf{U}_y)\\
&\tilde{b}_3^* = \tilde{b}_3(\mathbf{U}_x \leftrightarrow \mathbf{U}_y);\ \tilde{c}_1^* = \tilde{c}_1(\mathbf{U}_x \leftrightarrow \mathbf{U}_y);\ \tilde{d}_1^* = \tilde{d}_1(\mathbf{U}_x \leftrightarrow \mathbf{U}_y)\\
&\tilde{e}_1^* = \tilde{e}_1(\mathbf{U}_x \leftrightarrow \mathbf{U}_y);\ \tilde{f}_1^* = \tilde{f}_1(\mathbf{U}_x \leftrightarrow \mathbf{U}_y);\ \tilde{g}_1^* = \tilde{g}_1(\mathbf{U}_x \leftrightarrow \mathbf{U}_y)\\
&\tilde{h}_1^* = \tilde{h}_1(\mathbf{U}_x \leftrightarrow \mathbf{U}_y).
\end{aligned} \tag{2.82}$$

The operation $\left(\mathbf{U}_x \leftrightarrow \mathbf{U}_y\right)$ means $\mathbf{U}_x$ and $\mathbf{U}_y$ are interchanged where available in the expression. In equation (2.80) and (2.81) the second to the twelfth term represent SOD, TOD, SPM, cross phase modulation (CPM), four-wave mixing (FWM), SS, SFS, cross steepening (CS), cross frequency shifting (CFS), steepening associated with four-wave mixing, and frequency shifting associated with four-wave mixing.

2.3 Soliton-Dragging and Trapping Inverter Gates

The analytical model of the soliton-dragging and trapping inverter logic gates consists of a specific length L_d of a high birefringent singlemode fiber, that ensures that the control pulse arrival delay time is different when a signal is present and when it is not. Typically the SDG/STG inverter gate length should be a few soliton periods. The Boolean time-shift keying logic operation of a SDG/STG inverter gate, schematically represented in Fig. 1.2, is given by

$$F(L_d) = \begin{cases} 1 & \Delta\tau_{cadt}(L_d) - 4T_0 \prec \Delta\tau_{csadt}(L_d) \prec \Delta\tau_{cadt}(L_d) + 4T_0 \\ 0 & \text{otherwise} \end{cases} . \tag{2.83}$$

Here $\Delta\tau_{csadt}$ is the arrival delay time of the control pulse with the control and signal pulses co-propagating over length L_d, and $\Delta\tau_{cadt}$ is the arrival delay time of the control pulse with the control pulse propagating alone over length L_d. T_0 is the half-width at e^{-1} intensity of the input control and signal pulses. The arrival delay times are referenced with respect to the peak-amplitude arrival delay time of the control pulse. The soliton-dragging and trapping gates mainly rely on CPM for their switching operation, and their switching operation is affected by the inclusion of higher order linear and nonlinear effects, namely TOD, SS, SFS, CS and CFS. The inclusion of the higher order linear and nonlinear effects arises from the use of soliton pulses with pulse widths below 100 fs (ultrashort pulses). Ultrashort pulses are required to achieve bit rates greater than 50 Gbps (ultrafast bit rates). The higher order linear and nonlinear effects tend to prevent the control pulse from arriving within the clock time window of the gate, namely from greater than $\Delta\tau_{cadt}(L_d) - 4T_0$ to less than $\Delta\tau_{cadt}(L_d) + 4T_0$, assuming a clock time window of $8T_0$ ($\approx$ 4 pulse widths). These effects will tend to affect the switching efficiency of a SDG/STG inverter gate. A relative measure of the switching efficiency of a SDG/STG inverter gate can be expressed as follows.

$$\eta(L_d) = \begin{cases} \dfrac{\left|\Delta\tau_{csadt}(L_d) - \Delta\tau_{cadt}(L_d)\right|}{4T_0} & \Delta\tau_{cadt}(L_d) - 4T_0 \le \Delta\tau_{csadt} \le \Delta\tau_{cadt}(L_d) + 4T_0 \\ 1.00 & \text{otherwise} \end{cases} \tag{2.84}$$

An efficiency $\eta(L_D)$ of less than one implies the SDG/STG inverter gate can not perform its function with respect to its clock time window. The value of the efficiency multiplied by the clock time window, $\eta(L_D)8T_0$, gives the maximum clock time window required for time-shift keying logic switching, and is a measure of the switching degradation.

2.4 NOLM, STG-NOLM GATE, STG-PSI-NOLM GATE MODELS

The analytical models of the STG-NOLM and STG-PSI-NOLM gates are based on the analytical models of the STG and NOLM. The analytical model of the STG has already been presented. This section now proceeds to give the analytical model of a NOLM.

The analytical model governing the interferometric switching characteristics of an NOLM, schematically represented in Fig. 1.3, are given by [2.21]

$$\begin{aligned} \boldsymbol{E}_3 &= \alpha^{1/2}\boldsymbol{E}_1 + i(1-\alpha)^{1/2}\boldsymbol{E}_2, \\ \boldsymbol{E}_4 &= i(1-\alpha)^{1/2}\boldsymbol{E}_1 + \alpha^{1/2}\boldsymbol{E}_2 \end{aligned} \tag{2.85}$$

The coefficient α is the coupling coefficient of the NOLM. For the STG-NOLM and STG-PSI-NOLM gates, the coupling coefficient $\alpha = 0.5$.

In the STG-NOLM gate and STG-PSI-NOLM gate, the analytical single soliton pulse propagation model is used for the counterclockwise control pulse in the NOLM, and the analytical two orthogonally polarized soliton pulse propagation model, is used for the clockwise control and signal pulses (constituting the STG). The interferometric switching in the two gates are effected via equations (2.85).

2.5 SINGLEMODE STEP-INDEX FIBER ANALYTICAL MODEL

The fiber SMF01 is a computer simulated singlemode birefringent step-index fiber model, based on the weakly-guiding approximation condition. Derivation of the field equations for the linearly-polarized (LP) "pseudo-modes", that is *EH* or *HE* modes, for a weakly-guiding singlemode fiber are given by Adams [2.13]. The properties of the total dispersion D of the singlemode fiber SMF01 is based on the following formula [2.14]

$$D = \frac{1}{c}\frac{dm}{d\lambda} = -\frac{\lambda}{c}\frac{d^2 n}{d\lambda^2}. \tag{2.86}$$

Here n and m are the effective refractive index and group index respectively, and λ is the wavelength. The velocity of light is denoted as c. The total dispersion D can be decomposed into the following components

$$D = -(M_{cmd} + M_{wd} + M_{cpd} + M_r). \tag{2.87}$$

Here M_{cmd}, M_{wd}, M_{cpd} and M_{r} represent the composite material dispersion, waveguide dispersion, composite profile dispersion and the remainder dispersion. The dispersion components are given as follows.

$$\begin{aligned}
M_{cmd} &= \frac{\lambda}{c}\left[\Gamma\frac{d^2 n_1}{d\lambda^2} + (1-\Gamma)\frac{d^2 n_2}{d\lambda^2}\right], \\
M_{wd} &= \frac{n_1\Delta}{c\lambda}\left(\frac{m_1}{n_1}\right)^2 v\frac{d^2(bv)}{dv^2}, \\
M_{cpd} &= \frac{n_1\Delta'}{c}\left(\frac{\lambda\Delta'}{4\Delta} - \frac{m_1}{n_1}\right)\left[2(\Gamma-b) + v\frac{d^2(bv)}{dv^2}\right], \\
M_r &= \frac{1}{\lambda c n}\left[m_1^2 2\Delta(\Gamma-b) - m^2 + m_1^2\Gamma + m_2^2(1-\Gamma)\right].
\end{aligned} \tag{2.88}$$

Here n_1 and n_2 are the refractive indices of the core and cladding respectively. The group indices of the core and cladding are m_1 and m_2 respectively. The relative refractive index is denoted as Δ, and $\Delta' = d(\Delta)/d\lambda$. The normalized propagation constant is denoted by b and is defined as

$$b = 1 - \left(\frac{u}{v}\right)^2, \tag{2.89}$$

where u and v are the radial phase parameter and the normalized frequency respectively. The parameter u and v are related to the cladding decay parameter w by

$$v^2 = u^2 + w^2. \tag{2.90}$$

The power confinement factor Γ is given by

$$\Gamma = \frac{1}{2}\left[b + \frac{d(bv)}{dv}\right]. \tag{2.91}$$

A first approximation of w is found through [2.15]

$$w \approx 1.1428v - 0.9960, \tag{2.92}$$

and then iteratively optimized using the characteristic equation for the fundamental mode of a step-index fiber [2.16, 2.17] given by

$$u\frac{J_1(u)}{J_0(u)} = w\frac{K_1(w)}{K_0(w)}. \tag{2.93}$$

The zeroth and first order Bessel functions of the first kind are denoted by J_0 and J_1 respectively. The zeroth and first order modified Bessel functions of the second kind are denoted by K_0 and K_1 respectively. The term $vd^2(bv)/dv^2$ can be evaluated using [2.18]

$$v\frac{d^2(bv)}{dv^2} = \frac{2u^2k_l(w)}{v^2w^2}\left\{\begin{array}{l}\left[3w^2 - 2k_l(w)(w^2 - u^2)\right] \\ + w\left(w^2 + u^2k_l(w)\right)\left(k_l(w) - 1\right)\left[\dfrac{K_{l-1}(w) + K_{l+1}(w)}{K_l(w)}\right]\end{array}\right\}. \tag{2.94a}$$

Here $k_l(w)$ is defined as

$$k_l(w) = \frac{K_l^2(w)}{K_{l-1}(w)K_{l+1}(w)}, \tag{2.94b}$$

and l denotes the first mode number of a LP_{lM} mode. For a singlemode fiber $l = 0$. The term $d(bv)/dv$ can be evaluated using [2.15]

$$\frac{d(bv)}{dv} = \frac{d}{dv}\left(v - \frac{u^2}{v}\right) = 1 - \frac{u^2}{v^2}\left[1 - 2k_l(w)\right]. \tag{2.95}$$

The composition of the core of SMF01 is based on 13.5 $^m/_o$ GeO_2 and 86.5 $^m/_o$ SiO_2 with the following Sellmeier parameters [2.19]

$$\begin{aligned} &A_1 = 0.711040,\ \lambda_1 = 0.064270\ \mu\text{m};\ A_2 = 0.451885,\ \lambda_2 = 0.129408\ \mu\text{m} \\ &A_3 = 0.704048,\ \lambda_3 = 9.425478\ \mu\text{m}. \end{aligned} \tag{2.96}$$

The composition of the cladding of SMF01 is based on SiO_2 with the following Sellmeier parameters [2.20]

$$\begin{aligned} &A_1 = 0.6961663,\ \lambda_1 = 0.0684043\ \mu\text{m};\ A_2 = 0.4079426,\ \lambda_2 = 0.1162414\ \mu\text{m} \\ &A_3 = 0.8974794,\ \lambda_3 = 9.896161\ \mu\text{m}. \end{aligned} \tag{2.97}$$

The refractive index is given by the Sellmeier equation

$$n^2(\lambda) = \sum_{i=1}^{3} \frac{A_i \lambda^2}{\left(\lambda^2 - \lambda_i^2\right)}. \tag{2.98}$$

The dispersion characteristic graphs for SMF01 with a core diameter of 4.54 μm are given in Appendix A. The birefringence of the fiber, $B = \left|n_x - n_y\right|$, is a simulation variable. The effective refractive indices of the two orthogonal polarizations are denoted by n_x and n_y, with $n_x > n_y$. This implies that the slow axis is the x-axis and the y–axis is the fast axis. It is assumed that the second and higher order dispersion parameters are equal for the two orthogonal polarizations.

CHAPTER III

NUMERICAL MODELLING

The analytical soliton pulse propagation models derived in Chapter II do not lend themselves to analytical solutions except for some specific in which the inverse scattering method [3.1] can be used. It is therefore necessary to employ numerical methods to solve the analytical models such as the finite-difference and pseudospectral methods. Pseudospectral methods are generally faster by an order of magnitude or more to achieve the same accuracy [3.2]. An extensively used pseudospectral method is the split-step Fourier method [3.3, 3.4], and a variant of this method is employed, namely the symmetrized split-step Fourier method [3.5], to solve the analytical models of Chapter II.

3.1 SYMMETRIZED SPLIT-STEP FOURIER METHOD

The split-step Fourier method is based on the following philosophy. The NLS equation can be formally written as

$$\frac{\partial A}{\partial z} = \left(\hat{\boldsymbol{D}} + \hat{\boldsymbol{N}}\right)A , \tag{3.1}$$

where $\hat{\boldsymbol{D}}$ is a differential operator that accounts for dispersion and absorption in a linear medium, and $\hat{\boldsymbol{N}}$ is a nonlinear operator that governs the effects of fiber nonlinearities on pulse propagation. In general, dispersion and nonlinearly act together along the length of a fiber. The split-step Fourier method obtains an approximate solution by assuming that in propagating the optical field over a small distance *h*, the dispersive and nonlinear effects can be pretended to act independently. Typically, propagation from *z* to *z*+*h* is carried out in two steps. In the first step, the nonlinearity acts alone, and $\hat{\boldsymbol{D}} = 0$. In the second step, dispersion acts alone, and $\hat{\boldsymbol{N}} = 0$. Mathematically, this is implemented as

$$A(z+h,T) \approx \exp(h\hat{\boldsymbol{D}})\exp(h\hat{\boldsymbol{N}})A(z,T) . \tag{3.2}$$

The nonlinear operation $B(z,T) = \exp(h\hat{\boldsymbol{N}})A(z,T)$ is executed in the time domain, followed by the dispersion operation executed in the frequency domain using

$$A(z+h,T) \approx \exp(h\hat{\boldsymbol{D}})B(z,T) = \left\{\boldsymbol{F}^{-1}\exp\left[h\hat{\boldsymbol{D}}(i\omega)\right]\boldsymbol{F}\right\}B(z,T) . \tag{3.3}$$

Here $\boldsymbol{F}$ denotes the Fourier transform operation, $\hat{\boldsymbol{D}}(i\omega)$ is the dispersion equation with the differential operator $\partial / \partial T$ replaced by $i\omega$, and ω is the frequency in the Fourier frequency domain. The use of the FFT algorithm [3.6] makes the numerical evaluation of (3.3) relatively fast.

The split-step Fourier method is accurate to second order in the step size *h*, in accordance to the Baker-Hausdorff formula [3.7], for two noncommuting operators such as $\hat{a}$ and $\hat{b}$, given by

$$\exp(\hat{a})\exp(\hat{b}) = \exp\left(\hat{a} + \hat{b} + \frac{1}{2}\left[\hat{a},\ \hat{b}\right] + \frac{1}{12}\left[\hat{a} - \hat{b},\ \left[\hat{a},\ \hat{b}\right]\right] + \ldots\right) . \tag{3.4}$$

Here $\left[\hat{a},\ \hat{b}\right] = \hat{a}\hat{b} - \hat{b}\hat{a}$. The dominant error for the split-step Fourier method is due to the single commutator $1/2\left[\hat{a},\ \hat{b}\right] = (1/2)h^2\left[\hat{\boldsymbol{D}},\ \hat{\boldsymbol{N}}\right]$, which is of second order in the step size *h*.

The accuracy of the split-step Fourier method can be improved by including the nonlinearly effect in the middle of the computation segment, rather than at the boundary as in

$$A(z+h,\ T) \approx \exp\left(\frac{h}{2}\hat{\boldsymbol{D}}\right)\exp\left(\int_z^{z+h}\hat{\boldsymbol{N}}(z')dz'\right)\exp\left(\frac{h}{2}\hat{\boldsymbol{D}}\right)A(z,\ T). \tag{3.5}$$

The integral in the middle exponential is approximated by $\exp(h\hat{N})$ since the step size *h* to be used is assumed to be small enough. For the symmetrized split-step Fourier method, the dominant error is due to the double commutator $1/12\left[\hat{a}-\hat{b},\ \left[\hat{a},\ \hat{b}\right]\right] = (1/12)h^3\left[\hat{\boldsymbol{D}}-\hat{\boldsymbol{N}},\ \left[\hat{\boldsymbol{D}},\ \hat{\boldsymbol{N}}\right]\right]$, which is of third order in the step size *h*.

Implementation of the symmetrized split-step Fourier method is as follows. The fiber length is divided into a large number of segments, of width *h*, that need not be of the same length. The optical pulse field *A*(*z*, *T*) is first propagated with dispersion for a distance *h*/2. At the midplane z+*h*/2, the field is multiplied by the nonlinear term that represents the effect of nonlinearly over the whole segment width *h*. Finally, the field is propagated the remaining distance *h*/2 with dispersion only to obtain *A*(*z*+*h*, *T*).

3.2 NUMERICAL PROPAGATION MODELS

In Chapter II two soliton propagation analytical models were derived, namely one for single soliton pulse propagation, and the other for the propagation of two orthogonally polarized soliton pulses. From the analytical propagation models the respective numerical propagation models were derived using the symmetrized split-step Fourier method.

3.2.1 Single Soliton Pulse Propagation Model

The numerical model for single soliton pulse propagation can be derived from equation (2.52) in the form

$$\begin{aligned}&\frac{\partial A}{\partial z}+\frac{\alpha}{2}A+\frac{i}{2}k_1''\frac{\partial^2 A}{\partial T^2}-\frac{1}{6}k_1'''\frac{\partial^3 A}{\partial T^3}\\&=i\gamma\left\{|A|^2A+\frac{2i}{\omega_1}\frac{\partial\left(|A|^2A\right)}{\partial T}-T_RA\frac{\partial|A|^2}{\partial T}\right\}.\end{aligned}\tag{3.6}$$

A is the slow varying complex amplitude of the pulse, $\gamma=(n_2\omega_1)/(cA_{eff})$, and T_R is related to the slope of the Raman gain [3.8] and its approximately 5 fs. The attenuation is denoted by α. Here the nonlinear refractive index coefficient n_2 and the effective fiber core area A_{eff} are defined as

$$\begin{aligned}n_2&=\frac{3}{8n}\chi^{(3)}_{xxxx},\\A_{eff}&=\pi w^2.\end{aligned}\tag{3.7}$$

Here n is the linear refractive index, and w is the cladding decay parameter, also known as the mode-width parameter. The following normalization is applied to (3.6)

$$\begin{aligned} &U=\frac{A}{\sqrt{P_0}},\ \xi=\frac{z}{L_D},\ \tau=\frac{T}{T_0}=\frac{t-z/v_g}{T_0} \\ &L_D=\frac{T_0^2}{\left|k_1^{''}\right|},\ L_{NL}=\frac{T_0^2}{\gamma P_0},\ \text{and}\ N^2=\frac{L_D}{L_{NL}}=\frac{\gamma P_0 T_0^2}{\left|k_1^{''}\right|}. \end{aligned} \tag{3.8}$$

Here L_D is the dispersion length, L_{NL} is the nonlinear length, P_0 is the peak pulse power, v_g is the group velocity, and T_0 is the half-width at the e^{-1} intensity. Applying the normalization leads to

$$\begin{aligned} &\frac{\partial U}{\partial \xi}=\left(\hat{\boldsymbol{D}}+\hat{\boldsymbol{N}}\right)U, \\ &\hat{\boldsymbol{D}}=-\operatorname{sgn}(k_1^{''})\frac{i}{2}\frac{\partial^2}{\partial\tau^2}+\delta\frac{\partial^3}{\partial\tau^3}-\frac{L_D\alpha}{2}, \\ &\hat{\boldsymbol{N}}=iN^2\left\{|U|^2+\frac{is}{U}\frac{\partial\left(|U|^2U\right)}{\partial\tau}-\tau_R\frac{\partial\left(|U|^2\right)}{\partial\tau}\right\}, \\ &\delta=\frac{k_1^{'''}}{6\left|k_1^{''}\right|T_0},\ s=\frac{2}{\omega_1 T_0},\ \text{and}\ \tau_R=\frac{T_R}{T_0}. \end{aligned} \tag{3.9a}$$

Using $\partial/\partial T \leftrightarrow i\omega$ and $T=\tau T_0$ gives

$$\begin{aligned} &\hat{\boldsymbol{D}}=\operatorname{sgn}(k_1^{''})\frac{i}{2}\omega^2T_0^2-i\delta\omega^3T_0^3-\frac{L_D\alpha}{2}, \\ &\omega=2\pi f_d;\ f_d=\frac{d}{N_0T_s};\ -\frac{\mathrm{N}_0}{2}\le d\le\frac{\mathrm{N}_0}{2}. \end{aligned} \tag{3.9b}$$

Here N_0 is the number of sampling points at sampling interval T_s. The numerical model given by (3.9) is used to simulate the propagation of a single soliton in a singlemode fiber, with zero attenuation.

3.2.2 Two Orthogonally Polarized Soliton Pulses Propagation Model

The numerical model for the co-propagation of two orthogonally polarized soliton pulses can also be similarly derived from equation (2.80) and (2.81) as follows. Making the following assumptions

$$\begin{aligned}
&T_0 = T_{x0} = T_{y0};\ k_{x1} \neq k_{y1};\ k_{x1}^{'} \neq k_{y1}^{'};\ k_1^{''} = k_{x1}^{''} = k_{y1}^{''};\ k_1^{'''} = k_{x1}^{'''} = k_{y1}^{'''} \\
&B = \Delta n = \left|n_x - n_y\right| \\
&n_y + \Delta n = n_x \ (\text{Since } n_x \succ n_y\ x-\text{axis is slow and } y\text{ - axis is fast}) \\
&\Delta k = k_{x1} - k_{y1}\ .
\end{aligned} \tag{3.10}$$

Here the subscripts x and y denotes the respective parameter in the orthogonal polarizations. B is the fiber modal birefringence with a range of values from 5.0×10^{-9} to 8.0×10^{-4} in practical fibers [3.9]. Taking only the x-polarization equation of the CHNLS equation (2.80) in the form

$$\frac{\partial A_x}{\partial z} + \frac{\alpha}{2} A_x + k_{x1}^{'} \frac{\partial A_x}{\partial T} + \frac{i}{2} k_{x1}^{''} \frac{\partial^2 A_x}{\partial T^2} - \frac{1}{6} k_{x1}^{'''} \frac{\partial^3 A_x}{\partial T^3}$$

$$= i\gamma \left\{ \begin{aligned}
&\left|A_x\right|^2 A_x + \frac{2}{3}\left|A_y\right|^2 A_x + \frac{1}{3} A_y^2 A_x^* e^{-i2(\Delta k)z} \\
&+ \frac{2i}{\omega_1}\left[\frac{\partial\left(\left|A_x\right|^2 A_x\right)}{\partial T} + \frac{2}{3}\frac{\partial\left\{\left|A_y\right|^2 A_x\right\}}{\partial T} + \frac{1}{3}\frac{\partial\left(A_y^2 A_x^*\right)}{\partial T} e^{-i2(\Delta k)z} \right] \\
&- T_R\left[A_x \frac{\partial\left|A_x\right|^2}{\partial T} + \frac{2}{3} A_x \frac{\partial\left|A_y\right|^2}{\partial T} + \frac{1}{3} A_y \frac{\partial\left(A_y A_x^*\right)}{\partial T} e^{-i2(\Delta k)z} \right]
\end{aligned} \right\}, \tag{3.11}$$

and applying similar normalization as in (3.8) gives

$$\frac{\partial U_x}{\partial \xi}+\frac{L_D\alpha}{2}U_x+k'_{x1}\frac{L_D}{T_0}\frac{\partial U_x}{\partial \tau}+\frac{i}{2}\operatorname{sgn}(k''_{x1})\frac{\partial^2 U_x}{\partial \tau^2}-\delta\frac{\partial^3 U_x}{\partial \tau^3}$$

$$=i\left\{\begin{array}{l}\gamma P_{0x}L_D|U_x|^2U_x+\frac{2}{3}\gamma P_{0y}L_D|U_y|^2U_x+\frac{1}{3}\gamma P_{0y}L_DU_y^2U_x^*e^{-i\Delta R\xi}\\ +is\left[\gamma P_{0x}L_D\frac{\partial\left(|U_x|^2U_x\right)}{\partial\tau}+\frac{2}{3}\gamma P_{0y}L_D\frac{\partial\left\{|U_y|^2U_x\right\}}{\partial\tau}+\frac{1}{3}\gamma P_{0y}L_D\frac{\partial\left(U_y^2U_x^*\right)}{\partial\tau}e^{-i\Delta R\xi}\right]\\ -\tau_R\left[\gamma P_{0x}L_DU_x\frac{\partial|U_x|^2}{\partial\tau}+\frac{2}{3}\gamma P_{0y}L_DU_x\frac{\partial|U_y|^2}{\partial\tau}+\frac{1}{3}\gamma P_{0y}L_DU_y\frac{\partial\left(U_yU_x^*\right)}{\partial\tau}e^{-i\Delta R\xi}\right]\end{array}\right\}. \qquad (3.12)$$

Here $R=4\omega_1T_0=\left(8\pi cT_0/\lambda_1\right)$ and $\Delta=(B/c)T_0/|k''_1|=\left(k'_{x1}-k'_{y1}\right)T_0/\left(2|k''_1|\right)$. The normalized reference frame group velocity is $1/\Delta$. Let

$$\begin{array}{l}N_x^2=\gamma P_{0x}L_D,\ N_y^2=\gamma P_{0y}L_D,\\ u_x=N_xU_x\cos\theta \text{ and } u_y=N_yU_y\sin\theta.\end{array} \qquad (3.13)$$

Here the polarization angle θ determines the relative strengths of the partial soliton pulses in each of the two polarizations. Substituting (3.13) into (3.12) gives the following.

$$\frac{\partial u_x}{\partial \xi}+\frac{L_D\alpha}{2}u_x+k'_{x1}\frac{L_D}{T_0}\frac{\partial u_x}{\partial \tau}+\frac{i}{2}\operatorname{sgn}(k''_{x1})\frac{\partial^2 u_x}{\partial \tau^2}-\delta\frac{\partial^3 u_x}{\partial \tau^3}$$

$$=i\left\{\begin{array}{l}|u_x|^2u_x+\frac{2}{3}|u_y|^2u_x+\frac{1}{3}u_y^2u_x^*e^{-i\Delta R\xi}\\ +is\left[\frac{\partial\left(|u_x|^2u_x\right)}{\partial\tau}+\frac{2}{3}\frac{\partial\left\{|u_y|^2u_x\right\}}{\partial\tau}+\frac{1}{3}\frac{\partial\left(u_y^2u_x^*\right)}{\partial\tau}e^{-i\Delta R\xi}\right]\\ -\tau_R\left[u_x\frac{\partial|u_x|^2}{\partial\tau}+\frac{2}{3}u_x\frac{\partial|u_y|^2}{\partial\tau}+\frac{1}{3}u_y\frac{\partial\left(u_yu_x^*\right)}{\partial\tau}e^{-i\Delta R\xi}\right]\end{array}\right\}. \qquad (3.14)$$

For this study the singlemode fiber is assumed to be strongly birefringent such that $\Delta R \succ\succ 1$ in the operating window. This implies that the exponential varying terms can therefore be neglected in the numerical model. Equation (3.14) can be amicably written in the form

$$\frac{\partial u_x}{\partial \xi} = \left(\hat{\boldsymbol{D}}_x + \hat{\boldsymbol{N}}_x\right)u_x,$$
$$\hat{\boldsymbol{D}}_x = -\frac{L_D\alpha}{2} + \Delta\frac{\partial}{\partial\tau} - \operatorname{sgn}(k_1^{''})\frac{i}{2}\frac{\partial^2}{\partial\tau^2} + \delta\frac{\partial^3}{\partial\tau^3}, \tag{3.15a}$$
$$\hat{\boldsymbol{N}}_x = i\left\{|u_x|^2 + \frac{2}{3}|u_y|^2 + \frac{is}{u_x}\left[\frac{\partial\left(|u_x|^2 u_x\right)}{\partial\tau} + \frac{2}{3}\frac{\partial\left\{|u_y|^2 u_x\right\}}{\partial\tau}\right] - \tau_R\left[\frac{\partial|u_x|^2}{\partial\tau} + \frac{2}{3}\frac{\partial|u_y|^2}{\partial\tau}\right]\right\}.$$

Using $\partial/\partial T \leftrightarrow i\omega$ and $T = \tau T_0$ in the dispersion operator $\hat{\boldsymbol{D}}_x$ gives

$$\hat{\boldsymbol{D}}_x = -\frac{L_D\alpha}{2} + i\Delta\omega T_0 + \operatorname{sgn}(\beta_2)\frac{i}{2}\omega^2 T_0^2 - i\delta\omega^3 T_0^3. \tag{3.15b}$$

Applying the same mathematical operation for the y-polarization equation of the CHNLS equation (2.81) gives

$$\frac{\partial u_y}{\partial \xi} = \left(\hat{\boldsymbol{D}}_y + \hat{\boldsymbol{N}}_y\right)u_y,$$
$$\hat{\boldsymbol{D}}_y = -\frac{L_D\alpha}{2} - \Delta\frac{\partial}{\partial\tau} - \operatorname{sgn}(k_1^{''})\frac{i}{2}\frac{\partial^2}{\partial\tau^2} + \delta\frac{\partial^3}{\partial\tau^3}, \tag{3.16a}$$
$$\hat{\boldsymbol{N}}_y = i\left\{|u_y|^2 + \frac{2}{3}|u_x|^2 + \frac{is}{u_y}\left[\frac{\partial\left\{|u_y|^2 u_y\right\}}{\partial\tau} + \frac{2}{3}\frac{\partial\left(|u_x|^2 u_y\right)}{\partial\tau}\right] - \tau_R\left[\frac{\partial|u_y|^2}{\partial\tau} + \frac{2}{3}\frac{\partial|u_x|^2}{\partial\tau}\right]\right\}.$$

Using $\partial/\partial T \leftrightarrow i\omega$ and $T = \tau T_0$ in the dispersion operator $\hat{\boldsymbol{D}}_y$ gives

$$\hat{\boldsymbol{D}}_y = -\frac{L_D\alpha}{2} - i\Delta\omega T_0 + \operatorname{sgn}(k_1^{''})\frac{i}{2}\omega^2 T_0^2 - i\delta\omega^3 T_0^3. \tag{3.16b}$$

The terms with Δ represent the polarization dispersion due to modal birefringence. It is assumed that in the positive direction of the delay time frame τ, the pulse arrival time is delayed, and hence the sign before Δ is positive in the x-polarization axis (slow axis) and negative in the y-polarization axis (fast axis).

The numerical model given by the coupled equation (3.15) and (3.16) is used to simulate the propagation of two orthogonally polarized soliton pulses in a STG and SDG, with zero attenuation.

CHAPTER IV

COMPUTER SIMULATION ALGORITHMS

The numerical soliton pulse propagation models developed in Chapter III, namely (3.9), (3.15) and (3.16), need to be implemented on a computer for numerical evaluation. This in turn enables the analysis of higher order linear and nonlinear effects, namely TOD, SS, SFS, CS and CFS. Here the computer program modules that facilitate the simulation of the propagation of a single soliton pulse, and two orthogonally polarized soliton pulses in a singlemode fiber are presented.

All C^{++} source code listing for the computer program modules are given in Appendix C. The programming modules are sufficiently commented to provide functionality information.

4.1 Soliton Pulse Propagation Modules

The executable main program module for single soliton pulse propagation is "wlength.exe" and its corresponding source code is "wlength.cpp". The program module "wlength.cpp" provides the execution shell, and invokes the program module "bpm.cpp" to simulate the single soliton pulse propagation. The program module "bpm.cpp" depends on the singlemode fiber characteristic and mathematics modules described in later sections.

Similarly the executable main program module for the propagation of two orthogonally polarized soliton pulses is "wlengtha.exe", and its corresponding source code is "wlengtha.cpp". The program module "wlengtha.cpp" provides the execution shell, and invokes the program module "bpma.cpp" to simulate the propagation of two orthogonally polarized soliton pulses. The program module "bpma.cpp" also depends on the singlemode fiber characteristic and mathematics modules described in later sections.

The following functions make up the modules bpm.cpp/bpma.cpp:

(1) void dfft(dcomplex *bprofile,unsigned int size,double type): Performs complex discrete fast Fourier and inverse Fourier transformation.

(2) void gf1solitonp(dcomplex *bprofile,double pulse_FWHM,double windowsize): Generates a normalized fundamental soliton pulse profile.

(3) void propsolitonp(dcomplex *bprofile,double pulse_FWHM, double zstep /*Fraction of solition period */,double zlength,double windowsize): Propagates a normalized soliton pulse in a singlemode fiber by the symmetrized split step Fourier method. Zero attenuation is assumed.

(4) void nolmpropsolitonp(unsigned int axis,dcomplex *bprofile,double pulse_FWHM, double zstep /*Fraction of soliton period */,double zlength,double windowsize):): Propagates a normalized soliton pulse in a NOLM by the symmetrized split step Fourier method. Zero attenuation is assumed.

(5) dcomplex dispersion_operator(unsigned int dfreq,double To,double Ts,double Delta,double beta2,unsigned int size,double LD): Single soliton pulse propagation dispersion operator.

(6) dcomplex dispersion_operatorxy(unsigned int axis,unsigned int dfreq,double To,double Ts,double Delta,double beta2,unsigned int size,double LD): The dispersion operator for the propagation of two orthogonally polarized soliton pulses.

(7) void propsolitonps(dcomplex *bprofilex,dcomplex *bprofiley,double pulse_FWHM1,double pulse_FWHM2,double zstep /*Fraction of solition period */,double zlength,double windowsize): Co-propagates two orthogonally polarized normalized soliton pulses, in a singlemode fiber by the symmetrized split step Fourier method. Zero attenuation is assumed.

(8) void nolmpropsolitonps(dcomplex *bprofilex,dcomplex *bprofiley,double pulse_FWHM,double zstep, /*Fraction of soliton period */double zlength,double windowsize): Co-propagates two orthogonally polarized normalized soliton pulses, in a NOLM by the symmetrized split step Fourier method. Zero attenuation is assumed.

(9) void stgpsinolm(dcomplex *inputprofile,double pulse_FWHM,double zstep, /*Fraction of soliton period */double zlength,double windowsize): Implements the STG-PSI-NOLM gate.

(10) void stgnolm(dcomplex *inputprofile,double pulse_FWHM,double zstep, /*Fraction of soliton period */double zlength,double windowsize): Implements the STG-NOLM gate.

(11) void nolmswitch(double xcalpha,dcomplex *Ein1,dcomplex *Ein2,dcomplex *Eout1,dcomplex *Eout2): Implements the NOLM interferometric switch.

(12) void profiletshift(unsigned int type,double timeshift,double cwindow,dcomplex *tprofile): Applies a positive/negative time-shift to a given pulse profile.

4.2 MATHEMATICS MODULE

Both "bpm.cpp" and "bpma.cpp" rely on the mathematics module "admathf.cpp" for their implementation. The following functions makeup the mathematics module "admathf.cpp". All mathematics modules below were obtained from [4.1] except for modules (14) to (16), that were coded by myself using the 3-point and 5-point Lagrangian interpolation formulae, and the trapezoidal rule:

(1) double bessj0(double x): Ordinary Bessel function J_0.

(2) double bessy0(double x): Ordinary Bessel function Y_0.

(3) double sign(double x): Returns the sign of a variable x.

(4) double bessj1(double x): Ordinary Bessel function J_1.

(5) double bessy1(double x): Ordinary Bessel function Y_1.

(6) double bessj(int n, double x): Ordinary Bessel function J_N.

(7) double bessy(int n, double x): Ordinary Bessel function Y_N.

(8) double bessi0(double x): Modified Bessel function I_0.

(9) double bessk0(double x): Modified Bessel function K_0.

(10) double bessi1(double x): Modified Bessel function I_1.

(11) double bessk1(double x): Modified Bessel function K_1.

(12) double bessi(int n,double x): Modified Bessel function I_N.

(13) double bessk(int n, double x): Modified Bessel function K_N.

(14) void L3pDt(dcomplex *bprofile,unsigned int size,double pulse_window,double pulse_HW): Calculates the numerical first derivative of a boundary-zero padded array using 3-point Lagrangian interpolation formulae.

(15) void L5pDt(dcomplex *bprofile,unsigned int size,double pulse_window,double pulse_HW): Calculates the numerical first derivative of a boundary-zero padded array using 5-point Lagrangian interpolation formulae.

(16) void TrapIz(dcomplex *bprofile1,dcomplex *bprofile2,unsigned int size,double zstep_size): Calculate the numerical integration of an array using the trapezoidal rule.

(17) double sgn(double number): Determines the mathematical sign of argument *number*.

4.3 SINGLEMODE FIBER CHARACTERISTICS MODULES

The singlemode fiber characteristics modules are "wlength.cpp"/"wlengtha.cpp", "nlbfunc.cpp", "wgbfunc.cpp", and "nlbfunc.cpp".

The following functions makeup "wlength.cpp/wlengtha.cpp":

(1) double nsellmeier(double wavelength, double B[3], double wlength[3]): Refractive index approximation using Sellmeier's equation.

(2) double D1nsellmeier(double wavelength, double B[3], double wlength[3]): First derivative of the material refractive index with respect to the radian frequency approximation using Sellmeier's equation.

(3) double D2nsellmeier(double wavelength, double B[3], double wlength[3]): Second derivative of the refractive index with respect to the radian frequency approximation using Sellmeier's equation.

(4) double sgroupnrf(double wavelength, double B[3], double wlength[3]): Group index based on Sellmeier's equation. Using radian frequency. Intensity dependent.

(5) double sgroupnwl(double wavelength, double B[3], double wlength[3]): Group index based on Sellmeier's equation. Using wavelength. Intensity dependent.

(6) double sgvelocity(double wavelength, double B[3], double wlength[3]): Group velocity based on Sellmeier's equation.

(7) double sgvd(double wavelength, double B[3], double wlength[3]): Group velocity dispersion (GVD) or β_2 based on Sellmeier's equation.

(8) double SD(double wavelength, double B[3], double wlength[3]): Dispersion Parameter (D) based on Sellmeier's equation.

(9) double D1lbdnsellmeier(double wavelength, double B[3], double wlength[3]): First derivative of the material refractive index with respect to the wavelength approximation using Sellmeier's equation.

(10) double D2lbdnsellmeier(double wavelength, double B[3], double wlength[3]): Second derivative of the refractive index with respect to the wavelength approximation using Sellmeier's equation.

(11) double DELTA(double wavelength): Relative refractive index difference using Sellmeier's equation.

(12) double D1DELTA(double wavelength): First Derivative of DELTA, the relative refractive index difference, using Sellmeier's equation.

(13) void fiberDdata(void): Generates composite material dispersion (CMD), waveguide dispersion (WD), composite profile dispersion (CPD), dispersion cross-products (MR) and the total dispersion (D=CMD+WD+CPD+MR).

The following functions makeup “wgbfunc.cpp”:

(1) double V(double wavelength, double coreradius): Returns the normalized frequency V.

(2) double Wrn(double V): Single mode optical fiber step-index cladding eignvalue W approximation by Rudolph and Neumann [4.2].

(3) double W(double v): Cladding eigenvalue W based on approximate eigenvalue equation type I for HE_{11}or LP_{01} mode [4.3, 4.4].

(4) double Urn(double v): Core eigenvalue U based on Rudolph and Neumann approximation.

(5) double U(double v): Core eigenvalue U based on approximate eigenvalue equation type I for HE_{11}or LP_{01} mode

(6) double brn(double v): Normalized propagation constant b based on Rudolph and Neumann approximation.

(7) double b(double v): Normalized propagation constant based on approximate eigenvalue equation type I for HE_{11}or LP_{01} mode.

(8) double Beta_z(double wavelength, double b): Total mode propagation constant.

(9) double eri(double wavelength, double b): Effective refractive index.

(10) double cow(double coreradius): Cut off wavelength for single mode fibers for operation in the range $1.2 \times 10^{-6} - 2.0 \times 10^{-6}$ m for a specified core and cladding composition, and core radius.

(11) double kappa0(double w): Returns $pow(bessk0(w) / bessk1(w), 2.0)$.

(12) double PCFactor(double v): Power confinement factor (Γ) for singlemode fiber fundamental mode based on approximate eigenvalue equation type I for HE_{11}or LP_{01} mode.

(13) double VD2BV(double v): Returns $vd^2(bv) / dv^2$ for HE_{11}or LP_{01} mode.

(14) double DBV(double v): Returns $d(bv) / dv$ for HE_{11}or LP_{01} mode.

(15) double cmd(double wavelength): Composite material dispersion based on approximate eigenvalue equation type I for HE_{11}or LP_{01} mode.

(16) double wgd(double wavelength): Waveguide dispersion based on approximate eigenvalue equation type I for HE_{11}or LP_{01} mode.

(17) double cpd(double wavelength): Composite profile dispersion.

(18) double dcp(double wavelength): Dispersion cross products as given by J.M. Adams [4.5].

(19) double D(double wavelength): Effective total dispersion, D, coefficient as define by D. Gloge [4.6].

(20) double GVD(double wavelength): Effective group velocity dispersion or β_2. Includes composite material dispersion waveguide dispersion, composite profile dispersion and cross-product terms.

(21) double egi(double wavelength): Effective group index.

(22) double egv(double wavelength): Effective group velocity.

(23) double ebeta1(double wavelength): Effective β_1.

The following functions makeup "nlbfunc.cpp":

(1) double gamma(double wavelength): Nonlinear coefficient γ.

(2) double nsolitonp(double wavelength,double sorder,double pwidth): Returns *n*-soliton peak power.

(3) double p1espotsize(double u,double w): Returns Peterman I effective spot size.

The STG, SDG, STG-NOLM, and STG-PSI-NOLM are implemented through the program modules "wlengtha.cpp" and "bpma.cpp", which simulate propagation of two orthogonally polarized soliton pulses and/or the propagation of a single soliton pulse along a specified length of fiber.

CHAPTER V

RESULTS

Numerical solutions for the single soliton pulse propagation using (3.9) in Phase I and III, and the propagation of two orthogonally polarized soliton pulses using (3.15) and (3.16) in Phase II and III are presented in this chapter.

Phase I was primarily introductory and conducted to gain knowledge of the numerical techniques involved in the propagation of a single soliton pulse, in a singlemode fiber. The introductory part of Phase II was also used to gain knowledge of the numerical techniques involved in the propagation of two orthogonally polarized soliton pulses, in a strongly birefringent singlemode fiber. Results from the introductory numerical solutions matched work done by reputable researchers such as Agrawal [5.1], Islam [5.2], and Menyuk [5.3, 5.4]. On this basis the computer modules were considered fully tested and thereafter enhanced to include the new cross Raman effects, due to CS and CFS, in Phase II and III.

The pulse characteristics used for soliton pulse propagation in a singlemode are as follows:

(1) Soliton amplitude (N): This is defined by

$$N = \sqrt{L_D / L_{NL}} \ . \tag{5.1}$$

Here the dispersion length $L_D = T_0^2 / |\beta_2|$ and the nonlinear length $L_{NL} = 1/\gamma P_0$. The peak pulse power is denoted by P_0, γ is the nonlinear coefficient, and T_0 is the half-width at the e^{-1} intensity. Soliton amplitudes of 1.0 and 2.0 correspond to peak powers of 2.0703 kW and 8.2811 kW respectively for a soliton pulse width (FWHM) of 50 fs at 1.55 μm, with β_2 = -10.1596 ps^2/km (SMF01).

(2) Pulse width (T_{FWHM}): The pulse width T_{FWHM} was set at 50 fs. The pulse width is based on the full-width at half power maximum, which is related to T_0 by

$$T_{FWHM} = 2\ln\left(1+\sqrt{2}\right)T_0 . \tag{5.2}$$

This gives a value of T_0 of 28.3684 fs. The time delay of the soliton pulses is measured in units of the half-width at the e^{-1} intensity T_0.

(3) Pulse carrier wavelength (λ_1): The carrier wavelength was set at 1.55 μm.

(4) Soliton period (z_0): The soliton period is defined as

$$z_0 = \frac{\pi}{2} L_D . \tag{5.3}$$

With T_{FWHM} = 50 fs and λ_1=1.55 μm, and β_2 = -10.1596 ps^2/km (SMF01) then $z_0 = 0.12439$ m. The propagation distance of the soliton pulses is measured in units of the soliton period.

(5) Walk-off length (l_{wo}): The walk-off length is the distance over which two orthogonal pulses start together, and then separate as the result of birefringence $(B = \Delta n)$. Walk-off length is defined as

$$l_{wo} = \left(cT_{FWHM}\right)/\Delta n. \qquad (5.4)$$

The numerical simulation parameters used for soliton pulse propagation in a singlemode fiber are:

(1) Sample points (N_0): The number of discrete sample points, N_0, used was set at 16,384 in the numerical simulations for the time and frequency domains.

(2) Computation window (T_W): The computation window T_W is the width of the numerical computation widow within which the pulse is capable of transiting. The computation window was set at either 1000 fs or 2000 fs depending upon the truncation and round-off errors encountered, and the amount of delay the pulse experiences over the computation window. The Phase I simulation computation window was set at T_W=2000 fs. For Phase II, T_W was either 1000 fs or 2000 fs. For Phase III, T_W was 2000 fs.

(3) Center frequency (f_0): Carrier center frequency $f = f_0$. The normalized frequency=$(f\text{-}f_0)T_0$=$(k\text{-}k_0)T_0/T_W$. Here $(k\text{-}k_0)/T_0$ is the discrete frequency.

(4) Normalized soliton power ($|U|^2$).

(5) Normalized power spectral density, PSD, ($|U(f,z/z_0)/U(f_0,z/z_0=0)|^2$).

5.1 PHASE I

This section gives the results obtained from propagating a single 50 fs (FWHM) soliton pulse at 1.55 μm in the singlemode fiber (SMF01), over 0.5 m $(z/z_0 = 4.0195)$. The time and frequency domain profiles of the pulse at $z/z_0 = 0.000$ (solid line) and $z/z_0 = 4.0195$ (dotted line) are given. All Phase I simulations include SOD and SPM.

For a soliton amplitude $N = 1.0$ and only the effects due to SOD and SPM included in the propagation, it can be seen from the time and frequency domain profiles in Figure 5.1.1 and Figure 5.1.2 respectively, that the pulse retains its shape in both domains.

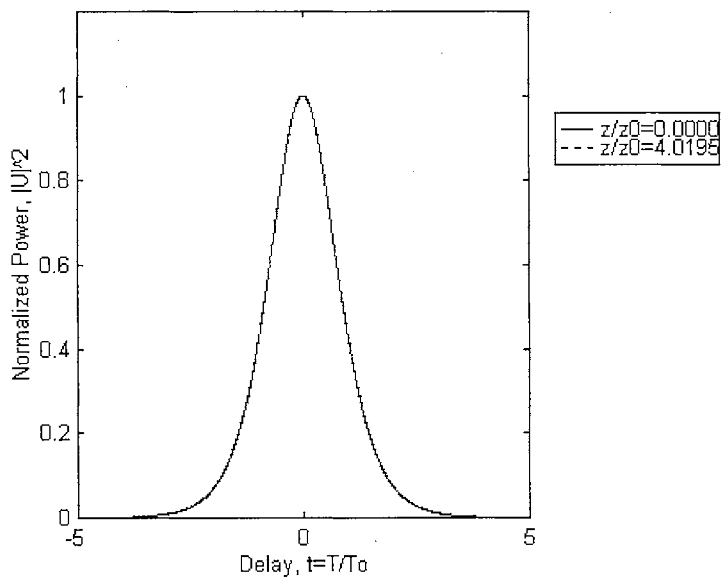


Figure 5.1.1 Time domain 50 fs soliton pulse at $N = 1.0$ (SOD+SPM).

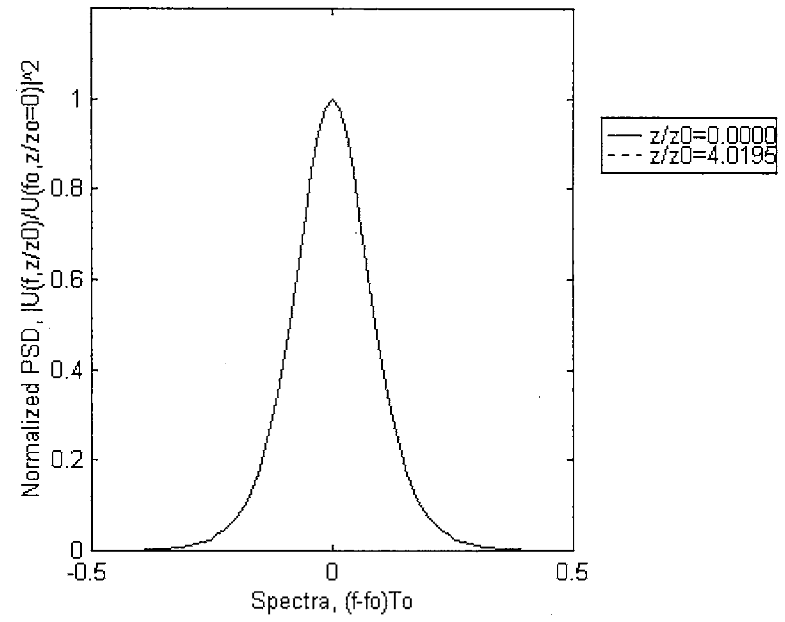


Figure 5.1.2 Frequency domain 50 fs soliton pulse at $N = 1.0$ (SOD+SPM).

In this particular case, without the inclusion of higher order linear and nonlinear effects, the fundamental soliton propagates without distortion in the time and frequency domain. However, this case does not truly represent the propagation of soliton pulses less than 100 fs. For soliton pulses less than 100 fs, higher nonlinear and dispersion effects need to be included, namely TOD, SS, and SFS.

For a soliton amplitude $N = 1.0$ with SOD, SPM and SFS effects included in the propagation, it can be seen from the time domain profile in Figure 5.2.1 that at $z/z_0 = 4.0195$ the soliton pulse is delayed by $1.20T_0$ due to SFS.

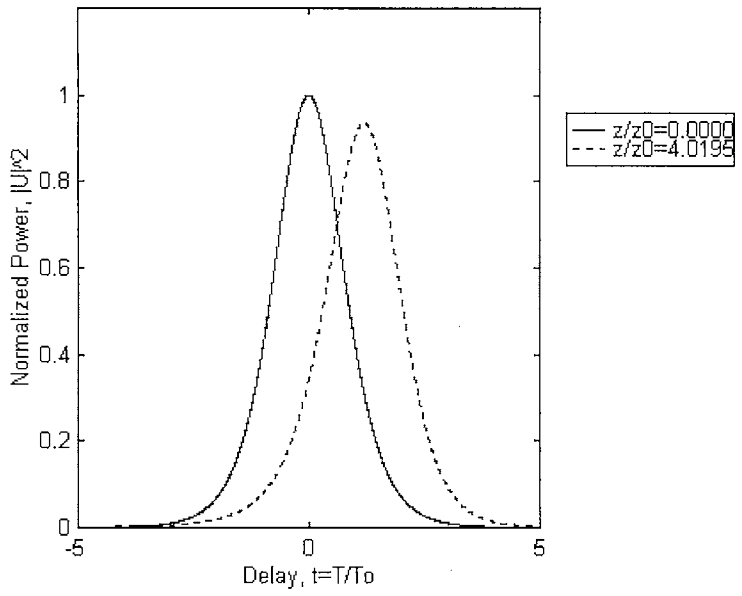


Figure 5.2.1 Time domain 50 fs soliton pulse at $N = 1.0$ (+SFS).

The frequency domain pulse profile given by Figure 5.2.2, shows that the pulse spectral shape is modified. The peak amplitude frequency has been shifted towards the low frequency spectrum by $0.0567T_0/T_W$.

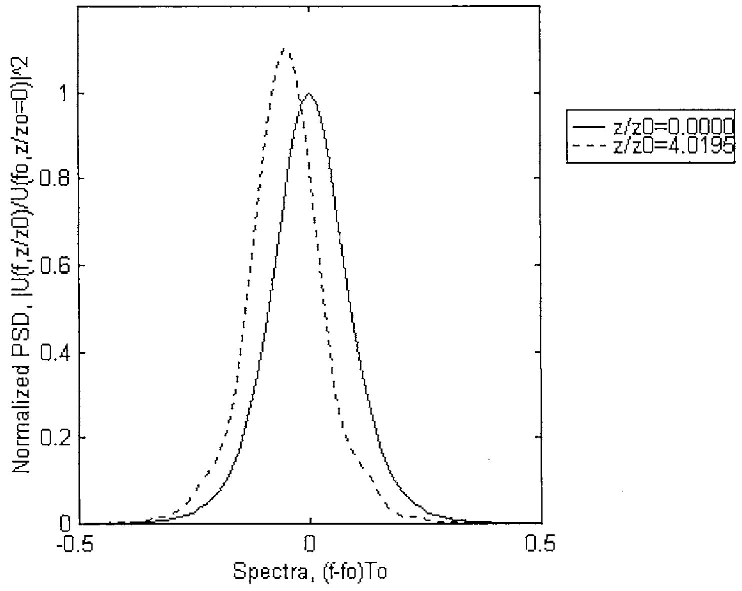


Figure 5.2.2 Frequency domain 50 fs soliton pulse at $N = 1.0$ (+SFS).

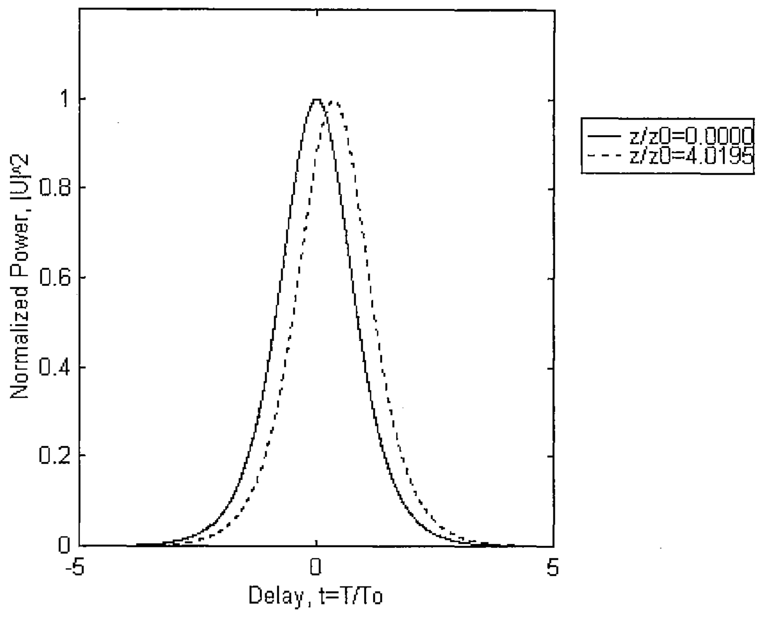


Figure 5.3.1 Time domain 50 fs soliton pulse at $N = 1.0$ (+SS).

For a soliton amplitude $N = 1.0$ with SOD, SPM and SS effects included in the propagation, it can be seen from the time domain profile in Figure 5.3.1 that at $z/z_0 = 4.0195$ the soliton pulse is delayed by $0.36T_0$ due to SS. The temporal delay due to SOD, SPM and SS is less that due to SOD, SPM, and SFS of $1.20T_0$. The frequency domain pulse profile given by Figure 5.3.2, shows that the pulse spectral shape is slightly modified. The peak amplitude frequency remains at zero.

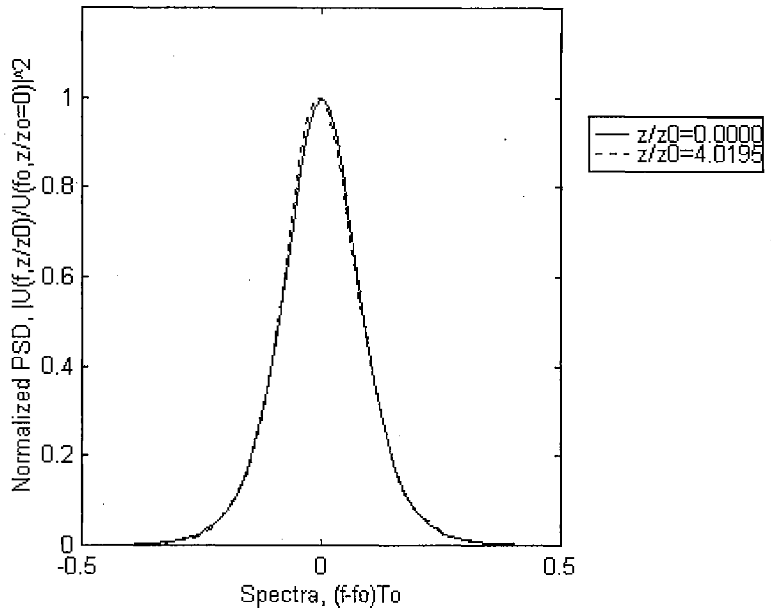


Figure 5.3.2 Frequency domain 50 fs soliton pulse at $N = 1.0$ (+SS).

For a soliton amplitude $N = 1.0$ with SOD, SPM, SS and SFS effects included in the propagation, it can be seen from the time domain profile in Figure 5.4.1 that at $z/z_0 = 4.0195$ the soliton pulse is delayed by $1.53T_0$ due to SS and SFS. The temporal delay due to SOD, SPM, SS and SFS is greater that due to SOD, SPM, and SS or SFS of $0.36T_0$ and $1.20T_0$ respectively. The frequency domain pulse profile given by Figure 5.4.2, shows that the pulse spectral shape is modified. The peak amplitude frequency has been shifted towards the low frequency spectrum by $0.0567T_0/T_W$., which is the same as that for the case with SOD, SPM and SFS.

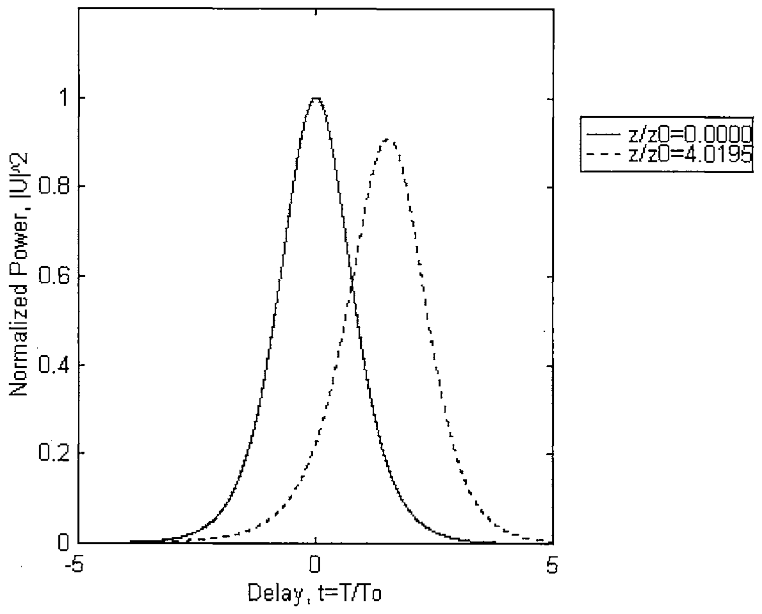


Figure 5.4.1 Time domain 50 fs soliton pulse at $N = 1.0$ (+SS+SFS).

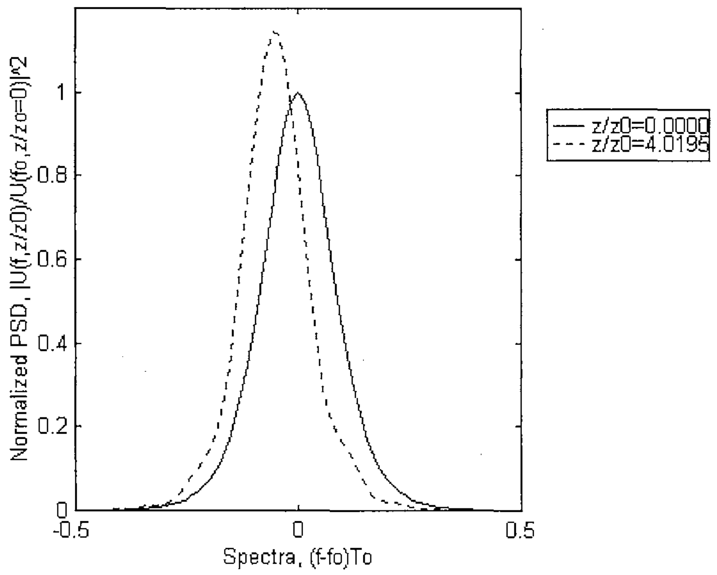


Figure 5.4.2 Frequency domain 50 fs soliton pulse at $N = 1.0$ (+SS+SFS).

For a soliton amplitude $N = 1.0$ with SOD, SPM, and TOD effects included in the propagation, it can be seen from the time domain profile in Figure 5.5.1 that at $z/z_0 = 4.0195$ the soliton pulse is slightly advanced by $0.0215T_0$ due to TOD. This implies that TOD speeds up the soliton pulse slightly as opposed to the effects of SS and SFS that delay the soliton pulse. The frequency domain pulse profile given by Figure 5.5.2, shows that the pulse spectral shape is not modified at all. Thus, the peak amplitude frequency remains at zero.

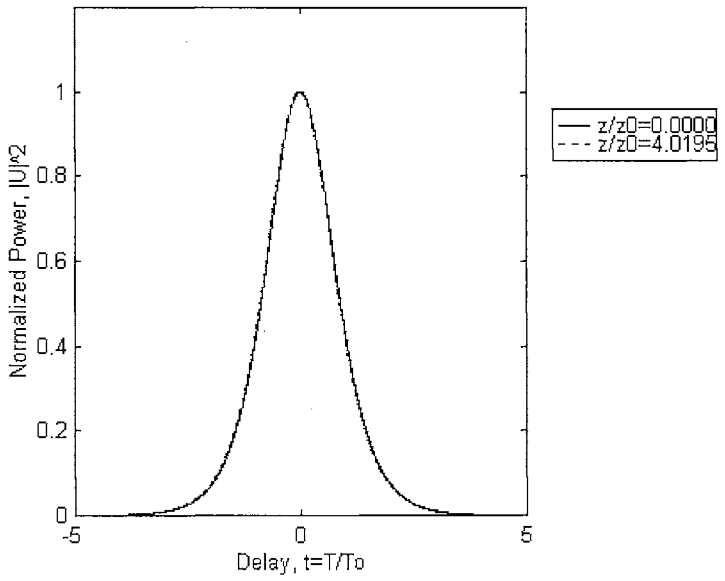


Figure 5.5.1 Time domain 50 fs soliton pulse at $N = 1.0$ (+TOD).

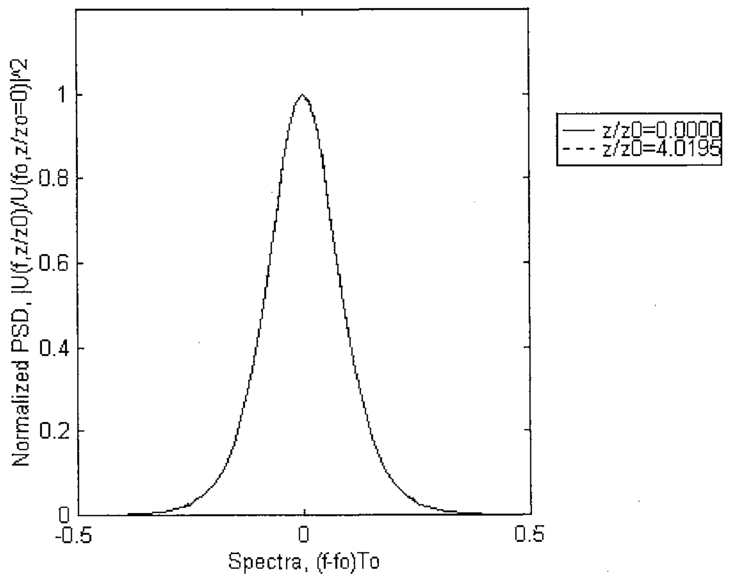


Figure 5.5.2 Frequency domain 50 fs soliton pulse at $N = 1.0$ (+TOD).

For a soliton amplitude $N = 1.0$ with SOD, SPM, TOD and SFS effects included in the propagation, it can be seen from the time domain profile in Figure 5.6.1 that at $z/z_0 = 4.0195$ the soliton pulse is delayed by $1.1835T_0$ due to TOD and SFS. TOD has speeded up the pulse by about $0.02T_0$ from the case with SOD, SPM, and SFS, which had a delay of $1.2T_0$.

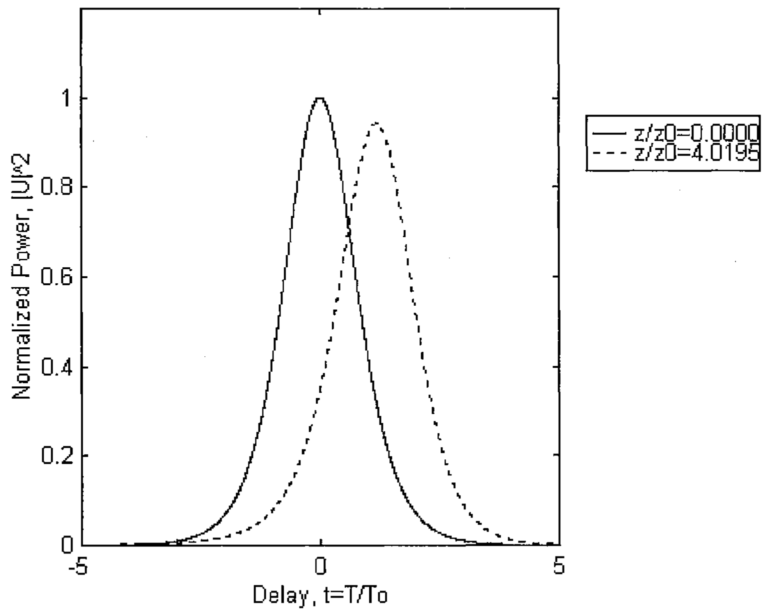


Figure 5.6.1 Time domain 50 fs soliton pulse at $N = 1.0$ (+TOD+SFS).

The frequency domain pulse profile given by Figure 5.6.2, shows that the pulse spectral shape is modified. The peak amplitude frequency has been shifted towards the low frequency spectrum by $0.0567T_0/T_W$, which is the same as for the cases with (a) SOD, SPM, and SFS and (b) SOD, SPM, SS and SFS.

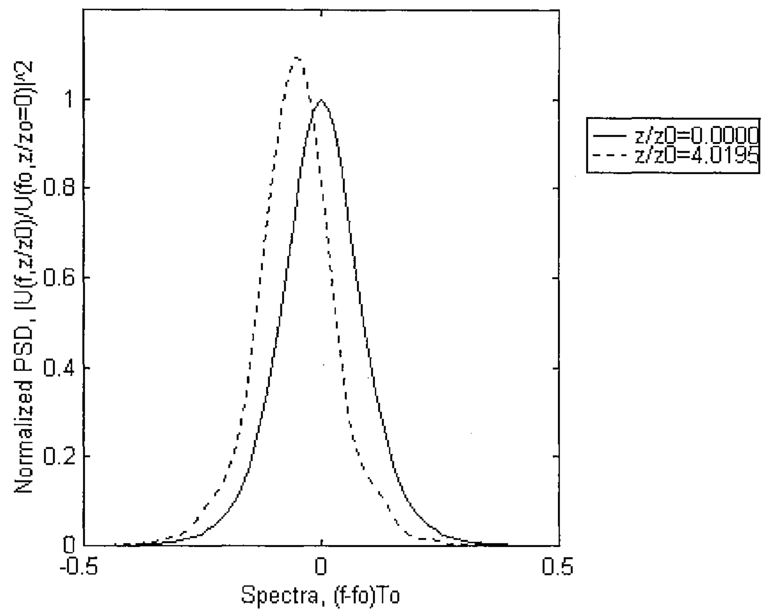


Figure 5.6.2 Frequency domain 50 fs soliton pulse at $N = 1.0$ (+TOD+SFS).

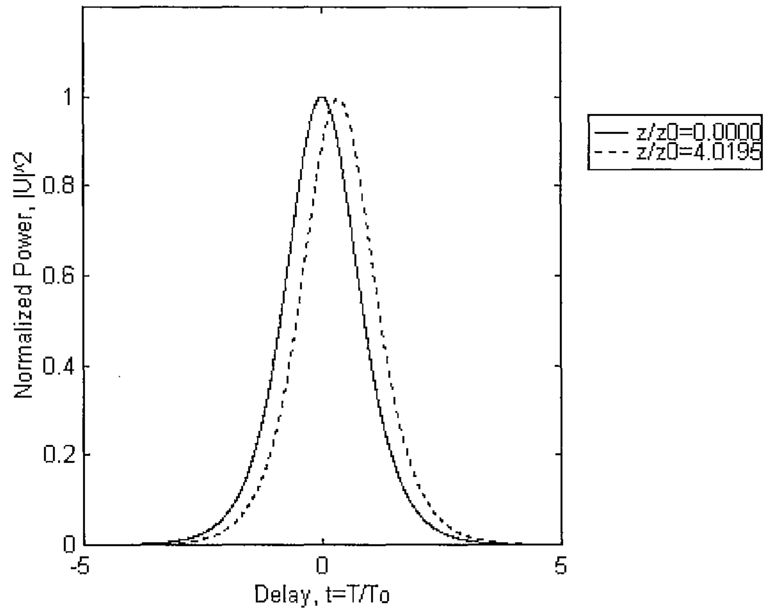


Figure 5.7.1 Time domain 50 fs soliton pulse at $N = 1.0$ (+TOD+SS).

For a soliton amplitude $N = 1.0$ with SOD, SPM, TOD and SS effects included in the propagation, it can be seen from the time domain profile in Figure 5.7.1 that at $z/z_0 = 4.0195$ the soliton pulse is delayed by $0.34T_0$ due to TOD and SS. Here again

TOD has speeded up the pulse by about $0.02T_0$ from the case with SOD, SPM, and SS, which had a delay of $0.36T_0$. The frequency domain pulse profile given by Figure 5.7.2, shows that the pulse spectral shape is slightly modified. The peak amplitude frequency remains at zero, which is the same case with SOD, SPM, and SS.

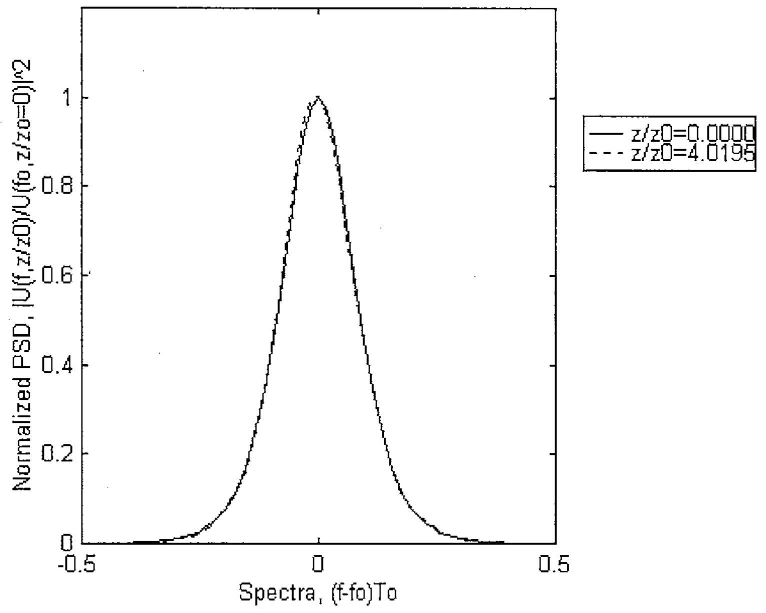


Figure 5.7.2 Frequency domain 50 fs soliton pulse at $N = 1.0$ (+TOD+SS).

For a soliton amplitude $N = 1.0$ with SOD, SPM, TOD, SS and SFS effects included in the propagation, it can be seen from the time domain profile in Figure 5.8.1 that at $z/z_0 = 4.0195$ the soliton pulse is delayed by $1.5150T_0$ due to TOD, SS and SFS. . TOD here again has speeded up the pulse by about $0.02T_0$ from the case with SOD, SPM, SS and SFS, which had a delay of $1.53T_0$.

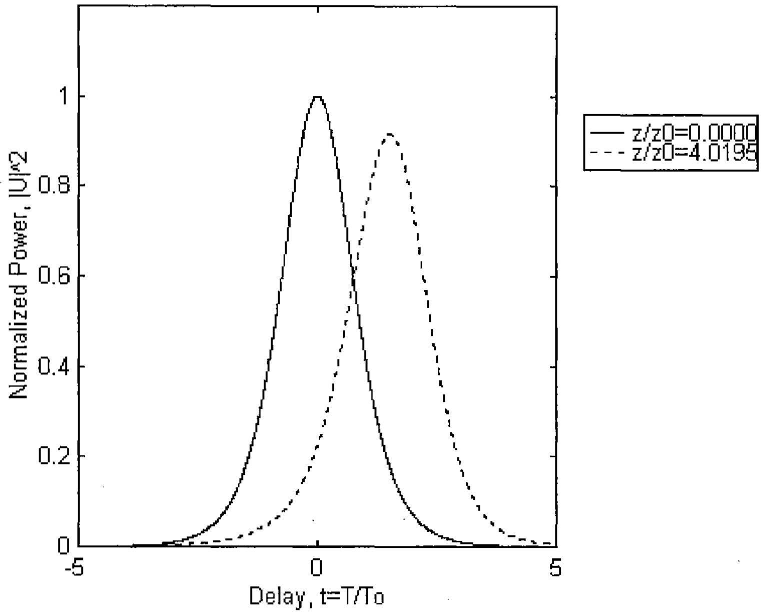


Figure 5.8.1 Time domain 50 fs soliton pulse at $N = 1.0$ (+TOD+SS+SFS).

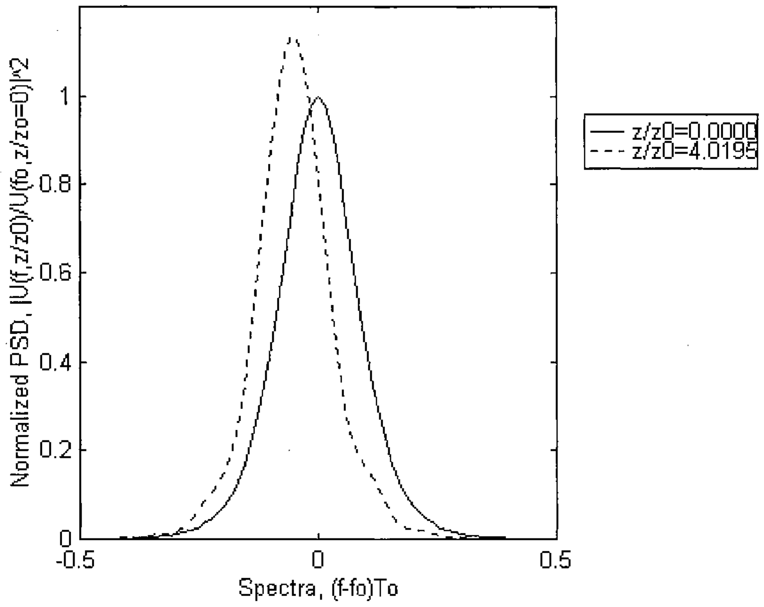


Figure 5.8.2 Frequency domain 50 fs soliton pulse at N=1.0 (+TOD+SS+SFS).

The frequency domain pulse profile given by Figure 5.8.2 shows that the pulse spectral shape is modified. The peak amplitude frequency has been shifted towards the low frequency spectrum by $0.0567T_0/T_W$, which is the same as for the cases with (a) SOD, SPM, SFS, (b) SOD, SPM, SS and SFS, and (c) SOD, SPM, TOD and SFS. In all these cases SFS is responsible for the peak amplitude frequency towards the low frequency spectrum by $0.0567T_0/T_W$.

For a soliton amplitude N=2.0 with SOD, SPM, TOD, SS and SFS effects included in the propagation, it can be seen from the time and frequency domain profiles in Figures 5.9.1 to 5.13.2 that at $z/z_0 = 4.0195$ the second order soliton pulse has split into two solitons. One of the solitons has a large amplitude with a broad low frequency spectrum (red shifted), and the other has a small amplitude and narrow high frequency spectrum (blue shifted). The large amplitude soliton grows at the expense of the small amplitude soliton, due to the continuous transfer of energy from the blue shifted components to the red shifted components, through Raman gain. The blue shifted soliton diminishes in amplitude and is advanced in time slightly, whereas the red shifted soliton is amplified and delayed in time significantly.

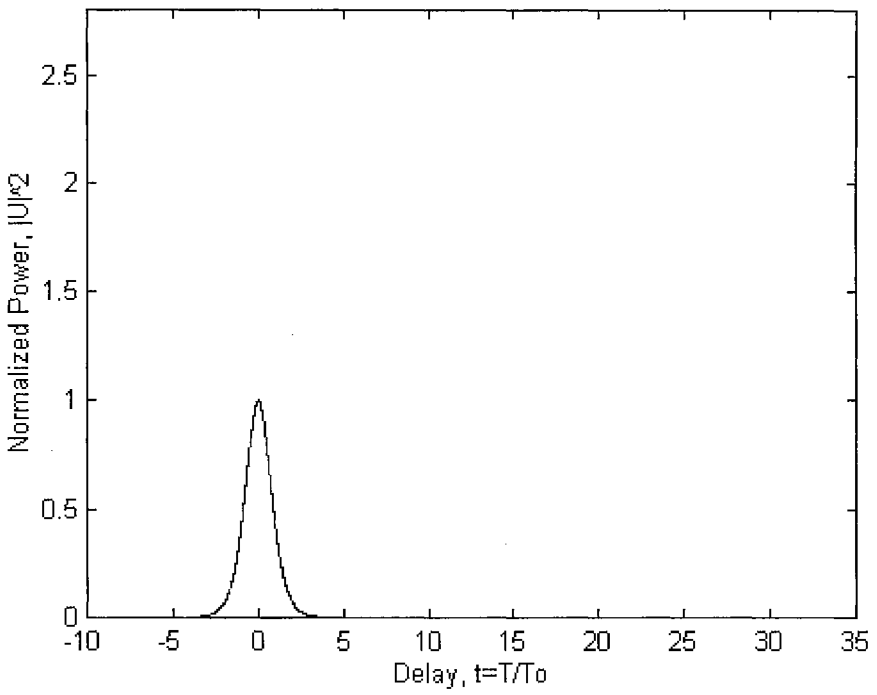


Figure 5.9.1 Time domain 50 fs soliton pulse at $N = 2.0$ for z/z_0=0.0000 (+TOD+SS+SFS).

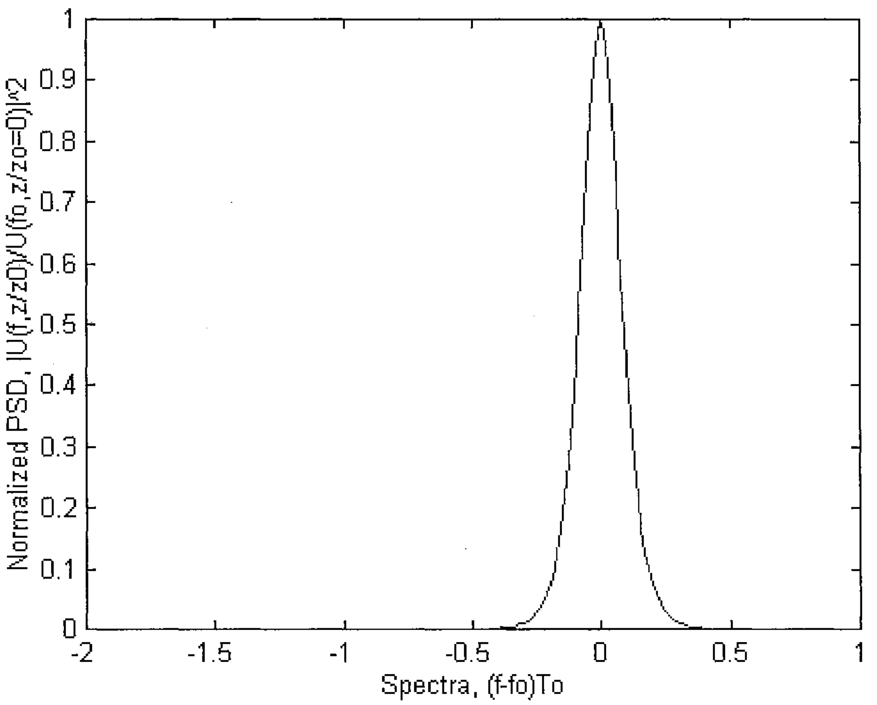


Figure 5.9.2 Frequency domain 50 fs soliton pulse at $N = 2.0$ for z/z_0=0.0000 (+TOD+SS+SFS).

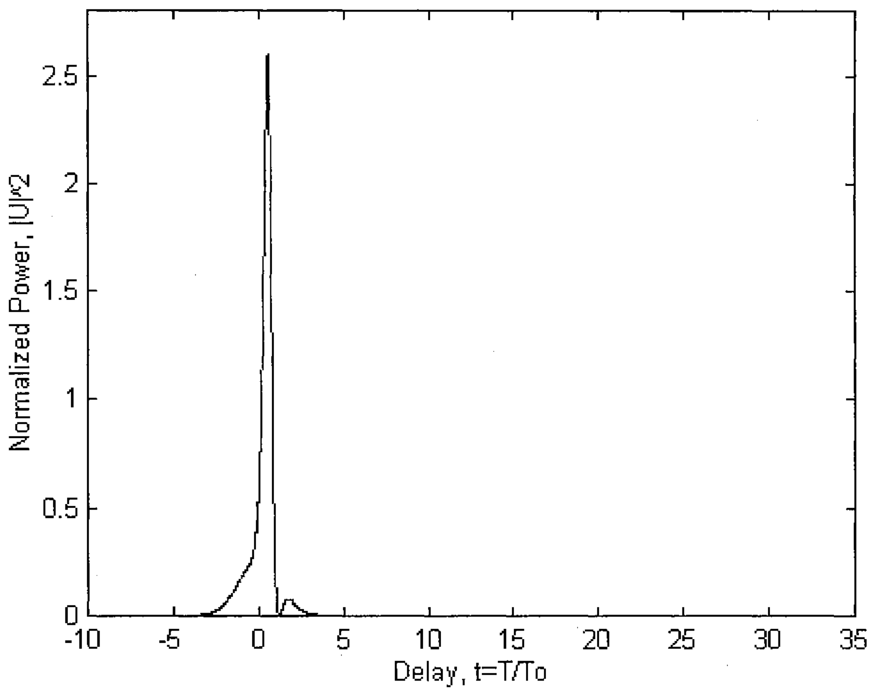


Figure 5.10.1 Time domain 50 fs soliton pulse at $N = 2.0$ for z/z_0=0.40195 (+TOD+SS+SFS).

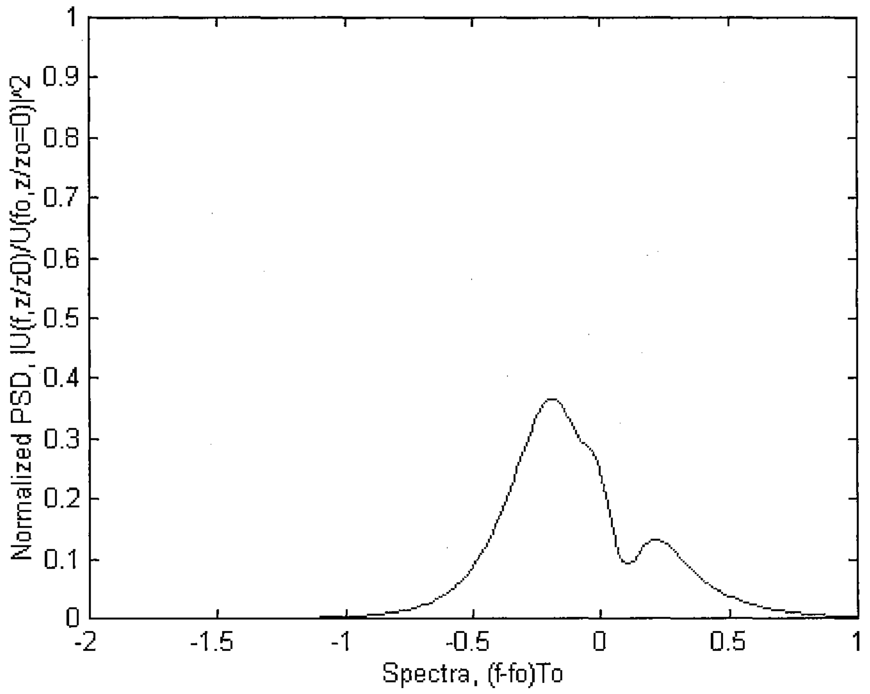


Figure 5.10.2 Frequency domain 50 fs soliton pulse at $N = 2.0$ for z/z_0=0.40195 (+TOD+SS+SFS).

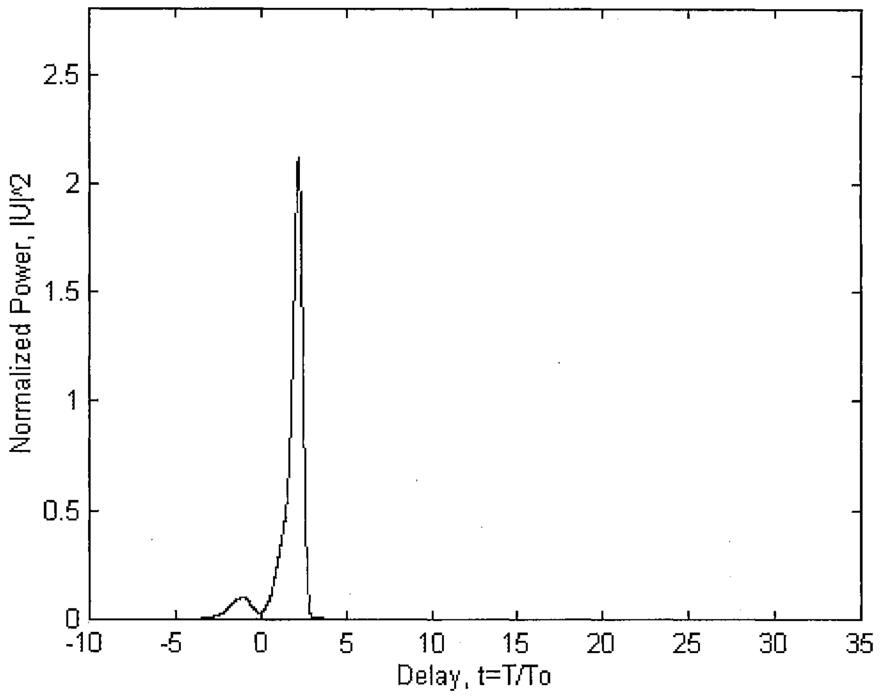


Figure 5.11.1 Time domain 50 fs soliton pulse at $N = 2.0$ for z/z_0=0.80390 (+TOD+SS+SFS).

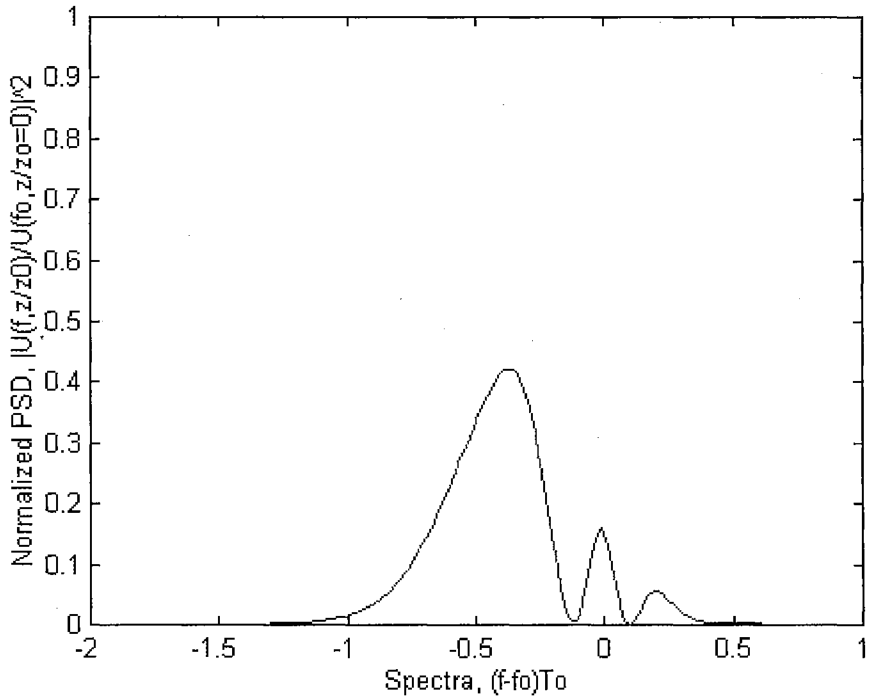


Figure 5.11.2 Frequency domain 50 fs soliton pulse at $N = 2.0$ for z/z_0=0.80390 (+TOD+SS+SFS).

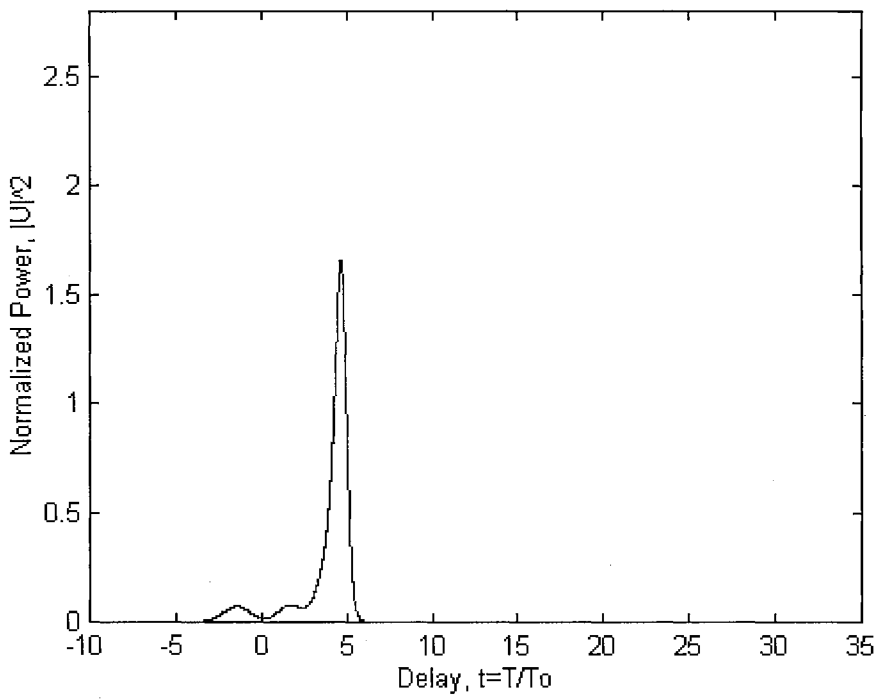


Figure 5.12.1 Time domain 50 fs soliton pulse at $N = 2.0$ for z/z_0=1.20585 (+TOD+SS+SFS).

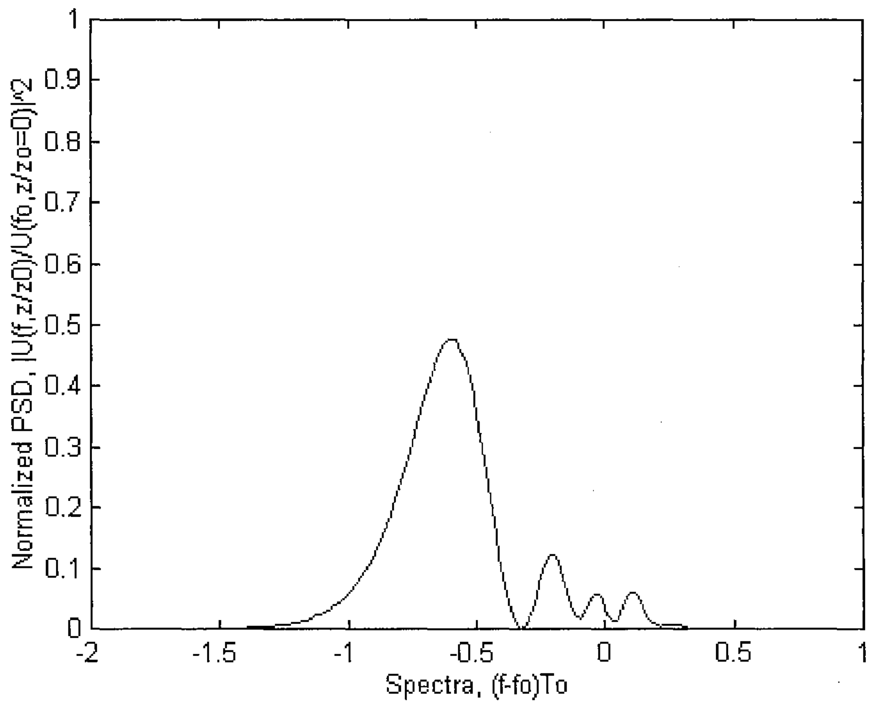


Figure 5.12.2 Frequency domain 50 fs soliton pulse at $N = 2.0$ for z/z_0=1.20585 (+TOD+SS+SFS).

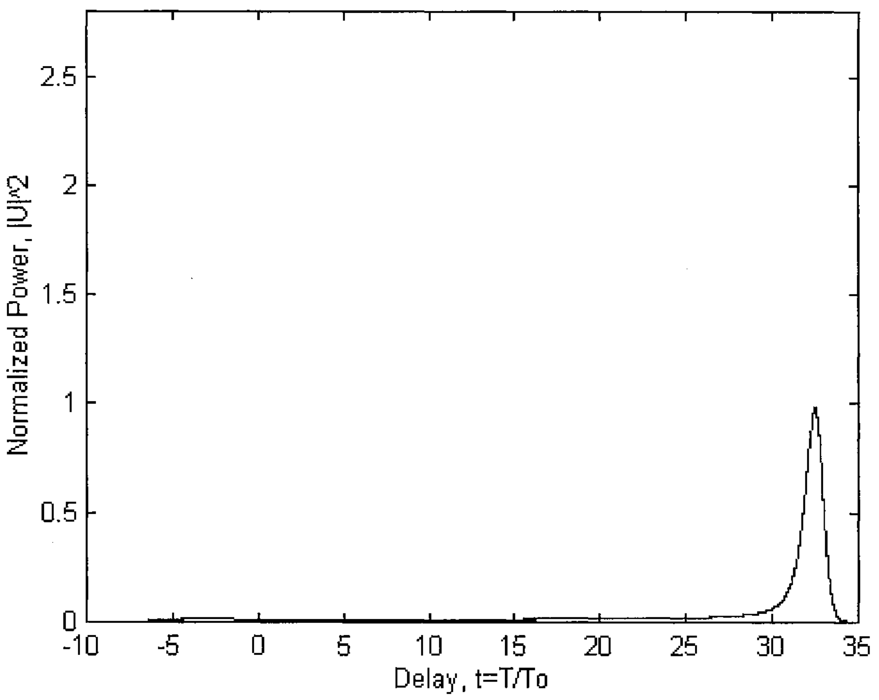


Figure 5.13.1 Time domain 50 fs soliton pulse at $N = 2.0$ for z/z_0=4.01950 (+TOD+SS+SFS).

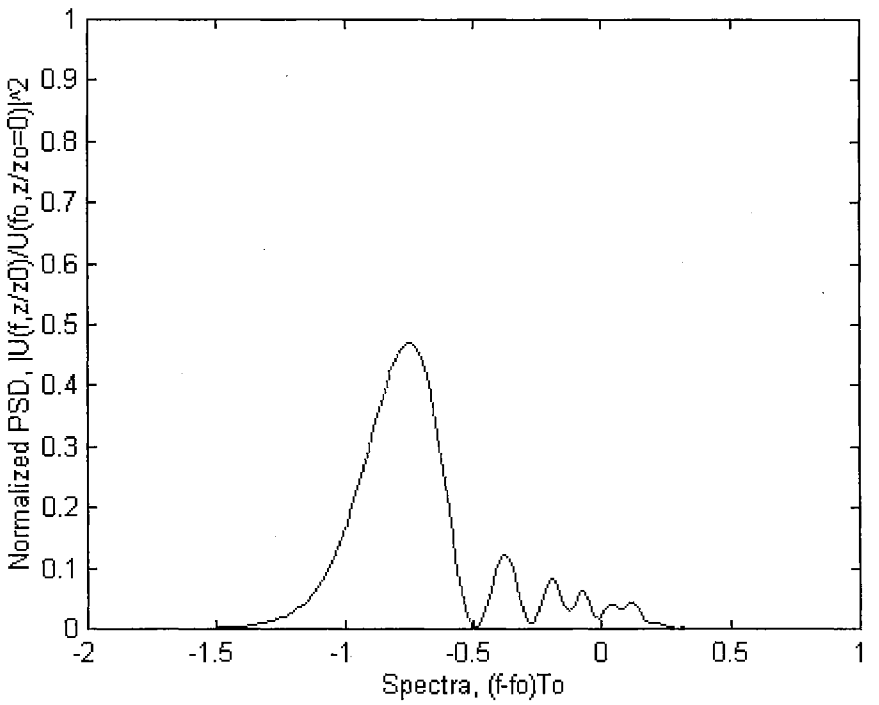


Figure 5.13.2 Frequency domain 50 fs soliton pulse at $N = 2.0$ for z/z_0=4.01950 (+TOD+SS+SFS).

5.2 PHASE II

This section gives the results obtained from propagating two orthogonally polarized 50 fs (FWHM) soliton pulses at 1.55 μm in a strongly birefringent singlemode fiber (SMF01), over 0.7464 m (z/z_0=6.0000) with a total normalized soliton amplitude of $N_T = 1.24$. The pulse widths of both orthogonal pulses were also varied from 30 fs to 100 fs at a constant birefringence of 1.1103×10^{-4}. The 0.7464 m length of fiber, plus a polarizer at the end of the fiber represents the inverter STG/SDG. All the STG/SDG time and frequency domain profiles of the *x*-polarization pulse (solid line) and *y*-polarization pulse (dashed line), at z/z_0=0.0000 and z/z_0=6.0000 are given in Appendix B. All Phase II simulations include SOD, SPM and CPM.

Phase II simulations involved various combinations of the higher order linear and nonlinear effects, namely TOD, SS, SFS, CS, and CFS. The effects are labeled for brevity as follows. The standard effect sequence SOD, SPM, CPM, TOD, SS, SFS, CS, and CFS is represented by SOD-CFS. The plus operator (+) effects inclusion of a particular effect in the sequence. Thus SOD-SFS+CFS represents the effect sequence SOD, SPM, CPM, SS, SFS and CFS, which excludes the effect of CS.

Due to the numerous time and frequency domain pulse profiles from the STG/SDG results, it was necessary to summarize the results. Monitoring the peak pulse amplitude delay, for the two orthogonal axes, with respect to propagation distance effected this.

The operation of the STG/SDG depends upon a total soliton amplitude threshold, N_{th}, and CPM. The total soliton amplitude threshold depends upon the half width at e^{-1}

intensity (T_0), birefringence (B), and the magnitude of the second order dispersion parameter $k_1^{''}$ as follows [5.5-5.7]

$$N_{th} = \sqrt{\frac{1+3\Delta^2}{1+\alpha}};\ \Delta = \frac{\left(k_{x1}^{'} - k_{y1}^{'}\right)T_0}{2\left|k_1^{''}\right|} = \frac{(B/c)T_0}{2\left|k_1^{''}\right|};\ \alpha = 2/3 \qquad (5.5)$$

A reduction or increase in the birefringence or pulse width results in the reduction or increase of the total soliton amplitude threshold respectively. An increase or reduction in the magnitude of the second order dispersion parameter $k_1^{''}$ results in the reduction or increase of the total soliton amplitude threshold respectively. It is important to note that the magnitude of the birefringence needs to be such that $\Delta R \succ\succ 1$, in order for the exponential varying terms to be neglected in equation (3.14). Thus a birefringence of $B = 1.1103 \times 10^{-4}$ was selected to give $\Delta = 0.5$ for a pulse width (FWHM) of 50 fs at 1.55 μm with $k_1^{''} = -10.1596\ \mathrm{ps}^2/\mathrm{km}$ (SMF01). This meets the condition of $\Delta R \succ\succ 1$.

For Phase II simulations $\Delta = 0.5$, which gives a total soliton amplitude threshold of $N_{th} = 1.0247$. For the STG, the polarization angle determining the pulse strengths in the two orthogonal axes, θ, is equal to 45° for the control pulse on either the x or y-axis. For the SDG the angle θ is 30° for the control pulse on the x-axis and 60° for the control pulse on the y-axis.

For $\Delta = 0.5$ and $\theta = 45^\circ$ (STG with the control on either the fast or slow axis), if the total soliton amplitude N_T is less than $N_{th} = 1.0247$, the two orthogonally polarized pulses spread because of dispersion and they separate due to birefringence. Above threshold, the effect of nonlinearly constrains both the spreading due to dispersion and the splitting due to birefringence, resulting in the trapping of the soliton pulses with equal and opposite frequency shifts. For $\Delta = 0.5$ and $\theta = 30^\circ$ (SDG with the control on slow axis) with the total soliton amplitude N_T =1.1, the original peak of the y-axis pulse (on fast axis) is not captured by the x-axis pulse (on slow axis); it advances steadily in time. At the same time, a new peak is created from the background of the y-axis pulse by the interaction with the x-axis pulse. This new peak is delayed in time and moves together with the x-axis pulse. The y-axis pulse consists of two parts; a disperse wave component which advances in time, spreading and diminishing in amplitude, and a portion which contributes to the soliton. The designation of an exact threshold is arbitrary in SDGs, as no sharp transition in the pulse behaviour is observed. Below the soliton amplitude threshold, however, no soliton is produced. These results were obtained by Menyuk [5.4] with SOD, SPM and CPM effects only for pulses above 500 fs. In this study, effects due to TOD, SS, SFS, CS, and CFS have been included, due to the use of sub-100 fs soliton pulses.

The following subsections give the results obtained for the STG and SDG. The pulse peak amplitude delay against propagation distance, of the two orthogonally polarized pulses, are compared for various combinations of higher order linear and nonlinear effects of interest.

5.2.1 Soliton Trapping Gate (STG)

For the STG, the two orthogonally polarized soliton pulses are of equal amplitude and therefore the angle $\theta = 45^\circ$. The time delay of the two orthogonally polarized soliton pulses (co-propagating) against the propagation distance, for the effect sequence SOD-

CPM, is compared with those of effect sequences: (a) SOD-SFS, (b) SOD-CFS, (c) SOD-CS, and (d) SOD-SFS+CFS in Figures 5.14.1 and 5.14.2.

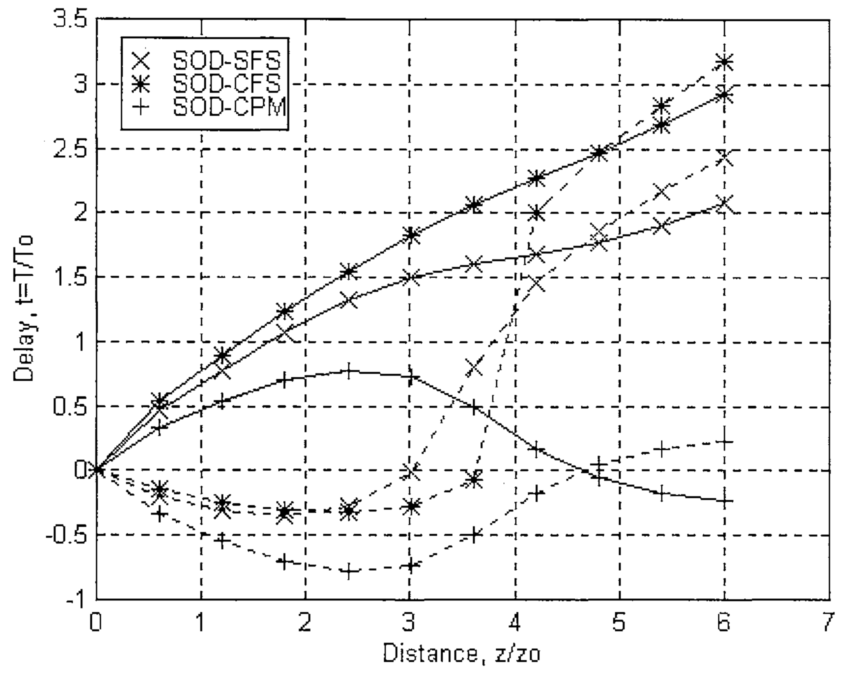


Figure 5.14.1 STG control pulse arrival delay time versus distance for SOD-SFS, and SOD-CFS. Plots with solid line represent the x-polarization control/signal pulse (on slow axis) and those with a dashed line represent the y-polarization signal/control pulse (on fast axis). Control and signal pulse widths = 50 fs, N_T = 1.24 and $\theta = 45^\circ$.

It can be observed from Figure 5.14.1 that the combined effect of TOD, SS and SFS delays the pulse on the fast and slow axis significantly by about $2.35T_0$. The pulses are further delayed by the inclusion of CS and CFS by about $0.82T_0$. Comparing Figure 5.14.2 to Figure 5.14.1 yields that the pulse delay, on both the fast and slow axis, due to CFS ($\approx 0.62T_0$) is larger than that due to CS ($\approx 0.\ 17T_0$) in the STG.

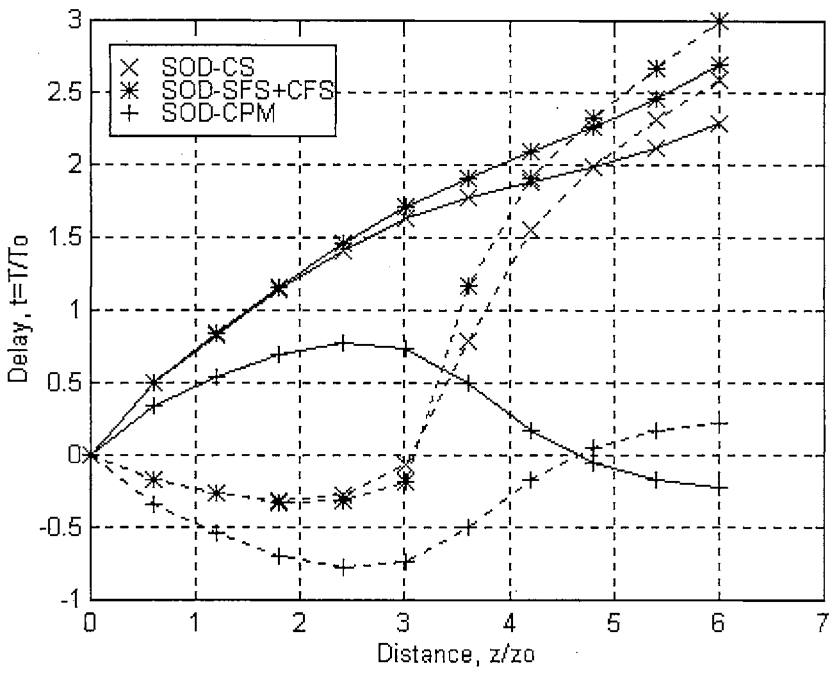


Figure 5.14.2 STG control pulse arrival delay time versus distance for SOD-CS, and SOD-SFS+CFS. Plots with solid line represent the x-polarization control/signal pulse (on slow axis) and those with a dashed line represent the y-polarization signal/control pulse (on fast axis). Control and signal pulse widths = 50 fs, $N_T = 1.24$ and $\theta = 45^\circ$.

In order to study how the switching efficiency of the STG is affected by the introduction of the effects due to TOD, SS, SFS, CS and CFS the following analysis was carried out. A single soliton pulse was propagated over the distance of z/z_0=6.0000 under effects due to (a) SOD-CPM and then finally under effects due to (b) SOD-CFS. The control pulse and the signal pulse were then propagated over the same length of fiber under the effects due to SOD-CFS. The single pulse propagation cases are denoted by (S), and the co-propagation case by (C) in Figures 5.15.1 and 5.15.2. Figure 5.15.1 gives the results for the control pulse on the x-polarization axis (slow axis), and Figure 5.15.2 gives the results for the control pulse on the y-polarization axis (fast axis). The efficiency of the STG, under SOD-CFS effects for the control pulse on the x and y-polarization axis, using (2.84) is 0.84 $\left(\text{maximum clock time window} = T_{mctw} = 0.84 \times 2 \times 4T_0 = 6.72T_0\right)$ and 1.00 ($T_{mctw} = 1.66 \times 2 \times 4T_0 = 13.28T_0$) respectively. This implies that the STG, under the influence of SOD-CFS effects, will switch more efficiently if the control pulse is on *y*-axis (fast axis). The efficiency of the STG under SOD-CPM effects, for the control pulse on the *x* and *y*-polarization axis is 1.00 (*x*-polarization $T_{mctw} = 9.84T_0$ and *y*-polarization $T_{mctw} = 10.24T_0$). On the fast axis (*y*-axis), the effects due to SOD-CFS increase T_{mctw}, under SOD-CPM effects, by 30%. However, on the slow axis (*x*-axis), the effects due to SOD-CFS reduce T_{mctw}, under SOD-CPM effects, by 32%.

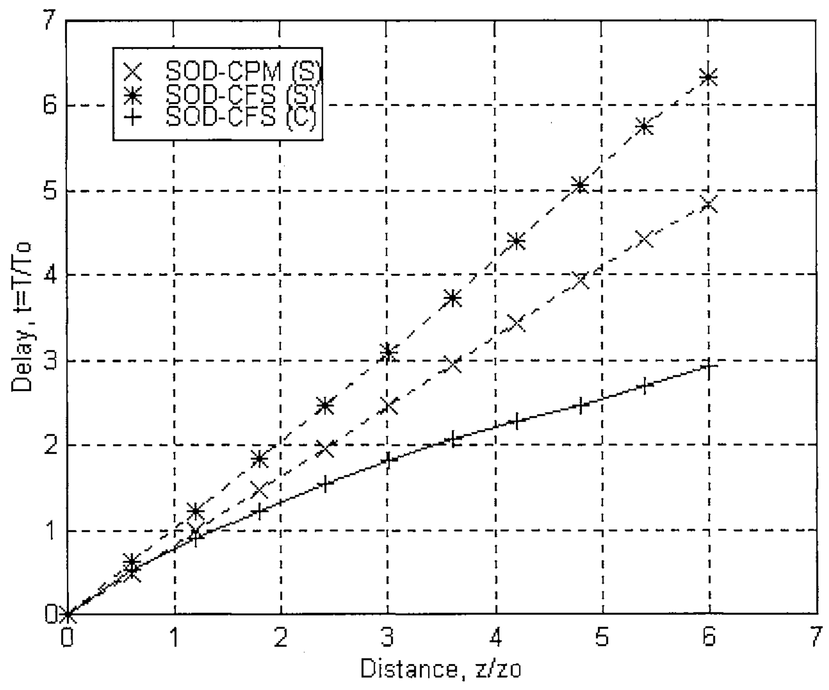


Figure 5.15.1 STG *x*-axis control pulse arrival delay time versus distance. Plot with solid line represents the *x*-polarization control pulse co-propagating with the signal pulse, and those with a dashed line represent the *x*-polarization control pulse propagating alone. Control and signal pulse widths = 50 fs, $N_T = 1.24$ and $\theta = 45^{\circ}$.

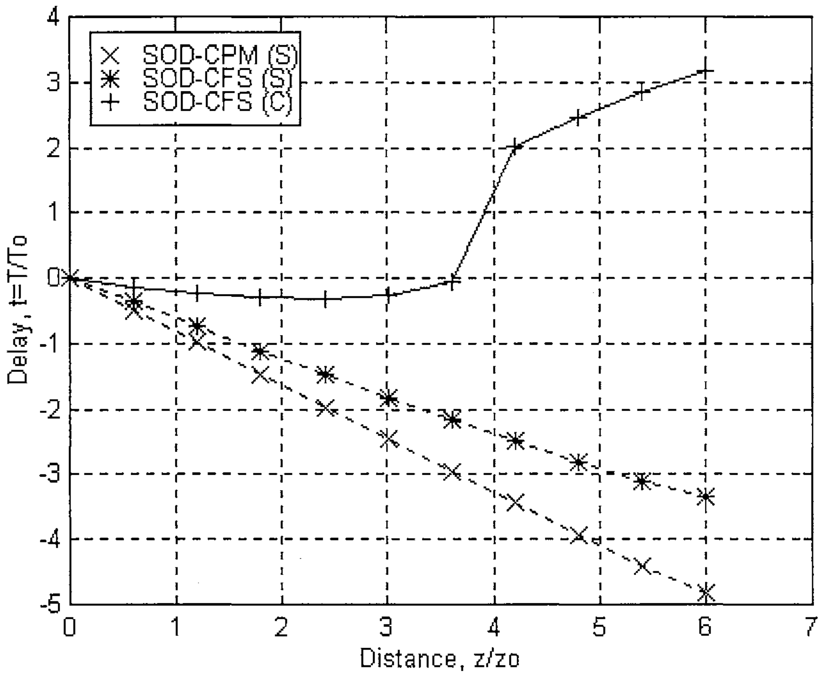


Figure 5.15.2 STG y-axis control pulse arrival delay time versus distance. Plot with solid line represent the y-polarization control pulse co-propagating with the signal pulse, and those with a dashed line represent the y-polarization control pulse propagating alone. Control and signal pulse widths = 50 fs, $N_T = 1.24$ and $\theta = 45^\circ$.

5.2.2 Soliton Dragging Gate (SDG)

For the SDG, the two orthogonally polarized soliton pulses are of unequal amplitude, such that the control pulse is always great than that of the signal pulse. This implies that the control pulse can either be on the x-polarization axis $(\theta = 30^\circ)$, or on the y-polarization axis $(\theta = 60^\circ)$.

For the case when $\theta = 30^\circ$, the time delay of the two orthogonally polarized soliton pulses (co-propagating) against the propagation distance, for the effect sequence SOD-CPM, is compared with those of effect sequences: (a) SOD-SFS, (b) SOD-CFS, (c) SOD-CS, and (d) SOD-SFS+CFS in Figures 5.16.1 and 5.16.2.

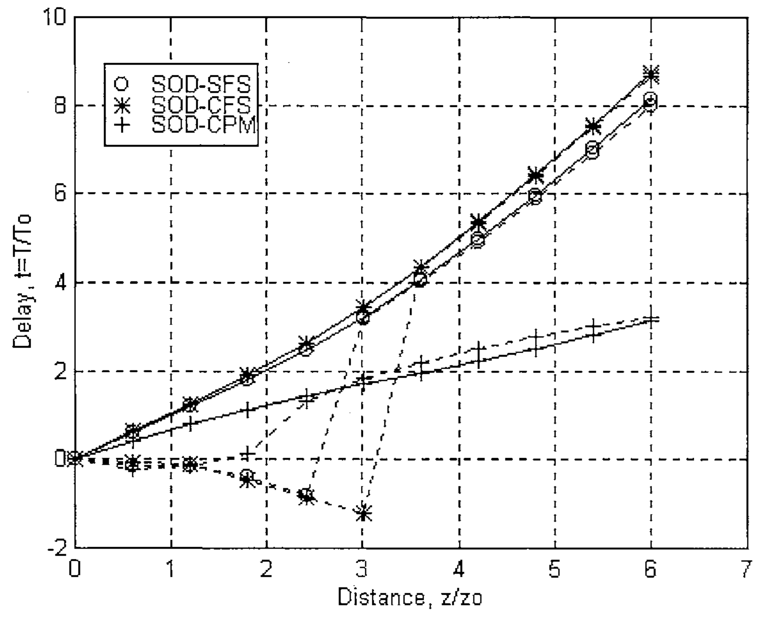


Figure 5.16.1 SDG control pulse arrival delay time versus distance for SOD-SFS, and SOD-CFS. Plots with solid line represent the x-polarization control pulse (on slow axis) and those with a dashed line represent the y-polarization signal pulse (on fast axis). Control and signal pulse widths = 50 fs, $N_T = 1.24$ and $\theta = 30^\circ$.

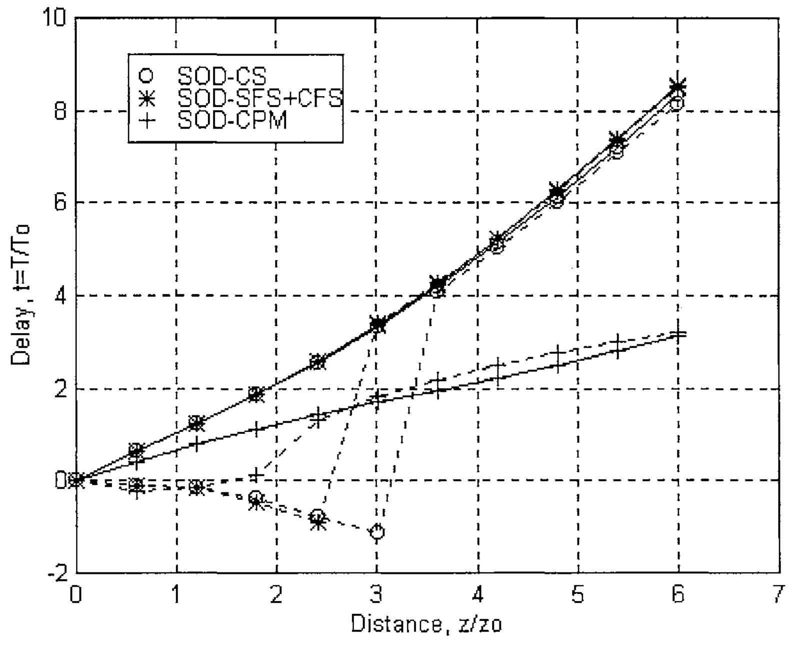


Figure 5.16.2 SDG control pulse arrival delay time versus distance for SOD-CS, and SOD-SFS+CFS. Plots with solid line represent the x-polarization control pulse (on slow axis) and those with a dashed line represent the y-polarization signal pulse (on fast axis). Control and signal pulse widths = 50 fs, $N_T = 1.24$ and $\theta = 30^\circ$.

It can be noted from Figure 5.16.1 that the combined effect of TOD, SS and SFS delays the control pulse on the slow axis significantly by about $4.92T_0$. The control pulse is further delayed by the inclusion of CS and CFS by about $0.78T_0$. Comparing Figure 5.16.2 to Figure 5.16.1 yields that the control pulse delay, on the slow axis, due to CFS ($\approx 0.46T_0$) is larger than that due to CS ($\approx 0.31T_0$) in the SDG.

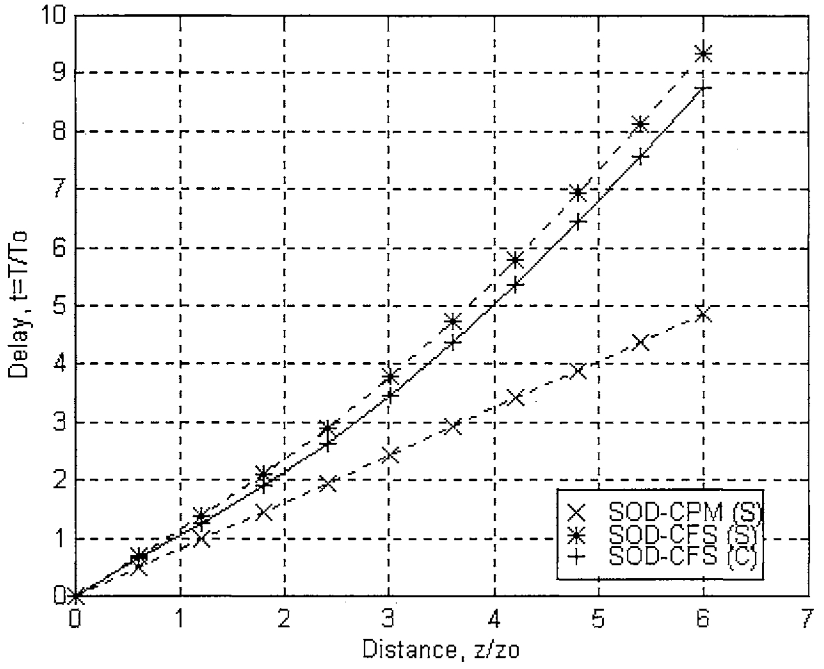


Figure 5.17.1 SDG x-axis control pulse arrival delay time versus distance. Plot with solid line represents the x-polarization control pulse co-propagating with the signal pulse, and those with a dashed line represent the x-polarization control pulse propagating alone. Control and signal pulse widths = 50 fs, $N_T = 1.24$ and $\theta = 30^\circ$.

In order to study how the switching efficiency of the SDG at $\theta = 30^\circ$ is affected by TOD, SS, SFS, CS and CFS, the same analysis was carried out as for the STG. Figure 5.17.1 gives the results for the control pulse on the x-polarization axis (slow axis), and Figure 5.17.2 gives the results for the signal pulse on the y-polarization axis (fast axis). The efficiency of the SDG at $\theta = 30^\circ$, under SOD-CFS effects, using equation (2.84) is

0.14 ($T_{mctw} = 1.12T_0$). The efficiency of the SDG at $\theta = 30^\circ$, under SOD-CPM effects, is 0.37 ($T_{mctw} = 2.96T_0$). This implies that the effects due to SOD-CFS reduce T_{mctw}, under SOD-CPM effects, by about 62%. Since in both cases the efficiency is less than 1.00, it implies that a reduced clock time window is required to implement the SDG at $\theta = 30^\circ$. A larger clock time window reduction is required for the case with SOD-CFS effects, than the case with SOD-CPM effects. Therefore, with respect to the efficiency equation (2.84), it is undesirable to operate the SDG at $\theta = 30^\circ$ under SOD-CFS effects.

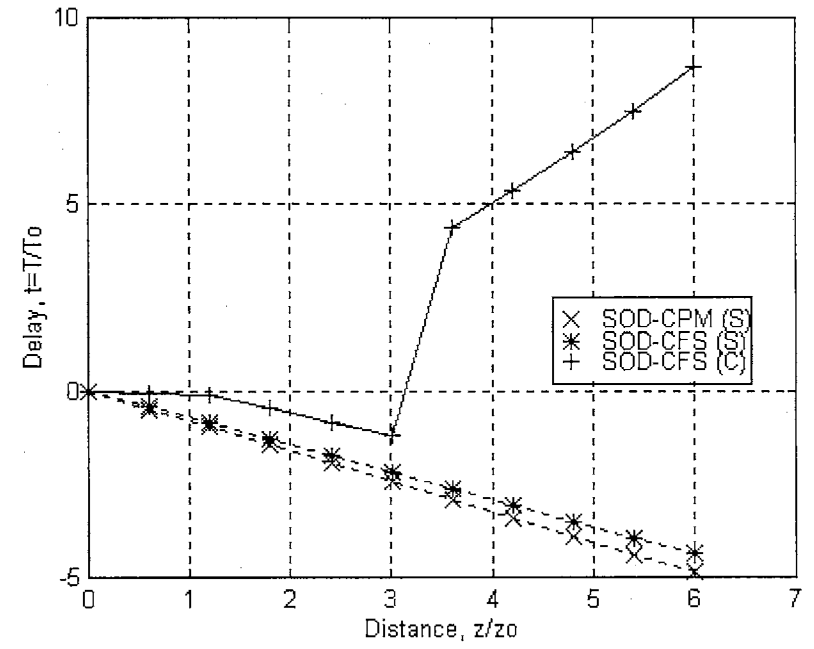


Figure 5.17.2 SDG y-axis signal pulse arrival delay time versus distance. Plot with solid line represents the y-polarization signal pulse co-propagating with the control pulse, and those with a dashed line represent the y-polarization signal pulse propagating alone. Control and signal pulse widths = 50 fs, $N_T = 1.24$ and $\theta = 30^\circ$.

For the case when $\theta = 60^\circ$, the time delay of the two orthogonally polarized soliton pulses (co-propagating) against the propagation distance, for the effect sequence SOD-CPM, is compared with those of effect sequences: (a) SOD-SFS, (b) SOD-CFS, (c) SOD-CS, and (d) SOD-SFS+CFS in Figures 5.18.1 and 5.18.2.

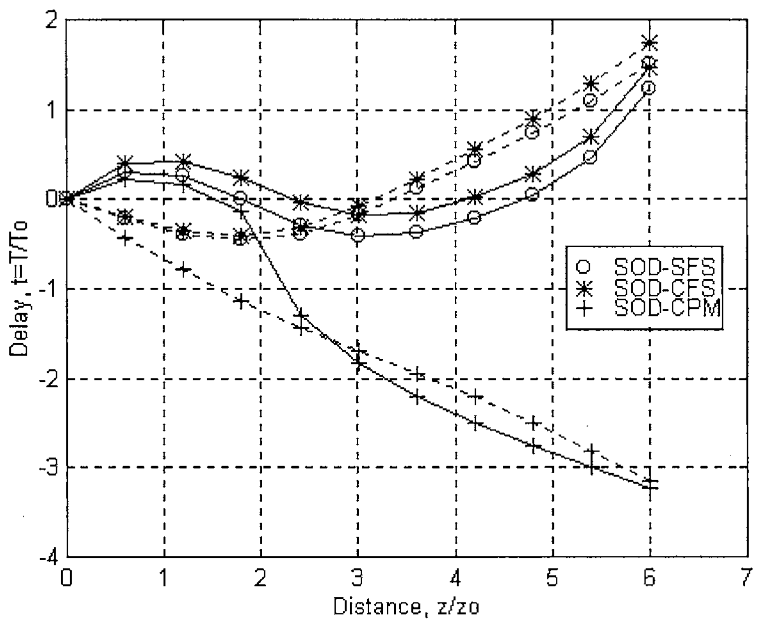


Figure 5.18.1 SDG control pulse arrival delay time versus distance for SOD-SFS, and SOD-CFS. Plots with solid line represent the x-polarization signal pulse (on slow axis) and those with a dashed line represent the y-polarization control pulse (on fast axis). Control and signal pulse widths = 50 fs, $N_T = 1.24$ and $\theta = 60^\circ$.

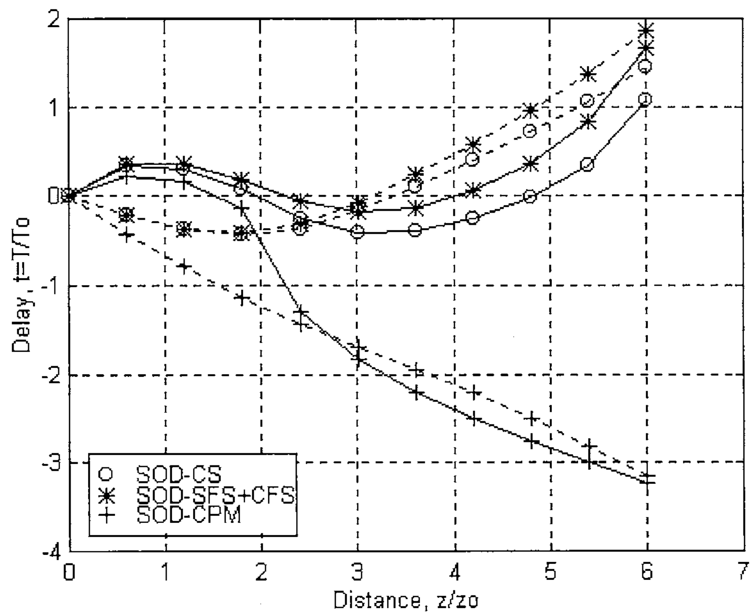


Figure 5.18.2 SDG control pulse arrival delay time versus distance for SOD-CS, and SOD-SFS+CFS. Plots with solid line represent the x-polarization signal pulse (on slow axis) and those with a dashed line represent the y-polarization control pulse (on fast axis). Control and signal pulse widths = 50 fs, $N_T = 1.24$ and $\theta = 60^\circ$.

It can be observed from Figure 5.18.1 that the combined effect of TOD, SS and SFS delays the control pulse on the fast axis significantly by about $4.62T_0$. The control pulse is further delayed by the inclusion of CS and CFS by about $0.19T_0$. Comparing Figure 5.18.2 to Figure 5.18.1 yields that the control pulse on the fast axis is speed-up by about $0.08T_0$ due to CS and delayed by about $0.31T_0$ due to CFS in the SDG.

A similar analysis of the switching efficiency of the SDG at $\theta = 60^\circ$, with effects due to TOD, SS, SFS, CS and CFS, can be carried out as for the STG. Figure 5.19.1 gives the results for the signal pulse on the x-polarization axis (slow axis), and Figure 5.19.2 gives the results for the control pulse on the y-polarization axis (fast axis). The efficiency of the SDG at $\theta = 60^\circ$, under SOD-CFS effects, using equation (2.84) is 0.53 $\left(T_{mctw} = 4.24T_0\right)$. The efficiency of the SDG at $\theta = 60^\circ$, under SOD-CPM effects, is 0.41 ($T_{mctw} = 3.28T_0$).

The effects due to SOD-CFS increase T_{mctw}, under SOD-CPM effects, by 28%. Since in both cases the efficiency is less than 1.00, it implies that a reduced clock time window is required to implement the SDG at $\theta = 60^\circ$. A larger clock time window reduction is required for the case with SOD-CPM effects, than the case with SOD-CFS effects. Therefore, with respect to the efficiency equation (2.84), it is desirable to operate the SDG at $\theta = 60^\circ$ under SOD-CFS effects.

Clearly the effects due to TOD, SS, CS, and CFS, with respect to SOD-CPM effects, reduce the switching efficiency of the SDG at $\theta = 30^\circ$, and increase it at $\theta = 60^\circ$. However, in both cases the efficiency is well below one.

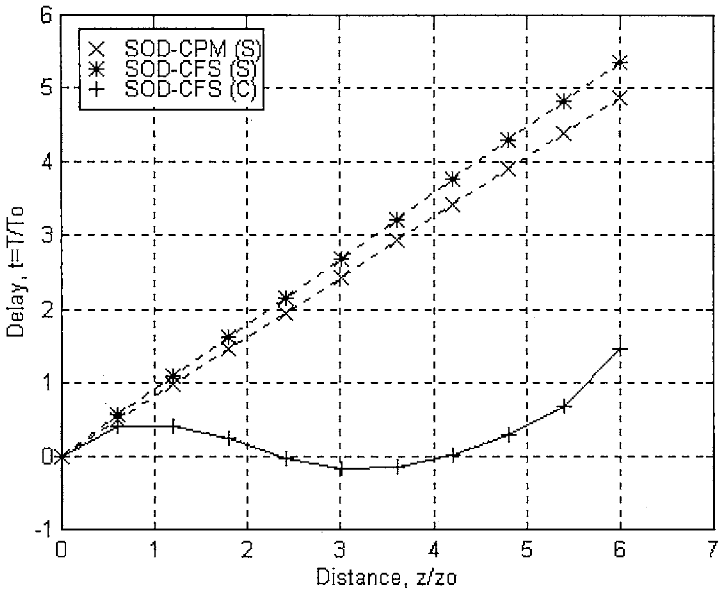


Figure 5.19.1 SDG x-axis signal pulse arrival delay time versus distance. Plot with solid line represents the x-polarization signal pulse co-propagating with the control pulse, and those with a dashed line represent the x-polarization signal pulse propagating alone. Control and signal pulse widths = 50 fs, $N_T = 1.24$ and $\theta = 60^\circ$.

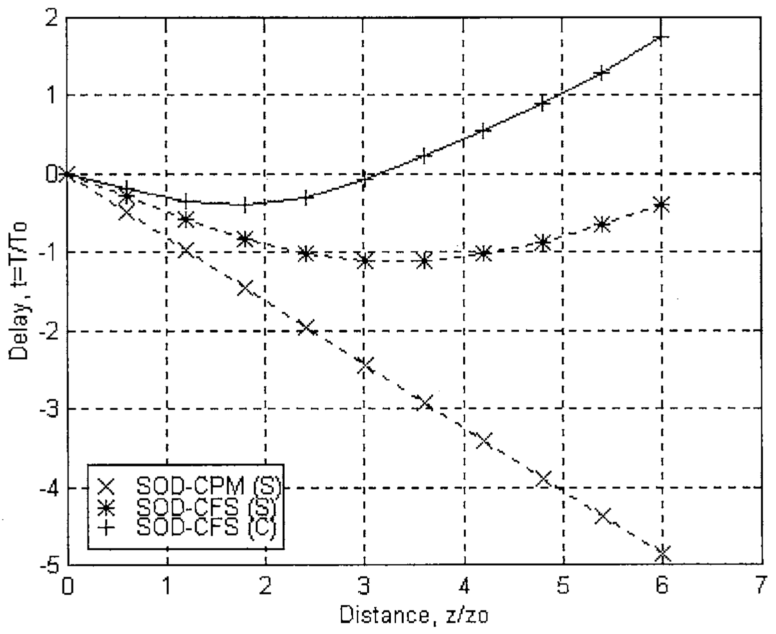


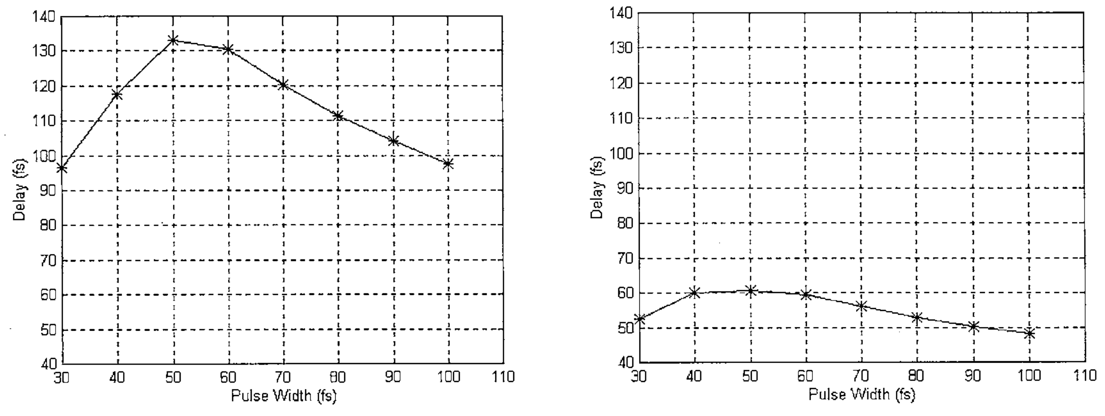
Figure 5.19.2 SDG y-axis control pulse arrival delay time versus distance. Plot with solid line represents the y-polarization control pulse co-propagating with the signal pulse, and those with a dashed line represent the y-polarization control pulse propagating alone. Control and signal pulse widths = 50 fs, $N_T = 1.24$ and $\theta = 60^\circ$.

5.2.3 Switching Maximum Clock Time Window Variation with Pulse Width at Constant Birefringence

At a constant birefringence of $B = 1.1103 \times 10^{-4}$, the pulse widths of the orthogonal pulses were varied from 30 fs to 100 fs, for the STG and SDG with the control on the fast axis. The delay between the peak centroid of the control pulse co-propagating with the signal pulse, and the control pulse propagation alone, gives the maximum clock time window. The peak centroid delay is defined as

$$T_d = \frac{\sum_{i=1}^{N_0} |u_i|^2 T_i}{\sum_{i=1}^{N_0} |u_i|^2} \tag{5.6}$$

Here N_0 is the number of sample points, u_i is the complex pulse envelope instantaneous amplitude at discrete time interval T_i. Figure 5.20(a) and Fig. 5.20(b) show the variation of the delay against the pulse width (FWHM), for the STG and SDG, with the control on the fast axis, respectively.



(a) (b)

Figure 5.20 Control pulse delay against pulse width (FWHM) for STG(a)/SDG(b).

From Figure 5.20, it can be observed that around the normalized birefringence of 0.5 the delay is a maximum. It is therefore crucial to design for a maximum clock time window at the normalized birefringence of about 0.5 for both gates. The STG has a clock time window nearly twice larger than that of the SDG. This implies that the STG switches more efficiency than the SDG for sub-100 fs soliton pulses, where both gates are operated with the control on the fast axis.

5.3 PHASE III

This section gives the results obtained from the STG-NOLM and STG-PSI-NOLM gates, for 50 fs soliton pulses at 1.55 μm in a strongly birefringent singlemode fiber over several walk-off lengths. The normalized total soliton amplitude at the input of the STG-NOLM and STG-PSI-NOLM gates is set at 1.24. The normalized soliton amplitude of the signal at the input polarizing beam splitter for the STG-NOLM and STG-PSI-NOLM gates is set at $1.24/\sqrt{2} = 0.8768$.

In order to observe the effects introduced by the higher order nonlinear and dispersion effects, the STG-NOLM and STG-PSI-NOLM gates where simulated under SOD-CPM effects and then under SOD-CFS. The switching characteristics of the gates where then evaluated under SOD-CFS, as this represents the complete case. For the STG-NOLM gate, the NOLM loop propagation length L is over $L = 11 l_{wo}$ (l_{wo} = walk-off length), with no cross-splices at all. The STG-PSI-NOLM gate is simulated with cross-splices over a NOLM loop length of $L = 11 l_{seg}$, at an odd number of segment lengths $\left(\text{with } l_{seg} = l_{wo}\right)$ or even number of cross-splices, to ensure the same polarization at the cross-coupler.

The STG-PSI-NOLM gate is also simulated over $L = 7l_{seg}$, at an odd number of segment lengths $\left(\text{with } l_{seg} = 3l_{wo}\right)$, for comparison of pulse characteristics. The walk-off length $l_{wo} = \left(cT_{FWHM}\right)/\Delta n$ for 50 fs (FWHM) pulses with a birefringence of $B = \Delta n = 1.1103\times10^{-4}$ gives $l_{wo} = 0.1350$ m. For each of the gates, the output control pulse transmission and centroid frequency shift, are determined over various normalized initial separations $\left(t_s = t/T_0\right)$ of the clockwise control pulse, and the clockwise signal pulse at the input polarization beam splitter. The centroid frequency shift is given by

$$f_c = \frac{\sum_{i=1}^{N_0} |u_i|^2 f_i}{\sum_{i=1}^{N_0} |u_i|^2}. \tag{5.7}$$

Here N_0 is the number of sample points, u_i is the complex pulse envelope instantaneous amplitude at discrete frequency f_i. This is followed by the output port control pulse, clockwise control and signal pulses, and counter clockwise control pulse characteristics.

5.3.1 STG-NOLM Gate

The output port control pulse transmission and frequency shift versus the initial separation, of the clockwise control and signal pulses are given in Figures 5.21.1 to 5.26.2.

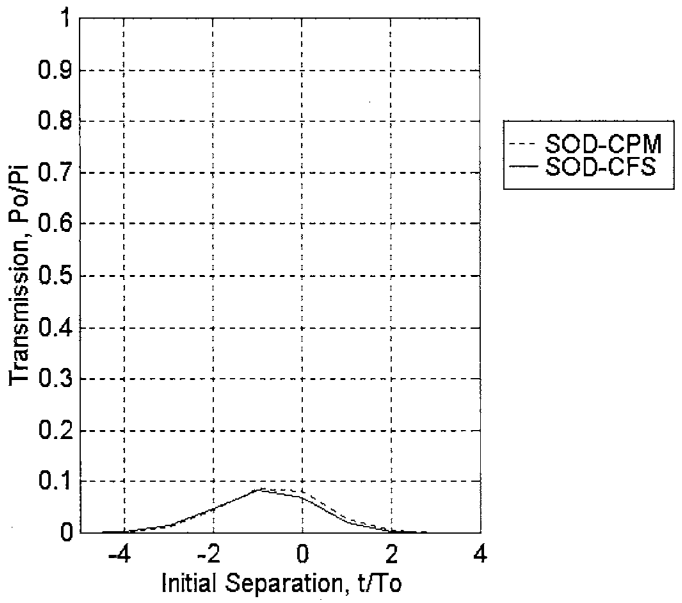


Figure 5.21.1 STG-NOLM gate output control pulse transmission against initial separation at $L = l_{wo}$.

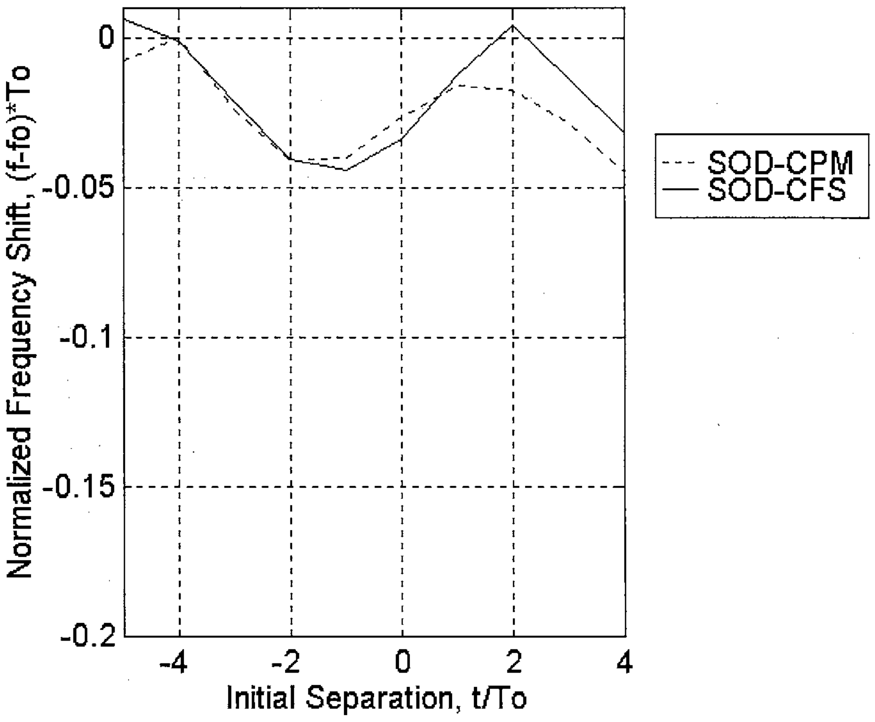


Figure 5.21.2 STG-NOLM gate output control pulse frequency shift against initial separation at $L = l_{wo}$.

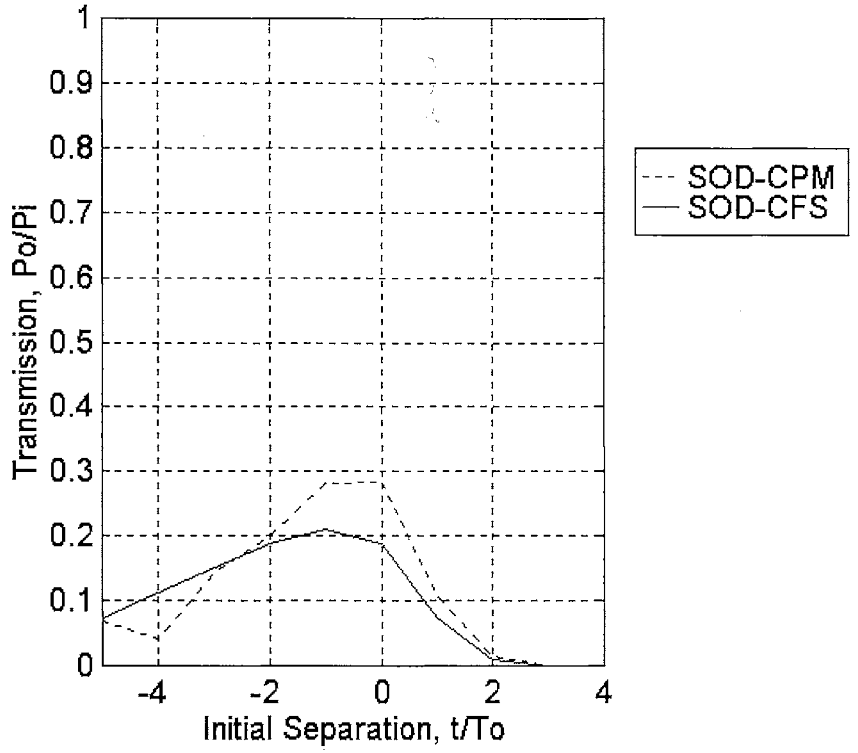


Figure 5.22.1 STG-NOLM gate output control pulse transmission against initial separation at $L = 3l_{wo}$.

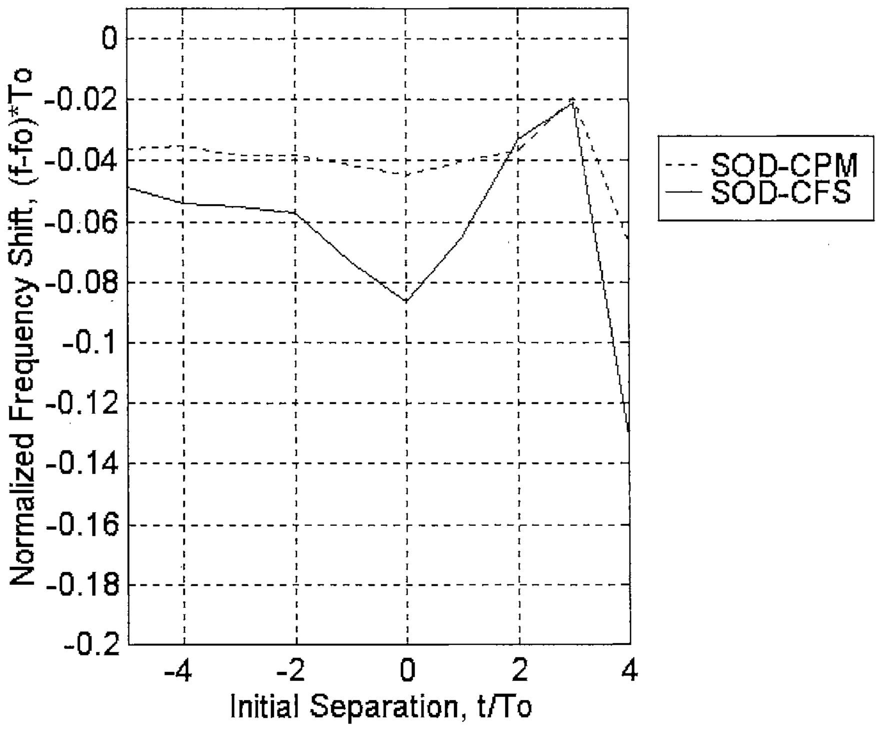


Figure 5.22.2 STG-NOLM gate output control pulse frequency shift against initial separation at $L = 3l_{wo}$.

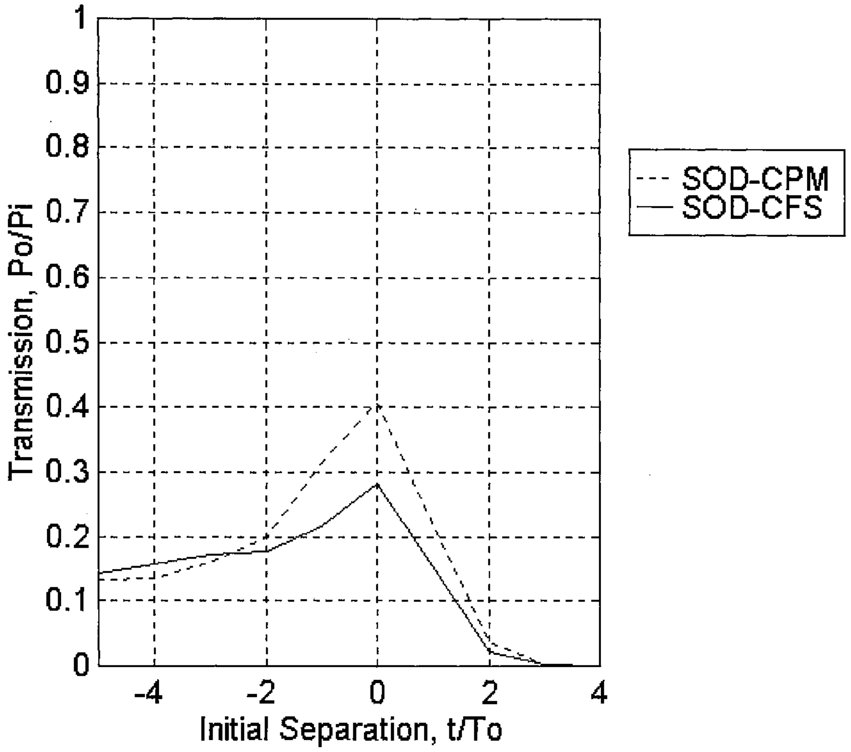


Figure 5.23.1 STG-NOLM gate output control pulse transmission against initial separation at $L = 5l_{wo}$.

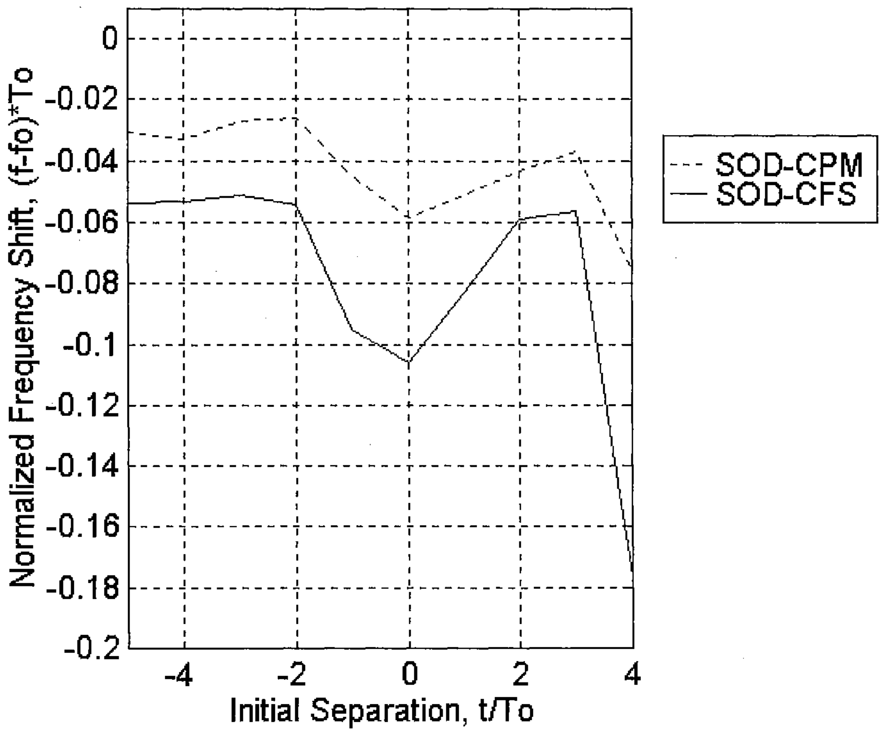


Figure 5.23.2 STG-NOLM gate output control pulse frequency shift against initial separation at $L = 5l_{wo}$.

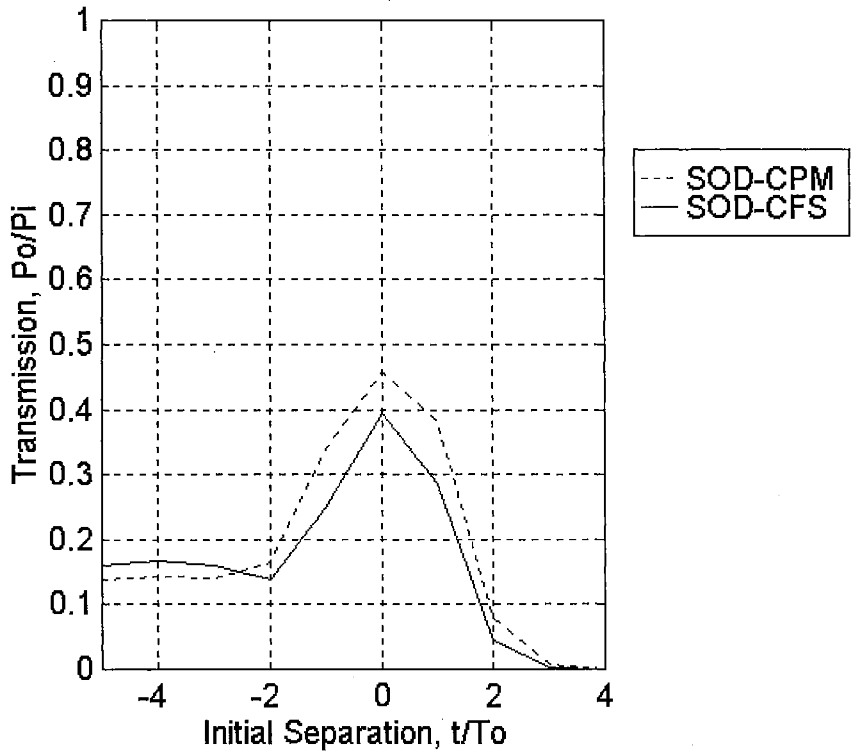


Figure 5.24.1 STG-NOLM gate output control pulse transmission against initial separation at $L = 7l_{wo}$.

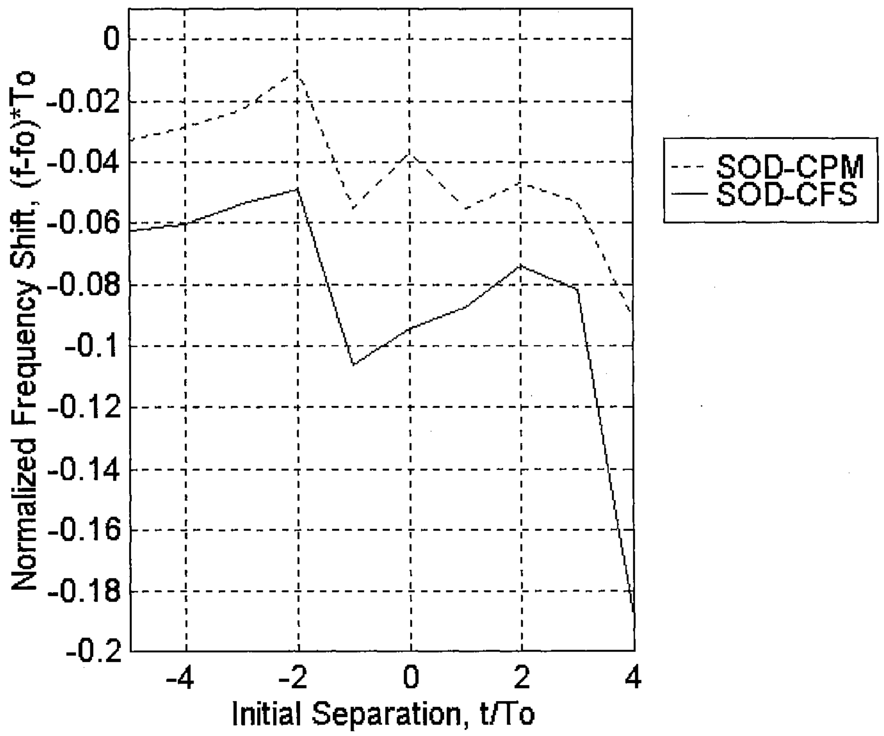


Figure 5.24.2 STG-NOLM gate output control pulse frequency shift against initial separation at $L = 7l_{wo}$.

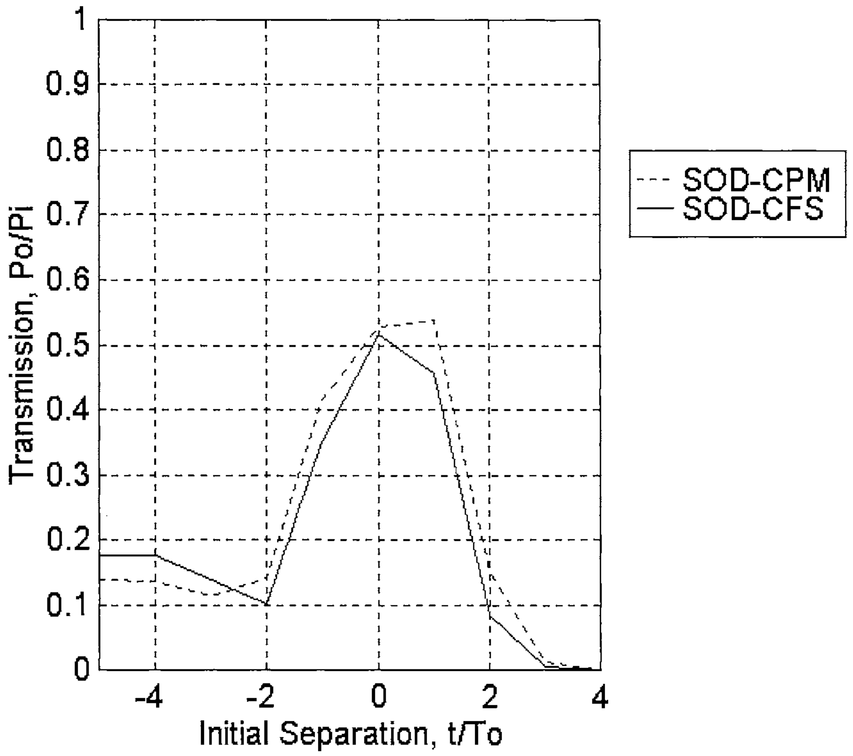


Figure 5.25.1 STG-NOLM gate output control pulse transmission against initial separation at $L = 9l_{wo}$.

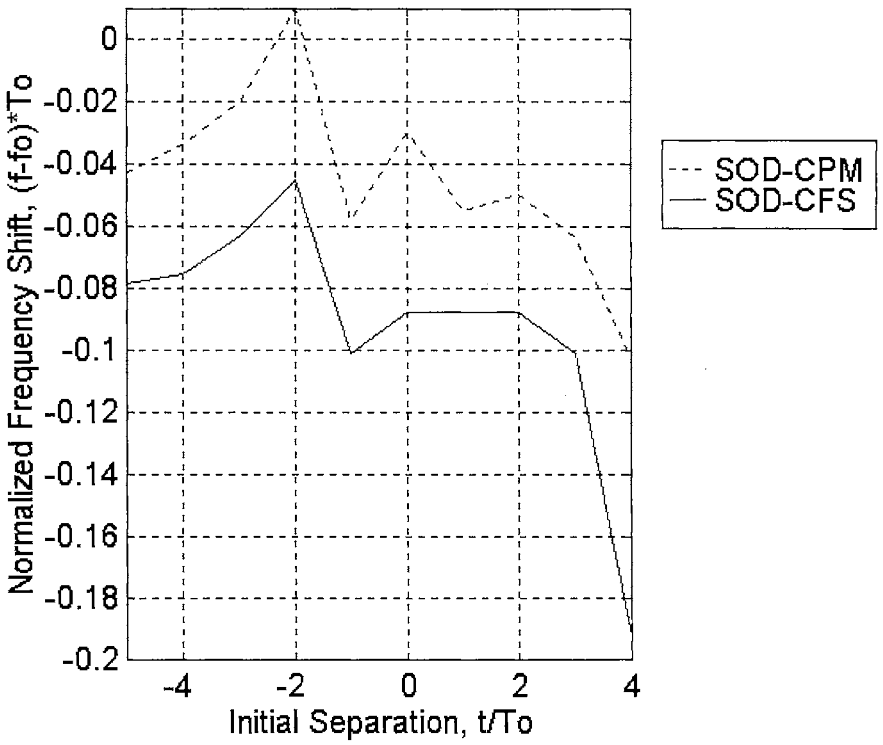


Figure 5.25.2 STG-NOLM gate output control pulse frequency shift against initial separation at $L = 9l_{wo}$.

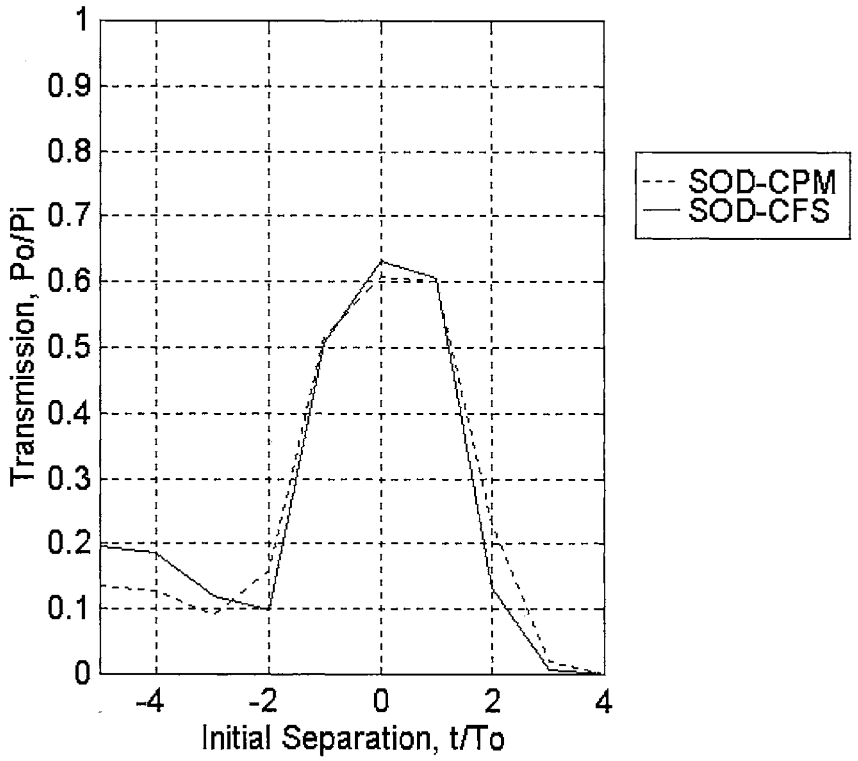


Figure 5.26.1 STG-NOLM gate output control pulse transmission against initial separation at $L = 11l_{wo}$.

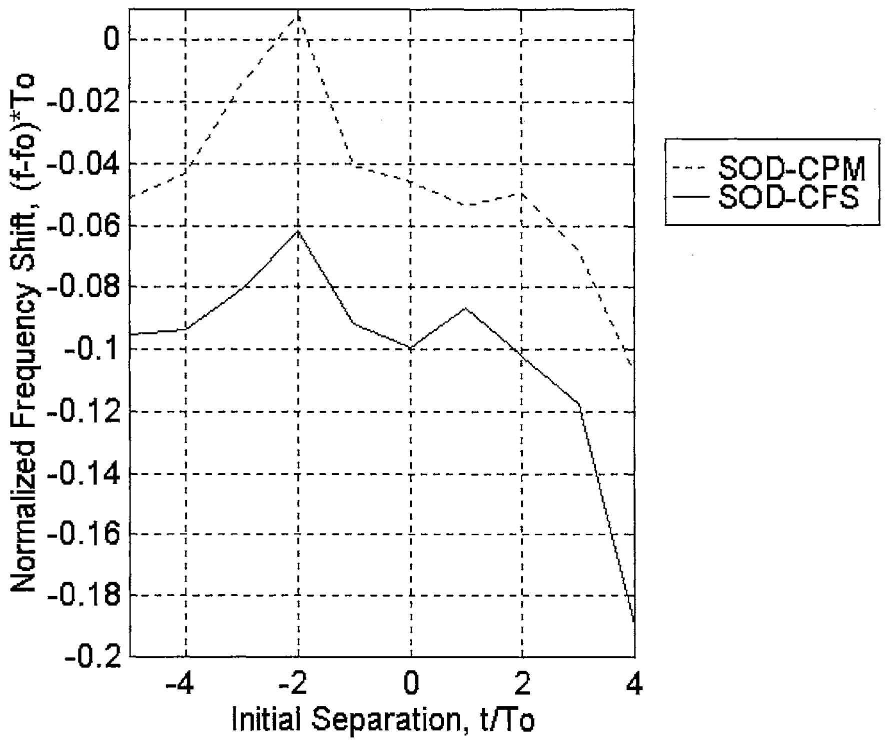


Figure 5.26.2 STG-NOLM gate output control pulse frequency shift against initial separation at $L = 11l_{wo}$.

The results for the STG-NOLM gate, under SOD-CFS effects as observed from Figures 5.21.1 to 5.26.2 show that, a peak output control pulse transmission of 0.2 occurs within a clock time window (at half power maximum) of about $-4.0 \leq t_s \leq 1.0$ (≈ 3 pulse widths) at $L = 3l_{wo}$, when the control pulse just begins to split. The peak output control pulse transmission, under SOD-CPM and SOD-CFS effects, increase in magnitude with the NOLM loop length L. The peak output control pulse transmission from $L = l_{wo}$ to $L = 11l_{wo}$ increases from 0.0857, for both SOD-CPM and SOD-CFS effects, to 0.5929 and 0.6143 under SOD-CPM and SOD-CFS effects respectively. The SOD-CFS effects increasingly reduce the transmission, under SOD-CPM, at $t_s = 0.0$ and $L = l_{wo}$ from 25% to 30% at $L = 5l_{wo}$, and thereafter the transmission reduction is diminished until the effects increase transmission by 4% at $L = 11l_{wo}$. Although a maximum transmission of 0.6143 is achieved at $t_s = 0.0$ and $L = 11l_{wo}$, the output control pulse starts splitting into two pulses at $L = 3l_{wo}$, and the separation of the split up increases with the NOLM loop length L. The STG-NOLM gate is a very poor optical fiber gate due to the pulse split up at short NOLM loop lengths, and the low transmission, of about 0.2 at $L = 3l_{wo}$, just when the output control pulse begins to split. The output control pulse centroid frequency shift generally increases from $L = l_{wo}$ to $L = 11l_{wo}$, with the frequency shift due to SOD-CFS effects becoming increasingly larger than the frequency shift due to SOD-CPM effects.

Figures 5.27.1 to 5.27.4 give the output port control pulse temporal and spectral characteristics, where the pulse splitting can be clearly seen to occur at $L = 3l_{wo}$. The initial separation is set to $t_s = 0.0$ for maximum output of the control pulse.

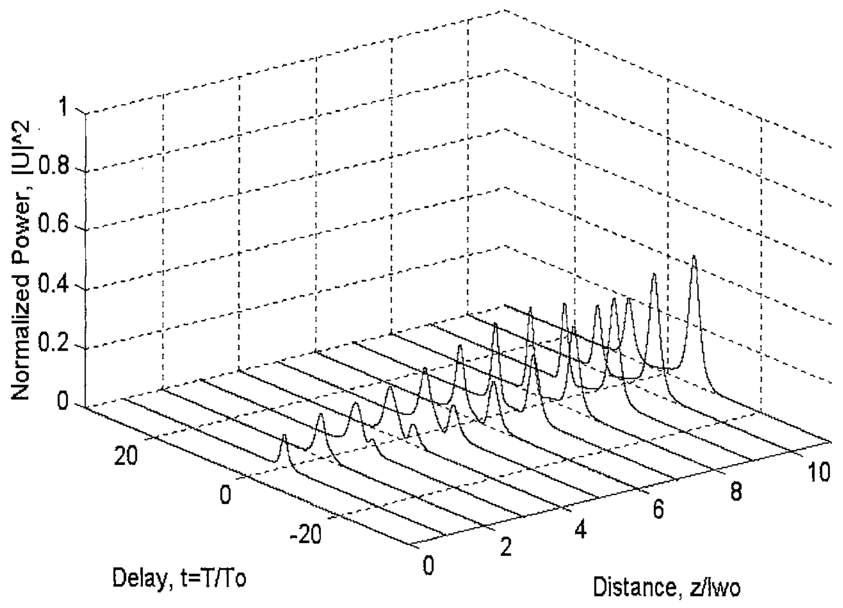


Figure 5.27.1 STG-NOLM gate under SOD-CFS control pulse time domain profile at $t_s = 0.0$ over $L = 11l_{wo}$.

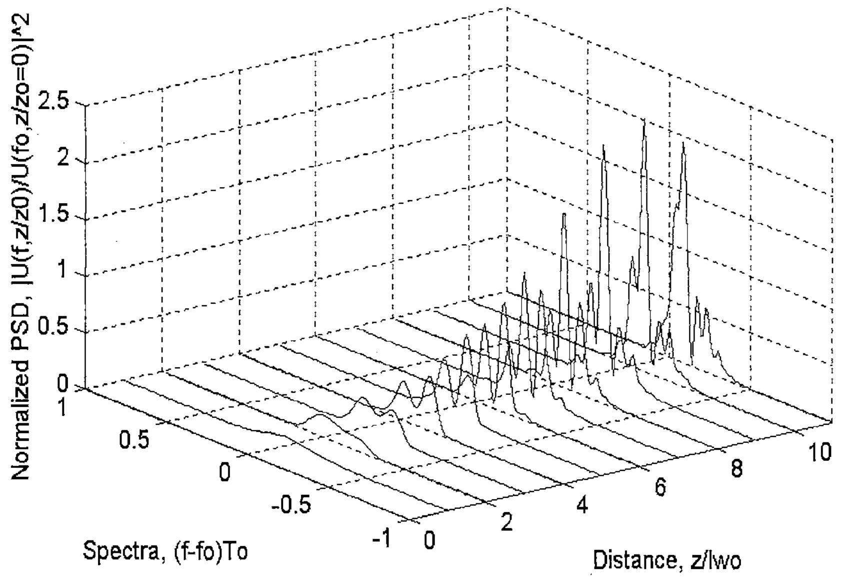


Figure 5.27.2 STG-NOLM gate under SOD-CFS control pulse frequency domain profile at $t_s = 0.0$ over $L = 11l_{wo}$.

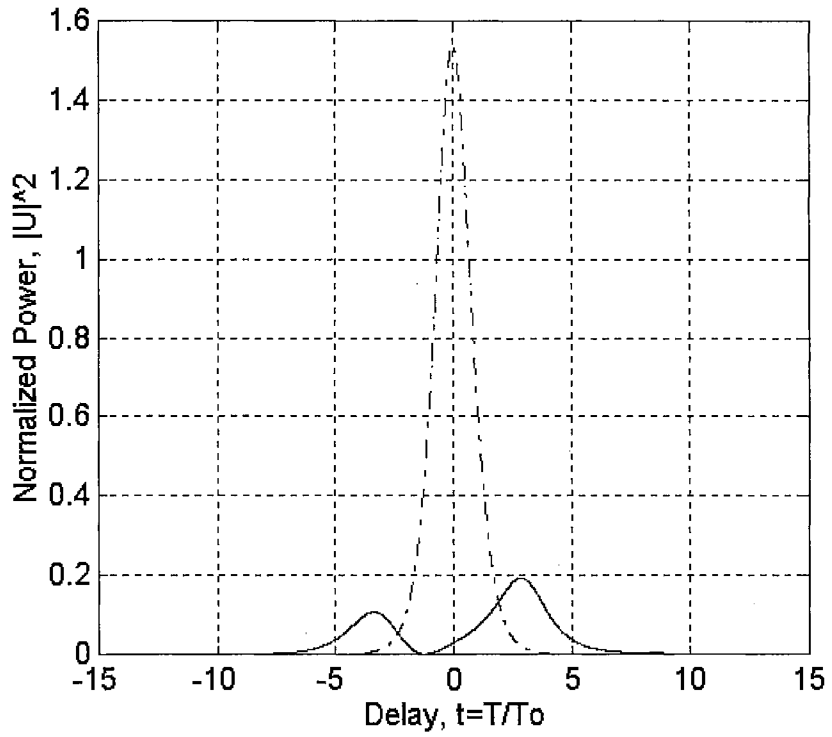


Figure 5.27.3 STG-NOLM gate under SOD-CFS control pulse time domain profile at $t_s = 0.0$ and $L = 5l_{wo}$. Dashdot and solid lines represent the input and output control pulses respectively.

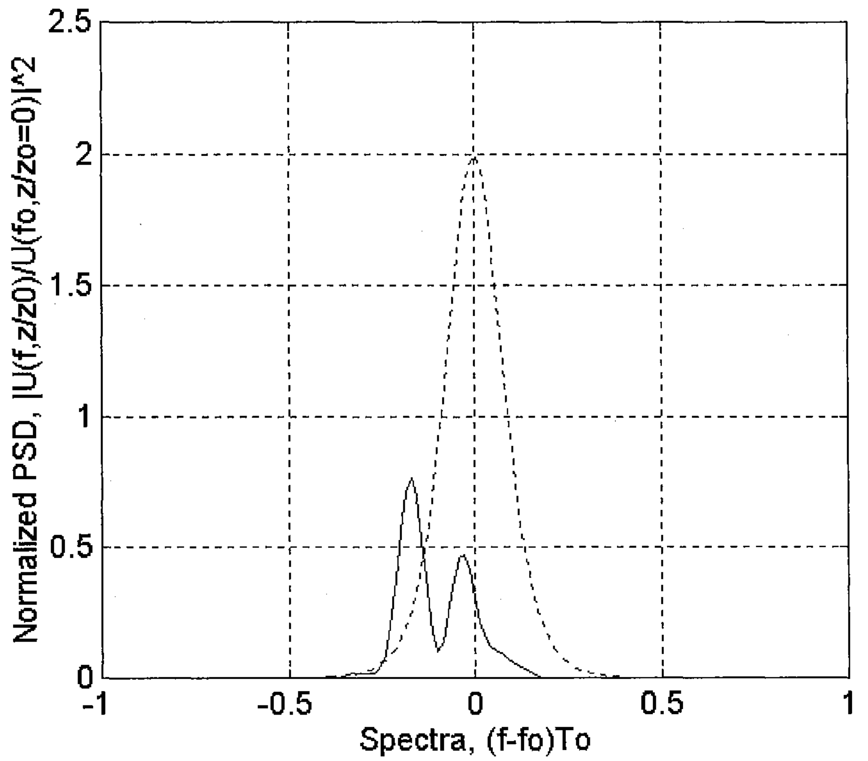


Figure 5.27.4 STG-NOLM gate under SOD-CFS control pulse frequency domain profile at $t_s = 0.0$ and $L = 5l_{wo}$. Dashdot and solid lines represent the input and output control pulses respectively.

In order to get an insight into why the switching performance of the STG-NOLM is very poor, one needs to observe the clockwise control and signal pulses, and counter clockwise control pulse just before the interferometric switch, in both the time and frequency domain. The clockwise control and signal pulses, and counter clockwise control pulse temporal and spectral characteristics, prior to switching, are give in Figures 5.27.5 to 5.27.10.

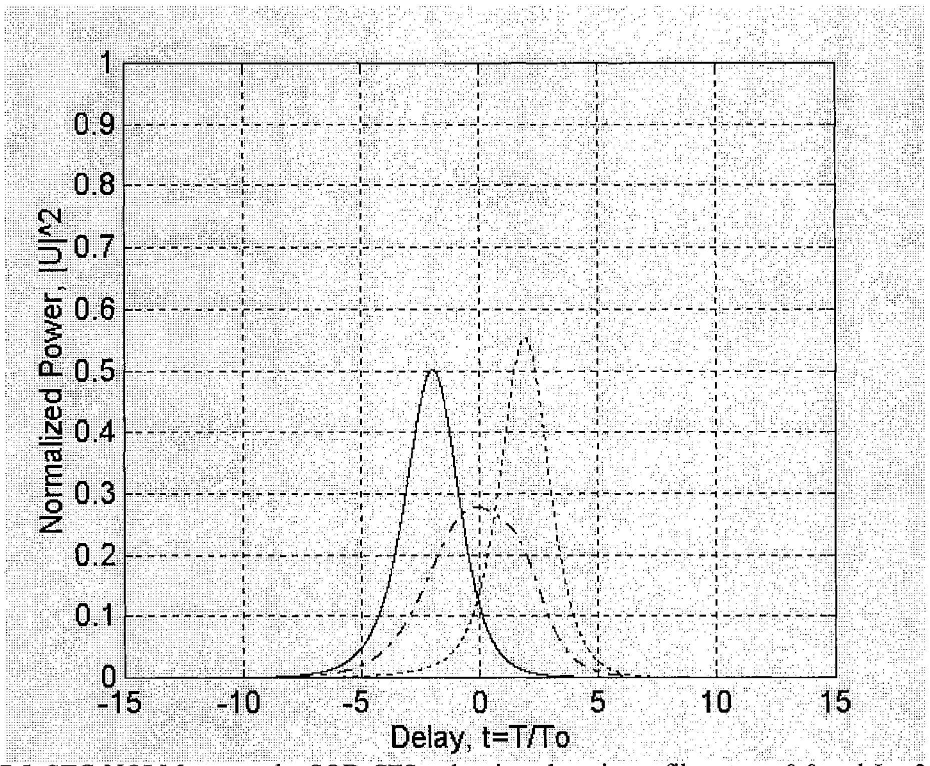


Figure 5.27.5 STG-NOLM gate under SOD-CFS pulse time domain profiles at $t_s = 0.0$ and $L = 3l_{wo}$. Dotted, dashdot, and solid lines represent the clockwise signal, clockwise control and counter-clockwise control pulses.

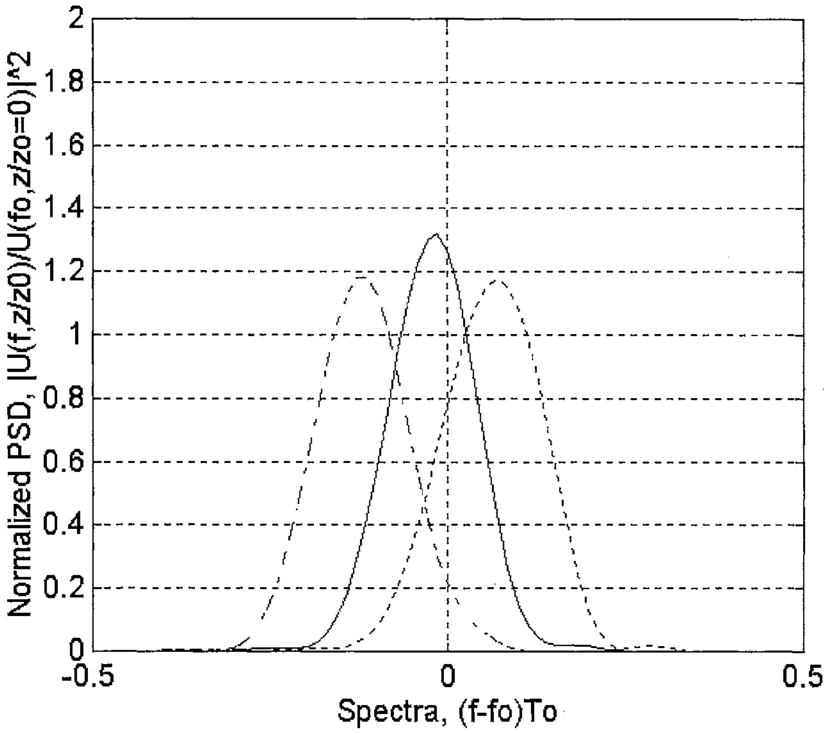


Figure 5.27.6 STG-NOLM gate under SOD-CFS pulse frequency domain profiles at $t_s = 0.0$ and $L = 3l_{wo}$. Dotted, dashdot, and solid lines represent the clockwise signal, clockwise control and counter-clockwise control pulses.

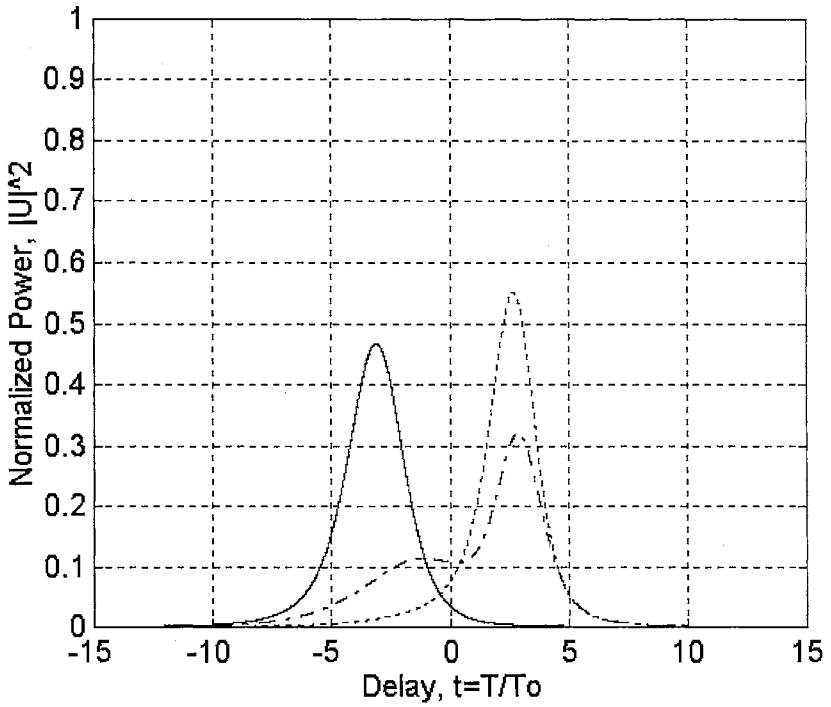


Figure 5.27.7 STG-NOLM gate under SOD-CFS pulse time domain profiles at $t_s = 0.0$ and $L = 5l_{wo}$. Dotted, dashdot, and solid lines represent the clockwise signal, clockwise control and counter-clockwise control pulses.

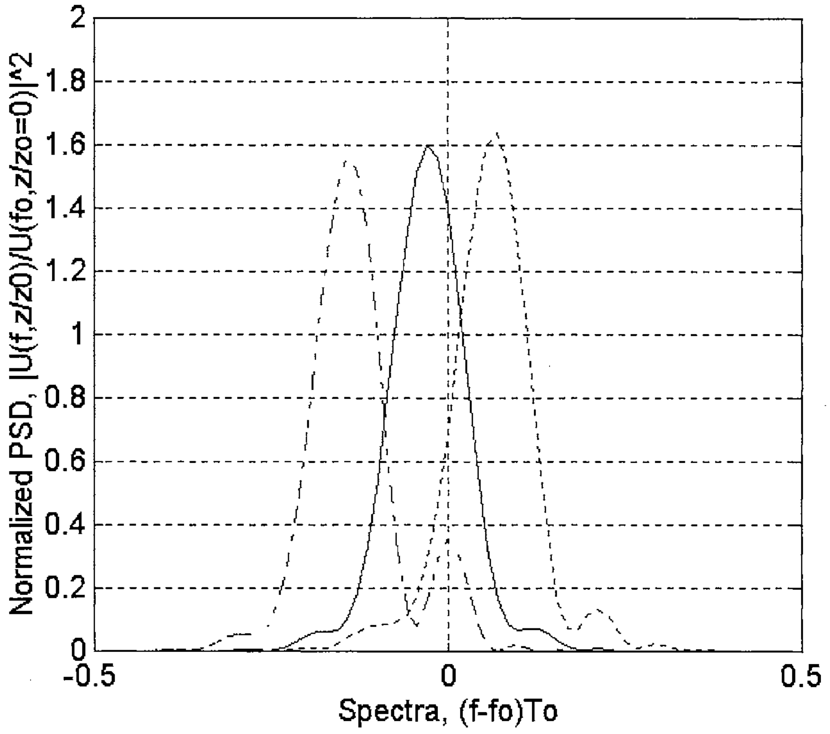


Figure 5.27.8 STG-NOLM gate under SOD-CFS pulse frequency domain profiles at $t_s = 0.0$ and $L = 5l_{wo}$. Dotted, dashdot, and solid lines represent the clockwise signal, clockwise control and counter-clockwise control pulses.

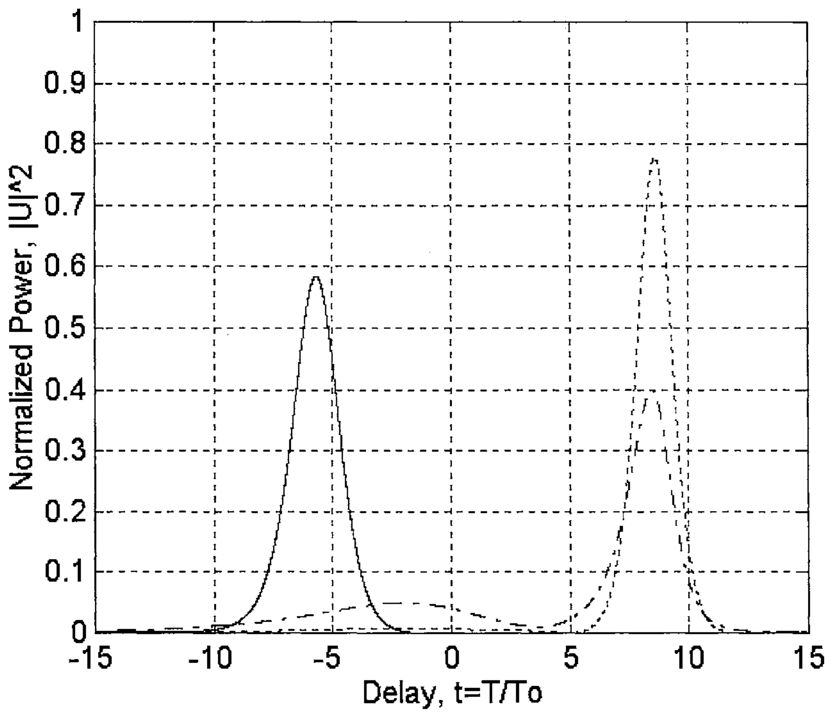


Figure 5.27.9 STG-NOLM gate under SOD-CFS pulse time domain profiles at $t_s = 0.0$ and $L = 11l_{wo}$. Dotted, dashdot, and solid lines represent the clockwise signal, clockwise control and counter-clockwise control pulses.

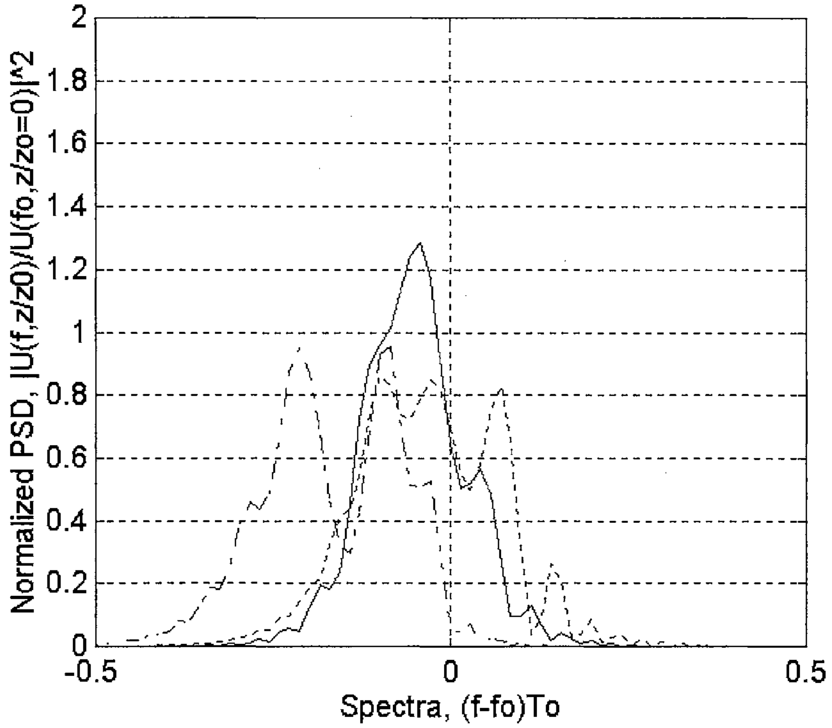


Figure 5.27.10 STG-NOLM gate under SOD-CFS pulse frequency domain profiles at $t_s = 0.0$ and $L = 11l_{wo}$. Dotted, dashdot, and solid lines represent the clockwise signal, clockwise control and counter-clockwise control pulses.

It can be observed from Figures 5.27.5 to 5.27.10 that trapping of the clockwise control pulse by the clockwise signal causes it to be delayed in time with respect to the counter clockwise control pulse. This time delay arises from the group velocity mismatch of the clockwise and counterclockwise control pulse. The clockwise control pulse is at a group velocity that is midway between that of the fast and slow axis, and the counter clockwise control pulse is at the fast axis group velocity. The group velocity mismatch causes the output control pulse to split up, well before the clockwise control pulse can acquired sufficient CPM with respect to the counter clockwise control pulse. With increasing NOLM loop length L, the spectral profiles of the clockwise control and signal pulses, and the counter clockwise signal pulse become severely distorted.

5.3.2 STG-PSI-NOLM gate with Segment Length Equal to the Walk-off Length

To improve the performance of the STG-NOLM gate, it was discovered that the clockwise control and signal pulses need to under go partial shuffle interaction (PSI) CPM. During PSI CPM the clockwise control and signal pulses profiles are made to be stay in temporal contact, and the clockwise and counter clockwise control pulse group velocity mismatch is made to be initially small by using cross-splices. The cross-splices are setup after each segment length (l_{seg}) equal to the walk-off length (l_{wo}). The improved performance, however, degrades as l_{seg} is increased as will later been seen for the case when $l_{seg} = 3l_{wo}$.

The output port control pulse transmission and frequency shift versus the initial separation of the clockwise control and signal pulses, are given in Figures 5.28.1 to 5.33.2.

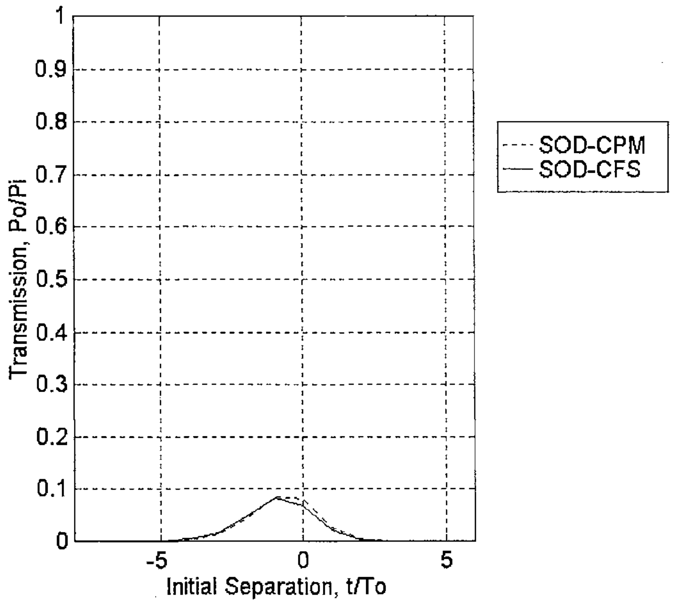


Figure 5.28.1 STG-PSI-NOLM gate output control pulse transmission against initial separation at $L = l_{seg}$, with $l_{seg} = l_{wo}$.

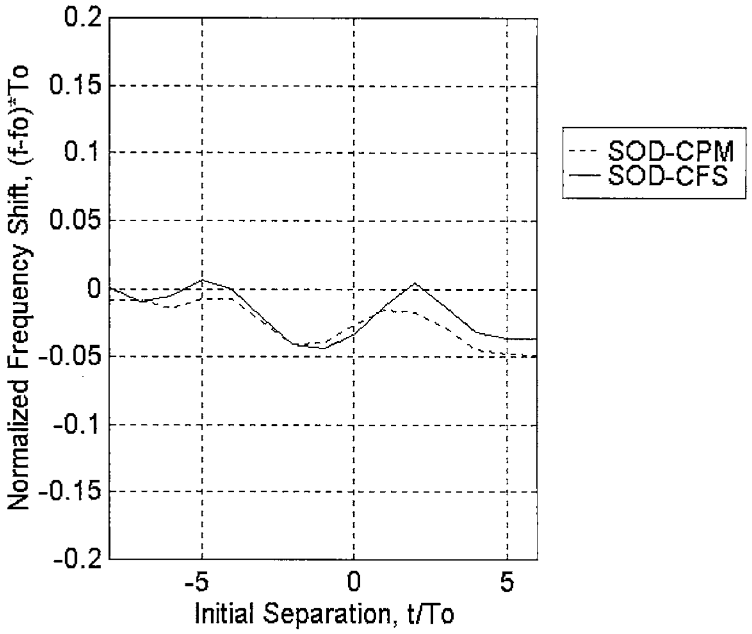


Figure 5.28.2 STG-PSI-NOLM gate output control pulse frequency shift against initial separation at $L = l_{seg}$, with $l_{seg} = l_{wo}$.

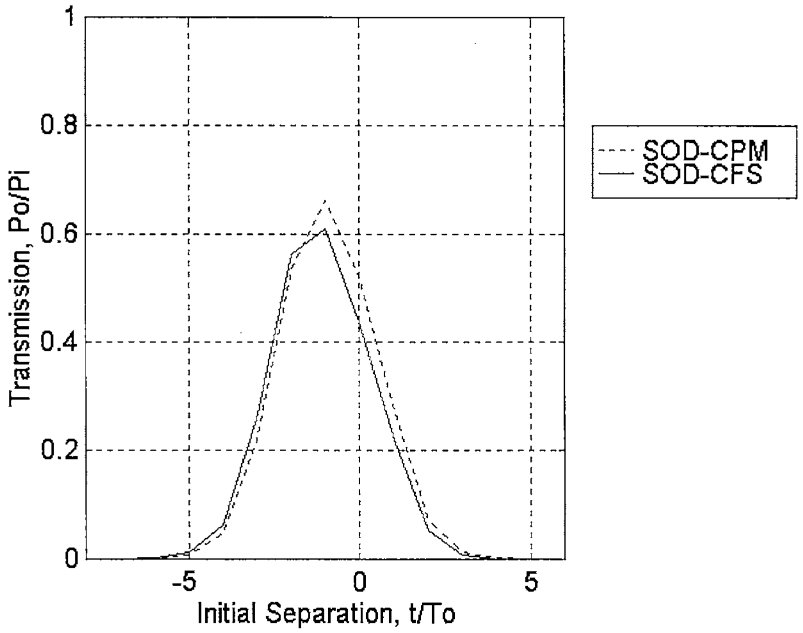


Figure 5.29.1 STG-PSI-NOLM gate output control pulse transmission against initial separation at $L = 3l_{seg}$, with $l_{seg} = l_{wo}$.

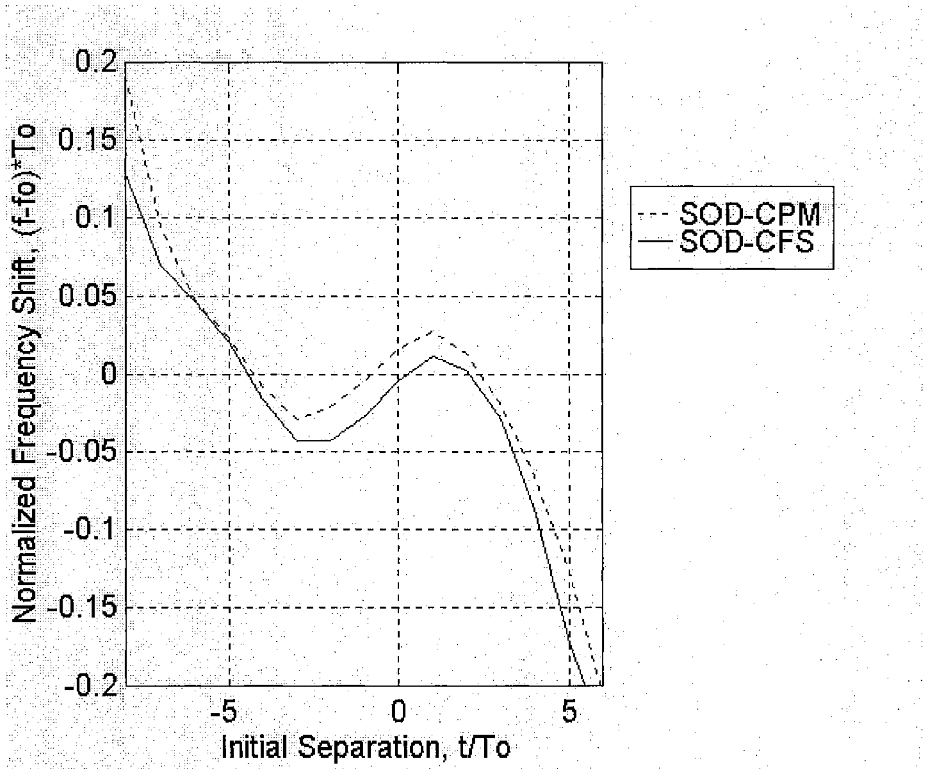


Figure 5.29.2 STG-PSI-NOLM gate output control pulse frequency shift against initial separation at $L = 3l_{seg}$, with $l_{seg} = l_{wo}$.

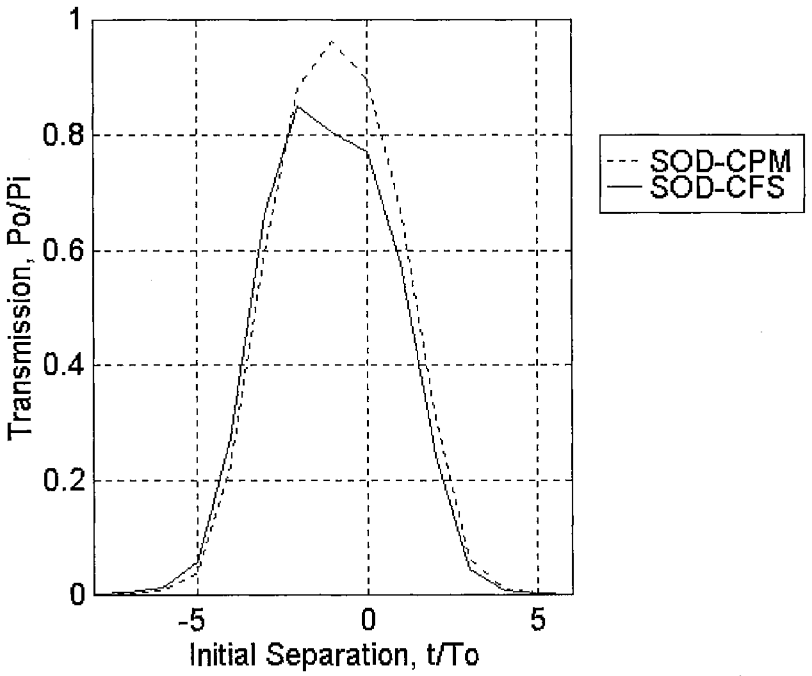


Figure 5.30.1 STG-PSI-NOLM gate output control pulse transmission against initial separation at $L = 5l_{seg}$, with $l_{seg} = l_{wo}$.

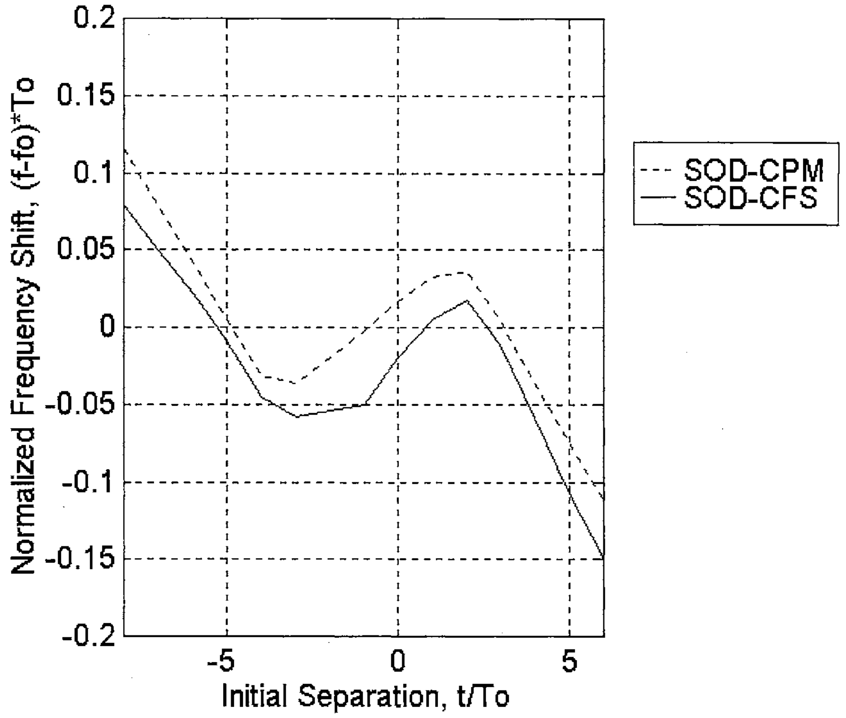


Figure 5.30.2 STG-PSI-NOLM gate output control pulse frequency shift against initial separation at $L = 5l_{seg}$, with $l_{seg} = l_{wo}$.

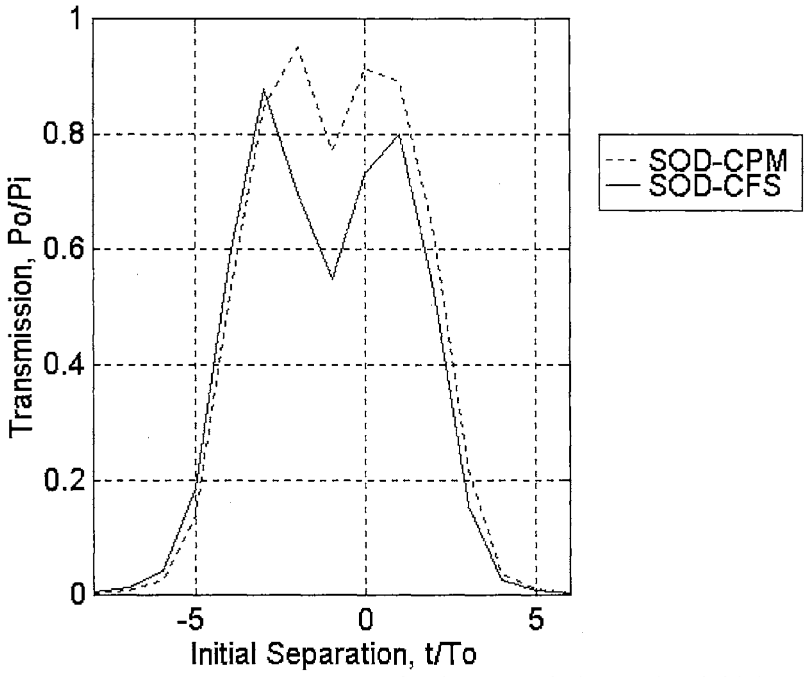


Figure 5.31.1 STG-PSI-NOLM gate output control pulse transmission against initial separation at $L = 7l_{seg}$, with $l_{seg} = l_{wo}$.

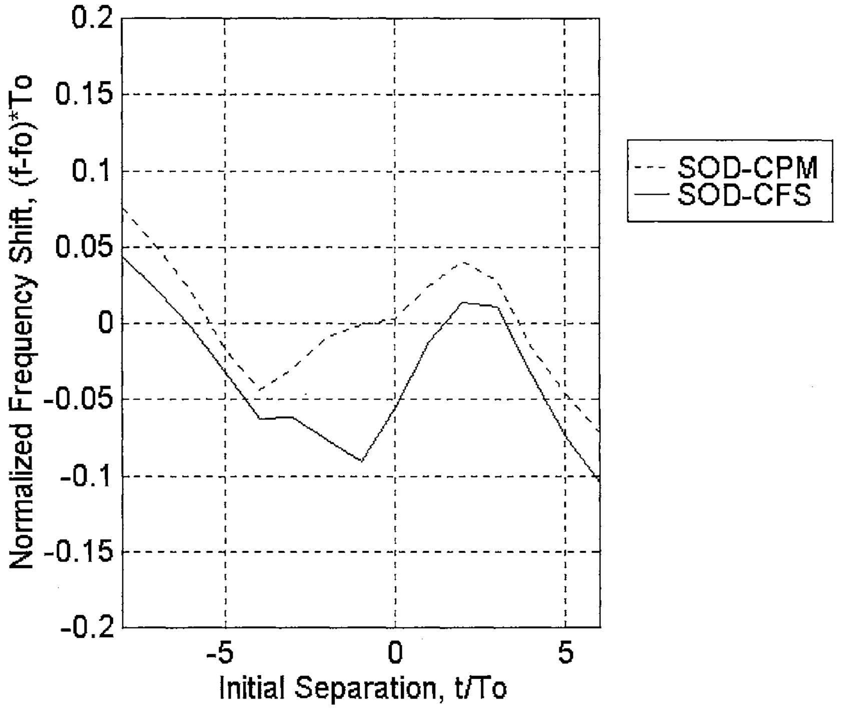


Figure 5.31.2 STG-PSI-NOLM gate output control pulse frequency shift against initial separation at $L = 7l_{seg}$, with $l_{seg} = l_{wo}$.

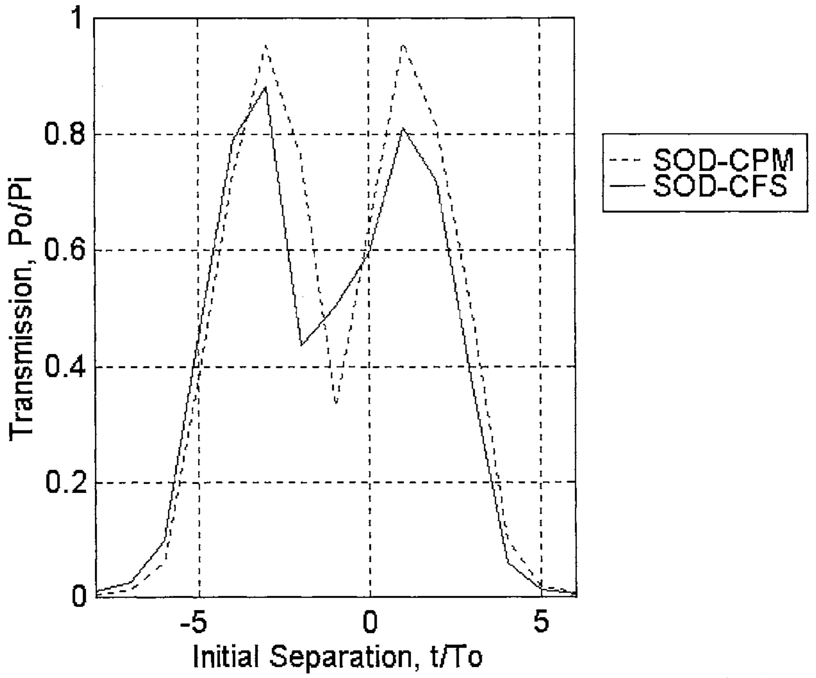


Figure 5.32.1 STG-PSI-NOLM gate output control pulse transmission against initial separation at $L = 9l_{seg}$, with $l_{seg} = l_{wo}$.

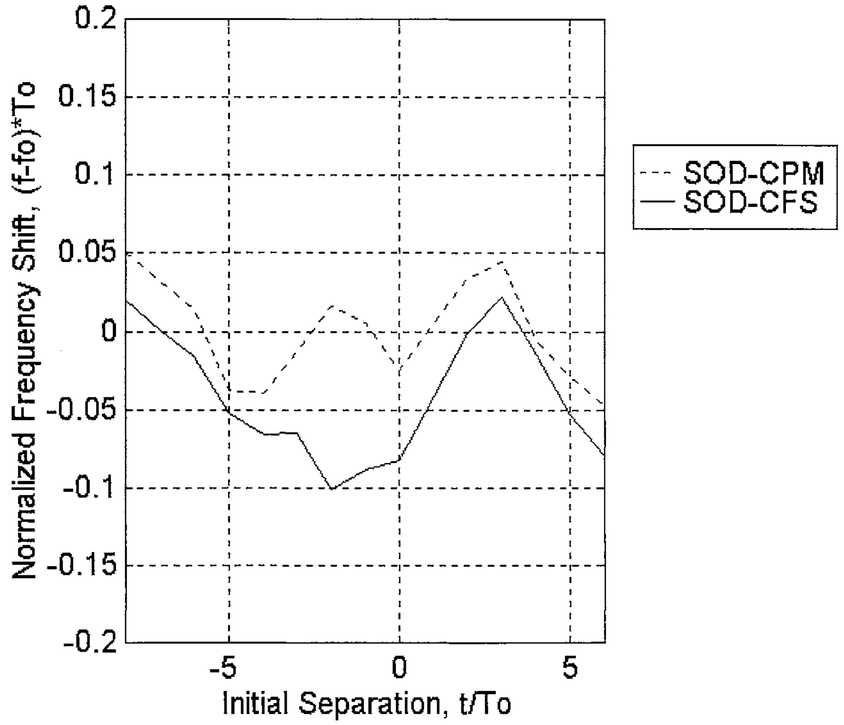


Figure 5.32.2 STG-PSI-NOLM gate output control pulse frequency shift against initial separation at $L = 9l_{seg}$, with $l_{seg} = l_{wo}$.

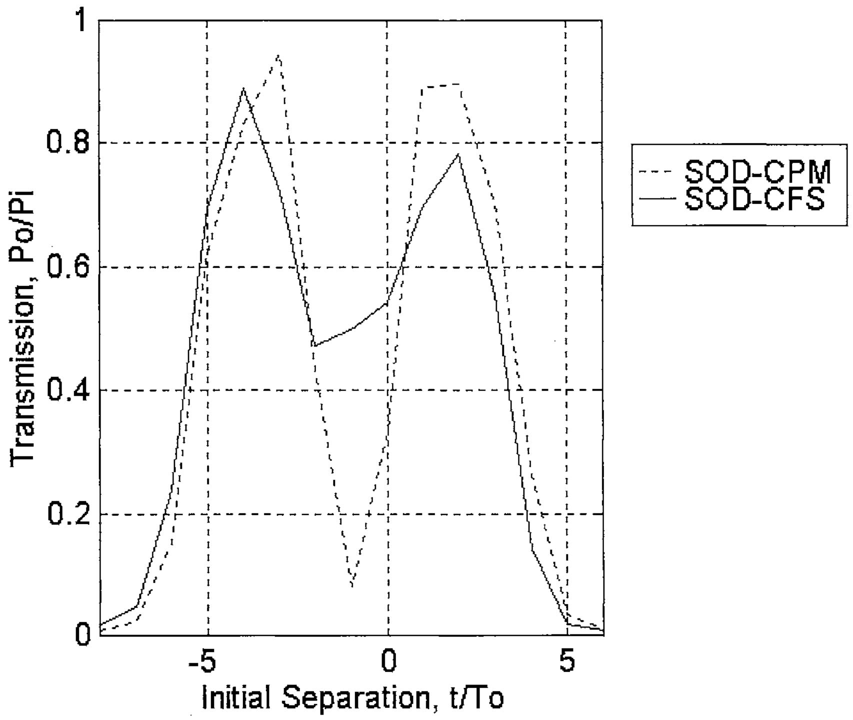


Figure 5.33.1 STG-PSI-NOLM gate output control pulse transmission against initial separation at $L = 11l_{seg}$, with $l_{seg} = l_{wo}$.

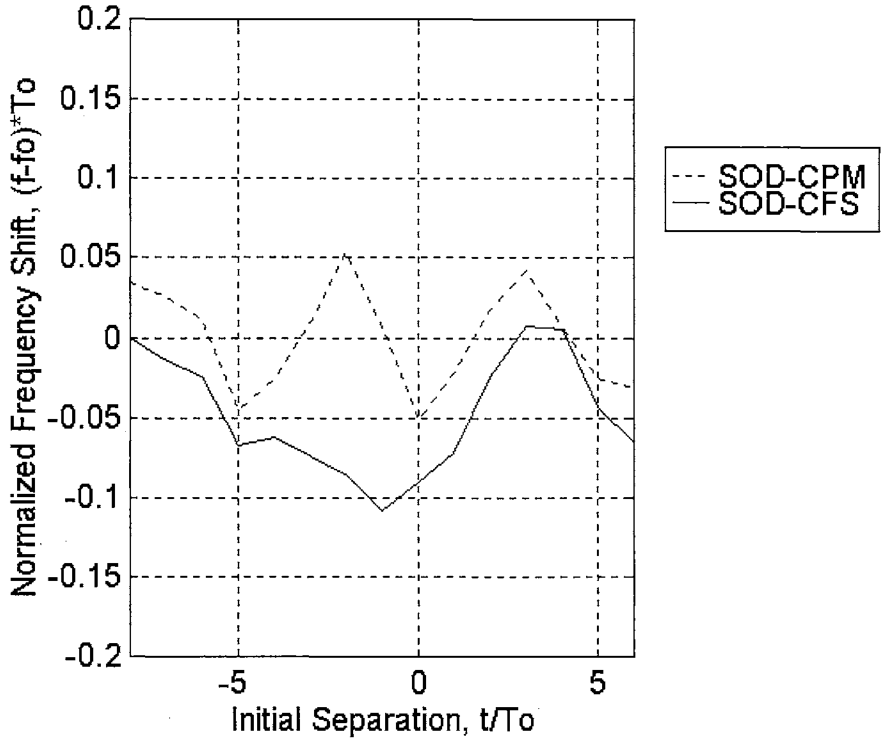


Figure 5.33.2 STG-PSI-NOLM gate output control pulse frequency shift against initial separation at $L = 11l_{seg}$, with $l_{seg} = l_{wo}$.

The results for the STG-PSI-NOLM gate with $l_{seg} = l_{wo}$, under SOD-CFS effects as observed from Figures 5.28.1 to 5.33.2, show that the peak output control pulse transmission increases and occurs within a clock time window (at half power maximum) of about $-3.6 \leq t_s \leq 1.7$ ($\approx$ 3 pulse widths), from $L = l_{seg}$ to $L = 5l_{seg}$. After $L = 5l_{seg}$ the control pulse transmission profile, under SOD-CPM and SOD-CFS effects, splits into two from $L = 7l_{seg}$ onwards, and the separation increases with the NOLM loop length L. The control pulse transmission profile splitting is due to the accumulated trapping effects, as will be evidenced in the temporal and spectral profiles of the clockwise control and signal pulses, and counter clockwise control pulse just before the interferometric switch. The SOD-CFS effects reduce the output control pulse transmission, under SOD-CPM, by

14% at $t_s = 0.0$ and $L = 5l_{seg}$ from 0.9120 to 0.7840. The output control pulse frequency shift initially increases up to $L = 3l_{seg}$, with a centroid frequency range of $-0.2 \le f_c \le 0.2$, and thereafter decreases with the NOLM loop length L. The output control pulse frequency shift under SOD-CFS effects is generally greater than that under SOD-CPM effects, across the various initial separation values.

Figures 5.34.1 to 5.34.4 give the output port control pulse temporal and spectral characteristics for the STG-PSI-NOLM gate with $l_{seg} = l_{wo}$, where the pulse splitting can be clearly seen to occur at $L = 9l_{wo}$, as opposed to $L = 3l_{wo}$ in the STG-NOLM gate. The longer NOLM loop length traversal before splitting occurs, is due to a smaller trapping effect in the first one to five walk-off lengths, in the STG-PSI-NOLM gate with $l_{seg} = l_{wo}$. The initial separation is set to $t_s = -2.0T_0$ for maximum output of the control pulse.

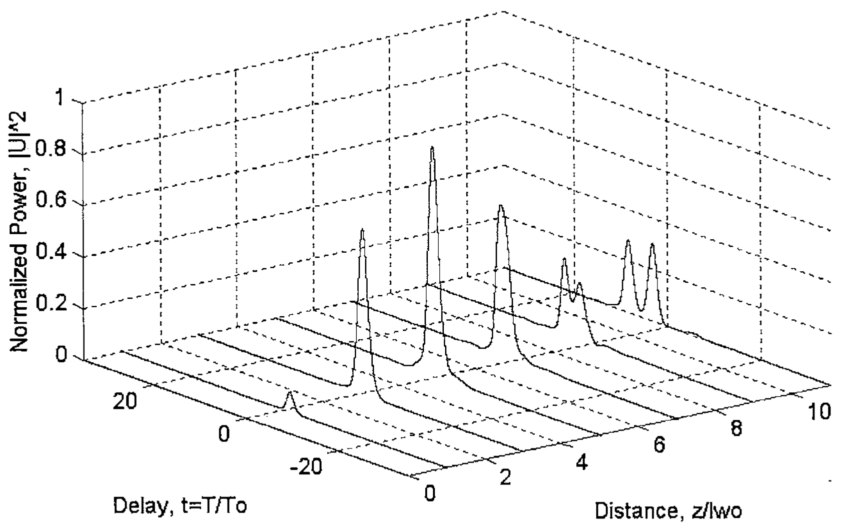


Figure 5.34.1 STG-PSI-NOLM gate under SOD-CFS control pulse time domain profile at t_s = -2.0T_0 and $l_{seg} = l_{wo}$ over $L = 11l_{seg}$.

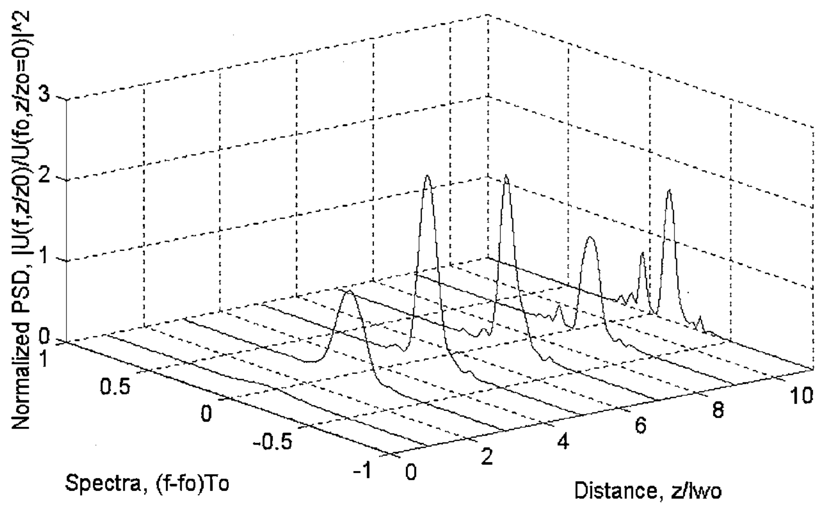


Figure 5.34.2 STG-PSI-NOLM gate under SOD-CFS control pulse frequency domain profile at $t_s = -2.0T_0$ and $l_{seg} = l_{wo}$ over $L = 11l_{seg}$.

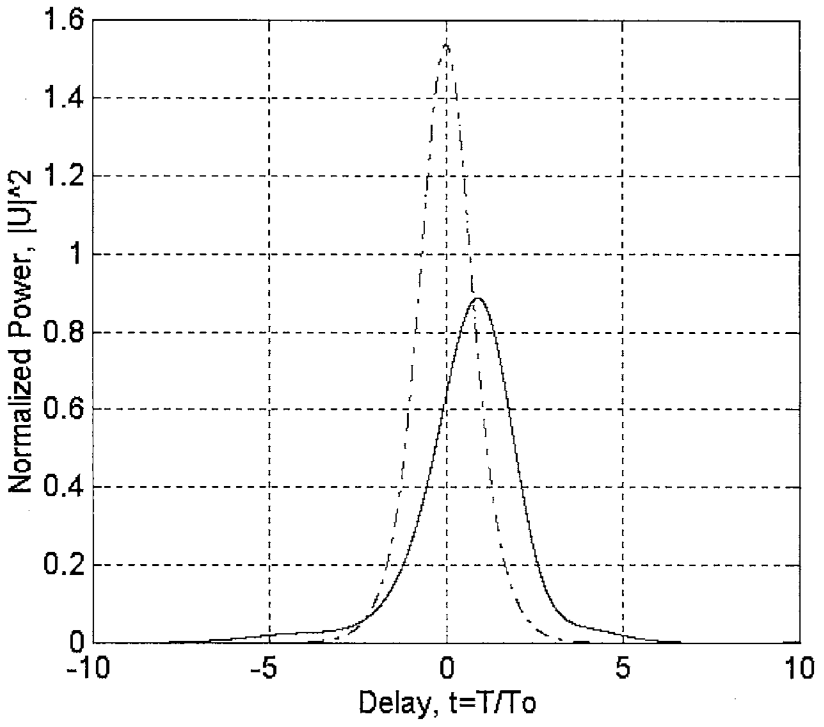


Figure 5.34.3 STG-PSI-NOLM gate under SOD-CFS control pulse time domain profile at $t_s = -2.0T_0$ and $L = 5l_{seg}$ ($l_{seg} = l_{wo}$) Dashdot and solid lines represent the input and output control pulses respectively.

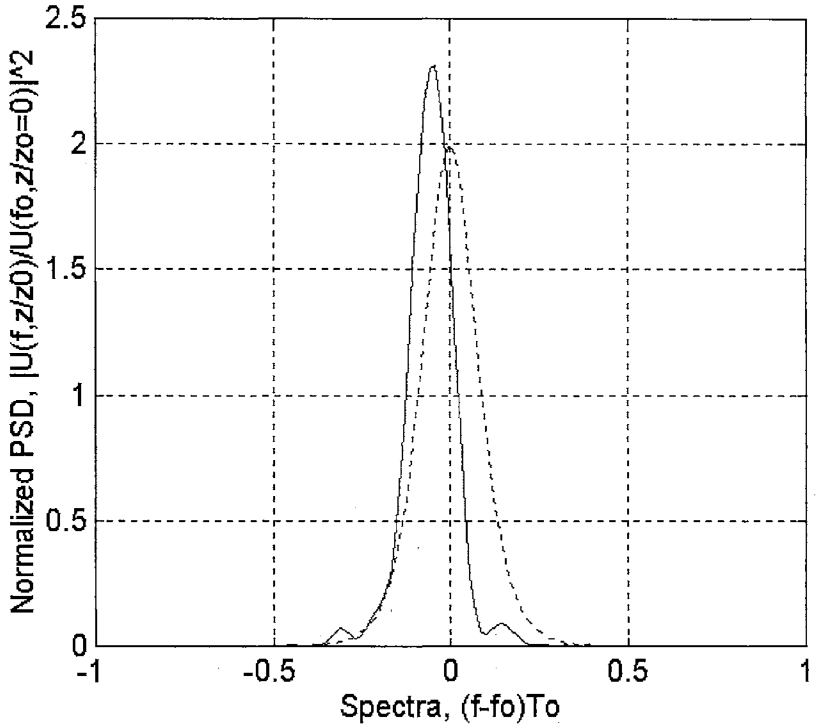


Figure 5.34.4 STG-PSI-NOLM gate under SOD-CFS control pulse frequency domain profile at $t_s = -2.0T_0$ and $L = 5l_{seg}$ ($l_{seg} = l_{wo}$). Dashdot and solid lines represent the input and output control pulses respectively.

From Figures 5.34.3 and 5.34.4, it can be seen that an output control pulse peak normalized soliton power of 0.8667 is achieved at a positive delay of $0.9574T_0$, with a normalized carrier frequency shift $(f - f_0)T_0 = -0.0426$.

In order to get an insight into why the switching performance of the STG-PSI-NOLM gate is superior to the STG-NOLM gate, one needs to observe the clockwise control and signal pulses, and counter clockwise control pulse just before the interferometric switch, in both the time and frequency domain. The clockwise control and signal pulses, and counter clockwise control pulse temporal and spectral characteristics, prior to switching, are give in Figures 5.34.5 to 5.34.10.

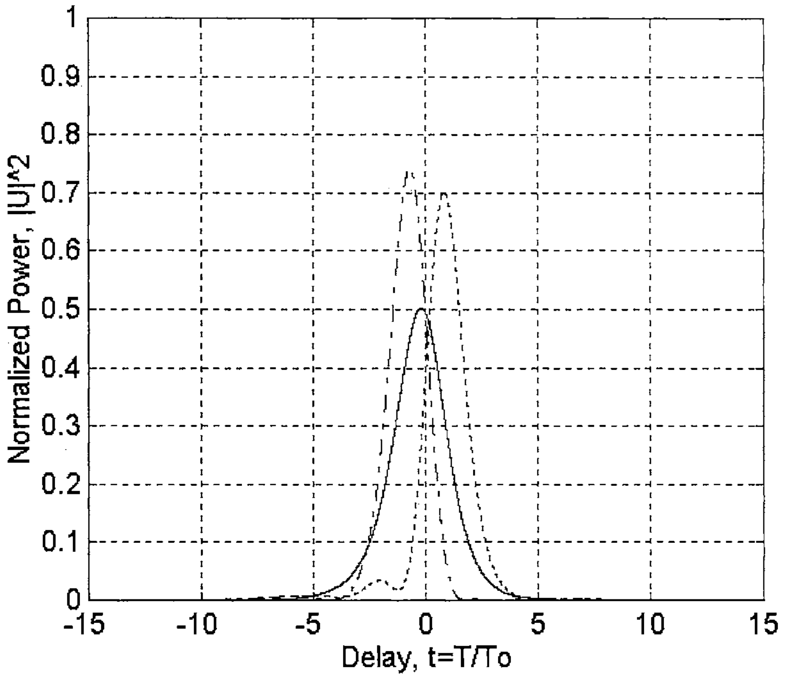


Figure 5.34.5 STG-PSI-NOLM gate under SOD-CFS pulse time domain profiles at $t_s = -2.0T_0$ and $L = 3l_{seg}$ ($l_{seg} = l_{wo}$). Dotted, dashdot, and solid lines represent the clockwise signal, clockwise control and counter-clockwise control pulses.

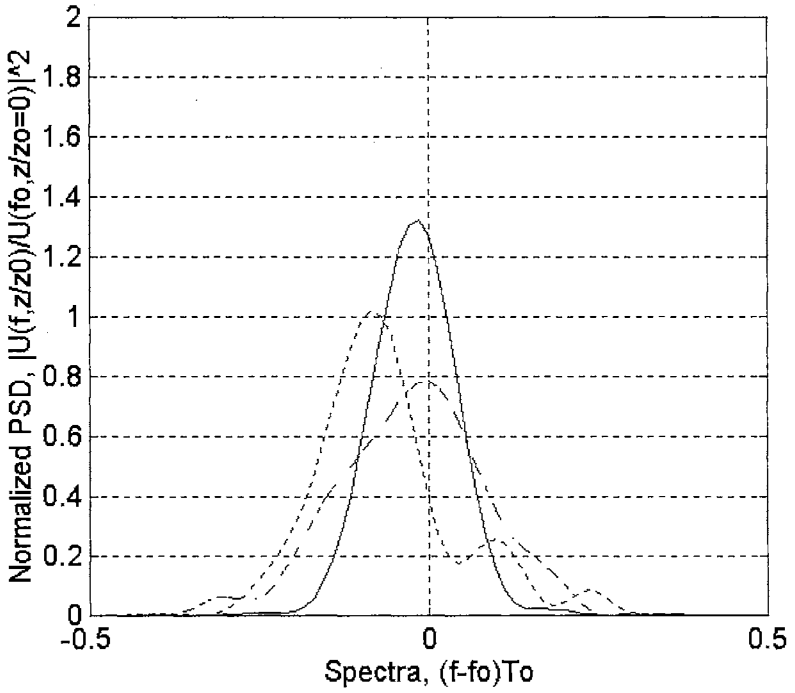


Figure 5.34.6 STG-PSI-NOLM gate under SOD-CFS pulse frequency domain profiles at $t_s = -2.0T_0$ and $L = 3l_{seg}$ ($l_{seg} = l_{wo}$). Dotted, dashdot, and solid lines represent the clockwise signal, clockwise control and counter-clockwise control pulses.

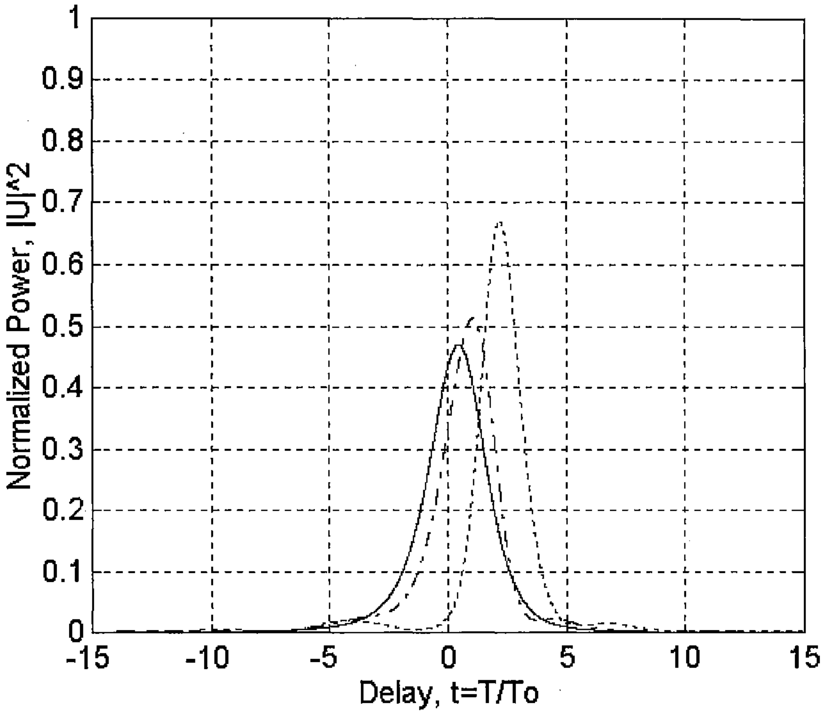


Figure 5.34.7 STG-PSI-NOLM gate under SOD-CFS pulse time domain profiles at $t_s = -2.0T_0$ and $L = 5l_{seg}$ ($l_{seg} = l_{wo}$). Dotted, dashdot, and solid lines represent the clockwise signal, clockwise control and counter-clockwise control pulses.

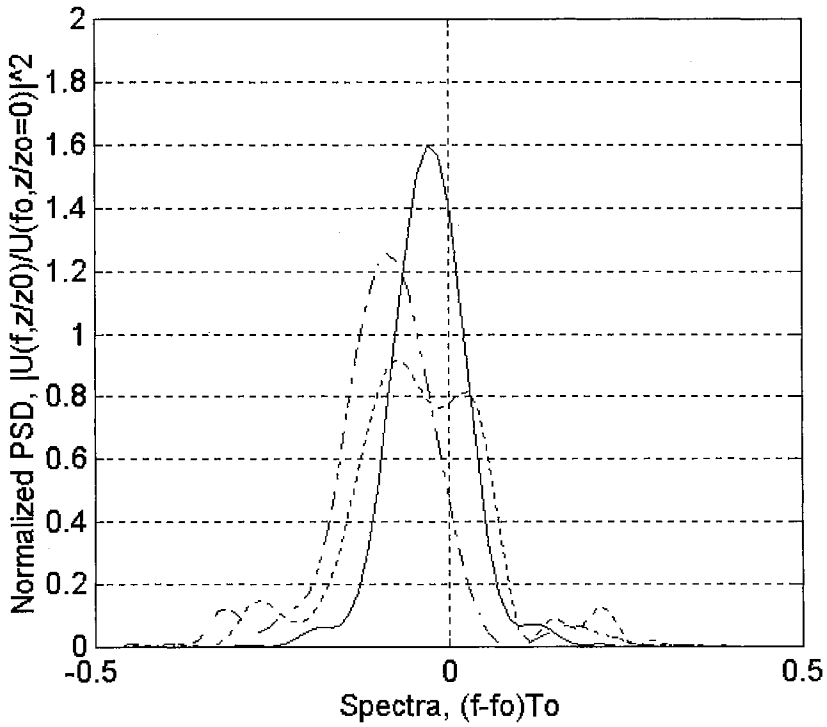


Figure 5.34.8 STG-PSI-NOLM gate under SOD-CFS pulse frequency domain profiles at $t_s = -2.0T_0$ and $L = 5l_{seg}$ ($l_{seg} = l_{wo}$). Dotted, dashdot, and solid lines represent the clockwise signal, clockwise control and counter-clockwise control pulses.

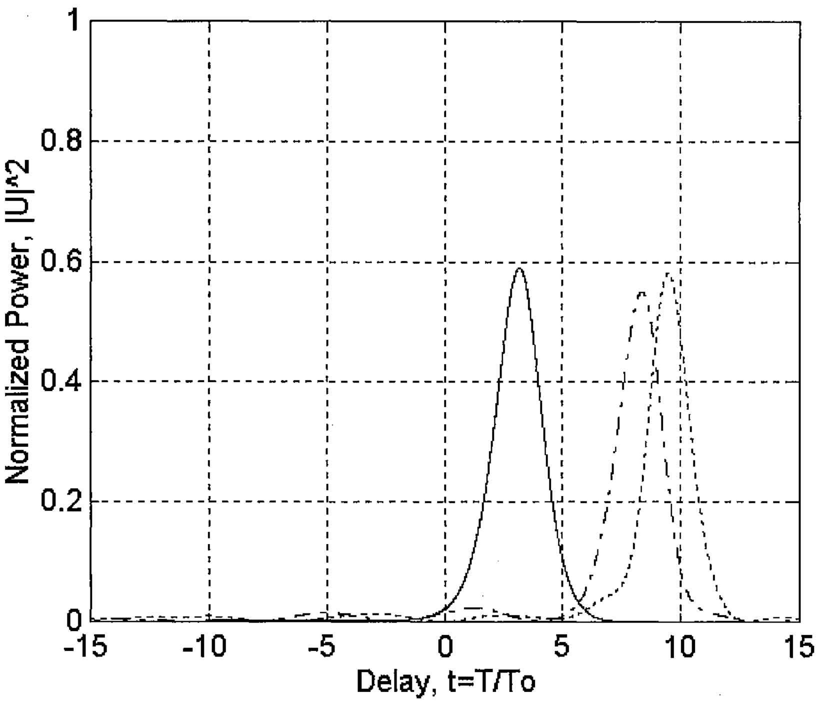


Figure 5.34.9 STG-PSI-NOLM gate under SOD-CFS pulse time domain profiles at $t_s = -2.0T_0$ and $L = 11l_{seg}$ ($l_{seg} = l_{wo}$). Dotted, dashdot, and solid lines represent the clockwise signal, clockwise control and counter-clockwise control pulses.

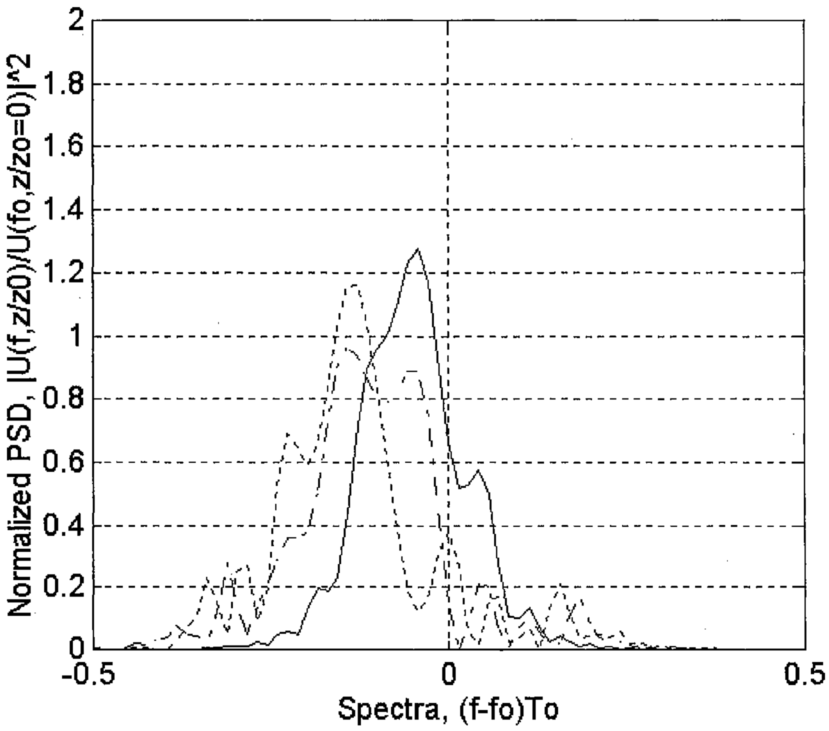


Figure 5.34.10 STG-PSI-NOLM gate under SOD-CFS pulse frequency domain profiles at $t_s = -2.0T_0$ and $L = 11l_{seg}$ ($l_{seg} = l_{wo}$). Dotted, dashdot, and solid lines represent the clockwise signal, clockwise control and counter-clockwise control pulses.

It can be observed from Figures 5.34.5 to 5.34.10 that trapping of the clockwise control pulse by the clockwise signal pulse causes it to split at $L = 9l_{seg}$. The reduced trapping effect prior to $L = 9l_{seg}$, produces a very small group velocity mismatch between the clockwise and counter clockwise control pulses, due to PSI CPM (through the cross-splices). The reduction in the trapping effect due to PSI-CPM can be evidenced by the reduced delay between the clockwise and counter clockwise control pulses, at $L = 11l_{wo}$ in Figures 5.27.9 and 5.34.9, from the STG-NOLM gate case of 14.4231T_0 to the STG-PSI-NOLM gate $\left(\text{with } l_{seg} = l_{wo}\right)$ case of 5.1923T_0 (a reduction of 64%). The clockwise control pulse acquires sufficient CPM, well before the trapping effect becomes significant and causes the output control pulse to split. With increasing NOLM loop length L, the spectral profiles of the clockwise control and signal pulses, and the counter clockwise signal pulse become severely distorted.

Clearly the STG-PSI-NOLM gate with $l_{seg} = l_{wo}$ is superior to the STG-NOLM gate, because it produces a maximum output control pulse transmission, under SOD-CFS effects, of 0.7840 with a clock time window of about 3 pulse widths at $L = 5l_{wo}$, without output control pulse breakup. This can be compared to a maximum output control pulse transmission of about 0.2 with a clock time window of about 3 pulse widths at $L = 3l_{wo}$, at the start of the control pulse breakup for the STG-NOLM gate.

5.3.3 STG-PSI-NOLM gate with Segment Length Equal to 3 Walk-off Lengths

The segment length of the STG-PSI-NOLM gate was increased such that $l_{seg} = 3l_{wo}$, and the initial separation of the clockwise control and signal pulses was set to $t_s = 0.0$, for maximum output of the control pulse. This was done to observe the effect of increasing l_{seg} from l_{wo} to $3l_{wo}$.

Figures 5.35.1 to 5.35.4 give the output port control pulse temporal and spectral characteristics for the STG-PSI-NOLM gate with $l_{seg} = 3l_{wo}$, where the pulse splitting can be clearly seen to occur at $L = 3l_{wo}$, as opposed to $L = 9l_{wo}$ in the STG-PSI-NOLM gate with $l_{seg} = l_{wo}$. The pulse split up in the STG-PSI-NOLM with $l_{seg} = 3l_{wo}$ occurs at the same NOLM loop length of $L = 3l_{wo} = l_{seg}$, as in the STG-NOLM gate. Here again, as was the case in the STG-NOLM gate, the clockwise control pulse in the STG-PSI-NOLM gate with $l_{seg} = 3l_{wo}$ is unable to acquire sufficient CPM before the output control pulse starts splitting at $L = l_{seg}$. Output control pulse splitting at $L = l_{seg}$ is due to the large trapping effect over the NOLM loop length l_{seg}, and thereafter a combination of the large trapping effect, long segment length, and the cross-splices causes the control pulses and the signal pulse in the NOLM loop to separate temporally.

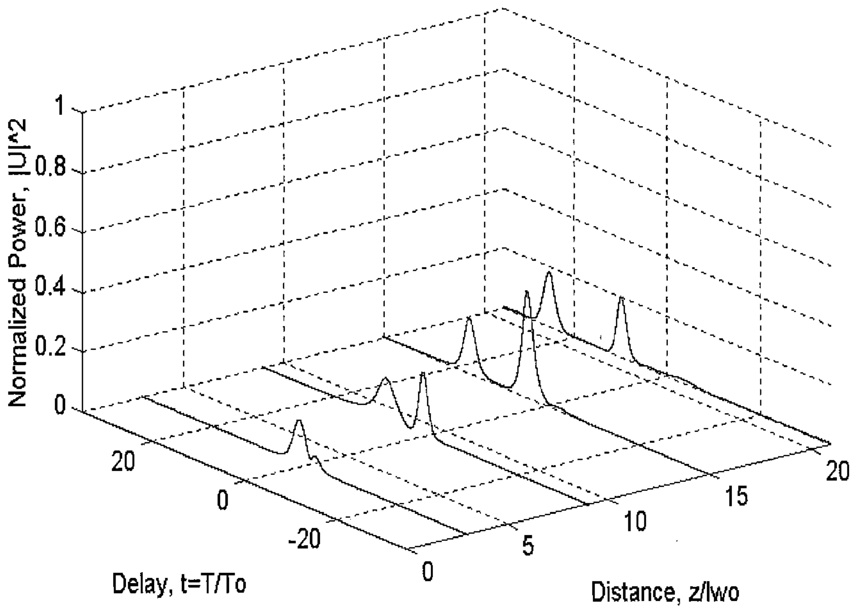


Figure 5.35.1 STG-PSI-NOLM gate under SOD-CFS control pulse time domain profile at $t_s = 0.0$ and $l_{seg} = 3l_{wo}$ over $L = 7l_{seg}$.

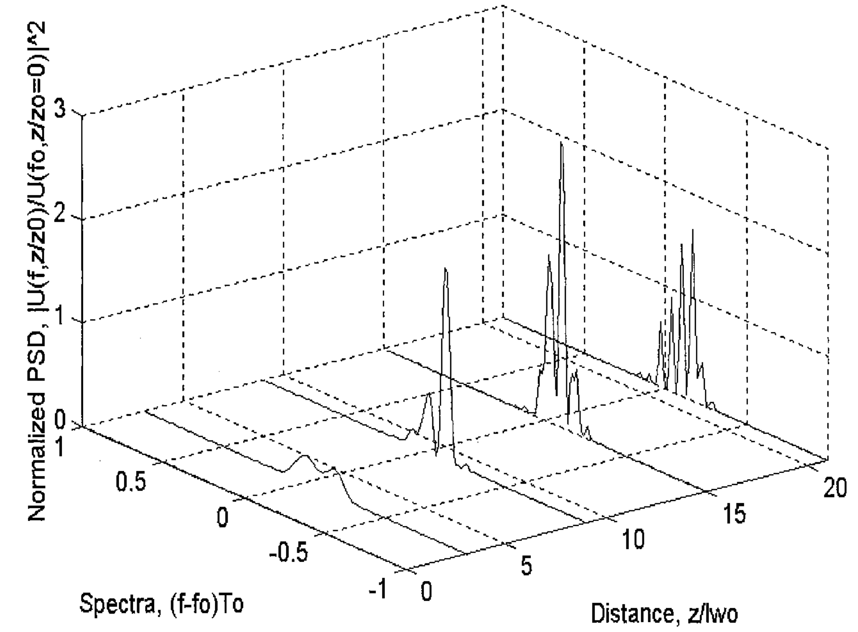


Figure 5.35.2 STG-PSI-NOLM gate under SOD-CFS control pulse frequency domain profile at $t_s = 0.0$ and $l_{seg} = 3l_{wo}$ over $L = 7l_{seg}$.

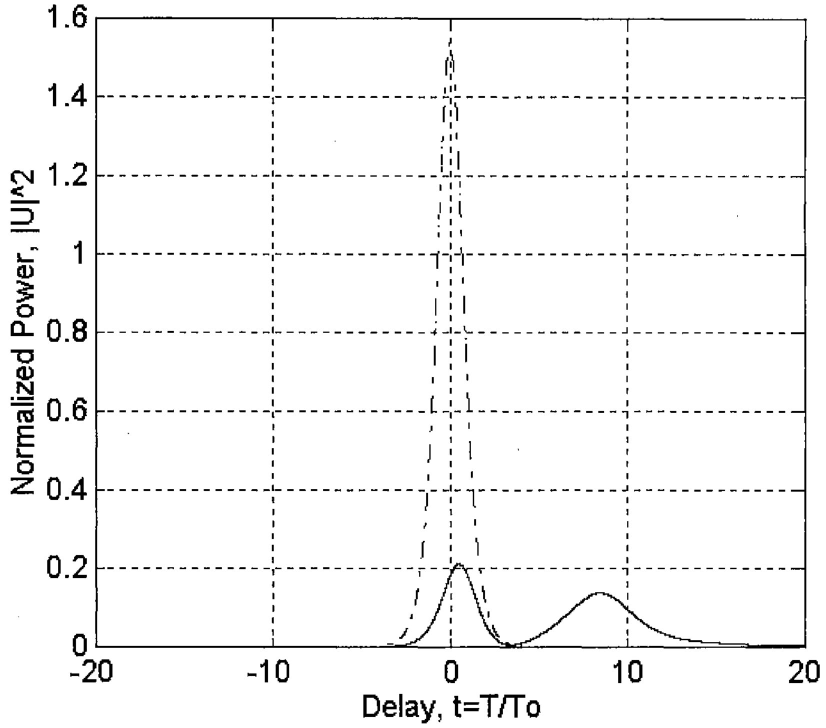


Figure 5.35.3 STG-PSI-NOLM gate under SOD-CFS control pulse time domain profile at $t_s = 0.0$ and $L = 3l_{seg}$ ($l_{seg} = 3l_{wo}$) Dashdot and solid lines represent the input and output control pulses respectively.

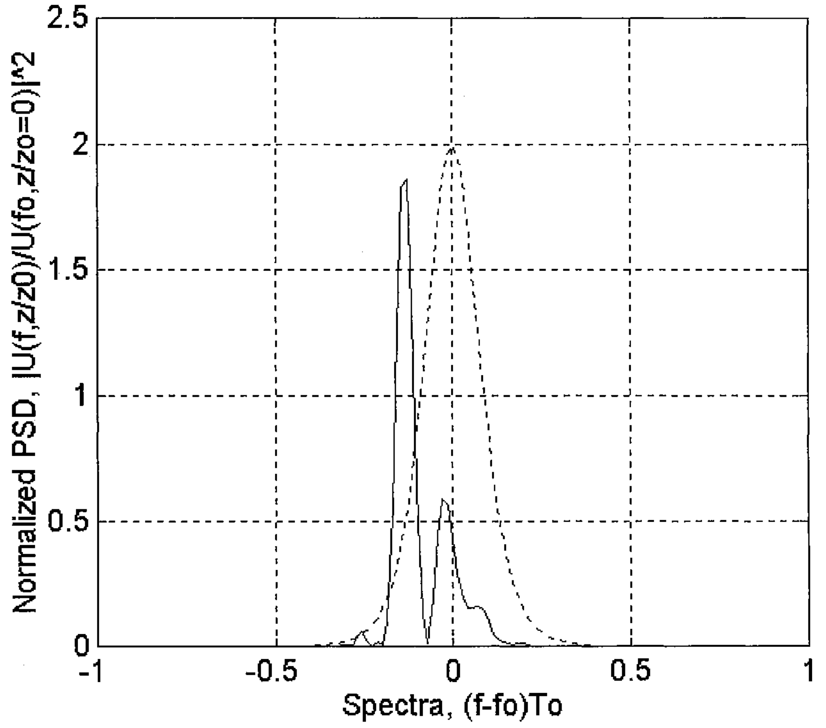


Figure 5.35.4 STG-PSI-NOLM gate under SOD-CFS control pulse frequency domain profile at $t_s = 0.0$ and $L = 3l_{seg}$ ($l_{seg} = 3l_{wo}$) Dashdot and solid lines represent the input and output control pulses respectively.

An understand of the pulse splitting in the STG-PSI-NOLM with $l_{seg} = 3l_{wo}$, can be obtained from the observation of the clockwise control and signal pulses, and counter clockwise control pulse just before the interferometric switch, in both the time and frequency domain. The clockwise control and signal pulses, and counter clockwise control pulse temporal and spectral characteristics, prior to switching, are give in Figures 5.35.5 to 5.35.10.

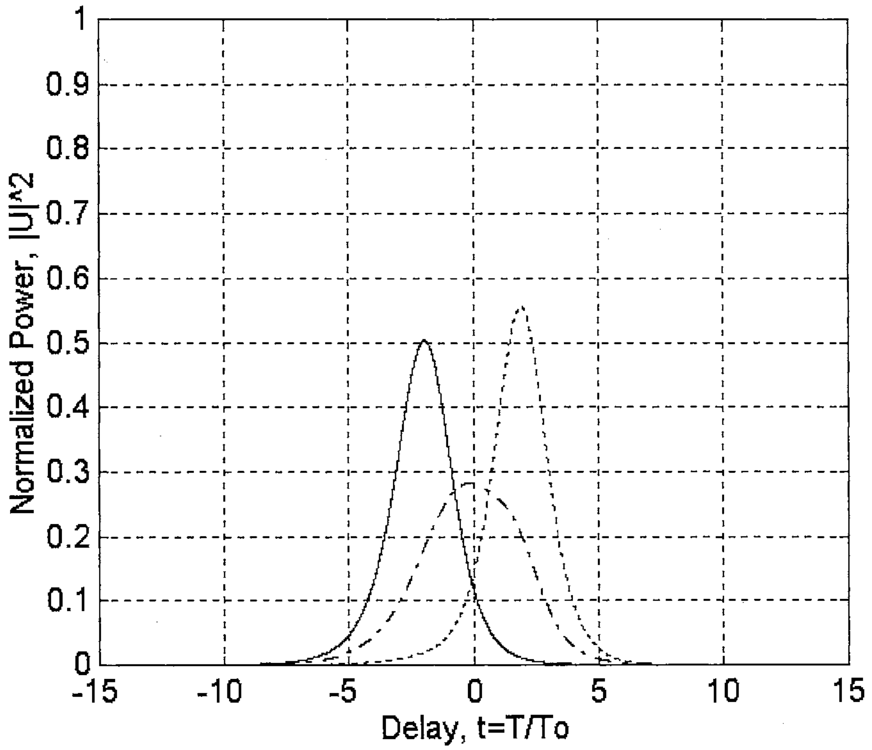


Figure 5.35.5 STG-PSI-NOLM gate under SOD-CFS pulse time domain profiles at $t_s = 0.0$ and $L = l_{seg}$ ($l_{seg} = 3l_{wo}$). Dotted, dashdot, and solid lines represent the clockwise signal, clockwise control and counter-clockwise control pulses.

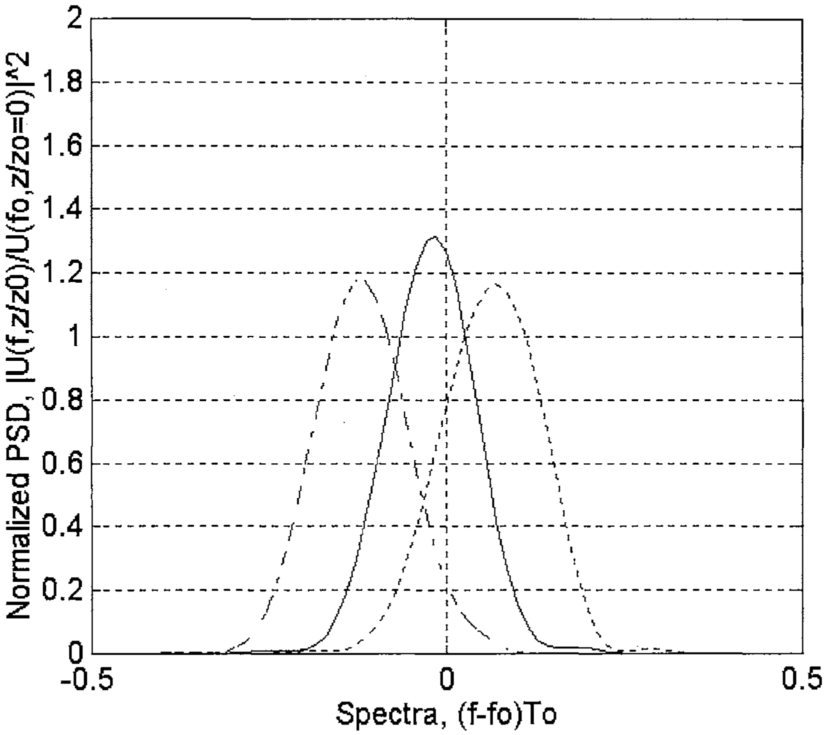


Figure 5.35.6 STG-PSI-NOLM gate under SOD-CFS pulse frequency domain profiles at $t_s = 0.0$ and $L = l_{seg}$ ($l_{seg} = 3l_{wo}$). Dotted, dashdot, and solid lines represent the clockwise signal, clockwise control and counter-clockwise control pulses.

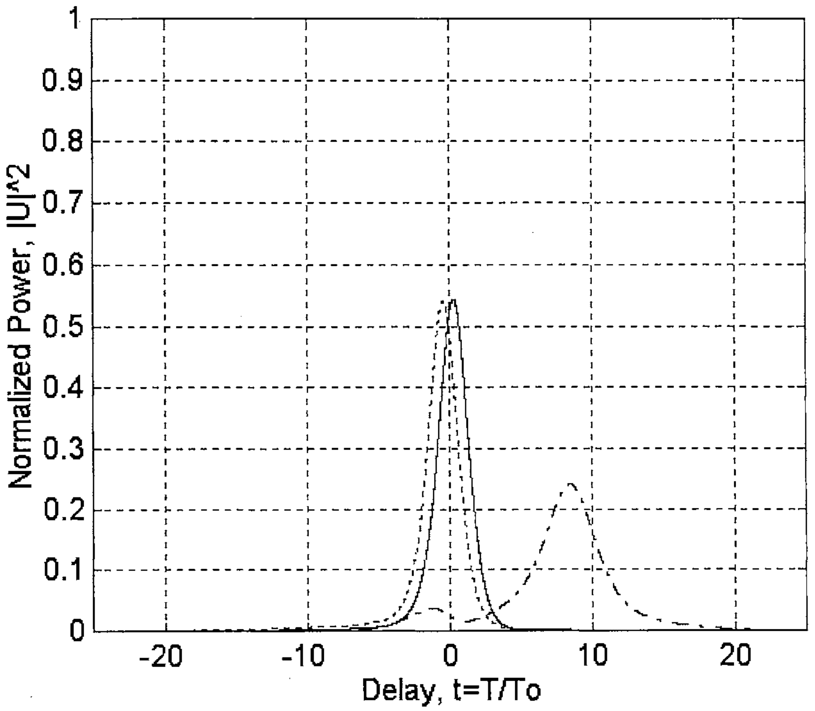


Figure 5.35.7 STG-PSI-NOLM gate under SOD-CFS pulse time domain profiles at $t_s = 0.0$ and $L = 3l_{seg}$ ($l_{seg} = 3l_{wo}$). Dotted, dashdot, and solid lines represent the clockwise signal, clockwise control and counter-clockwise control pulses.

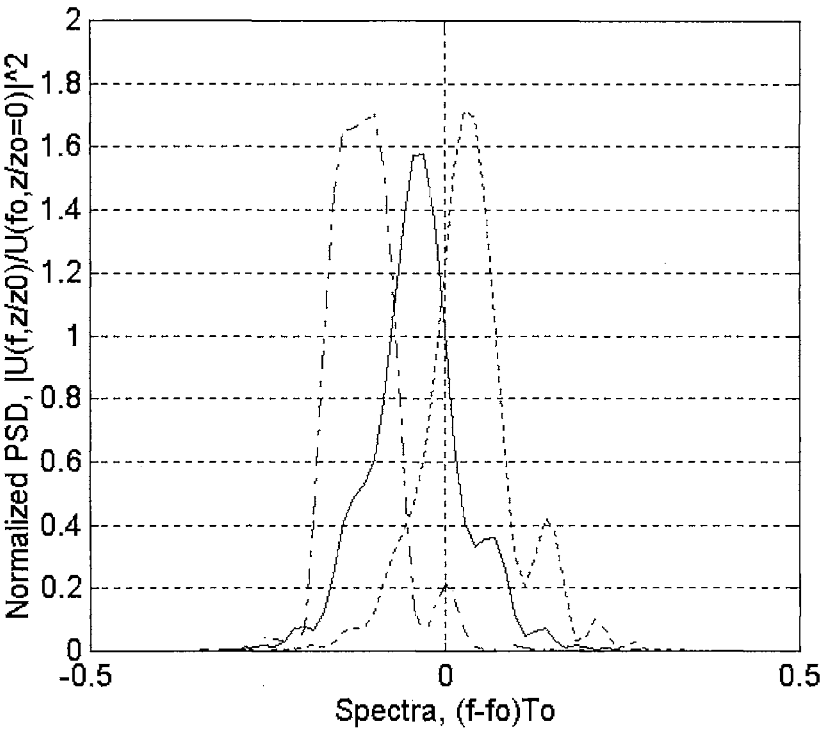


Figure 5.35.8 STG-PSI-NOLM gate under SOD-CFS pulse frequency domain profiles at $t_s = 0.0$ and $L = 3l_{seg}$ ($l_{seg} = 3l_{wo}$). Dotted, dashdot, and solid lines represent the clockwise signal, clockwise control and counter-clockwise control pulses.

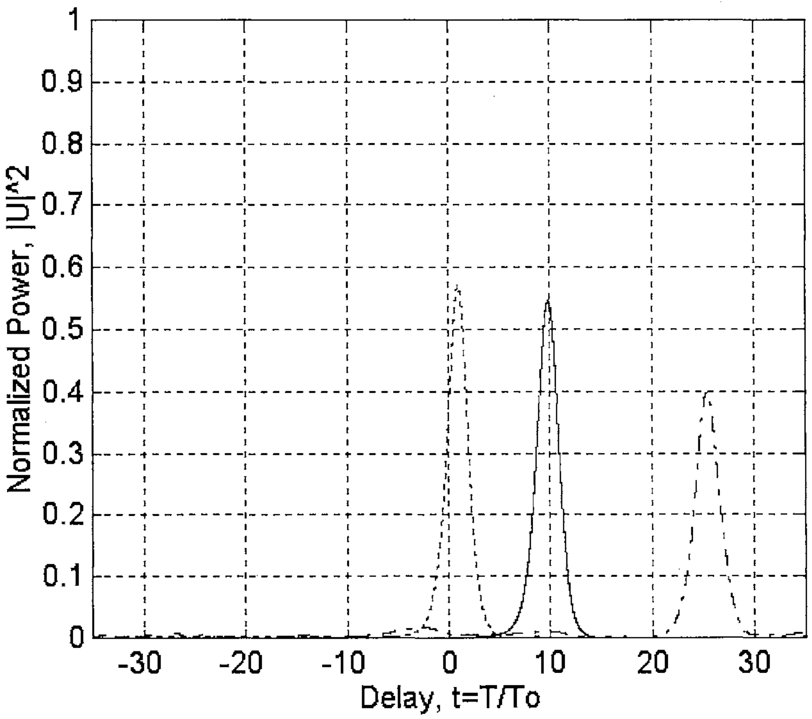


Figure 5.35.9 STG-PSI-NOLM gate under SOD-CFS pulse time domain profiles at $t_s = 0.0$ and $L = 7l_{seg}$ ($l_{seg} = 3l_{wo}$). Dotted, dashdot, and solid lines represent the clockwise signal, clockwise control and counter-clockwise control pulses.

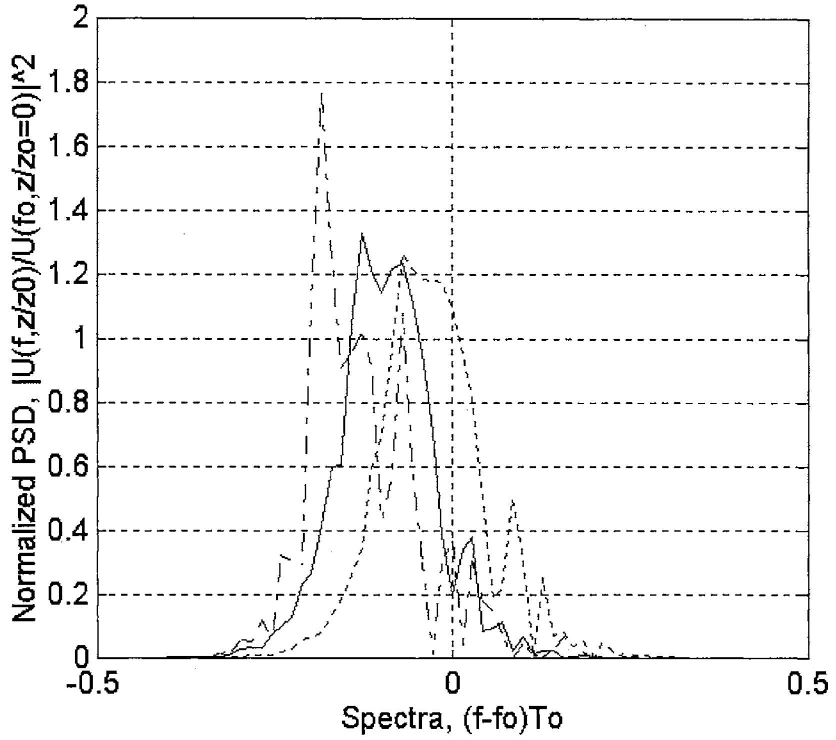


Figure 5.35.10 STG-PSI-NOLM gate under SOD-CFS pulse frequency domain profiles at $t_s = 0.0$ and $L = 7l_{seg}$ ($l_{seg} = 3l_{wo}$). Dotted, dashdot, and solid lines represent the clockwise signal, clockwise control and counter-clockwise control pulses.

Pulse splitting at $L = l_{seg}$, for the STG-PSI-NOLM with $l_{seg} = 3l_{wo}$, is due to the large trapping effect over the NOLM loop length l_{seg}, and thereafter a combination of the large trapping effect, long segment length, and the cross-splices causes the control pulses and the signal pulse to separate temporally. The theory behind this is that at each cross-splice, the trapping effects on the clockwise control and signal pulses are reversed. This means that if the control pulse was on the fast axis in the previous NOLM segment length and was being slowed down, then in the next NOLM segment length it will be on the slow axis, and will be sped up. This is the key towards the minimization of the trapping effect through PSI-CPM when $l_{seg} = l_{wo}$, to produce a minimum group velocity mismatch between the clockwise and counter clockwise control pulses. It should also be noted that all the pulses in the NOLM loop switch their group velocity, from either the fast axis group velocity to the slow axis group velocity or vice-versa, at each cross-splice. In the case of the STG-PSI-NOLM with $l_{seg} = 3l_{wo}$, the segment length is so long that the control pulses and the signal pulses eventually completely temporally separate at $L = 7l_{seg}$, and their respective spectral profiles are severely distorted.

CHAPTER VI

SUMMARY, CONCLUSIONS, AND RECOMMENDATIONS

A summary and conclusion is presented from the results of the simulation of propagation of a single soliton pulse, two orthogonally polarized soliton pulses (STG/SDG) in a singlemode fiber, and the STG-NOLM and STG-PSI-NOLM gates with soliton pulse widths of 50 fs (FWHM). Recommendations are put forward for further study of the sub-100 fs optical fiber gates.

The results obtained in this study were numerical results from simulations. It would be desirable to confirm the effects of cross steepening on the switching efficiency of STGs and SDGs and the switching performance of the STG-PSI-NOLM gate, through physical experimental work. The experimental set-up for such work is very complex to build from scratch. A reliable 50 fs soliton laser needs to be built with sufficient output power, such as the one built by Mitschke and Mollenauer [6.1].

6.1 Summary

The results from Phase I agree with those obtained from a reputable researcher, namely Agrawal [6.2]. The results, with respect to pulse delay characteristics, can be summarized as follows. The results from the propagation of a $N = 1$ single soliton in a singlemode fiber, shows that the effect of either SS and SFS on the pulse arrival time is to delay it, irrespective of the polarization axis. The pulse delay due to SFS is much larger than that due to SS. TOD, however, advances the pulse arrival time slightly. For the propagation of an $N = 2$ single soliton, the soliton splits into two solitons. One of the soliton is a red shifted large amplitude soliton, with a broad spectrum. The other soliton is a blue shifted small amplitude soliton, with a narrow spectrum. The red-shifted soliton grows in amplitude at the expense of the blue-shifted soliton, through Raman gain. Raman gain of the red-shifted soliton is through energy transfer, from the high frequency blue-shifted soliton to the low frequency red-shifted soliton.

In Phase II, the propagation results of two orthogonally polarized solitons, in a strongly birefringent singlemode fiber (STG/SDG) with effects due to SOD, SPM and CPM (SOD-CPM), agree with those of reputable researchers, such as Islam [6.3] and Menyuk [6.4, 6.5]. The pulse delay characteristics obtained from the addition of a combination of TOD, SS, SFS, CS and CFS (TOD-CFS), compared to SOD-CPM in the STG/SDG, can be summarized as follows. In the STG, the addition of TOD-SFS effects significantly delays the two orthogonally polarized solitons, irrespective of their polarization axis. The introduction of CS or CFS delays the pulses further, with the delay due to CFS being greater than that of CS. The STG switches more efficiently, with the control pulse on the fast axis, under the effects of TOD-CFS. An explanation for this is that SS-CFS effects act in the same direction as the strong trapping effect on the fast axis. It is easily seen that SS, SFS and CFS delay the pulses irrespective of the polarization axis, but not for CS. When trapping occurs, the two orthogonal pulses are locked together as a whole and essentially propagate at an average group velocity. This implies that CS on either polarization axis is in the same direction as SS, thus delaying the pulses. The trapping action due to CPM, and birefringence is such that the pulse on the fast axis is slowed down, and the pulse on the slow axis is sped up by equal amounts. Thus the delay effects due to SS-CFS on the fast axis are significantly increased by the speed up due to trapping. The opposite is true for the slow axis. For the SDGs at 30° and 60°, the addition of TOD-SFS significantly delays the two orthogonally polarized solitons, irrespective of their polarization axis. For the SDG at 30°, introduction of CS or CFS delays the pulses further, with the delay due to CFS being greater than that due to CS. The SDG operated at 30° switches less efficiently than at 60°. An explanation for this is that SS-CFS effects act in the opposite direction of the weak dragging effect on the slow axis. The dragging action is such that the pulse on the fast axis is slowed down and the pulse on the slow axis is sped up by unequal amounts, proportional to the amplitude of their co-propagating orthogonal pulse. The two orthogonal pulses are not trapped together as a whole. The trailing edge of the signal pulse on the fast axis is dragged with the slow axis control pulse, and its leading edge speeds up and broadens with distance. CS of the control pulse from the dragged signal pulse trailing edge (slowed down due to dragging effect), on the fast axis, delays it slightly. As before SS, SFS and CFS delay the control pulse on the slow axis. The speed up of the control pulse, due to dragging, is significantly reduced due to the effects of SS-CFS. This diminishes the clock time window for time-shift keying logic. For the SDG at 60°, further introduction of CS or CFS advances or delays the pulses respectively. The SDG operated at 60° switches more efficiently than at 30°. An explanation for this is that SS-CFS effects act in the same direction as the weak dragging effect on the fast axis. Here the leading edge of the signal pulse on the slow axis is dragged with the fast axis control pulse, and its trailing edge slows down and broadens with distance. CS of the control pulse from the dragged signal pulse leading edge (sped up due to dragging effect), on the slow axis, speeds it up slightly. As before SS, SFS and CFS delay the control pulse on the fast axis. The delay of the control pulse due to dragging, SS, SFS, and CFS is greater than the speed up due to CS. This gives a larger clock time window for time-shift keying logic for the SDG operated at 60° than at 30°.

In phase III, the switching performance of the STG-NOLM gate, under SOD-CFS, was very poor with an output control pulse peak transmission of 0.2, within a clock time window of about 3 pulse widths, at an NOLM loop length of 3 walk-off lengths. The STG-PSI-NOLM gate (with segment length equal to walk-off length) proved to have a superior switching performance than the STG-NOLM gate. The STG-PSI-NOLM gate had an output control pulse peak transmission of 0.7840, within a clock time window of about 3 pulse widths, at an NOLM loop length of 5 walk-off lengths, under SOD-CFS effects. The switching performance of the STG-PSI-NOLM gate, however, degrades as the segment length is increased, as was seen in the case when the segment length was equal to 3 walk-off lengths. The poor switching performance in the STG-NOLM gate was due to the trapping effect. In the STG-PSI-NOLM gate (with segment length equal to walk-off length), the trapping effect is minimized over one to five walk-off lengths by PSI CPM, through the use of cross-splices and each segment length equal to the walk-off length. However, when the segment length is increased sufficiently (for instance when the segment length equals to 3 walk-off lengths), the trapping effect, and the cross-splices eliminate PSI CPM, and the NOLM loop pulses separate temporally. The temporal separation of the clockwise pulse, and counter clockwise control pulse leads to a very poor switching performance. The higher order nonlinear and dispersion effects due to TOD-CFS slightly attenuates the transmission of the output control pulse.

For the STG and the SDG, with the control on the slow axis, the switching maximum clock time windows are reduced by 32% and 62% respectively, due to the self and cross Raman effects. However, for the STG and SDG operated with the control on the fast axis, it was found that the switching maximum clock time windows are increased by 30% and 28% respectively, due to the self and cross Raman effects. It is essential to design for a maximum clock time window at the normalized birefringence of about 0.5 for both gates. The STG with the control on the fast axis is better than SDGs for sub-100 fs operation. The STG-PSI-NOLM gate (with segment length equal to walk-off length) also has good switching characteristics than the STG-NOLM, for sub-100 fs operation.

The cross Raman effects have been experimentally observed, with the effects due to CFS accounted for in numerical solutions [6.6, 6.7]. However, in these numerical solutions, the effects of CS have been excluded. Experimental verification of the delay characteristics of CS needs to be conducted to verify the numerical results. The STG-PSI-NOLM gate has not to date been physically built, therefore there is need to experimentally confirm the numerical performance of this new type of optical fiber gate.

6.2 Conclusions

It is clear from the summary that the inverter STG, with the control pulse on the fast axis, is much more immune to the delay effects introduced by the combined effects of TOD, SS, SFS, CS and CFS (TOD-CFS). The STG is still capable of switching efficiently under these delaying effects. The inverter SDG, however, is less immune to these effects, especially when the control pulse is on the slow axis (at angle 30°). The time clock window needs only to be slightly reduced for the SDG, when the control pulse is on the fast axis (at angle 60°). Inference from these results indicates that a STG is better than a SDG, when operated under the effects of TOD-CFS. The switch performance of the STG-PSI-NOLM gate (with segment length equal to walk-off length) is very superior to the STG-NOLM gate, which has very poor switching characteristics, and the output control pulse transmission of both gates are only slightly attenuated by the effects of TOD-CFS.

6.3 Recommendations

Using numerical simulations, other types of fiber based soliton optical gates can be studied under the effects of TOD, SS, SFS, CS and CFS. The experimental verification of the CS delay characteristics, and the PSI CPM effect in the STG-PSI-NOLM (with segment length equal to walk-off length) needs to be conducted. This means building up an experimental set-up for generating sub-100 fs solitons, and implementing the STG, SDG, and STG-PSI-NOLM gate.

REFERENCES

CHAPTER I

(1) Mollenauer L.F., Gordon J.P. and Evangelides S.G., "Multigigabit Soliton Transmission Traverse Ultralong Distances", Laser Focus World, 159, (November 1991)

(2) Islam M.N., *Ultrafast Fiber Switching Devices and Systems*, 1st ed. (Cambridge University Press, 1992).

(3) Nussbaum E., "Communication Network Needs and Technologies - A Place for Photonic Switching?", IEEE Journal on Selected Areas in Communications **6**, 1036 (1988).

(4) MultiG/T Workshop Proceedings 1-5, KTH/SICS, 1990-1992.

(5) RACE Common Functional Specifications, Document 5: IBC Access, CFS E210-E500. Published by RACE R1045 Project Consensus Management.

(6) Smith P.W., "On the Physical Limits of Digital Optical Switching and Logic Elements", Bell System Tech. J. **61**, 1975 (1982).

(7) Mollenauer L.F., Gordon J.P. and Islam M.N., "Soliton Propagation in Long Fibers with Periodically Compensated Loss", IEEE J. of Quantum Electron. **QE-22**, 157(1986).

(8) Mollenauer L.F., Evangelides S.G. and Haus H.A., "Long-Distance Soliton Propagation Using Lumped Amplifiers and Dispersion Shifted Fiber", IEEE Journal of Lightwave Technology **9**, 194 (1991).

(9) Hasegawa A. and Kodama Y., "Signal Transmission by Optical Solitons in Monomode Fiber", Proc. of the IEEE **69**, 1145 (1981).

(10) Menyuk C.R., Islam M.N., and Gordon J.P., “Raman Effect in Birefringent Optical Fibers”, Opt. Lett. **16**, 566 (1991).

(11) Bloembergen N., *Nonlinear Optics*, (Benjamin, Reading, MA, 1977) Chapter 1.

(12) Buerger M.J., Elementary Crystallography, (Wiley, New York, 1963).

(13) Alfano R.R., Li Q.X., Jimbo T., Manassah J.T., and Ho P.P, Opt. Lett. **14**, 626 (1986).

(14) Alfano R.R., Baldeck P.L., Raccah F., and Ho P.P., Appl. Opt. **26**, 3491 (1987).

(15) Islam M.N., Mollenauer L.F., Stolen R.H., Simpson J.R., and Shang H.T., Opt. Lett. **12**, 625 (1987).

(16) Agrawal G.P., Phys. Rev. Lett. **59**, 880 (1987).

(17) Schadt D., and Jaskorzynska B., J. Opt. Soc. Am. B **4**, 856 1987).

(18) Agrawal G.P., *Nonlinear Fiber Optics*, 1st ed. (Academic Press, 1989).

(19) Marcuse D., *Light Transmission Optics*, (van Nostrand Reinhold, New York, 1982), Chapter 8 and 12.

(20) Bourkoff E., Zhao W., Joseph R.I., and Christoulides D.N, Opt. Lett. **12**, 272 (1987).

(21) Golovchenko E.A., Dianov E.M., Pilipetskii A.N., Prokhorov A.M., and Serkin V.N., JETP Lett. **45**, 91 (1987).

(22) DeMartini F., Townes C.H., Gustafson T.K., and Kelley P.L., Phys. Rev. **164**, 312 (1967).

(23) Grischkowsky D., Courtens E., and Armstrong J.A., Phys. Rev. Lett. **31**, 422 (1973).
(24) Tzoar N., and Jain M., Phys. Rev. A **23**, 1266 (1981).
(25) Anderson D., and Lisak M., Rev. A **27**, 1393 (1983).
(26) Yang G., and Shen Y.R., Opt. Lett. **9**, 510 (1984).
(27) Manassah J.T., Mustafa M.A., Alfano R.A., and Ho P.P., IEEE J. Quantum Electron. **QE-22**, 197 (1986).
(28) Golovchenko E.A., Dianov E.M., Prokhorov A.M. and Serkin V.N, Sov. Phys. Dokl. **31**, 494 (1986).
(29) Ohkuma K., Ichikawa Y.H., and Abe Y., Opt. Lett. **22**, 516 (1987).
(30) Kodama Y., and Nozaki K., Opt. Lett. **12**, 1038 (1987).
(31) Mitschke F.M., and Mollenauer L.F., Opt. Lett. **11**, 659 (1986).
(32) Gordon J.P., Opt. Lett. **11**, 662 (1986).
(33) Ostrovski L.A., Sov. Phys. JETP **24**, 797 (1967).
(34) Jonek R.J., and Landauer R., Phys. Lett. **24A**, 228 (1967).
(35) Shimizu F.., Phys. Rev. Lett. **19**, 1097 (1967).
(36) Gustafson T.K., Taran J.P., Haus H.A., Lifsitz J.R., and Kelly P.L., Phys. Rev. **177**, 306 (1969).
(37) Cubeddu R., Polloni R., Sacchi C.A., and Svelto O., Phys. Rev. A. **2**, 1955 (1970).
(38) Alfano R.R., and Shapiro S.L., Phys. Rev. Lett. **24**, 592 (1970); **24**, 1217 (1970).
(39) Shen Y.R., and Loy M.M.T., Phys. Rev. A. **3**, 2099 (1971).
(40) Kodama Y., and Hasegawa A., IEEE J. Quantum Electron. **QE-23**, 510 (1987).
(41) Taniuti T., "Reductive Perturbation Method and Far Fields of Wave Equations", Prog. Theor. Phys. (Japan), Suppl. **55**, 1 (1974).
(42) Hasegawa A., and Kodama Y., *Solitons in Optical Communications*, 1st ed. (Oxford University Press, 1995), Chapter 3.
(43) Islam M.N., Menyuk C.R., Chen C.-J, and Soccolich C.E., Opt. Lett. **16**, 214 (1991).
(44) Menyuk C.R., Opt. Lett. **12**, 614 (1987).
(45) Menyuk C.R., J. Opt. Soc. Am. B. **5**, 392 (1988).
(46) Duguay M.A., and Hansen J.W., Appl. Phys. Lett. **15**, 192 (1969).
(47) Stolen R.H., Botineau J., and Ashkin A., Opt. Lett. **7**, 512 (1982); Stolen R.H., and Ashkin A., Appl. Phys. Lett. **22**, 294 (1973).
(48) Halas N.J., Krokel D., and Grischkowsky D., Appl. Phys. Lett. **50**, 886 (1987).
(49) Morioka T., Saruwatari M., and Takeda A., Electron Lett. **23**, 453 (1987); Morioka T., and Saruwatari M., IEEE J. Select. Areas Commun. **6**, 1186 (1988).
(50) Islam M.N., Soccolich C.E., Gordon J.P., and Paek U.C., Opt. Lett. **15**, 21 (1990).
(51) Andrekson P.A., Olsson N.A., Simpson J.R., Tanbun-Ek T., Logan R.A., and Haner M., Electron. Lett. **27**, 922 (1991).
(52) Islam M.N., Dijaili S.P., and Gordon J.P., Opt. Lett. **13**, 518 (1988).
(53) Soccolich C.E., and Islam M.N., Opt. Lett. **14**, 645 (1989).
(54) Friberg S.R., Weiner A.M., Silberberg Y., Sfez G., and Smith P.W., Opt. Lett. **13**, 904 (1988).
(55) Trillo S., Wabnitz S., Stolen R.H., Assanto G., Seaton C.T., and Stegeman G.I., Appl. Phys. Lett. **49**, 1224 (1986).
(56) Trillo S., Wabnitz S., Finlayson N., Banyai W.C., Seaton C.T., and Stegeman G.I., Appl. Phys. Lett. **53**, 837 (1988).
(57) Doran N.J., and Wood D., Opt. Lett. **13**, 56 (1988).
(58) Blow K.J., Doran N.J., and Nayar B.K., Opt. Lett. **14**, 754 (1989).
(59) Islam M.N., Sunderman E.R., Stolen R.H., Pleibel W., and Simpson J.R., Opt. Lett. **14**, 811 (1989).
(60) Blow K.J., Doran N.J., Nayar B.K., and Nelson B.P., Opt. Lett. **15**, 248 (1990).
(61) Moores, Bergman K., Haus H.A., and Ippen E.P., Opt. Lett. **16**, 138 (1991).
(62) Avramopoulos H., French P.M.W., Gabriel M.C., and Whitaker N.A., IEEE Phot. Tech. Lett. **3**, 235 (1991).
(63) Mitschke F.M., and Mollenauer L.F., Opt. Lett. **11**, 659 (1986).
(64) Pearson G.W., Zanoni R., and Krasinski J.S., Optics Comm. **103**, 507 (1993).
(65) Lee H.K., Kim K.H., Park S.Y., and Lee E., IEEE Phot. Tech. Lett. **7**, 1441 (1995).
(66) Islam M.N., Opt. Lett. **15**, 417 (1990).
(67) Islam M.N., Menyuk C.R., Chen C.-J., and Soccolich C.E., Opt. Lett. **16**, 214 (1991).
(68) Islam M.N., Soccolich C.E., and Miller D.A.B., Opt. Lett. **15**, 909 (1990).

(69) Islam M.N., Soccolich C.E., Chen C.-J., Kim K.S., Simpson J.R., and Paek U.C., Electron. Lett. **27**, 130 (1991).
(70) Menyuk C.R., IEEE J. Quantum Electron. **QE-23**, 174 (1987).
(71) Chen C.-J., Menyuk C.R., Islam M.N., and Stolen R.H., Opt. Lett. **16**, 1647 (1991).
(72) Headly III C., and Agrawal G.P., J. Opt. Soc. Am. B. 13, 2170 (1996).
(73) Ahn K.H, Vaziri M., Barnett B.C., Williams G.R., Cao X.D., Islam M.N., Malo B., Hill K.O., and Chowdhury D.Q., J. Lightwave Tech. 14, 1768 (1996).
(74) Williams G.R., Vaziri M., Ahn K.H., Barnett B.C., and Islam M.N., Opt. Lett. **20**, 1671 (1995).
(75) Whitaker N.A., Avromopoulos H., French P.M.W., Gabriel M.C., and Lamarche R.E., Opt. Lett. **16**, 1838 (1991).
(76) Cao X.D., Barnett B.C., Ahn K.H., Liang Y., Williams G.R., Vaziri M., and Islam M.N., Opt. Lett. 21, 1211 (1996).
(77) Chimfwembe P.C., and Krasinski J.S., "Cross Kerr and Raman effects between two orthogonally polarized femtosecond pulses in a singlemode fiber", Photonics West, Lase'99, San Jose, CA USA, 23-29 January 1999.

CHAPTER II

(1) Asano N., and Taniuti T., "Reductive Perturbation Method for Nonlinear Wave Propagation in Inhomogeneous Media. I", J. Phys. Soc. (Japan), **27**, 1059 (1969).
(2) Asano N., and Taniuti T, "Reductive Perturbation Method for Nonlinear Wave Propagation in Inhomogeneous Media. II", J. Phys. Soc. (Japan), **29**, 209 (1970).
(3) Asano N., "Reductive Perturbation Method for Nonlinear Wave Propagation in Inhomogeneous Media. II", J. Phys. Soc. (Japan), **29**, 220 (1970).
(4) Taniuti T., "Reductive Perturbation Method and Far Fields of Wave Equations", Prog. Theor. Phys. (Japan), Suppl. **55**, 1 (1974).
(5) Kodama Y., and Hasegawa A., IEEE J. Quantum Electron. **QE-23**, 510 (1987).
(6) Lindstedt A., "Ueber die Integration einer fur die strorungstheorie wichtigen Differentialgleichung", Aston. Nach., **103**, 211 (1882).
(7) Lighthill M.J., "A Technique for Rendering Approximate Solutions to Physical Problems Uniformly Valid", Phil. Mag., **40**, 1179 (1949).
(8) Lighthill M.J., "A Technique for Rendering Approximate Solutions to Physical Problems Uniformly Valid", Z. Flugwiss., 9, 267 (1961).
(9) Nayfeh A.H., *Perturbation Methods*, (John Wiley and Sons, 1973), Chapter 3.
(10) Hasegawa A., and Kodama Y., *Solitons in Optical Communications*, 1st ed. (Oxford University Press, 1995), Chapter 3.
(11) Shen Y.R., *The Principle of Nonlinear Optics*, 1st ed. (John Wiley and Sons, New York, 1984), Chapter 16.
(12) Duguay M.A., and Hansen J.W., Appl. Phys. Lett. **15**, 192 (1969).
(13) Adams M.J., *An Introduction to Optical Waveguides*, 1st ed. (John Wiley and Sons, 1981), Chapter 7.
(14) Gloge D., "Dispersion in Weakly-Guiding Fibers", Appl. Opt. **10**, 2442 (1971).
(15) Rudolph H.-D, and Neumann E.-G, "An Approximation for the Eigenvalues of the Fundamental Mode of Step-Index Glass Fiber Waveguide", Electron. Lett. 29, 328 (1976).
(16) Synder A.W., "Asymptotic Expressions for Eigenfunctions and Eigenvalues of Dielectric or Optical Waveguide", IEEE Trans. **MTT-17**, 1130 (1969).
(17) Gloge D., "Weakly-Guiding Fibers", Appl. Opt. **10**, 2252 (1971).
(18) Chang C.T., "Minimum Dispersion in a Single-Mode Step-Index Optical Fiber", Appl. Opt. **18**, 2516 (1979).
(19) Fleming J.W., "Material Dispersion in Lightguide Glasses", Electron. Lett. **14**, 326 (1978).
(20) Mallitson I.H., "Interspecimen Comparison of the Refractive Index of Fused Silica", J. Opt. Soc. Am. 55, 1205 (1965).
(21) Doran N.J., and Wood D., Opt. Lett. **13**, 56 (1988).

CHAPTER III

(1) Zakharov V.E., and Shabat A.B., Sov. Phys. JETP **34**, 62 (1972).
(2) Taha T.R., and Ablowitz M.J., J. Comp. Phys. **55**, 203 (1984).
(3) Hardin R.H., and Tappert F.D., SIAM Rev. Chronicle **15**, 423 (1973).
(4) Fisher R.A. and Bischel W.K., Appl. Phys. Lett. **23**, 661 (1973); J.Appl. Phys. **46**, 4921 (1975).
(5) Fleck J.A., Morris J.R., and Feit M.D., Appl. Phys. **10**, 129 (1976).
(6) Cooley J.W., and Tukey J.W., Math. Comput. **19**, 297 (1965).
(7) Weiss G.H., and Maradudin A.A., J. Math. Phys. 3, 771 (1962).
(8) Gordon J.P., Opt. Lett. 11, 662 (1986).
(9) Kaminow I.P., "Polarization in optical fibers", IEEE J. Quantum Electron**.** **QE-17**, 15 (1981).

CHAPTER IV

(1) Press W.H., Flannery B.P., Teukolsky S.A., and Vetterling W.T., *Numerical Recipes: The Art of Scientific Computing*, 1st ed. (Cambridge University Press, 1986).
(2) Rudolph H.-D, and Neumann E.-G, "An Approximation for the Eigenvalues of the Fundamental Mode of Step-Index Glass Fiber Waveguide", Electron. Lett. 29, 328 (1976).
(3) Synder A.W., "Asymptotic Expressions for Eigenfunctions and Eigenvalues of Dielectric or Optical Waveguide", IEEE Trans. **MTT-17**, 1130 (1969).
(4) Gloge D., "Weakly-Guiding Fibers", Appl. Opt. **10**, 2252 (1971).
(5) Adams M.J., *An Introduction to Optical Waveguides*, 1st ed. (John Wiley and Sons, 1981), Chapter 7.
(6) Gloge D., "Dispersion in Weakly-Guiding Fibers", Appl. Opt. **10**, 2442 (1971).

CHAPTER V

(1) Agrawal G.P., *Nonlinear Fiber Optics*, 1st ed. (Academic Press, 1989).
(2) Islam M.N., *Ultrafast Fiber Switching Devices and Systems*, 1st ed. (Cambridge University Press, 1992).
(3) Menyuk C.R., Opt. Lett. **12**, 614 (1987).
(4) Menyuk C.R., J. Opt. Soc. Am. B. **5**, 392 (1988).
(5) Hasegawa A., Opt. Lett. **5**, 416 (1980).
(6) Mesentsev V.K., and Turitsyn S.K., Opt. Lett. **17**, 1497 (1992).
(7) Cao X.D., and McKinstrie C.J., J. Opt. Soc. Am. B. **10**, 1202 (1993).

CHAPTER VI

(1) Mitschke F.M., and Mollenauer L.F., Opt. Lett. 12, 407 (1987).
(2) Agrawal G.P., *Nonlinear Fiber Optics*, 1st ed. (Academic Press, 1989).
(3) Islam M.N., *Ultrafast Fiber Switching Devices and Systems*, 1st ed. (Cambridge University Press, 1992).
(4) Menyuk C.R., Opt. Lett. **12**, 614 (1987).
(5) Menyuk C.R., J. Opt. Soc. Am. B. **5**, 392 (1988).
(6) Menyuk C.R., Islam M.N., and Gordon J.P., Opt. Lett. **16**, 566 (1991).
(7) Chen C.-J., Menyuk C.R., Islam M.N., and Stolen R.H., Opt. Lett. **16**, 1647.

APPENDIXES

APPENDIX A—DISPERSION CHARACTERISTICS OF SINGLEMODE FIBER

SMF01

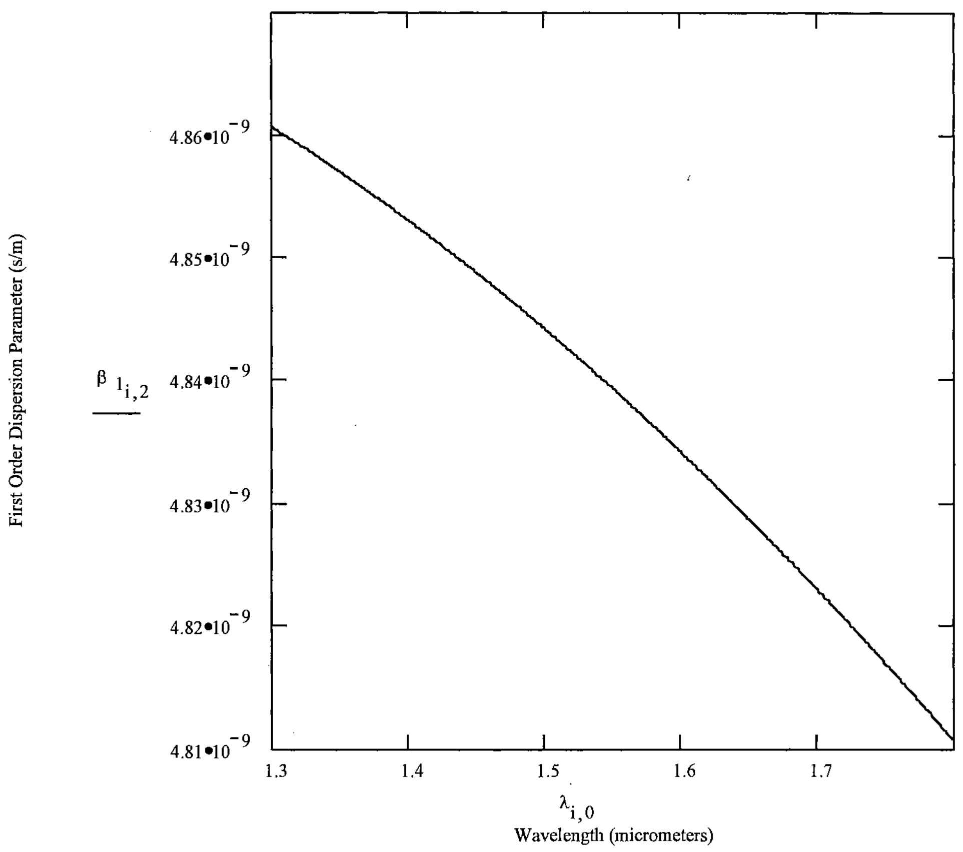


Figure A1 First order dispersion β_1 (s/m) versus wavelength (μm).

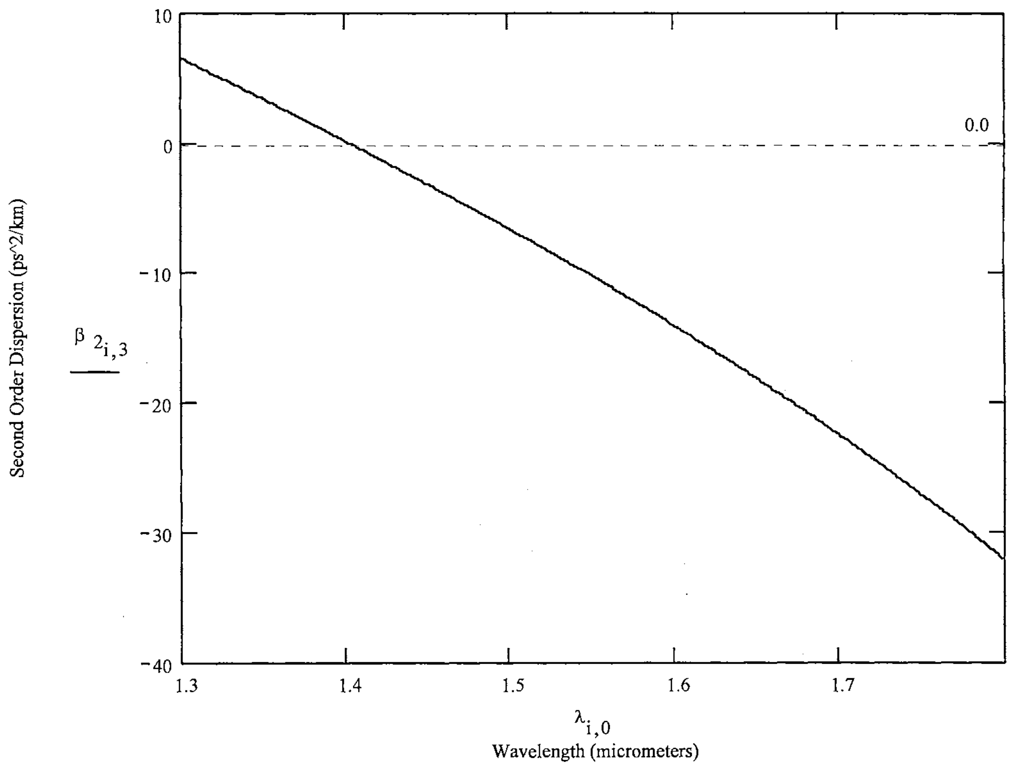


Figure A2 Second order dispersion β_2 (ps^2/km) versus wavelength (μm).

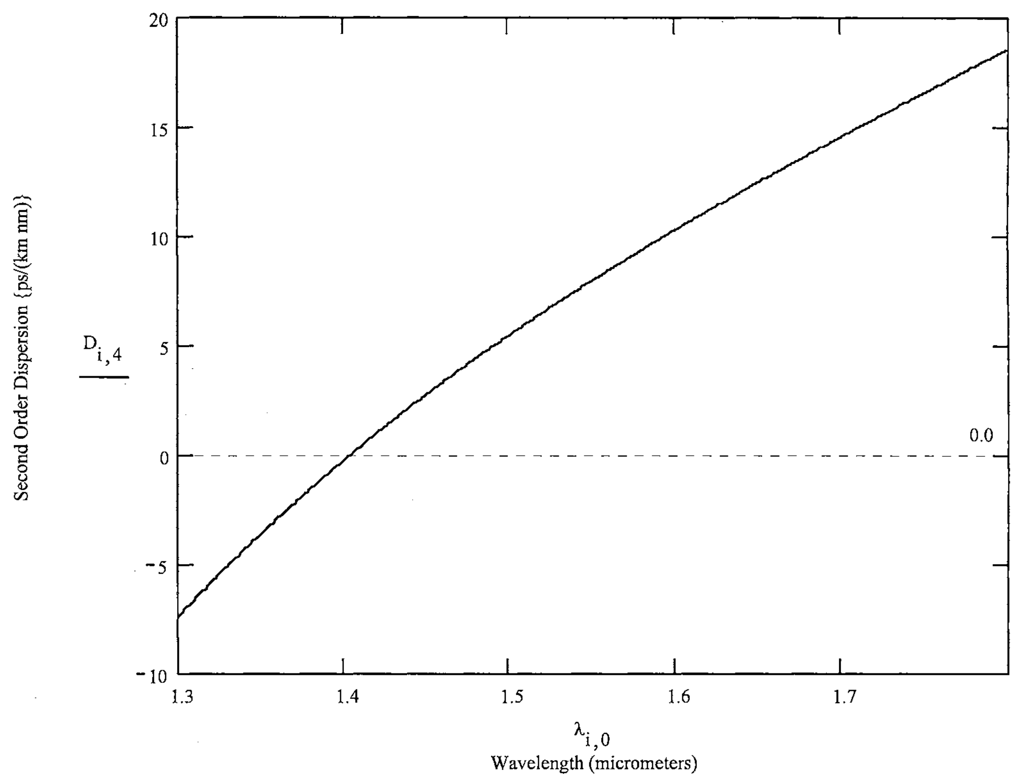


Figure A3 Second order dispersion *D* (ps/km nm) versus wavelength (μm).

APPENDIX B—PHASE II TIME AND FREQUENCY DOMAIN STG/SDG PROPAGATION PROFILES

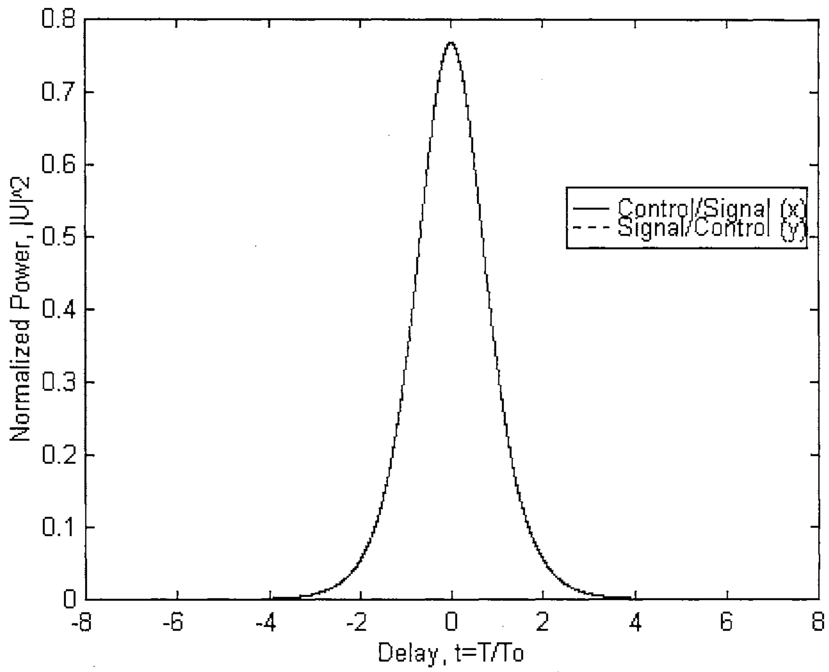


Figure B1.1 Time domain profile of STG (SOD-CPM) at z/z0=0.0000.

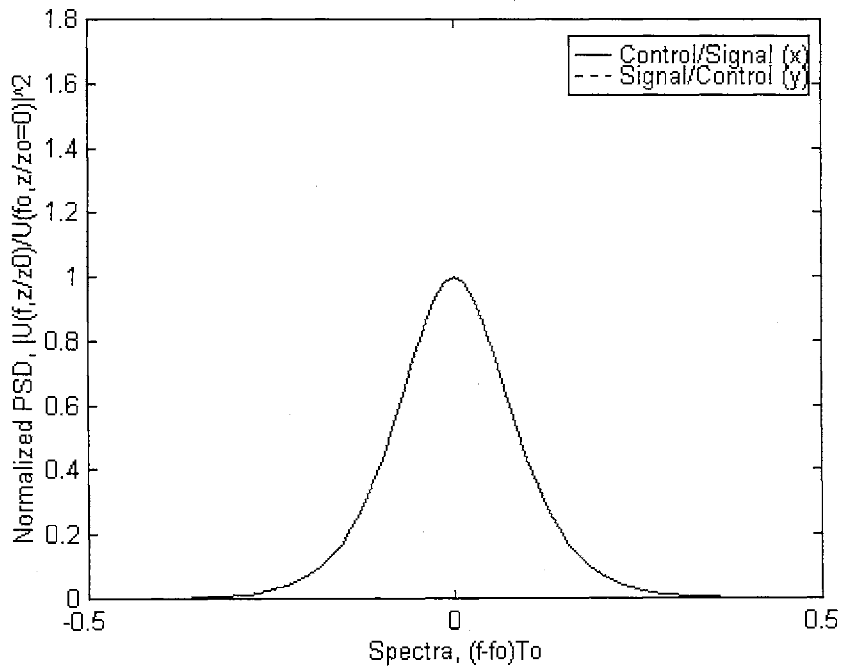


Figure B1.2 Frequency domain profile of STG (SOD-CPM) at z/z0=0.0000.

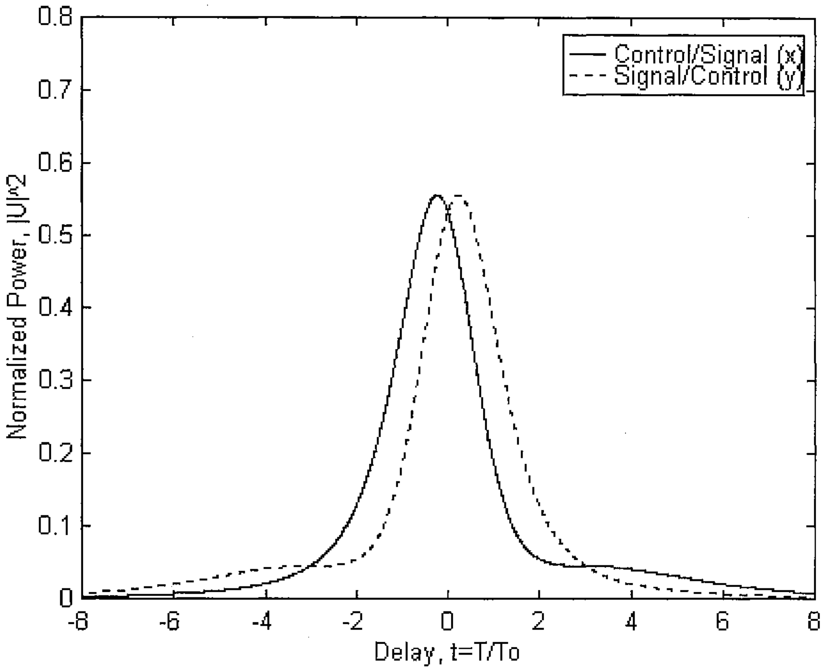


Figure B1.3 Time domain profile of STG (SOD-CPM) at z/z0=6.0000.

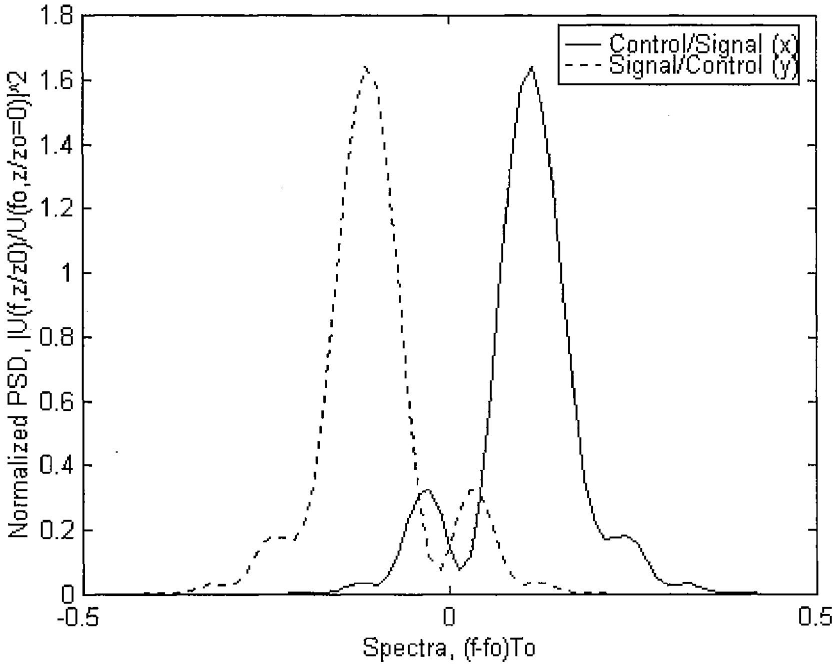


Figure B1.4 Frequency domain profile of STG (SOD-CPM) at z/z0=6.0000.

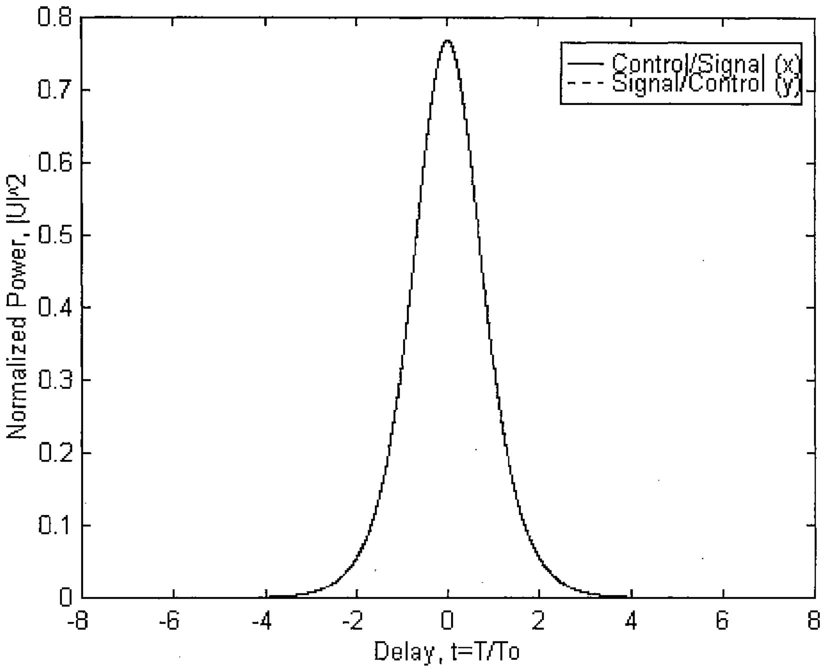


Figure B2.1 Time domain profile of STG (SOD-CFS) at z/z0=0.0000.

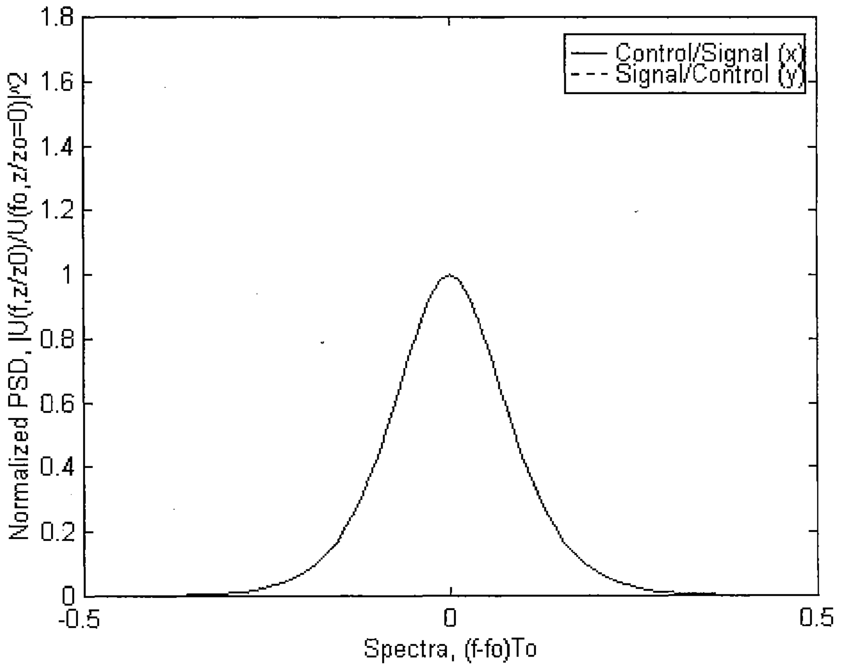


Figure B2.2 Frequency domain profile of STG (SOD-CFS) at z/z0=0.0000.

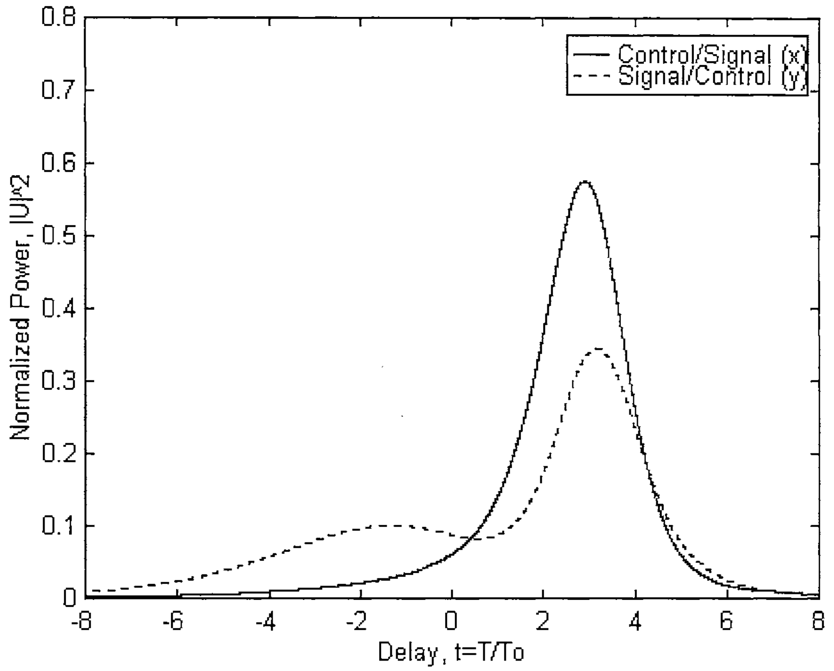


Figure B2.3 Time domain profile of STG (SOD-CFS) at z/z0=6.0000.

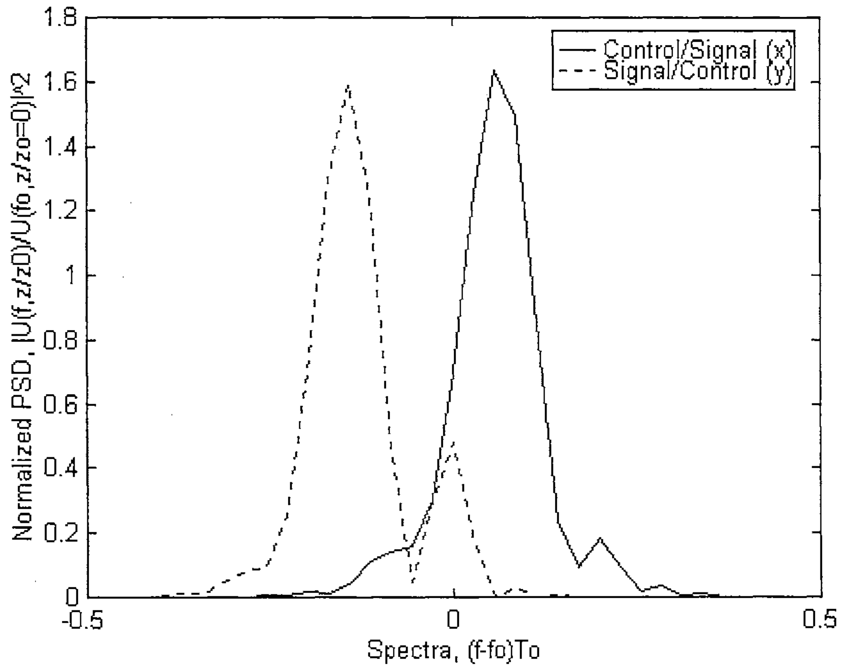


Figure B2.4 Frequency domain profile of STG (SOD-CFS) at z/z0=6.0000.

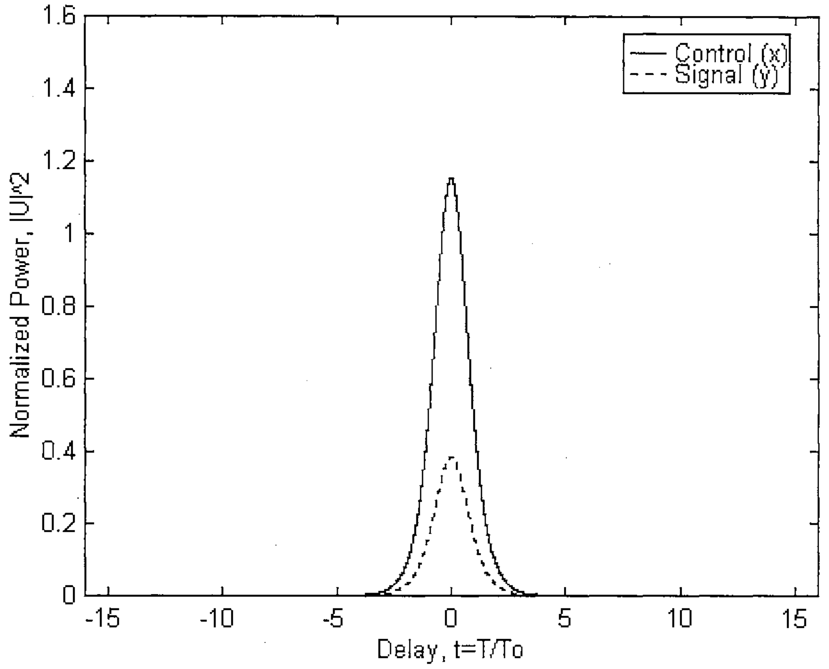


Figure B3.1 Time domain profile of SDG (SOD-CPM) at θ=30° and z/z0=0.0000.

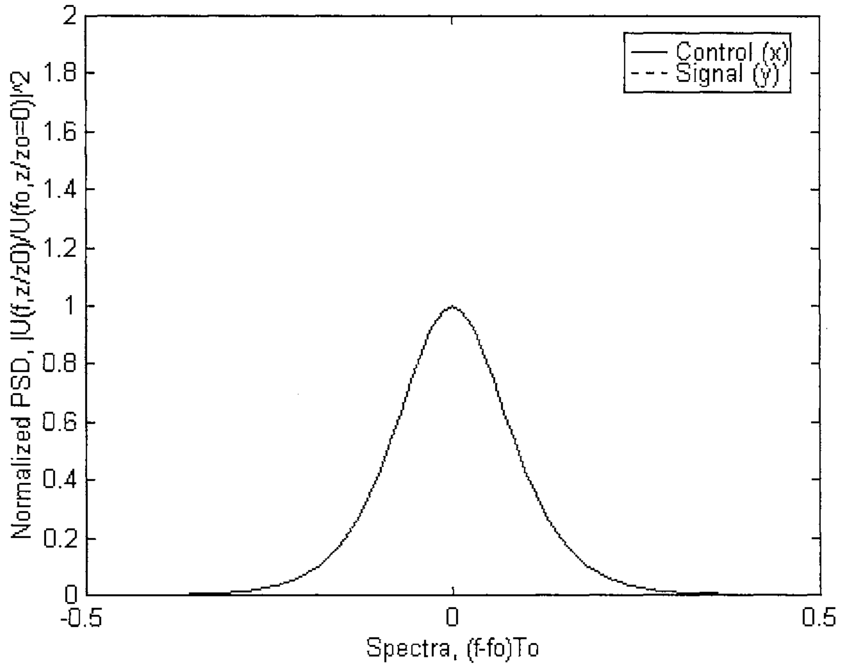


Figure B3.2 Frequency domain profile of SDG (SOD-CPM) at θ=30° and z/z0=0.0000.

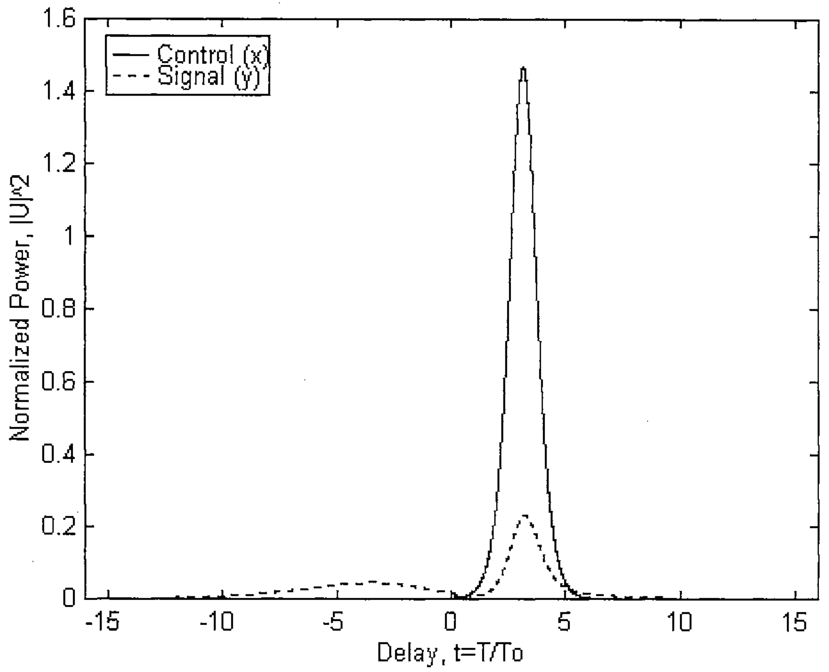


Figure B3.3 Time domain profile of SDG (SOD-CPM) at θ=30° and z/z0=6.0000.

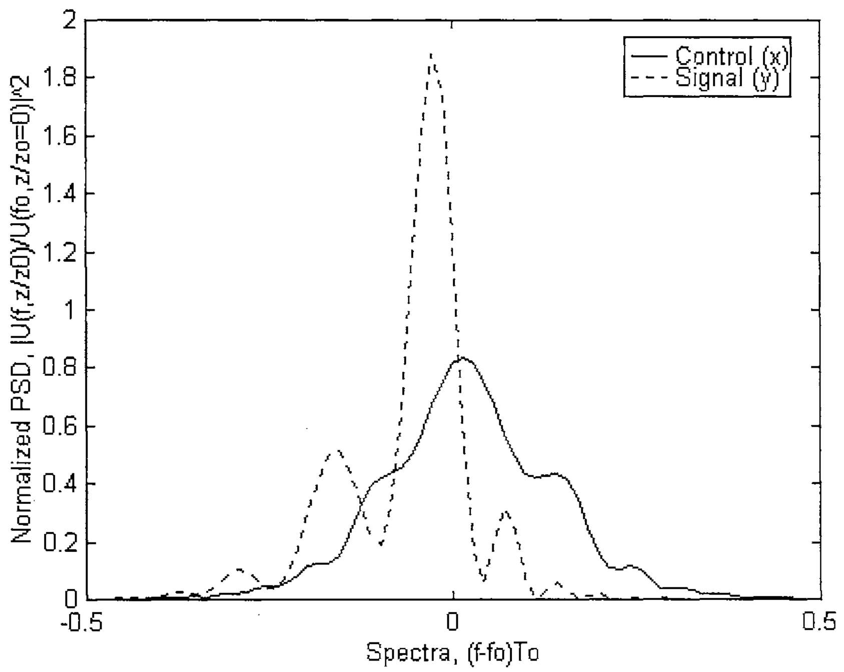


Figure B3.4 Frequency domain profile of SDG (SOD-CPM) at θ=30° and z/z0=6.0000.

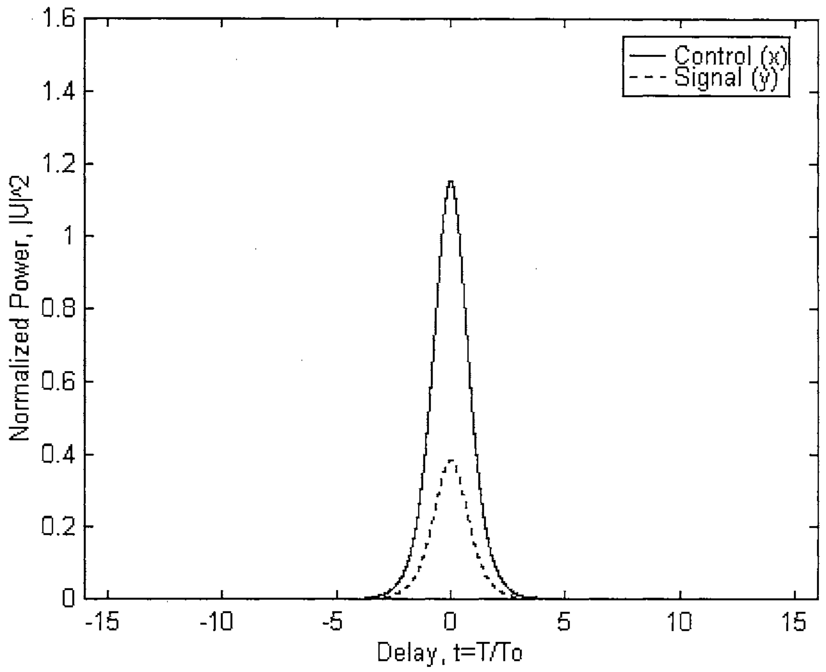


Figure B4.1 Time domain profile of SDG (SOD-CFS) at θ=30° and z/z0=0.0000.

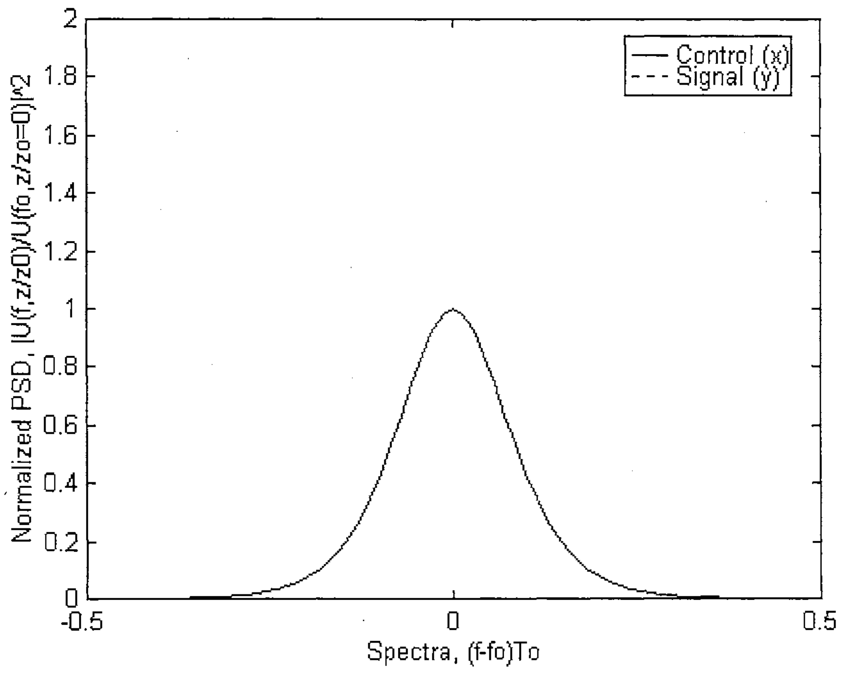


Figure B4.2 Frequency domain profile of SDG (SOD-CFS) at θ=30° and z/z0=0.0000.

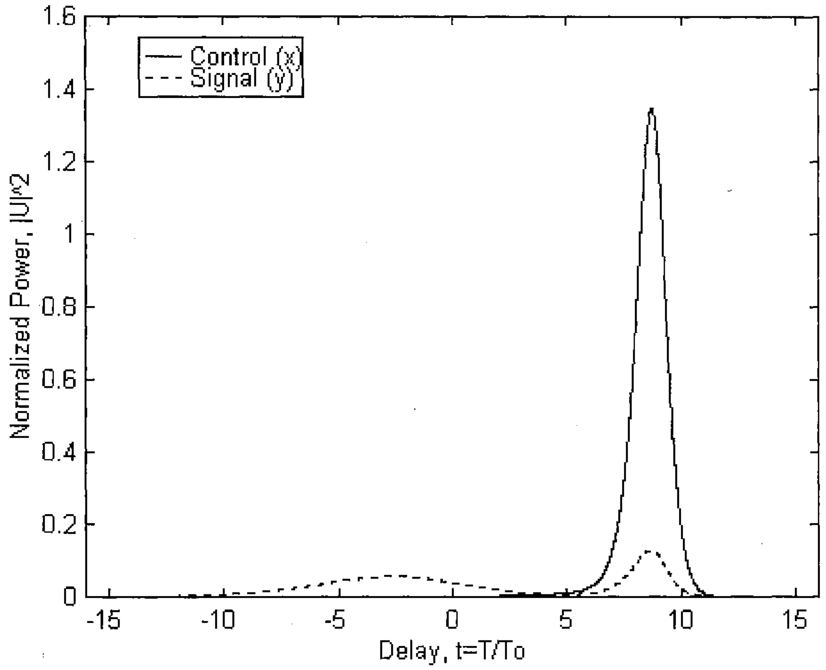


Figure B4.3 Time domain profile of SDG (SOD-CFS) at θ=30° and z/z0=6.0000.

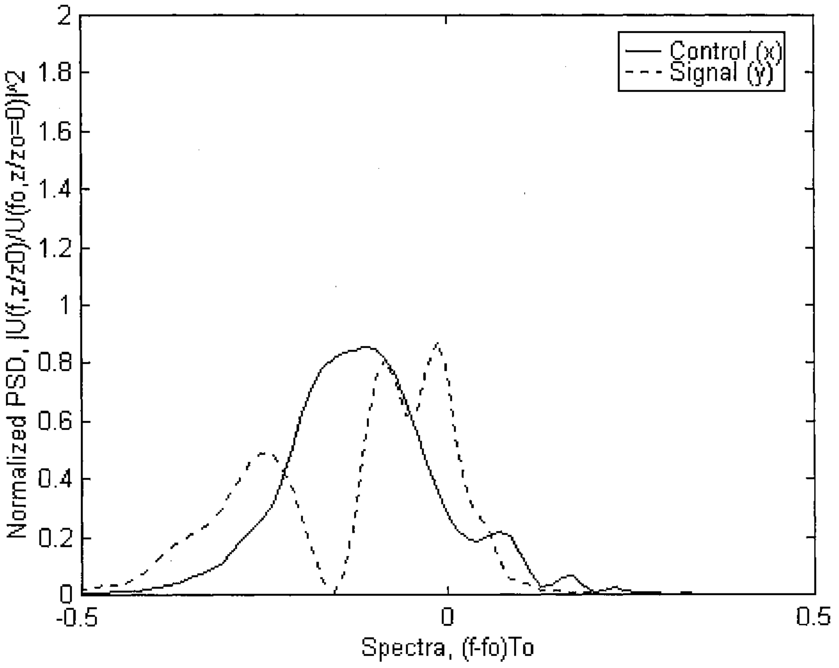


Figure B4.4 Frequency domain profile of SDG (SOD-CFS) at θ=30° and z/z0=6.0000.

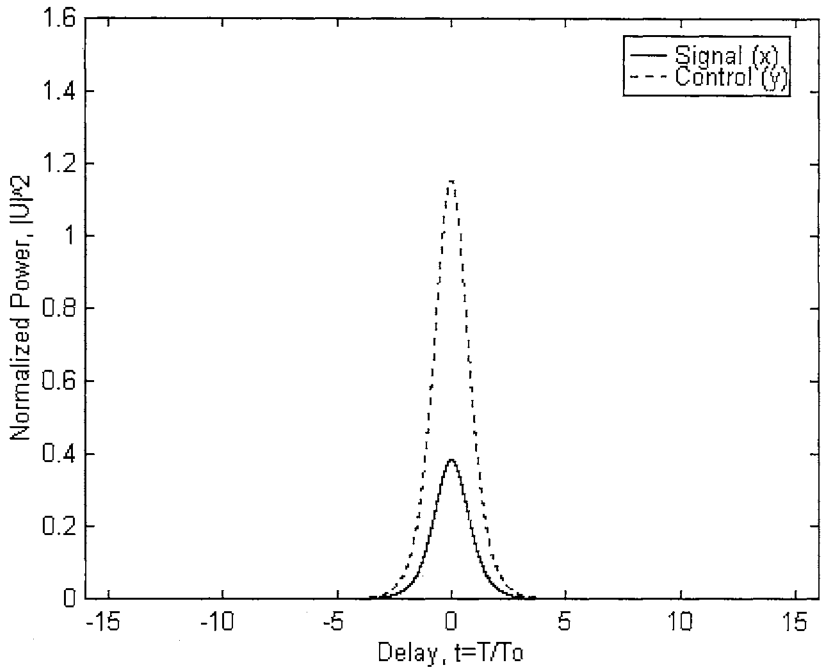


Figure B5.1 Time domain profile of SDG (SOD-CPM) at θ=60° z/z0=0.0000.

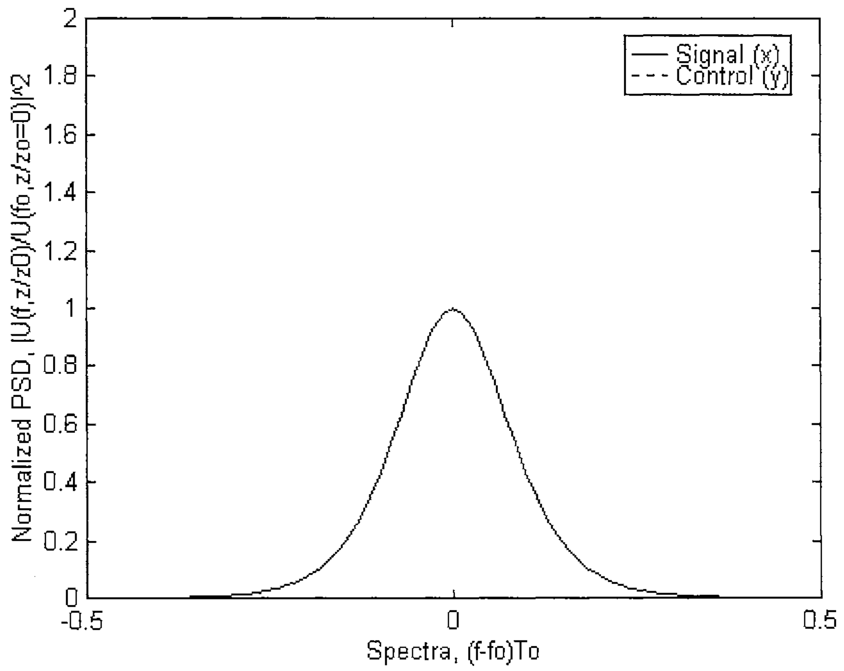


Figure B5.2 Frequency domain profile of SDG (SOD-CPM) at θ=60° z/z0=0.0000.

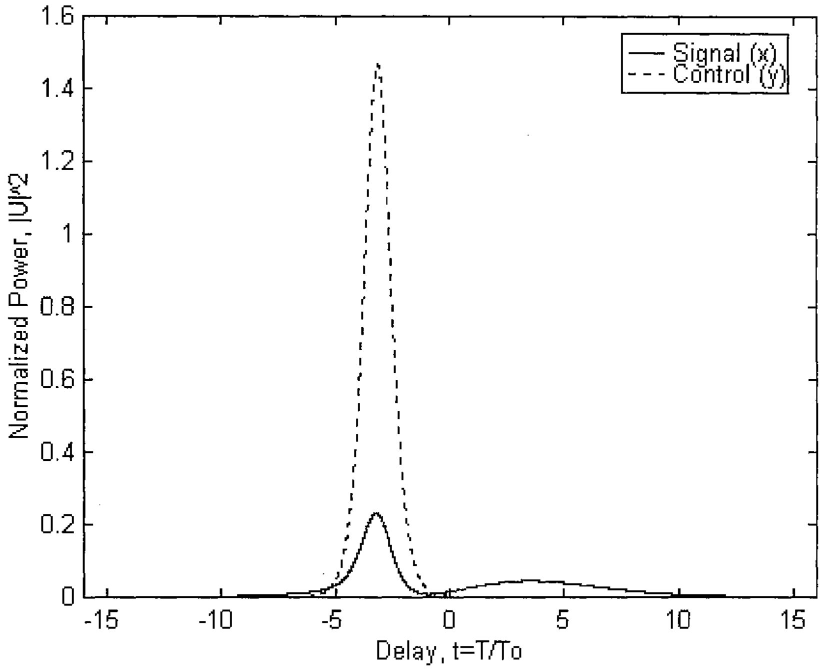


Figure B5.3 Time domain profile of SDG (SOD-CPM) at θ=60° z/z0=6.0000.

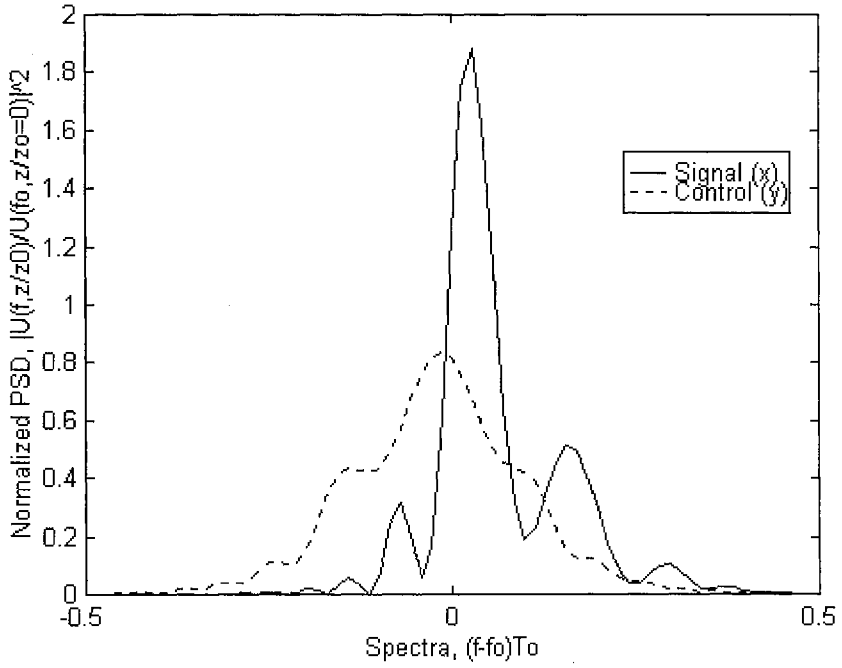


Figure B5.4 Frequency domain profile of SDG (SOD-CPM) at θ=60° z/z0=6.0000.

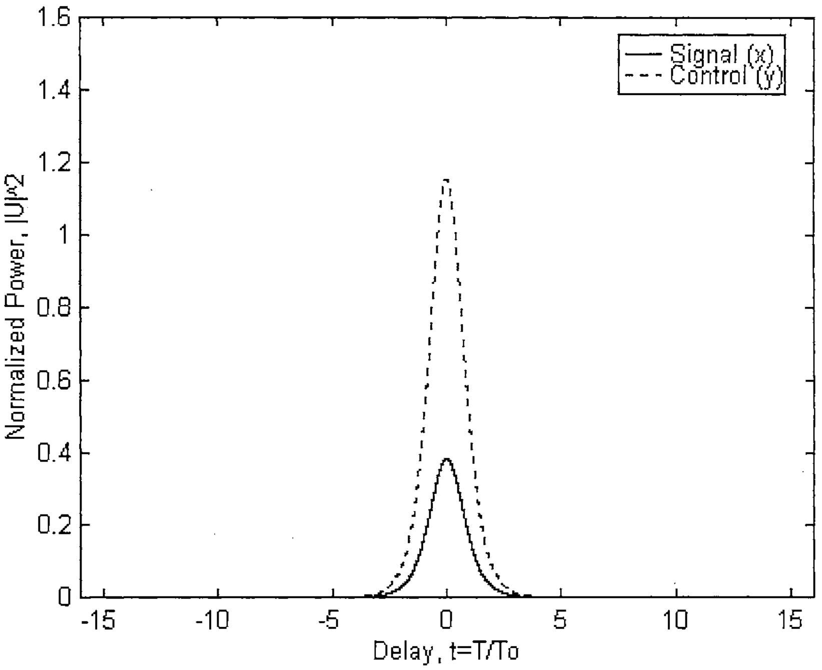


Figure B6.1 Time domain profile of SDG (SOD-CFS) at θ=60° z/z0=0.0000.

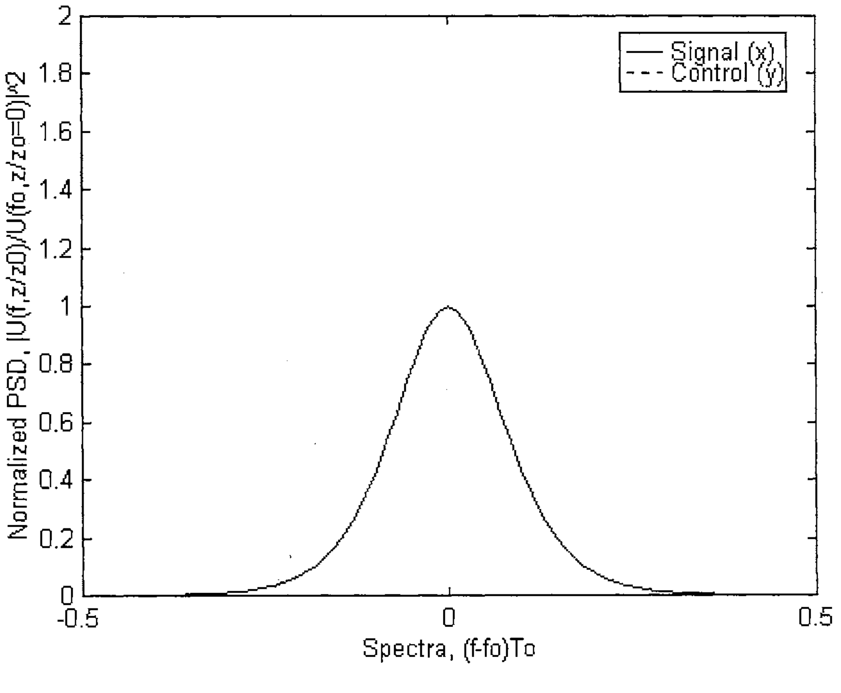


Figure B6.2 Frequency domain profile of SDG (SOD-CFS) at θ=60° z/z0=0.0000.

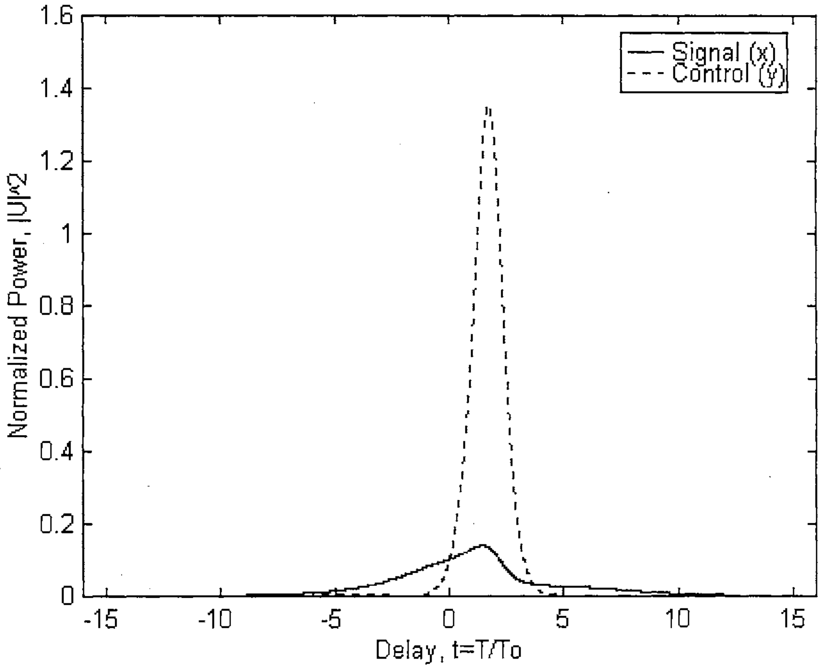


Figure B6.3 Time domain profile of SDG (SOD-CFS) at θ=60° z/z0=6.0000.

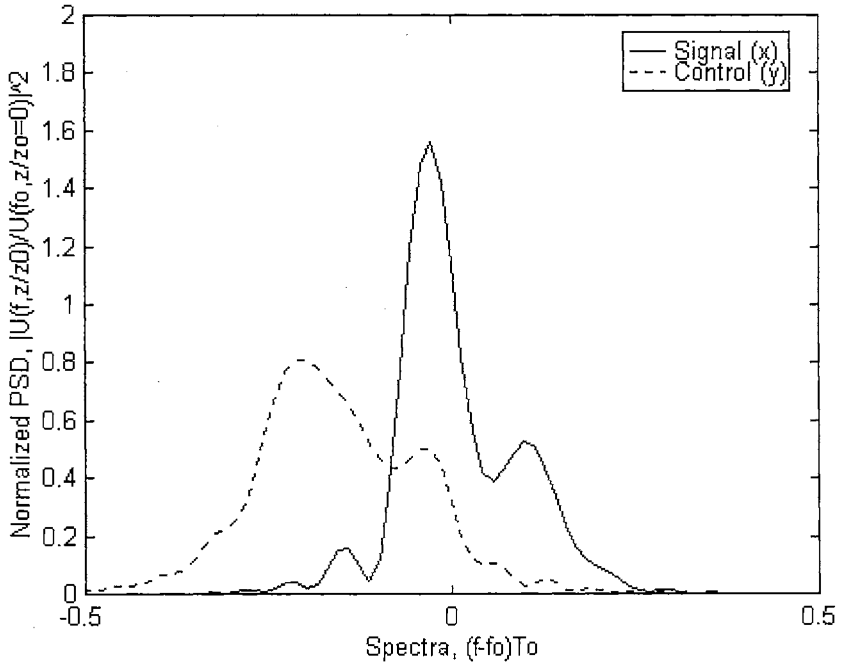


Figure B6.4 Frequency domain profile of SDG (SOD-CFS) at θ=60° z/z0=6.0000.

APPENDIX C—COMPUTER PROGRAM LISTING

```
/*
**********************************************************************
******
File: wlength.cpp
Author: Patrick C. Chimfwembe
Creation: 06/20/96
Revised: 10/16/97
**********************************************************************
********
Function:
Determines the walk-off  of two optical signals at different
wavelengths and power in a single-
mode waveguide and generates a file, wl.dat, that tabulates the group-
velocity , pulse intensity
and walk-off against wavelength
*/


# include <stdio.h>
# include <utility.h>
# include <process.h>
# include <stdlib.h>
# include <alloc.h>
# include <complex>
# include "wlglobal.h"
# include "wlength.h"
# include "wgbfunc.h"
# include "nlbfunc.h"
# include "bpm.h"
# include "admathf.h"


/* Step Index Single Mode Fiber Material Variables */

/* Core Refractive Index Data */
double GeSiO135865[3]={0.711040,0.451885,0.704048};/* Fleming (1978) */
double GeSiO135865wl[3]={0.064270e-6,0.129408e-6,9.425478e-6};

/* Cladding Refractive Index Data */
double SiO2[3]={0.6961663,0.4079426,0.8974794};/* Mallitson (1965) */
double SiO2wl[3]={0.0684043e-6,0.1162414e-6,9.896161e-6};

/* Core and cladding dynamic variables */
double *core=GeSiO135865, *corewl=GeSiO135865wl, *cladding=SiO2,
*claddingwl=SiO2wl;

double Px, Py ;// Peak power for x and y axes (W).
double xintensity, yintensity;//  Peak intensity for x and y axes
(W/m^2).

dcomplex  *profilex, *profiley;
```

```
void main(void){
  //FILE *dfp;
  //double pumpI,signalI;
  int i;
  if((profilex=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer profile\n");
    exit(1);  /* terminate program if out of memory */
    }
  //dcomplex cbeam[8]={dcomplex(-2.0,0.0),dcomplex(-1.0,-
0.0),dcomplex(0.0,0.0),dcomplex(1.0,0.0),dcomplex(2.0,0.0),dcomplex(2.0
,0.0),dcomplex(2.0,0.0),dcomplex(2.0,0.0)};
  //dcomplex temp;
  //temp=cbeam[0]+cbeam[1];
  /* pumpI=nsolitonp(pwlength,order,ppwidth)/FXSA;
  signalI=nsolitonp(swlength,order,spwidth)/FXSA; */
 /* if((dfp=fopen("eigenv.dat","w+"))==NULL){
    printf("Unable to open output file eigenv.dat\n");
    return;
    } */
 /*  fprintf(dfp,"Reference Power and intensity  for optical
fiber.\n");
  fprintf(dfp,"Power at 5 W has intensity %.8le\n",5.0/FXSA);
  fprintf(dfp,"Signal\n");
  fprintf(dfp,"Wavelength:%.8le\n",swlength);
  fprintf(dfp,"Signal GVD
(Beta2)=%.8le\n",GVD(swlength,0.0,0.0)*1.0e24*1.0e3);
  fprintf(dfp,"Intensity:%.8le\n",signalI);
  fprintf(dfp,"Pump\n");
  fprintf(dfp,"Wavelength:%.8le\n",pwlength);
  fprintf(dfp,"Pump GVD
(Beta2)=%.8le\n",GVD(pwlength,0.0,0.0)*1.0e24*1.0e3);
  fprintf(dfp,"Pump Intensity  PGVelocity      SGVelocity      d12(sp)
Walk-off\n");
  for(i=1;i<=200;i++){
    d12=(1.0/sgvelocity(swlength,core,corewl,signalI,pumpI))-
(1.0/sgvelocity(pwlength,core,corewl,(pumpI/100.0)*i,signalI));
    fprintf(dfp,"%.8le  %.8le  %.8le  %.8le
%.8le\n",(pumpI/100.0)*i,sgvelocity(pwlength,core,corewl,(pumpI/100.0)*
i,

signalI),sgvelocity(swlength,core,corewl,signalI,(pumpI/100.0)*i),d12,p
pwidth/d12);
    }
  fclose(dfp); */
 /* printf("Beta-2=%.8le ps^2/km at wavelength=%.8le micrometers\n",

sgvd(wavelength,SiO2,SiO2wl)*1.0e3*1.0e24,wavelength*1.0e6);
  printf("Fundamental Soliton Power=%.8le W at wavelength=%.8le
micrometers;Gamma=%.8le\n",
               nsolitonp(wavelength,SiO2,SiO2wl,order, spwidth),
wavelength*1.0e6, gamma(wavelength));
   printf("D=%.8le ps/km nm at wavelength=%.8le micrometers\n",
              SD(wavelength,core,corewl)*1.0e12*1.0e3*1.0e-
9,wavelength*1.0e6); */
  /* x=a;
  for(i=1;i<=10;i++){
   wavelength=cow(x);
   printf("Cut-off wavelength=%.8le; V=%.8le;
Radius=%.8le\n",wavelength,V(wavelength,x),x);
   x+=0.05e-6;
   }*/
   /* x=1.0;
   fprintf(dfp,"V    U    W    b    d(vb)/dv    vd^2(vb)/dv^2    Gamma\n");
   for(i=1;i<=50;i++){
     fprintf(dfp,"%.8le %.8le %.8le %.8le %.8le %.8le %.8le\n",x,U(x),
                 W(x),b(x),DBV(x),VD2BV(x),PCFactor(x));
     x+=0.1;
     }
    fclose(dfp); */
      // printf("V*VD^2(bV)=%.8le; U=%.8le;
W=%.8le\n",VD2BV(V(wavelength,a)),U(V(wavelength,a)),
Wrn(V(wavelength,a)));
 // for(i=15;i<=25;i++)
/* x=14.1;
 for(i=1;i<=10;i++){
   printf("Bessel Y1(%.16le):%.16le\n",x,bessy1(x));
   x+=0.1;
   } */
  //x=19.2;
 //printf("V=%.8le\nV*VD^2(bV)=%.8le\n GAMMA=%.8le\n
b=%.8le\n",x,VD2BV(x),PCFactor(x),b(x) );
 //
printf("K0(%.8le)=%.8le\nK1(%.8le)=%.8le\n",(double)(i)/10.0,bessk0((do
uble)(i)/10.0),(double)(i)/10.0,bessk1((double)(i)/10.0));
//for(i=1;i<=10;i++){
 //printf("x=%.8le\n",x);
   // printf("e^(x)K0(x)=%.25le\n
e^(x)K1(x)=%.25le\n",bessk0(x)*exp(x),bessk1(x)*exp(x));
   //printf("x^(2)K2(x)=%.25le\n",bessk(2,x)*exp(x));
   //printf("e^(-x)I0(x)=%.25le\n e^(-x)I1(x)=%.25le\n",bessi0(x)*exp(-
x),bessi1(x)*exp(-x));
 // printf("x^(-2)I2(x)=%.25le\n",bessi(2,x)*exp(-x));
 // printf("J2(x)=%.25le\n Y2(x)=%.25le\n",bessj(2,x),bessy(2,x));
 // x+=0.1;
 // }
 //fiberDdata();
 //dfft(cbeam,8,1);
 //gf1solitonp(profile,50.0e-15,90.0e-15);
 //printf("n=%.6le\n",nsellmeier(swlength,core,corewl));
 propsolitonp(profilex,spwidth,1.0e-5,0.5,2000.0e-15); //Setting
nzstep=1.0e-5 give about 16 hours computation time/shot of 10.
 free(profilex); //Free profilex
}
```

```
  /* Refractive index approximation using Sellmeier's equation */
double nsellmeier(double wavelength, double B[3], double wlength[3]){
  double w[3],result=0.0;
  int j;
  for(j=0;j<=2;j++)
    w[j]=2*PI*C/wlength[j];
  for(j=0;j<=2;j++)
    result=result+(B[j]*pow(w[j],2.0)/(pow(w[j],2.0)-
pow(2*PI*C/wavelength,2.0)));
  result=sqrt(1+result);
  return result;
  }

  /* First derivative of the material refractive index with respect to
the  radian frequency approximation
   using Sellmeier's equation */
double D1nsellmeier(double wavelength, double B[3], double wlength[3]){
  double w[3],radw,result=0.0;
  int j;
  radw=2.0*PI*C/wavelength;
  for(j=0;j<=2;j++)
    w[j]=2*PI*C/wlength[j];
 for(j=0;j<=2;j++)
    result=result+(B[j]*pow(w[j],2.0)*radw/pow((pow(w[j],2.0)-
pow(radw,2.0)),2.0));
  result=result/nsellmeier(wavelength,B,wlength);
  return result;
  }

  /* Second derivative of the Refracitive index with respect to the
radian frequency approximation
      using Sellmeier's equation */
double D2nsellmeier(double wavelength, double B[3], double wlength[3]){
  double w[3], radw, result=0.0;
  int j;
  radw=2.0*PI*C/wavelength;
  for(j=0;j<=2;j++)
    w[j]=2*PI*C/wlength[j];
  for(j=0;j<=2;j++)
    result=result+(((B[j]*pow(w[j],2.0)*pow((pow(w[j],2.0)-
pow(radw,2.0)),2.0))+
                 (4*B[j]*pow(w[j],2.0)*pow(radw,2.0)*(pow(w[j],2.0)-
pow(radw,2.0))))
                 /pow((pow(w[j],2.0)-pow(radw,2.0)),4.0));
  result=result-pow(D1nsellmeier(wavelength,B,wlength),2.0);
  result=result/nsellmeier(wavelength,B,wlength);
  return result;
  }

/* Group index based on Sellmeier's equation.  Using radian frequency.
Intensity dependent. */
double sgroupnrf(double wavelength, double B[3], double wlength[3]){
  double radw=2.0*PI*C/wavelength;
  return (nsellmeier(wavelength,
B,wlength)+(radw*D1nsellmeier(wavelength,B,wlength)));
  }

 /* Group index based on Sellmeier's equation.  Using wavelength.
Intensity dependent. */
 double sgroupnwl(double wavelength, double B[3], double wlength[3]){
  return (nsellmeier(wavelength,
B,wlength)+(wavelength*D1lbdnsellmeier(wavelength,B,wlength)));
  }

/* Group velocity based on Sellmeier's equation */
double sgvelocity(double wavelength, double B[3], double wlength[3]){
   return C/sgroupnrf(wavelength,B,wlength);
   }

/* Group Velocity Dispersion (GVD) or Beta-2 based on Sellmeier's
equation */
double sgvd(double wavelength, double B[3], double wlength[3]){
  double radw=2.0*PI*C/wavelength;
  return
((1.0/C)*(2*D1nsellmeier(wavelength,B,wlength)+radw*D2nsellmeier(wavele
ngth,B,wlength)));
  }
/* Dispersion Parameter (D) based on Sellmeier's equation */
double SD(double wavelength, double B[3], double wlength[3]){
  double radw=2.0*PI*C/wavelength;
  return (sgvd(wavelength,B,wlength)*(-2*PI*C/pow(wavelength,2.0)));
  }

 /* First derivative of the material refractive index with respect to
the  wavelength approximation
   using Sellmeier's equation */
double D1lbdnsellmeier(double wavelength, double B[3], double
wlength[3]){
  double result=0.0;
  int j;
 for(j=0;j<=2;j++)

result=result+(B[j]*pow(wlength[j],2.0)*wavelength/pow((pow(wavelength,
2.0)-pow(wlength[j],2.0)),2.0));
  result=-result/nsellmeier(wavelength,B,wlength);
  return result;
  }

  /* Second derivative of the Refractive index with respect to the
wavelength approximation
      using Sellmeier's equation */
double D2lbdnsellmeier(double wavelength, double B[3], double
wlength[3]){
  double result=0.0;
  int j;
  for(j=0;j<=2;j++)

result=result+(B[j]*pow(wlength[j],2.0)*(3.0*pow(wavelength,2.0)+pow(wl
ength[j],2.0)))
                /pow((pow(wavelength,2.0)-pow(wlength[j],2.0)),3.0);
  result=result-pow(D1lbdnsellmeier(wavelength,B,wlength),2.0);
  result=result/nsellmeier(wavelength,B,wlength);
  return result;
  }

/* Relative refractive index difference using Sellmeier's equation */
double DELTA(double wavelength){
  double n1= nsellmeier(wavelength,core,corewl);
  double n2=nsellmeier(wavelength,cladding,claddingwl);
  return ((pow(n1,2.0)-pow(n2,2.0))/(2.0*pow(n1,2.0)));
  }

 /* First Derivative of DELTA, the relative refractive index
difference, using Sellmeier's equation */
 double D1DELTA(double wavelength){
   double n1= nsellmeier(wavelength,core,corewl);
   double n2=nsellmeier(wavelength,cladding,claddingwl);
   return ((-
n2/pow(n1,3.0))*((n1*D1lbdnsellmeier(wavelength,cladding,claddingwl))-

(n2*D1lbdnsellmeier(wavelength,core,corewl))));
   }

/* Generates composite material dispersion (CMD), waveguide dispersion
(WD), composite profile
    dispersion (CPD), dispersion cross-products (MR) and the total
dispersion (D=CMD+WD+CPD+MR)
*/
void fiberDdata(void){
   FILE *dfp;
  int i;
  double wavelength=swlength,step=1.0e-9,x=0.0; // Step:1.0e-9;  Cut-
off wavelength is 1.30070109e-6 m at fiber core radius, a=(4.0e-6/2.0)
m
  if((dfp=fopen("fddata.dat","w+"))==NULL){
    printf("Unable to open output file fddata.dat\n");
    return;
    }
  x=a;
  for(i=1;i<=10;i++){
   wavelength=cow(x);
  fprintf(dfp,"Cut-off wavelength=%.8le; V=%.8le;
Radius=%.8le\n",wavelength,V(wavelength,x),x);
   x+=0.01e-6;
   }
  wavelength=1.3e-6; /* 1.3e-6;1.45e-6 */
  fprintf(dfp,"Wavelength (x10^-6 m)  Group Velocity (m/s)   Beta1
 (s/m)  Beta2 (ps^2/km) D (ps/(km*nm))\n");
```

```
  while(wavelength<=1.8e-6){/* 1.8e-6;1.6e-6 */
    fprintf(dfp,"%.8le  %.8le  %.8le  %.8le
%.8le\n",wavelength*1.0e6,egv(wavelength),ebeta1(wavelength),GVD(wavele
ngth)*1.0e24*1.0e3,D(wavelength)*1.0e12*1.0e3/1.0e9);
    wavelength+=step;
   }
   wavelength=1.3e-6; /* 1.3e-6;1.45e-6 */
   fprintf(dfp,"Wavelength (x10^-6 m)  CMD (ps/(km*nm)) WGD
(ps/(km*nm)) CPD (ps/(km*nm)) MR  (ps/(km*nm)) D (ps/(km*nm))\n");
   while(wavelength<=1.80e-6){ /* 1.8e-6;1.6e-6 */
   fprintf(dfp,"%.8le  %.8le  %.8le  %.8le  %.8le
%.8le\n",wavelength*1.0e6,cmd(wavelength)*1.0e12*1.0e3/1.0e9,wgd(wavele
ngth)*1.0e12*1.0e3/1.0e9,cpd(wavelength)*1.0e12*1.0e3/1.0e9,dcp(wavelen
gth)*1.0e12*1.0e3/1.0e9,D(wavelength)*1.0e12*1.0e3/1.0e9);
   wavelength+=step;
    }
  fclose(dfp);
  }
/*
**********************************************************************
******
File: wlengtha.cpp
Author: Patrick C. Chimfwembe
Creation: 06/20/96
Revised: 03/31/99
**********************************************************************
********
Function:
Determines the walk-off  of two optical signals at different
wavelengths and power in a single-
mode waveguide and generates a file, wl.dat, that tabulates the group-
velocity , pulse intensity
and walk-off against wavelength
*/


# include <stdio.h>
# include <utility.h>
# include <process.h>
# include <stdlib.h>
# include <alloc.h>
# include <complex>
# include "wlglobal.h"
# include "wlength.h"
# include "wgbfunc.h"
# include "nlbfunc.h"
# include "bpm.h"
# include "admathf.h"


/* Step Index Single Mode Fiber Material Variables */

/* Core Refractive Index Data */
```

```
double GeSiO135865[3]={0.711040,0.451885,0.704048};/* Fleming (1978) */
double GeSiO135865wl[3]={0.064270e-6,0.129408e-6,9.425478e-6};

/* Cladding Refractive Index Data */
double SiO2[3]={0.6961663,0.4079426,0.8974794};/* Mallitson (1965) */
double SiO2wl[3]={0.0684043e-6,0.1162414e-6,9.896161e-6};

/* Core and cladding dynamic variables */
double *core=GeSiO135865, *corewl=GeSiO135865wl, *cladding=SiO2,
*claddingwl=SiO2wl;

double Px,Py, EiPy;// Peak power for x and y axes (W).
double xintensity, yintensity;//  Peak intensity for x and y axes
(W/m^2).

dcomplex  *profilex, *profiley;

void main(void){
  //FILE *dfp;
  //double pumpI,signalI;
  int i;
  //if((profilex=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    //printf("Not enough memory to allocate buffer profilex\n");
    //exit(1);  /* terminate program if out of memory */
    //}
  if((profiley=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer profiley\n");
    exit(1);  /* terminate program if out of memory */
    }
  //dcomplex cbeam[8]={dcomplex(-2.0,0.0),dcomplex(-1.0,-
0.0),dcomplex(0.0,0.0),dcomplex(1.0,0.0),dcomplex(2.0,0.0),dcomplex(2.0
,0.0),dcomplex(2.0,0.0),dcomplex(2.0,0.0)};
  //dcomplex temp;
  //temp=cbeam[0]+cbeam[1];
  /* pumpI=nsolitonp(pwlength,order,ppwidth)/FXSA;
  signalI=nsolitonp(swlength,order,spwidth)/FXSA; */
 /* if((dfp=fopen("eigenv.dat","w+"))==NULL){
    printf("Unable to open output file eigenv.dat\n");
    return;
    } */
 /*  fprintf(dfp,"Reference Power and intensity  for optical
fiber.\n");
  fprintf(dfp,"Power at 5 W has intensity %.8le\n",5.0/FXSA);
  fprintf(dfp,"Signal\n");
  fprintf(dfp,"Wavelength:%.8le\n",swlength);
  fprintf(dfp,"Signal GVD
(Beta2)=%.8le\n",GVD(swlength,0.0,0.0)*1.0e24*1.0e3);
  fprintf(dfp,"Intensity:%.8le\n",signalI);
  fprintf(dfp,"Pump\n");
  fprintf(dfp,"Wavelength:%.8le\n",pwlength);
  fprintf(dfp,"Pump GVD
(Beta2)=%.8le\n",GVD(pwlength,0.0,0.0)*1.0e24*1.0e3);
```

```
  fprintf(dfp,"Pump Intensity  PGVelocity      SGVelocity      d12(sp)
Walk-off\n");
  for(i=1;i<=200;i++){
    d12=(1.0/sgvelocity(swlength,core,corewl,signalI,pumpI))-
(1.0/sgvelocity(pwlength,core,corewl,(pumpI/100.0)*i,signalI));
    fprintf(dfp,"%.8le  %.8le  %.8le  %.8le
%.8le\n",(pumpI/100.0)*i,sgvelocity(pwlength,core,corewl,(pumpI/100.0)*
i,

signalI),sgvelocity(swlength,core,corewl,signalI,(pumpI/100.0)*i),d12,p
pwidth/d12);
    }
  fclose(dfp); */
 /* printf("Beta-2=%.8le ps^2/km at wavelength=%.8le micrometers\n",

sgvd(wavelength,SiO2,SiO2wl)*1.0e3*1.0e24,wavelength*1.0e6);
  printf("Fundamental Soliton Power=%.8le W at wavelength=%.8le
micrometers;Gamma=%.8le\n",
               nsolitonp(wavelength,SiO2,SiO2wl,order, spwidth),
wavelength*1.0e6, gamma(wavelength));
   printf("D=%.8le ps/km nm at wavelength=%.8le micrometers\n",
              SD(wavelength,core,corewl)*1.0e12*1.0e3*1.0e-
9,wavelength*1.0e6); */
  /* x=a;
  for(i=1;i<=10;i++){
   wavelength=cow(x);
   printf("Cut-off wavelength=%.8le; V=%.8le;
Radius=%.8le\n",wavelength,V(wavelength,x),x);
   x+=0.05e-6;
   }*/
   /* x=1.0;
   fprintf(dfp,"V   U   W   b   d(vb)/dv   vd^2(vb)/dv^2   Gamma\n");
   for(i=1;i<=50;i++){
     fprintf(dfp,"%.8le %.8le %.8le %.8le %.8le %.8le %.8le\n",x,U(x),
                W(x),b(x),DBV(x),VD2BV(x),PCFactor(x));
     x+=0.1;
     }
    fclose(dfp); */
      // printf("V*VD^2(bV)=%.8le; U=%.8le;
W=%.8le\n",VD2BV(V(wavelength,a)),U(V(wavelength,a)),
Wrn(V(wavelength,a)));
 // for(i=15;i<=25;i++)
/* x=14.1;
 for(i=1;i<=10;i++){
   printf("Bessel Y1(%.16le):%.16le\n",x,bessy1(x));
   x+=0.1;
   } */
  //x=19.2;
 //printf("V=%.8le\nV*VD^2(bV)=%.8le\n GAMMA=%.8le\n
b=%.8le\n",x,VD2BV(x),PCFactor(x),b(x) );
 //
printf("K0(%.8le)=%.8le\nK1(%.8le)=%.8le\n",(double)(i)/10.0,bessk0((do
uble)(i)/10.0),(double)(i)/10.0,bessk1((double)(i)/10.0));
```

```
//for(i=1;i<=10;i++){
 //printf("x=%.8le\n",x);
   // printf("e^(x)K0(x)=%.25le\n
e^(x)K1(x)=%.25le\n",bessk0(x)*exp(x),bessk1(x)*exp(x));
   //printf("x^(2)K2(x)=%.25le\n",bessk(2,x)*exp(x));
   //printf("e^(-x)I0(x)=%.25le\n e^(-x)I1(x)=%.25le\n",bessi0(x)*exp(-
x),bessi1(x)*exp(-x));
 // printf("x^(-2)I2(x)=%.25le\n",bessi(2,x)*exp(-x));
 // printf("J2(x)=%.25le\n Y2(x)=%.25le\n",bessj(2,x),bessy(2,x));
 // x+=0.1;
 // }
 //fiberDdata();
 //dfft(cbeam,8,1);
 //gf1solitonp(profile,50.0e-15,90.0e-15);
 //printf("n=%.6le\n",nsellmeier(swlength,core,corewl));
 //propsolitonps(profilex,profiley,spwidth,1.0e-2,2.985469092,2000.0e-
15); // z/z0=6.0000 ;(11)1.48505697;(11)4.45517090;(21)2.83510875
 stgpsinolm(profiley,spwidth,1.0e-2,1.48505697,2000.0e-15);//4seg. at
3.0lwo=1.62006215;5 seg. at 3.0lwo=2.02507768 ;8seg. at 3.0l:3.24012429
 //free(profilex); //Free profile
 free(profiley);
}



  /* Refractive index approximation using Sellmeier's equation */
double nsellmeier(double wavelength, double B[3], double wlength[3]){
  double w[3],result=0.0;
  int j;
  for(j=0;j<=2;j++)
    w[j]=2*PI*C/wlength[j];
  for(j=0;j<=2;j++)
    result=result+(B[j]*pow(w[j],2.0)/(pow(w[j],2.0)-
pow(2*PI*C/wavelength,2.0)));
  result=sqrt(1+result);
  return result;
  }

  /* First derivative of the material refractive index with respect to
the  radian frequency approximation
   using Sellmeier's equation */
double D1nsellmeier(double wavelength, double B[3], double wlength[3]){
  double w[3],radw,result=0.0;
  int j;
  radw=2.0*PI*C/wavelength;
  for(j=0;j<=2;j++)
    w[j]=2*PI*C/wlength[j];
 for(j=0;j<=2;j++)
    result=result+(B[j]*pow(w[j],2.0)*radw/pow((pow(w[j],2.0)-
pow(radw,2.0)),2.0));
  result=result/nsellmeier(wavelength,B,wlength);
  return result;
  }
  /* Second derivative of the Refracitive index with respect to the
radian frequency approximation
      using Sellmeier's equation */
double D2nsellmeier(double wavelength, double B[3], double wlength[3]){
  double w[3], radw, result=0.0;
  int j;
  radw=2.0*PI*C/wavelength;
  for(j=0;j<=2;j++)
    w[j]=2*PI*C/wlength[j];
  for(j=0;j<=2;j++)
    result=result+(((B[j]*pow(w[j],2.0)*pow((pow(w[j],2.0)-
pow(radw,2.0)),2.0))+
                  (4*B[j]*pow(w[j],2.0)*pow(radw,2.0)*(pow(w[j],2.0)-
pow(radw,2.0))))
                  /pow((pow(w[j],2.0)-pow(radw,2.0)),4.0));
  result=result-pow(D1nsellmeier(wavelength,B,wlength),2.0);
  result=result/nsellmeier(wavelength,B,wlength);
  return result;
  }

/* Group index based on Sellmeier's equation.  Using radian frequency.
Intensity dependent. */
double sgroupnrf(double wavelength, double B[3], double wlength[3]){
  double radw=2.0*PI*C/wavelength;
  return (nsellmeier(wavelength,
B,wlength)+(radw*D1nsellmeier(wavelength,B,wlength)));
  }

 /* Group index based on Sellmeier's equation.  Using wavelength.
Intensity dependent. */
 double sgroupnwl(double wavelength, double B[3], double wlength[3]){
  return (nsellmeier(wavelength,
B,wlength)+(wavelength*D1lbdnsellmeier(wavelength,B,wlength)));
  }

/* Group velocity based on Sellmeier's equation */
double sgvelocity(double wavelength, double B[3], double wlength[3]){
   return C/sgroupnrf(wavelength,B,wlength);
   }

/* Group Velocity Dispersion (GVD) or Beta-2 based on Sellmeier's
equation */
double sgvd(double wavelength, double B[3], double wlength[3]){
  double radw=2.0*PI*C/wavelength;
  return
((1.0/C)*(2*D1nsellmeier(wavelength,B,wlength)+radw*D2nsellmeier(wavele
ngth,B,wlength)));
  }
/* Dispersion Parameter (D) based on Sellmeier's equation */
double SD(double wavelength, double B[3], double wlength[3]){
  double radw=2.0*PI*C/wavelength;
  return (sgvd(wavelength,B,wlength)*(-2*PI*C/pow(wavelength,2.0)));
  }

 /* First derivative of the material refractive index with respect to
the  wavelength approximation
   using Sellmeier's equation */
double D1lbdnsellmeier(double wavelength, double B[3], double
wlength[3]){
  double result=0.0;
  int j;
 for(j=0;j<=2;j++)

result=result+(B[j]*pow(wlength[j],2.0)*wavelength/pow((pow(wavelength,
2.0)-pow(wlength[j],2.0)),2.0));
  result=-result/nsellmeier(wavelength,B,wlength);
  return result;
  }

  /* Second derivative of the Refractive index with respect to the
wavelength approximation
      using Sellmeier's equation */
double D2lbdnsellmeier(double wavelength, double B[3], double
wlength[3]){
  double result=0.0;
  int j;
  for(j=0;j<=2;j++)

result=result+(B[j]*pow(wlength[j],2.0)*(3.0*pow(wavelength,2.0)+pow(wl
ength[j],2.0)))
                /pow((pow(wavelength,2.0)-pow(wlength[j],2.0)),3.0);
  result=result-pow(D1lbdnsellmeier(wavelength,B,wlength),2.0);
  result=result/nsellmeier(wavelength,B,wlength);
  return result;
  }

/* Relative refractive index difference using Sellmeier's equation */
double DELTA(double wavelength){
  double n1= nsellmeier(wavelength,core,corewl);
  double n2=nsellmeier(wavelength,cladding,claddingwl);
  return ((pow(n1,2.0)-pow(n2,2.0))/(2.0*pow(n1,2.0)));
  }

 /* First Derivative of DELTA, the relative refractive index
difference, using Sellmeier's equation */
 double D1DELTA(double wavelength){
   double n1= nsellmeier(wavelength,core,corewl);
   double n2=nsellmeier(wavelength,cladding,claddingwl);
   return ((-
n2/pow(n1,3.0))*((n1*D1lbdnsellmeier(wavelength,cladding,claddingwl))-

 (n2*D1lbdnsellmeier(wavelength,core,corewl))));
   }
```

```
/* Generates composite material dispersion (CMD), waveguide dispersion
(WD), composite profile
     dispersion (CPD), dispersion cross-products (MR) and the total
dispersion (D=CMD+WD+CPD+MR)
*/
void fiberDdata(void){
   FILE *dfp;
  int i;
  double wavelength=swlength,step=1.0e-9,x=0.0; // Step:1.0e-9;  Cut-
off wavelength is 1.30070109e-6 m at fiber core radius, a=(4.0e-6/2.0)
m
  if((dfp=fopen("fddata.dat","w+"))==NULL){
    printf("Unable to open output file fddata.dat\n");
    return;
    }
  x=a;
  for(i=1;i<=10;i++){
   wavelength=cow(x);
  fprintf(dfp,"Cut-off wavelength=%.8le; V=%.8le;
Radius=%.8le\n",wavelength,V(wavelength,x),x);
   x+=0.01e-6;
   }
  wavelength=1.3e-6; /* 1.3e-6;1.45e-6 */
  fprintf(dfp,"Wavelength (x10^-6 m)  Group Velocity (m/s)   Beta1
(s/m)  Beta2 (ps^2/km) D (ps/(km*nm))\n");
  while(wavelength<=1.8e-6){/* 1.8e-6;1.6e-6 */
    fprintf(dfp,"%.8le  %.8le  %.8le  %.8le
%.8le\n",wavelength*1.0e6,egv(wavelength),ebeta1(wavelength),GVD(wavele
ngth)*1.0e24*1.0e3,D(wavelength)*1.0e12*1.0e3/1.0e9);
    wavelength+=step;
   }
   wavelength=1.3e-6; /* 1.3e-6;1.45e-6 */
   fprintf(dfp,"Wavelength (x10^-6 m)  CMD (ps/(km*nm)) WGD
(ps/(km*nm)) CPD (ps/(km*nm)) MR  (ps/(km*nm)) D (ps/(km*nm))\n");
   while(wavelength<=1.80e-6){ /* 1.8e-6;1.6e-6 */
   fprintf(dfp,"%.8le  %.8le  %.8le  %.8le  %.8le
%.8le\n",wavelength*1.0e6,cmd(wavelength)*1.0e12*1.0e3/1.0e9,wgd(wavele
ngth)*1.0e12*1.0e3/1.0e9,cpd(wavelength)*1.0e12*1.0e3/1.0e9,dcp(wavelen
gth)*1.0e12*1.0e3/1.0e9,D(wavelength)*1.0e12*1.0e3/1.0e9);
   wavelength+=step;
    }
  fclose(dfp);
  }

/**********************************************************************
**************************
File: bpm.cpp
Author: P.C. Chimfwembe
Created: 02/11/97
Modified: 04/11/98
***********************************************************************
*************************
Function:
```

```
Provides beam propagation utility functions.
**********************************************************************
************************
*/
# include <stdio.h>
# include <process.h>
# include <stdlib.h>
# include <alloc.h>
# include <time.h>
# include <values.h>
# include <complex>
# include "wlglobal.h"
# include "bpm.h"
# include "admathf.h"
# include "wgbfunc.h"
# include "nlbfunc.h"


//extern ostream_withassign cout;


// Beam profile characteristics
typedef struct{
  double nzlength;
  double cnplength;
  dcomplex bprofile[bpsize];
  } SAVE;


/*********************************************************************
*************************
Name: dfft()
Function:  Calculate the dfft or the idfft of an array of size 2^n of
type complex
Authors: A.V. Oppenhiem and R.W. Schafer (Fortran Original).
       P.C. Chimfwembe (C++ Modified)
Comments:
Fourier transform type converted to physics prefered format,
ie -j --->i. Thus type=1.0 -->Fourier Transform
and type=-1.0 -->Inverse Fourier Transform.
Date: 02/22/97
Modified: 06/25/97
**********************************************************************
************************
*/
void dfft(dcomplex *bprofile,unsigned int size,double type){
   } // See A.V. Oppenhiem and R.W.Schafer Fortran original code.

/*********************************************************************
*************************
Name: gf1solitonp()
Function:  Generates a normalized fundamental soliton pulse profile
Author: P.C. Chimfwembe
```

```
Date: 02/22/97
Modified: 04/14/98
***********************************************************************
*************************
*/

void gf1solitonp(dcomplex *bprofile,double pulse_FWHM,double
windowsize){
  unsigned int i;
  double Ts=windowsize/bpsize;
  double pulse_HW=pulse_FWHM/(2.0*log(1.0+sqrt(2.0)));
  double ts=Ts/pulse_HW;
  for(i=0;i<bpsize;i++)
    bprofile[i]=1.0/cosh(dcomplex((i-bpsize/2.0)*ts,0.0));
  }


/**********************************************************************
**************************
Name: propsolitonp()
Function:  Propagates a normalized soliton pulse in a optical fiber by
Symmetrized
         Split Step Fourier Method.
         Zero attenuation is assumed.
Author: P.C. Chimfwembe
Date: 06/25/97
Modified: 07/08/98
***********************************************************************
*************************
*/
void propsolitonp(dcomplex *bprofile,double pulse_FWHM,
              double zstep /*Fraction of solition period  */
              ,double zlength,double windowsize){
  FILE *dfp,*dfpR,*dfpF;;// Data file and recovery "last profile
recorded" data file.
  dcomplex *U,*U2,*DU2t,*DU2Ut,*Nz1;//Data file and computational
scratchpad files.
  double Ts=windowsize/bpsize; /* Sample interval */
  double beta2=GVD(swlength); /* Fiber effective Beta2 (GVD)*/
  double beta3=6.2e-42;// 3rd order dispersion (s^3/m). Fiber core
radius=2.27e-6 m; Core:13.5m/o GeO2,85.5m/o SiO2;Cladding:SiO2
  double pulse_HW=pulse_FWHM/(2.0*log(1.0+sqrt(2.0)));//To:Half width
at 1/e intensity point.
  double wo=2*PI*C/swlength; // Carrier radian frequency.
  double delta,s,nX3rt;
  double LD,LNL,nzstep,nzlength;// Dispersion length,Nonlinear length
and normalized zstep
  double nz0=PI/2.0,z0;  // Normalized soliton period and soliton
period.
  unsigned int j,k,toprint=1,done=0;
  double cnplength=0.0,snplength=0.0; // Current and sub- normalized
propagation length.
  double N=2.0; //// Soliton order.
  double *W,*F,fpsdmax=0.0; // Time and frequency intensities.
  double shots=10.0; // No. of snap shots.
  SAVE R; // Recovery data array.
  if((U=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer U\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((U2=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer U2\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((DU2t=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer DU2t\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((DU2Ut=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer DU2Ut\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((Nz1=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer Nz1\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((W=(double *)calloc(bpsize,sizeof(double)))==NULL){
    printf("Not enough memory to allocate buffer W\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((F=(double *)calloc(bpsize,sizeof(double)))==NULL){
    printf("Not enough memory to allocate buffer F\n");
    exit(1);  /* terminate program if out of memory */
    }
  //Setting of delta,s, and nX3rt.
  delta=beta3/(6.0*fabs(beta2)*pulse_HW);
  s=2.0/(wo*pulse_HW);
  nX3rt=X3RT/pulse_HW; // Normalized Chi-3 response time.
  LD=pow(pulse_HW,2.0)/fabs(beta2);// Dispersion length
  nzstep=zstep*nz0;// Normalized z step.  Fraction of a soliton period.
  z0=nz0*LD;
  nzlength=zlength/LD;// Normalized z length.
  Px=pow(N,2.0)*fabs(beta2)/(gamma(swlength)*pow(pulse_HW,2.0));
  LNL=1.0/(gamma(swlength)*Px);// Non-linear length
  printf("Signal
Power=%.8le;Beta2=%.8le;Gamma=%.8le\n",Px,beta2,gamma(swlength));
  printf("Soliton Period(z0)=%.8le;Dispersion
Length(LD)=%.8le\n",z0,LD);
  printf("Non-Linear Length(LNL)=%.8le\n",LNL);
  printf("Delta=%.8le; Beta3=%.8le\n",delta,beta3);
  printf("nX3rt=%.8le; s=%.8le; N=%.8le\n",nX3rt,s,N);
  printf("Total Iterations=%.8le\n",ceil(nzlength/nzstep));
  gf1solitonp(bprofile,pulse_FWHM,windowsize);// Initialise soliton
beam profile.
  dfft(bprofile,bpsize,1.0);//Generate FFT
  fpsdmax=pow(abs(bprofile[0]),2.0)/pow(bpsize,2.0);// Power spectral
density for frequency zero - maximum power component.
  dfft(bprofile,bpsize,-1.0);//Generate IFFT
  for(j=0;j<bpsize;j++) // Normalise IFFT
      bprofile[j]=bprofile[j]/((double)(bpsize));
  if((dfpR=fopen("ppdatar.m","a+b"))==NULL){ // Profile complex data
file for recovery.
    printf("Unable to open output file ppdatar.m\n");
    exit(1);
    }
  fread(&R,sizeof(SAVE),1,dfpR); // Detect end of file with fread()
else ftell() or rewind has no effect on current file pointer position.
  if(ftell(dfpR)){ //If file contains past beam profile snap reload to
continue.
    toprint=0; // Reset print flag.  Recovering from power failure
termination of program.
    printf("Recovering from power failure termination ...\n");
    rewind(dfpR);
    fread(&R,sizeof(SAVE),1,dfpR);  // Initialise R context and
profile.
    fclose(dfpR);
    if(ceil(R.cnplength/nzstep)>=ceil(R.nzlength/nzstep)){
      printf("At end of propagation length.\n");
      printf("Please re-initialize total propagation length (m):");
      scanf("%le",&zlength);
      zlength=fabs(zlength);
      nzlength=zlength/LD;// Normalized z length.
      R.nzlength=nzlength;R.cnplength=cnplength=0.0;
      if(!R.nzlength)//If zero or non-real character was entered, then
terminate.
        exit(1); // Exit.
      }
    else{
      nzlength=R.nzlength;cnplength=R.cnplength;
      }
    printf("nzlength/nz0=%.8le;
cnplength/nz0=%.8le\n",R.nzlength/nz0,R.cnplength/nz0);
    printf("Iterations left=%.8le\n",ceil((R.nzlength-
R.cnplength)/nzstep));
    for(k=0;k<bpsize;k++)// Load last saved profile.
      bprofile[k]=R.bprofile[k];
    }
  else
    fclose(dfpR);
  while(ceil(cnplength/nzstep)<=ceil(nzlength/nzstep)){

if((snplength>=nzlength/shots)||(ceil(cnplength/nzstep)==ceil(nzlength/
nzstep))) //1.456500e+04 for 1m. @1.5m 2.184750e+04;@3.0m 4.369490e+04
      toprint=1;
    if(toprint){
      printf("At z/z0 Position:%le\n",cnplength/nz0);
      for(j=0;j<bpsize;j++)
```

```
            W[j]=pow(abs(bprofile[j]),2.0); //Returns the squared
magnitude of bprofile[j].
      if((dfp=fopen("ppdata.m","a+b"))==NULL){// Profile Matlab data
plot file.
        printf("Unable to open output file ppdata.m\n");
        exit(1);
        }
      if((dfpF=fopen("ppdataf.m","a+b"))==NULL){ // Frequency power
spectral density data file.
        printf("Unable to open output file ppdataf.m\n");
        exit(1);
        }
      if((dfpR=fopen("ppdatar.m","w+b"))==NULL){ // Profile complex
data file for recovery.
        printf("Unable to open output file ppdatar.m\n");
        exit(1);
        }
      fwrite(W,sizeof(double),bpsize,dfp);  // Save profile Matlab data
plot file.
      R.nzlength=nzlength;R.cnplength=cnplength;// Save context.
      for(j=0;j<bpsize;j++)
        R.bprofile[j]=bprofile[j];
      fwrite(&R,sizeof(SAVE),1,dfpR);  // Save profile complex data
file for recovery.
      dfft(bprofile,bpsize,1.0);//Generate FFT; Frequency domain
observation.
      for(j=0;j<bpsize/2;j++)//Re-organise discrete frequencies from -
N/2 - 0 - N/2.  FFT has it inside out.

F[j+bpsize/2]=pow(abs(bprofile[j]),2.0)/(pow(bpsize,2.0)*fpsdmax);//Fre
quency power spectral density.
      for(j=bpsize/2;j<bpsize;j++)
        F[j-
bpsize/2]=pow(abs(bprofile[j]),2.0)/(pow(bpsize,2.0)*fpsdmax);//Frequen
cy power spectral density.
      dfft(bprofile,bpsize,-1.0);//Generate IFFT
      for(j=0;j<bpsize;j++) // Normalise IFFT
        bprofile[j]=bprofile[j]/((double)(bpsize));
      fwrite(F,sizeof(double),bpsize,dfpF); // Save frequency power
spectral density data file.
      fclose(dfp);  // Close all data files.
      fclose(dfpR);
      fclose(dfpF);
      snplength=0.0;toprint=0;
      }
    for(k=0;k<bpsize;k++){
      U[k]=bprofile[k];//Store U pow(abs(bprofile[k]),2.0)
      DU2t[k]=U2[k]=dcomplex(pow(abs(bprofile[k]),2.0),0.0);//
Calculate and store abs(U)^2.
      DU2Ut[k]=pow(abs(bprofile[k]),2.0)*bprofile[k];// Calculate
(abs(U)^2)*U.
      }
    L5pDt(DU2t,bpsize,windowsize,pulse_HW);//Determine DU2t and store.
    L5pDt(DU2Ut,bpsize,windowsize,pulse_HW);//Determine DU2Ut and
store.
    for(j=0;j<bpsize;j++)

Nz1[j]=dcomplex(0.0,pow(N,2.0))*(U2[j]+((dcomplex(0.0,s)/U[j])*DU2Ut[j]
)-(nX3rt*DU2t[j]));
    cnplength+=nzstep;// Increment current normalized propagation
length.
    snplength+=nzstep;
    dfft(bprofile,bpsize,1.0);//Generate FFT of beam profile.
    for(j=0;j<bpsize;j++){ // Dispersion step at nz=nzstep/2.
      if((j>=(bpsize/2)-1-7680)&&(j<=(bpsize/2)-1+7680)) //(Was at
7168) Filter very high frequency round-off and
        bprofile[j]=dcomplex(0.0,0.0);// trancation noise with high
pass filter cutoff at fk=+/-1k/N.
      else{
        if(pow(abs(bprofile[j]),2.0)>=1.0e-6)

bprofile[j]=exp((nzstep/2.0)*dispersion_operator(j,pulse_HW,Ts,delta,be
ta2,bpsize,LD))*bprofile[j];
        }
      }
    dfft(bprofile,bpsize,-1.0);//Generate IFFT
    for(j=0;j<bpsize;j++) // Normalise IFFT
      bprofile[j]=bprofile[j]/((double)(bpsize));
    for(j=0;j<bpsize;j++){// Nonlinear step at nz=nzstep.
      if(pow(abs(U[j]),2.0)>=1.0e-6)// Clamp TR noise for N>=2.0.
        bprofile[j]=exp(nzstep*Nz1[j])*bprofile[j];
      }
    dfft(bprofile,bpsize,1.0);//Generate FFT of beam profile.
    for(j=0;j<bpsize;j++){ // Dispersion step at nz=nzstep/2.
      if((j>=(bpsize/2)-1-7680)&&(j<=(bpsize/2)-1+7680)) //(Was at
7168) Filter very high frequency round-off and
        bprofile[j]=dcomplex(0.0,0.0);// trancation noise with high
pass filter cutoff at fk=+/-1k/N.
      else{
        if(pow(abs(bprofile[j]),2.0)>=1.0e-6)

bprofile[j]=exp((nzstep/2.0)*dispersion_operator(j,pulse_HW,Ts,delta,be
ta2,bpsize,LD))*bprofile[j];
        }
      }
    dfft(bprofile,bpsize,-1.0);//Generate IFFT.
    for(j=0;j<bpsize;j++) // Normalise IFFT
      bprofile[j]=bprofile[j]/((double)(bpsize));
    }
  free(U);
  free(U2);
  free(DU2Ut);
  free(DU2t);
  free(Nz1);
  free(W);
  free(F);
  }

/* Dispersion Operator*/

dcomplex dispersion_operator(unsigned int dfreq,double To,double
Ts,double Delta,double beta2,unsigned int size,double LD){
  double radfreq;// Spectral radian frequency
  if(dfreq<=size/2)
    radfreq=2.0*PI*dfreq/(size*Ts);
  else
    radfreq=-2.0*PI*(size-dfreq)/(size*Ts);
  return(dcomplex(-
alpha*LD/2.0,0.5*sgn(beta2)*pow(radfreq*To,2.0))+dcomplex(0.0,-
Delta*pow(radfreq*To,3.0)));
  }

/* Dispersion Operator xy*/

dcomplex dispersion_operatorxy(unsigned int axis,unsigned int
dfreq,double To,double Ts,
                               double Delta,double beta2,unsigned int
size,double LD){
  double radfreq;// Spectral radian frequency
  double DELTA; // DELTA normalized group velocity frame.
  double OFF=0.0; // Effect switches.
  if(dfreq<=size/2)
    radfreq=2.0*PI*dfreq/(size*Ts);
  else
    radfreq=-2.0*PI*(size-dfreq)/(size*Ts);
  DELTA=(dn/C)*To/(2.0*fabs(beta2));
  if(axis==XAXIS) // X/1 polarisation dispersion operator.
    return(dcomplex(-
alpha*LD/2.0,(DELTA*radfreq*To)+0.5*sgn(beta2)*pow(radfreq*To,2.0))+dco
mplex(0.0,-Delta*pow(radfreq*To,3.0)));
  else // Y/2 polarisation dispersion operator.
    return(dcomplex(-alpha*LD/2.0,-
(DELTA*radfreq*To)+0.5*sgn(beta2)*pow(radfreq*To,2.0))+dcomplex(0.0,-
Delta*pow(radfreq*To,3.0)));
  }




/**********************************************************************
*************************
 Name: propsolitonps()
Function:  Co-propagates two orthogonal normalized soliton pulses in a
optical fiber by Symmetrized Split Step Fourier Method.  Zero
attenuation is assumed.
Author: P.C. Chimfwembe
Date: 03/19/98
Modified: 04/14/98
```

```
**********************************************************************
*************************
*/
void propsolitonps(dcomplex *bprofilex,dcomplex *bprofiley,double
pulse_FWHM,
                         double zstep, /*Fraction of solition period
*/
                         double zlength,double windowsize){
  FILE *dfpx,*dfpy,*dfpRx,*dfpRy,*dfpFx,*dfpFy; //Data file pointers.
  dcomplex *Ux,*aUx2,*DaUx2t,*DaUx2Uxt,*Nz1x;//Data computational
scratchpad array pointers for pulse x.
  dcomplex *Uy,*aUy2,*DaUy2t,*DaUy2Uyt,*Nz1y;//Data computational
scratchpad array pointers for pulse y.
  dcomplex *DaUx2Uyt,*DaUy2Uxt;//Data computational scratchpad array
pointers for cross terms of pulses x and y.
  double Ts=windowsize/bpsize; /* Sample interval and normalized sample
interval */
  double beta2=GVD(swlength); /* Fiber effective Beta2 (GVDx=GVDy) */
  double beta3=6.2e-42;//x and y 3rd order dispersions (s^3/m)
(TODx=TODy). Fiber core radius=2.27e-6 m; Core:13.5m/o GeO2,85.5m/o
SiO2;Cladding:SiO2
  double pulse_HW=pulse_FWHM/(2.0*log(1.0+sqrt(2.0)));//To:Half width
at 1/e intensity point.
  double wo=2*PI*C/swlength; // Carrier radian frequency.
  double delta,s,nX3rt;
  double LD,LNLx,LNLy,nzstep,nzlength;// Dispersion length,Nonlinear
length and normalized zstep.
  double nz0=PI/2.0,z0;  // Normalized soliton period and soliton
period.
  unsigned int j,k,toprint=1;
  double cnplength=0.0,snplength=0.0; // Current and sub- normalized
propagation length.
  double ANGLE=(30.0/180.0)*PI;//(84.0/180.0)*PI; // Splitting soliton
amplitude angle.
  double Nx=1.24*cos(ANGLE)*0.0,Ny=1.24*sin(ANGLE);
//(3)30;(2)60;(1)84(1.24SDG);45 (STG) (SDG 45<ANGLE>45)cos(ANGLE),
sin(ANGLE) Soliton order for x and y axes.
  double OFF=0.0; // Effect switch.
  double  *Wx,*Wy,*Fx,*Fy,fpsdxmax=0.0,fpsdymax=0.0; // Time and
frequency intensities.
  double shots=10.0; // No. of snap shots.
  SAVE Rx,Ry; // Recovery data arrays.
  if((Ux=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer Ux\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((aUx2=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer aUx2\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((DaUx2t=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer DaUx2t\n");
    exit(1);  /* terminate program if out of memory */
  }
if((DaUx2Uxt=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
  printf("Not enough memory to allocate buffer DaUx2Uxt\n");
  exit(1);  /* terminate program if out of memory */
  }
if((Nz1x=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
  printf("Not enough memory to allocate buffer Nz1x\n");
  exit(1);  /* terminate program if out of memory */
  }
if((Uy=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
  printf("Not enough memory to allocate buffer Uy\n");
  exit(1);  /* terminate program if out of memory */
  }
if((aUy2=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
  printf("Not enough memory to allocate buffer aUy2\n");
  exit(1);  /* terminate program if out of memory */
  }
if((DaUy2t=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
  printf("Not enough memory to allocate buffer DaUy2t\n");
  exit(1);  /* terminate program if out of memory */
  }
if((DaUy2Uyt=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
  printf("Not enough memory to allocate buffer DU2Uty\n");
  exit(1);  /* terminate program if out of memory */
  }
if((Nz1y=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
  printf("Not enough memory to allocate buffer Nz1y\n");
  exit(1);  /* terminate program if out of memory */
  }
if((DaUx2Uyt=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
  printf("Not enough memory to allocate buffer DU2Uty\n");
  exit(1);  /* terminate program if out of memory */
  }
if((DaUy2Uxt=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
  printf("Not enough memory to allocate buffer DU2Uty\n");
  exit(1);  /* terminate program if out of memory */
  }
if((Wx=(double *)calloc(bpsize,sizeof(double)))==NULL){
  printf("Not enough memory to allocate buffer Wx\n");
  exit(1);  /* terminate program if out of memory */
  }
if((Wy=(double *)calloc(bpsize,sizeof(double)))==NULL){
  printf("Not enough memory to allocate buffer Wy\n");
  exit(1);  /* terminate program if out of memory */
  }
if((Fx=(double *)calloc(bpsize,sizeof(double)))==NULL){
  printf("Not enough memory to allocate buffer Fx\n");
  exit(1);  /* terminate program if out of memory */
  }
if((Fy=(double *)calloc(bpsize,sizeof(double)))==NULL){
  printf("Not enough memory to allocate buffer Fy\n");
  exit(1);  /* terminate program if out of memory */
  }
  beta3=6.2e-42;// Fiber core radius=2.27e-6 m; Core:13.5m/o
GeO2,85.5m/o SiO2;Cladding:SiO2
  delta=beta3/(6.0*fabs(beta2)*pulse_HW);
  s=2.0/(wo*pulse_HW);
  nX3rt=X3RT/pulse_HW; // Normalized Chi-3 response time.
  LD=pow(pulse_HW,2.0)/fabs(beta2);// Dispersion length.
  z0=LD*PI/2.0;  // Soliton period in x polarization as reference
plane.
  nzstep=zstep*nz0;// Normalized z step.  Fraction of a soliton period.
  nzlength=zlength/LD;// Normalized z length referenced to x
polarization plane.
  Px=pow(Nx,2.0)*fabs(beta2)/(gamma(swlength)*pow(pulse_HW,2.0));
  Py=pow(Ny,2.0)*fabs(beta2)/(gamma(swlength)*pow(pulse_HW,2.0));
  if(Px!=0.0)
    LNLx=1.0/(gamma(swlength)*Px);// Non-linear length of signal.
  if(Py!=0.0)
    LNLy=1.0/(gamma(swlength)*Py);// Non-linear length of pump.
  printf("Peak
Powerx=%.8le;Beta2=%.8le;Gamma=%.8le\n",Px,beta2,gamma(swlength));
  printf("Peak Powery=%.8le\n",Py);
  printf("Soliton Period(z0)=%.8le;Dispersion
Length(LD)=%.8le\n",z0,LD);
  if(Px!=0.0)
    printf("Non-Linear Lengthx(LNLx)=%.8le\n",LNLx);
  else
    printf("Non-Linear Lengthx(LNLx)=infinity\n");
  if(Py!=0.0)
    printf("Non-Linear Lengthy(LNLy)=%.8le\n",LNLy);
  else
    printf("Non-Linear Lengthy(LNLy)=infinity\n");
  printf("s=%.8le;nX3rt=%.8le\nDelta=%.8le\n",s,nX3rt,delta);
  printf("Beta3=%.8le\nNx=%.8le;Ny=%.8le\n",beta3,Nx,Ny);
  printf("Total Iterations=%.8le\n",ceil(nzlength/nzstep));
  gf1solitonp(bprofilex,pulse_FWHM,windowsize);// Initialise soliton
beam profile x.
  gf1solitonp(bprofiley,pulse_FWHM,windowsize);// Initialise soliton
beam profile y.
  for(j=0;j<bpsize;j++){
    bprofilex[j]=bprofilex[j]*Nx;
    bprofiley[j]=bprofiley[j]*Ny;
    }
  dfft(bprofilex,bpsize,1.0);//Generate FFT of beam profile x.
  dfft(bprofiley,bpsize,1.0);//Generate FFT of beam profile y.
  fpsdxmax=pow(abs(bprofilex[0]),2.0)/pow(bpsize,2.0);// Power spectral
density for frequency zero - maximum power component.
  fpsdymax=pow(abs(bprofiley[0]),2.0)/pow(bpsize,2.0);
  dfft(bprofilex,bpsize,-1.0);//Generate IFFT for profile x.
  dfft(bprofiley,bpsize,-1.0);//Generate IFFT for profile y.
  for(j=0;j<bpsize;j++){ // Normalise IFFT for profiles x and y.
    bprofilex[j]=bprofilex[j]/((double)(bpsize));
    bprofiley[j]=bprofiley[j]/((double)(bpsize));
    }
  if((dfpRx=fopen("ppdatarx.m","a+b"))==NULL){ // Profile complex data
file for recovery.
    printf("Unable to open output file ppdatarx.m\n");
    return;
    }
  if((dfpRy=fopen("ppdatary.m","a+b"))==NULL){ // Profile complex data
file for recovery.
    printf("Unable to open output file ppdatary.m\n");
    return;
    }
  fread(&Rx,sizeof(SAVE),1,dfpRx); // Detect end of file with fread()
else ftell() or rewind has no effect on current file pointer position.
  fread(&Ry,sizeof(SAVE),1,dfpRy);
  if(ftell(dfpRx)&&ftell(dfpRy)){ //If file contains past beam profile
snap shots reload to continue.
    toprint=0; // Reset print flag.
    printf("Recovering from power failure termination ...\n");
    rewind(dfpRx);rewind(dfpRy);
    fread(&Rx,sizeof(SAVE),1,dfpRx);fread(&Ry,sizeof(SAVE),1,dfpRy); //
Initialise Rx and Ry context and profile.
    fclose(dfpRx);fclose(dfpRy);
    if(ceil(Rx.cnplength/nzstep)>=ceil(Rx.nzlength/nzstep)){ // x
reference plane.
      printf("At end of propagation length.\n");
      printf("Please re-initialize total propagation length (m):");
      scanf("%le",&zlength);
      zlength=fabs(zlength);
      nzlength=zlength/LD;// Normalized z length in x reference plane.
      Rx.nzlength=nzlength;Rx.cnplength=cnplength=0.0;
      if(!Rx.nzlength)//If zero or non-real character was entered, then
terminate.
        exit(1); // Exit.
      }
    else{
      nzlength=Rx.nzlength;cnplength=Rx.cnplength;
      }
    printf("nzlength/nz0=%.8le;
cnplength/nz0=%.8le\n",Rx.nzlength/nz0,Rx.cnplength/nz0);
    printf("Iterations left=%.8le\n",ceil((Rx.nzlength-
Rx.cnplength)/nzstep));
    for(k=0;k<bpsize;k++){// Load last saved profilex and profiley.
      bprofilex[k]=Rx.bprofile[k];
      bprofiley[k]=Ry.bprofile[k];
      }
    }
  else{
    fclose(dfpRx);fclose(dfpRy);
    }
  while(ceil(cnplength/nzstep)<=ceil(nzlength/nzstep)){

if((snplength>=nzlength/shots)||(ceil(cnplength/nzstep)==ceil(nzlength/
nzstep)))
      toprint=1;
    if(toprint){
      printf("At z/z0 Position:%le\n",cnplength/nz0);
      for(j=0;j<bpsize;j++){
        Wx[j]=pow(abs(bprofilex[j]),2.0); //Returns the squared
magnitude of bprofilex[j] and bprofiley[j].
        Wy[j]=pow(abs(bprofiley[j]),2.0);
            }
      if((dfpx=fopen("ppdatax.m","a+b"))==NULL){// Profile Matlab data
plot file.
        printf("Unable to open output file ppdatax.m\n");
        return;
        }
      if((dfpy=fopen("ppdatay.m","a+b"))==NULL){// Profile Matlab data
plot file.
        printf("Unable to open output file ppdatay.m\n");
        return;
        }
      if((dfpFx=fopen("ppdatafx.m","a+b"))==NULL){ // Frequency power
spectral density data file.
        printf("Unable to open output file ppdatafx.m\n");
        exit(1);
        }
      if((dfpFy=fopen("ppdatafy.m","a+b"))==NULL){ // Frequency power
spectral density data file.
        printf("Unable to open output file ppdatafy.m\n");
        exit(1);
        }
      if((dfpRx=fopen("ppdatarx.m","w+b"))==NULL){ // Profile complex
data file for recovery.
        printf("Unable to open output file ppdatarx.m\n");
        return;
        }
      if((dfpRy=fopen("ppdatary.m","w+b"))==NULL){ // Profile complex
data file for recovery.
        printf("Unable to open output file ppdatary.m\n");
        return;
        }
      fwrite(Wx,sizeof(double),bpsize,dfpx);// Save profile Matlab data
plot files.
      fwrite(Wy,sizeof(double),bpsize,dfpy);
      Rx.nzlength=nzlength;Rx.cnplength=cnplength;// Save context.
      for(j=0;j<bpsize;j++){
        Rx.bprofile[j]=bprofilex[j];
        Ry.bprofile[j]=bprofiley[j];
        }

fwrite(&Rx,sizeof(SAVE),1,dfpRx);fwrite(&Ry,sizeof(SAVE),1,dfpRy);  //
Save profile complex data files for recovery.
      dfft(bprofilex,bpsize,1.0);//Generate FFT of beam profile x.
      dfft(bprofiley,bpsize,1.0);//Generate FFT of beam profile y.
      for(j=0;j<bpsize/2;j++){//Re-organise discrete frequencies from -
N/2 - 0 - N/2.  FFT has it inside out.
        if(Px!=0.0)
Fx[j+bpsize/2]=pow(abs(bprofilex[j]),2.0)/(pow(bpsize,2.0)*fpsdxmax);//
Frequency power spectral density.
        else
          Fx[j+bpsize/2]=0.0;
        if(Py!=0.0)

Fy[j+bpsize/2]=pow(abs(bprofiley[j]),2.0)/(pow(bpsize,2.0)*fpsdymax);
        else
          Fy[j+bpsize/2]=0.0;
        }
      for(j=bpsize/2;j<bpsize;j++){
        if(Px!=0.0)
          Fx[j-
bpsize/2]=pow(abs(bprofilex[j]),2.0)/(pow(bpsize,2.0)*fpsdxmax);//Frequ
ency power spectral density.
        else
          Fx[j-bpsize/2]=0.0;
        if(Py!=0.0)
          Fy[j-
bpsize/2]=pow(abs(bprofiley[j]),2.0)/(pow(bpsize,2.0)*fpsdymax);
        else
          Fy[j-bpsize/2]=0.0;
        }
      dfft(bprofilex,bpsize,-1.0);//Generate IFFT for profile x.
      dfft(bprofiley,bpsize,-1.0);//Generate IFFT for profile y.
      for(j=0;j<bpsize;j++){ // Normalise IFFT for profiles x and y.
        bprofilex[j]=bprofilex[j]/((double)(bpsize));
        bprofiley[j]=bprofiley[j]/((double)(bpsize));
        }
      fwrite(Fx,sizeof(double),bpsize,dfpFx); // Save frequency power
spectral density data file.
      fwrite(Fy,sizeof(double),bpsize,dfpFy);
      fclose(dfpx);fclose(dfpy); // Close all data files.
      fclose(dfpRx);fclose(dfpRy);
      fclose(dfpFx);fclose(dfpFy);
      snplength=0.0;toprint=0;
      }
    for(k=0;k<bpsize;k++){
      Ux[k]=bprofilex[k];//Store Ux.
      Uy[k]=bprofiley[k];//Store Uy.
      DaUx2t[k]=aUx2[k]=dcomplex(pow(abs(bprofilex[k]),2.0),0.0);//
Calculate and store abs(Ux)^2.
      DaUy2t[k]=aUy2[k]=dcomplex(pow(abs(bprofiley[k]),2.0),0.0);//
Calculate and store abs(Uy)^2.
      DaUx2Uxt[k]=pow(abs(bprofilex[k]),2.0)*bprofilex[k];// Calculate
(abs(Ux)^2)*Ux.
      DaUy2Uyt[k]=pow(abs(bprofiley[k]),2.0)*bprofiley[k];// Calculate
(abs(Uy)^2)*Uy.
      DaUx2Uyt[k]=pow(abs(bprofilex[k]),2.0)*bprofiley[k];// Calculate
(abs(Ux)^2)*Uy.
      DaUy2Uxt[k]=pow(abs(bprofiley[k]),2.0)*bprofilex[k];// Calculate
(abs(Uy)^2)*Ux.
```

```
        } // Time derivatives.
    L5pDt(DaUx2t,bpsize,windowsize,pulse_HW);//Determine DaUx2t and
store.
    L5pDt(DaUy2t,bpsize,windowsize,pulse_HW);//Determine DaUy2t and
store.
    L5pDt(DaUx2Uxt,bpsize,windowsize,pulse_HW);//Determine DaUx2Uxt and
store.
    L5pDt(DaUy2Uyt,bpsize,windowsize,pulse_HW);//Determine DaUy2Uyt and
store.
    L5pDt(DaUx2Uyt,bpsize,windowsize,pulse_HW);//Determine DaUx2Uyt and
store.
    L5pDt(DaUy2Uxt,bpsize,windowsize,pulse_HW);//Determine DaUy2Uxt and
store.
    for(j=0;j<bpsize;j++){//  Compute nonlinear operator.
      if(Px!=0.0)
        Nz1x[j]=dcomplex(0.0,1.0)*(aUx2[j]+((2.0/3.0)*aUy2[j])

+((dcomplex(0.0,s)/Ux[j])*(DaUx2Uxt[j]+((2.0/3.0)*DaUy2Uxt[j])))
               -(nX3rt*(DaUx2t[j]+((2.0/3.0)*DaUy2t[j])))); // x
polarized pulse.
      else
        Nz1x[j]=dcomplex(0.0,0.0);
      if(Py!=0.0)
        Nz1y[j]=dcomplex(0.0,1.0)*(aUy2[j]+((2.0/3.0)*aUx2[j])

+((dcomplex(0.0,s)/Uy[j])*(DaUy2Uyt[j]+((2.0/3.0)*DaUx2Uyt[j])))
               -(nX3rt*(DaUy2t[j]+((2.0/3.0)*DaUx2t[j]))));  // y
polarized pulse.
      else
        Nz1y[j]=dcomplex(0.0,0.0);
      }
    cnplength+=nzstep;// Increment current normalized propagation
length.
    snplength+=nzstep;
    dfft(bprofilex,bpsize,1.0);//Generate FFT of beam profile x.
    dfft(bprofiley,bpsize,1.0);//Generate FFT of beam profile y.
    for(j=0;j<bpsize;j++){ // Dispersion step at nz=nzstep/2.
      if((j>=(bpsize/2)-1-7680)&&(j<=(bpsize/2)-1+7680)){ //(Was at
7168) Filter very high frequency round-off and
        bprofilex[j]=dcomplex(0.0,0.0);// trancation noise with high
pass filter cutoff at fk=+/-1k/N.
        bprofiley[j]=dcomplex(0.0,0.0);//7168(7k);8192(8k)
        }
      else{
        if(pow(abs(bprofilex[j]),2.0)>=1.0e-6)

bprofilex[j]=exp((nzstep/2.0)*dispersion_operatorxy(XAXIS,j,pulse_HW,Ts
,delta,beta2,bpsize,LD))*bprofilex[j];
        if(pow(abs(bprofiley[j]),2.0)>=1.0e-6)

bprofiley[j]=exp((nzstep/2.0)*dispersion_operatorxy(YAXIS,j,pulse_HW,Ts
,delta,beta2,bpsize,LD))*bprofiley[j];
        }
        }
      dfft(bprofilex,bpsize,-1.0);//Generate IFFT for profile x.
      dfft(bprofiley,bpsize,-1.0);//Generate IFFT for profile y.
      for(j=0;j<bpsize;j++){ // Normalise IFFT for profiles x and y.
        bprofilex[j]=bprofilex[j]/((double)(bpsize));
        bprofiley[j]=bprofiley[j]/((double)(bpsize));
        }
      for(j=0;j<bpsize;j++){// Nonlinear step at nz=nzstep.
        bprofilex[j]=exp(nzstep*Nz1x[j])*bprofilex[j];
        bprofiley[j]=exp(nzstep*Nz1y[j])*bprofiley[j];
        }
      dfft(bprofilex,bpsize,1.0);//Generate FFT of beam profile x.
      dfft(bprofiley,bpsize,1.0);//Generate FFT of beam profile y.
      for(j=0;j<bpsize;j++){ // Dispersion step at nz=nzstep/2.
        if((j>=(bpsize/2)-1-7680)&&(j<=(bpsize/2)-1+7680)){ //(Was at
7168) Filter very high frequency round-off and
          bprofilex[j]=dcomplex(0.0,0.0);// trancation noise with high
pass filter cutoff at fk=+/-1k/N.
          bprofiley[j]=dcomplex(0.0,0.0);//7168(7k);8192(8k)
          }
        else{
          if(pow(abs(bprofilex[j]),2.0)>=1.0e-6)

bprofilex[j]=exp((nzstep/2.0)*dispersion_operatorxy(XAXIS,j,pulse_HW,Ts
,delta,beta2,bpsize,LD))*bprofilex[j];
          if(pow(abs(bprofiley[j]),2.0)>=1.0e-6)

bprofiley[j]=exp((nzstep/2.0)*dispersion_operatorxy(YAXIS,j,pulse_HW,Ts
,delta,beta2,bpsize,LD))*bprofiley[j];
          }
        }
      dfft(bprofilex,bpsize,-1.0);//Generate IFFT for profile x.
      dfft(bprofiley,bpsize,-1.0);//Generate IFFT for profile y.
      for(j=0;j<bpsize;j++){ // Normalise IFFT for profiles x and y.
        bprofilex[j]=bprofilex[j]/((double)(bpsize));
        bprofiley[j]=bprofiley[j]/((double)(bpsize));
        }
      }
    free(Ux);
    free(aUx2);
    free(DaUx2t);
    free(DaUx2Uxt);
    free(Nz1x);
    free(Uy);
    free(aUy2);
    free(DaUy2t);
    free(DaUy2Uyt);
    free(Nz1y);
    free(DaUx2Uyt);
    free(DaUy2Uxt);
    free(Wx);
    free(Wy);
    free(Fx);
```

```
  free(Fy);
  }

/*********************************************************************
*************************
File: bpma.cpp
Author: P.C. Chimfwembe
Created: 02/11/97
Modified: 03/31/98
*********************************************************************
************************
Function:
Provides beam propagation utility functions.
*********************************************************************
************************
*/
# include <stdio.h>
# include <process.h>
# include <stdlib.h>
# include <alloc.h>
# include <time.h>
# include <values.h>
# include <complex>
# include "wlglobal.h"
# include "bpm.h"
# include "admathf.h"
# include "wgbfunc.h"
# include "nlbfunc.h"


//extern ostream_withassign cout;


// Beam profile characteristics
typedef struct{
  double nzlength;
  double cnplength;
  dcomplex bprofile[bpsize];
  } SAVE;

typedef struct{
  unsigned int segmentcounter;
  double fpsdx1max;
  double fpsdy1max;
  double fpsdy2max;
  dcomplex bprofilex1[bpsize];
  dcomplex bprofiley1[bpsize];
  dcomplex bprofiley2[bpsize];
  } SAVESWITCH;


/*********************************************************************
*************************
```

```
Name: dfft()
Function:  Calculate the dfft or the idfft of an array of size 2^n of
type complex
Authors: A.V. Oppenhiem and R.W. Schafer (Fortran Original).
       P.C. Chimfwembe (C++ Modified)
Comments:
Fourier transform type converted to physics prefered format,
ie -j --->i. Thus type=1.0 -->Fourier Transform
and type=-1.0 -->Inverse Fourier Transform.
Date: 02/22/97
Modified: 06/25/97
**********************************************************************
*************************
*/
void dfft(dcomplex *bprofile,unsigned int size,double type){
   } // See A.V. Oppenhiem and R.W.Schafer Fortran original code.

/*********************************************************************
**************************
Name: gf1solitonp()
Function:  Generates a normalized fundamental soliton pulse profile
Author: P.C. Chimfwembe
Date: 02/22/97
Modified: 04/14/98
**********************************************************************
*************************
*/

void gf1solitonp(dcomplex *bprofile,double pulse_FWHM,double
windowsize){
  unsigned int i;
  double Ts=windowsize/bpsize;
  double pulse_HW=pulse_FWHM/(2.0*log(1.0+sqrt(2.0)));
  double ts=Ts/pulse_HW;
  for(i=0;i<bpsize;i++)
    bprofile[i]=1.0/cosh(dcomplex((i-bpsize/2.0)*ts,0.0));
  }


/*********************************************************************
**************************
Name: propsolitonp()
Function:  Propagates a normalized soliton pulse in a optical fiber by
Symmetrized
         Split Step Fourier Method.
         Zero attenuation is assumed.
Author: P.C. Chimfwembe
Date: 06/25/97
Modified: 07/08/98
**********************************************************************
*************************
*/
void propsolitonp(dcomplex *bprofile,double pulse_FWHM,
              double zstep /*Fraction of soliton period  */
              ,double zlength,double windowsize){
  FILE *dfp,*dfpR,*dfpF;;// Data file and recovery "last profile
recorded" data file.
  dcomplex *U,*U2,*DU2t,*DU2Ut,*Nz1;//Data file and computational
scratchpad files.
  double Ts=windowsize/bpsize; /* Sample interval */
  double beta2=GVD(swlength); /* Fiber effective Beta2 (GVD)*/
  double beta3=6.2e-42;// 3rd order dispersion (s^3/m). Fiber core
radius=2.27e-6 m; Core:13.5m/o GeO2,85.5m/o SiO2;Cladding:SiO2
  double pulse_HW=pulse_FWHM/(2.0*log(1.0+sqrt(2.0)));//To:Half width
at 1/e intensity point.
  double wo=2*PI*C/swlength; // Carrier radian frequency.
  double delta,s,nX3rt;
  double LD,LNL,nzstep,nzlength;// Dispersion length,Nonlinear length
and normalized zstep
  double nz0=PI/2.0,z0;  // Normalized soliton period and soliton
period.
  unsigned int j,k,toprint=1,done=0;
  double cnplength=0.0,snplength=0.0; // Current and sub- normalized
propagation length.
  double N=2.0; //// Soliton order.
  double *W,*F,fpsdmax=0.0; // Time and frequency intensities.
  double shots=10.0; // No. of snap shots.
  SAVE R; // Recovery data array.
  if((U=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer U\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((U2=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer U2\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((DU2t=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer DU2t\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((DU2Ut=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer DU2Ut\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((Nz1=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer Nz1\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((W=(double *)calloc(bpsize,sizeof(double)))==NULL){
    printf("Not enough memory to allocate buffer W\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((F=(double *)calloc(bpsize,sizeof(double)))==NULL){
    printf("Not enough memory to allocate buffer F\n");
    exit(1);  /* terminate program if out of memory */
    }
  //Setting of delta,s, and nX3rt.
  delta=beta3/(6.0*fabs(beta2)*pulse_HW);
  s=2.0/(wo*pulse_HW);
  nX3rt=X3RT/pulse_HW; // Normalized Chi-3 response time.
  LD=pow(pulse_HW,2.0)/fabs(beta2);// Dispersion length
  nzstep=zstep*nz0;// Normalized z step.  Fraction of a soliton period.
  z0=nz0*LD;
  nzlength=zlength/LD;// Normalized z length.
  Px=pow(N,2.0)*fabs(beta2)/(gamma(swlength)*pow(pulse_HW,2.0));
  LNL=1.0/(gamma(swlength)*Px);// Non-linear length
  printf("Signal
Power=%.8le;Beta2=%.8le;Gamma=%.8le\n",Px,beta2,gamma(swlength));
  printf("Soliton Period(z0)=%.8le;Dispersion
Length(LD)=%.8le\n",z0,LD);
  printf("Non-Linear Length(LNL)=%.8le\n",LNL);
  printf("Delta=%.8le; Beta3=%.8le\n",delta,beta3);
  printf("nX3rt=%.8le; s=%.8le; N=%.8le\n",nX3rt,s,N);
  printf("Total Iterations=%.8le\n",ceil(nzlength/nzstep));
  gf1solitonp(bprofile,pulse_FWHM,windowsize);// Initialise soliton
beam profile.
  dfft(bprofile,bpsize,1.0);//Generate FFT
  fpsdmax=pow(abs(bprofile[0]),2.0)/pow(bpsize,2.0);// Power spectral
density for frequency zero - maximum power component.
  dfft(bprofile,bpsize,-1.0);//Generate IFFT
  for(j=0;j<bpsize;j++) // Normalise IFFT
      bprofile[j]=bprofile[j]/((double)(bpsize));
  if((dfpR=fopen("ppdatar.m","a+b"))==NULL){ // Profile complex data
file for recovery.
    printf("Unable to open output file ppdatar.m\n");
    exit(1);
    }
  fread(&R,sizeof(SAVE),1,dfpR); // Detect end of file with fread()
else ftell() or rewind has no effect on current file pointer position.
  if(ftell(dfpR)){ //If file contains past beam profile snap reload to
continue.
    toprint=0; // Reset print flag.  Recovering from power failure
termination of program.
    printf("Recovering from power failure termination ...\n");
    rewind(dfpR);
    fread(&R,sizeof(SAVE),1,dfpR);  // Initialise R context and
profile.
    fclose(dfpR);
    if(ceil(R.cnplength/nzstep)>=ceil(R.nzlength/nzstep)){
      printf("At end of propagation length.\n");
      printf("Please re-initialize total propagation length (m):");
      scanf("%le",&zlength);
      zlength=fabs(zlength);
      nzlength=zlength/LD;// Normalized z length.
      R.nzlength=nzlength;R.cnplength=cnplength=0.0;
      if(!R.nzlength)//If zero or non-real character was entered, then
terminate.
        exit(1); // Exit.
      }
    else{
      nzlength=R.nzlength;cnplength=R.cnplength;
      }
    printf("nzlength/nz0=%.8le;
cnplength/nz0=%.8le\n",R.nzlength/nz0,R.cnplength/nz0);
    printf("Iterations left=%.8le\n",ceil((R.nzlength-
R.cnplength)/nzstep));
    for(k=0;k<bpsize;k++)// Load last saved profile.
      bprofile[k]=R.bprofile[k];
    }
  else
    fclose(dfpR);
  while(ceil(cnplength/nzstep)<=ceil(nzlength/nzstep)){

if((snplength>=nzlength/shots)||(ceil(cnplength/nzstep)==ceil(nzlength/
nzstep))) //1.456500e+04 for 1m. @1.5m 2.184750e+04;@3.0m 4.369490e+04
      toprint=1;
    if(toprint){
      printf("At z/z0 Position:%le\n",cnplength/nz0);
      for(j=0;j<bpsize;j++)
           W[j]=pow(abs(bprofile[j]),2.0); //Returns the squared
magnitude of bprofile[j].
      if((dfp=fopen("ppdata.m","a+b"))==NULL){// Profile Matlab data
plot file.
        printf("Unable to open output file ppdata.m\n");
        exit(1);
        }
      if((dfpF=fopen("ppdataf.m","a+b"))==NULL){ // Frequency power
spectral density data file.
        printf("Unable to open output file ppdataf.m\n");
        exit(1);
        }
      if((dfpR=fopen("ppdatar.m","w+b"))==NULL){ // Profile complex
data file for recovery.
        printf("Unable to open output file ppdatar.m\n");
        exit(1);
        }
      fwrite(W,sizeof(double),bpsize,dfp);  // Save profile Matlab data
plot file.
      R.nzlength=nzlength;R.cnplength=cnplength;// Save context.
      for(j=0;j<bpsize;j++)
        R.bprofile[j]=bprofile[j];
      fwrite(&R,sizeof(SAVE),1,dfpR);  // Save profile complex data
file for recovery.
      dfft(bprofile,bpsize,1.0);//Generate FFT; Frequency domain
observation.
      for(j=0;j<bpsize/2;j++)//Re-organise discrete frequencies from -
N/2 - 0 - N/2.  FFT has it inside out.

F[j+bpsize/2]=pow(abs(bprofile[j]),2.0)/(pow(bpsize,2.0)*fpsdmax);//Fre
quency power spectral density.
      for(j=bpsize/2;j<bpsize;j++)
```

```
         F[j-
bpsize/2]=pow(abs(bprofile[j]),2.0)/(pow(bpsize,2.0)*fpsdmax);//Frequen
cy power spectral density.
      dfft(bprofile,bpsize,-1.0);//Generate IFFT
      for(j=0;j<bpsize;j++) // Normalise IFFT
        bprofile[j]=bprofile[j]/((double)(bpsize));
      fwrite(F,sizeof(double),bpsize,dfpF); // Save frequency power
spectral density data file.
      fclose(dfp);  // Close all data files.
      fclose(dfpR);
      fclose(dfpF);
      snplength=0.0;toprint=0;
      }
    for(k=0;k<bpsize;k++){
      U[k]=bprofile[k];//Store U pow(abs(bprofile[k]),2.0)
      DU2t[k]=U2[k]=dcomplex(pow(abs(bprofile[k]),2.0),0.0);//
Calculate and store abs(U)^2.
      DU2Ut[k]=pow(abs(bprofile[k]),2.0)*bprofile[k];// Calculate
(abs(U)^2)*U.
      }
    L5pDt(DU2t,bpsize,windowsize,pulse_HW);//Determine DU2t and store.
    L5pDt(DU2Ut,bpsize,windowsize,pulse_HW);//Determine DU2Ut and
store.
    for(j=0;j<bpsize;j++)

Nz1[j]=dcomplex(0.0,pow(N,2.0))*(U2[j]+((dcomplex(0.0,s)/U[j])*DU2Ut[j]
)-(nX3rt*DU2t[j]));
    cnplength+=nzstep;// Increment current normalized propagation
length.
    snplength+=nzstep;
    dfft(bprofile,bpsize,1.0);//Generate FFT of beam profile.
    for(j=0;j<bpsize;j++){ // Dispersion step at nz=nzstep/2.
      if((j>=(bpsize/2)-1-7680)&&(j<=(bpsize/2)-1+7680)) //(Was at 7168
now 7680) Filter very high frequency round-off and
        bprofile[j]=dcomplex(0.0,0.0);// trancation noise with high
pass filter cutoff at fk=+/-1k/N.
      else{
        if(pow(abs(bprofile[j]),2.0)>=1.0e-6)

bprofile[j]=exp((nzstep/2.0)*dispersion_operator(j,pulse_HW,Ts,delta,be
ta2,bpsize,LD))*bprofile[j];
        }
      }
    dfft(bprofile,bpsize,-1.0);//Generate IFFT
    for(j=0;j<bpsize;j++) // Normalise IFFT
      bprofile[j]=bprofile[j]/((double)(bpsize));
    for(j=0;j<bpsize;j++){// Nonlinear step at nz=nzstep.
      if(pow(abs(U[j]),2.0)>=1.0e-6)// Clamp TR noise for N>=2.0.
        bprofile[j]=exp(nzstep*Nz1[j])*bprofile[j];
      }
    dfft(bprofile,bpsize,1.0);//Generate FFT of beam profile.
    for(j=0;j<bpsize;j++){ // Dispersion step at nz=nzstep/2.
```

```
      if((j>=(bpsize/2)-1-7680)&&(j<=(bpsize/2)-1+7680)) //(Was at 7168
now 7680) Filter very high frequency round-off and
        bprofile[j]=dcomplex(0.0,0.0);// trancation noise with high
pass filter cutoff at fk=+/-1k/N.
      else{
        if(pow(abs(bprofile[j]),2.0)>=1.0e-6)

bprofile[j]=exp((nzstep/2.0)*dispersion_operator(j,pulse_HW,Ts,delta,be
ta2,bpsize,LD))*bprofile[j];
        }
      }
    dfft(bprofile,bpsize,-1.0);//Generate IFFT.
    for(j=0;j<bpsize;j++) // Normalise IFFT
      bprofile[j]=bprofile[j]/((double)(bpsize));
    }
  free(U);
  free(U2);
  free(DU2Ut);
  free(DU2t);
  free(Nz1);
  free(W);
  free(F);
  }

/**********************************************************************
*************************
Name: nolmpropsolitonp()
Function:  Propagates a normalized soliton pulse in a optical fiber
NOLM by Symmetrized
         Split Step Fourier Method.
         Zero attenuation is assumed.
Author: P.C. Chimfwembe
Date: 01/17/99
Modified: 02/03/99
***********************************************************************
************************
*/
void nolmpropsolitonp(unsigned int axis,dcomplex *bprofile,double
pulse_FWHM,
              double zstep /*Fraction of soliton period  */
              ,double zlength,double windowsize){
  dcomplex *U,*U2,*DU2t,*DU2Ut,*Nz1;//Data file and computational
scratchpad files.
  double Ts=windowsize/bpsize; /* Sample interval */
  double beta2=GVD(swlength); /* Fiber effective Beta2 (GVD)*/
  double beta3=6.2e-42;// 3rd order dispersion (s^3/m). Fiber core
radius=2.27e-6 m; Core:13.5m/o GeO2,85.5m/o SiO2;Cladding:SiO2
  double pulse_HW=pulse_FWHM/(2.0*log(1.0+sqrt(2.0)));//To:Half width
at 1/e intensity point.
  double wo=2*PI*C/swlength; // Carrier radian frequency.
  double delta,s,nX3rt;
  double LD,LNL,nzstep,nzlength;// Dispersion length,Nonlinear length
and normalized zstep
```

```
  double nz0=PI/2.0,z0;  // Normalized soliton period and soliton
period.
  unsigned int j,k,toprint=0,done=0;
  double cnplength=0.0,snplength=0.0; // Current and sub- normalized
propagation length.
  double OFF=0.0; // Effect switch.
  double shots=1.0; // No. of snap shots.
  if((U=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer U\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((U2=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer U2\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((DU2t=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer DU2t\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((DU2Ut=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer DU2Ut\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((Nz1=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer Nz1\n");
    exit(1);  /* terminate program if out of memory */
    }
  //Setting of delta,s, and nX3rt.
  delta=beta3/(6.0*fabs(beta2)*pulse_HW);
  s=2.0/(wo*pulse_HW);
  nX3rt=X3RT/pulse_HW; // Normalized Chi-3 response time.
  LD=pow(pulse_HW,2.0)/fabs(beta2);// Dispersion length
  nzstep=zstep*nz0;// Normalized z step.  Fraction of a soliton period.
  z0=LD*PI/2.0;
  nzlength=zlength/LD;// Normalized z length.
  while(ceil(cnplength/nzstep)<=ceil(nzlength/nzstep)){
    for(k=0;k<bpsize;k++){
      U[k]=bprofile[k];//Store U pow(abs(bprofile[k]),2.0)
      DU2t[k]=U2[k]=dcomplex(pow(abs(bprofile[k]),2.0),0.0);//
Calculate and store abs(U)^2.
      DU2Ut[k]=pow(abs(bprofile[k]),2.0)*bprofile[k];// Calculate
(abs(U)^2)*U.
      }
    L5pDt(DU2t,bpsize,windowsize,pulse_HW);//Determine DU2t and store.
    L5pDt(DU2Ut,bpsize,windowsize,pulse_HW);//Determine DU2Ut and
store.
    if(((Px!=0.0)&&(axis==XAXIS))||((Py!=0.0)&&(axis==YAXIS))){
      for(j=0;j<bpsize;j++)

Nz1[j]=dcomplex(0.0,1.0)*(U2[j]+((dcomplex(0.0,s)/U[j])*DU2Ut[j])-
(nX3rt*DU2t[j]));
      }
    else
```

```
      Nz1[j]=dcomplex(0.0,0.0);
    cnplength+=nzstep;// Increment current normalized propagation
length.
    snplength+=nzstep;
    dfft(bprofile,bpsize,1.0);//Generate FFT of beam profile.
    for(j=0;j<bpsize;j++){ // Dispersion step at nz=nzstep/2.
      if((j>=(bpsize/2)-1-7680)&&(j<=(bpsize/2)-1+7680)) //(Was at 7168
now 7680) Filter very high frequency round-off and
        bprofile[j]=dcomplex(0.0,0.0);// trancation noise with high
pass filter cutoff at fk=+/-1k/N.
      else{
        if(pow(abs(bprofile[j]),2.0)>=1.0e-6)

bprofile[j]=exp((nzstep/2.0)*dispersion_operatorxy(axis,j,pulse_HW,Ts,d
elta,beta2,bpsize,LD))*bprofile[j];
        }
      }
    dfft(bprofile,bpsize,-1.0);//Generate IFFT
    for(j=0;j<bpsize;j++) // Normalise IFFT
      bprofile[j]=bprofile[j]/((double)(bpsize));
    for(j=0;j<bpsize;j++){// Nonlinear step at nz=nzstep.
      if(pow(abs(U[j]),2.0)>=1.0e-6)// Clamp TR noise for N>=2.0.
        bprofile[j]=exp(nzstep*Nz1[j])*bprofile[j];
      }
    dfft(bprofile,bpsize,1.0);//Generate FFT of beam profile.
    for(j=0;j<bpsize;j++){ // Dispersion step at nz=nzstep/2.
      if((j>=(bpsize/2)-1-7680)&&(j<=(bpsize/2)-1+7680)) //(Was at 7168
now 7680) Filter very high frequency round-off and
        bprofile[j]=dcomplex(0.0,0.0);// trancation noise with high
pass filter cutoff at fk=+/-1k/N.
      else{
        if(pow(abs(bprofile[j]),2.0)>=1.0e-6)

bprofile[j]=exp((nzstep/2.0)*dispersion_operatorxy(axis,j,pulse_HW,Ts,d
elta,beta2,bpsize,LD))*bprofile[j];
        }
      }
    dfft(bprofile,bpsize,-1.0);//Generate IFFT.
    for(j=0;j<bpsize;j++) // Normalise IFFT
      bprofile[j]=bprofile[j]/((double)(bpsize));
    }
  free(U);
  free(U2);
  free(DU2Ut);
  free(DU2t);
  free(Nz1);
  }


/* Dispersion Operator*/

dcomplex dispersion_operator(unsigned int dfreq,double To,double
Ts,double Delta,double beta2,unsigned int size,double LD){
  double radfreq;// Spectral radian frequency
  if(dfreq<=size/2)
    radfreq=2.0*PI*dfreq/(size*Ts);
  else
    radfreq=-2.0*PI*(size-dfreq)/(size*Ts);
  return(dcomplex(-
alpha*LD/2.0,0.5*sgn(beta2)*pow(radfreq*To,2.0))+dcomplex(0.0,-
Delta*pow(radfreq*To,3.0)));
  }

/* Dispersion Operator xy*/

dcomplex dispersion_operatorxy(unsigned int axis,unsigned int
dfreq,double To,double Ts,
                               double Delta,double beta2,unsigned int
size,double LD){
  double radfreq;// Spectral radian frequency
  double DELTA; // DELTA normalized group velocity frame.
  double OFF=0.0; // Effect switches.
  if(dfreq<=size/2)
    radfreq=2.0*PI*dfreq/(size*Ts);
  else
    radfreq=-2.0*PI*(size-dfreq)/(size*Ts);
  DELTA=(dn/C)*To/(2.0*fabs(beta2));
  //printf("DELTA=%.8le\n",DELTA);
  //getchar();
  //exit(1);
  if(axis==XAXIS) // X/1 polarisation dispersion operator.
    return(dcomplex(-
alpha*LD/2.0,(DELTA*radfreq*To)+0.5*sgn(beta2)*pow(radfreq*To,2.0))+dco
mplex(0.0,-Delta*pow(radfreq*To,3.0)));
  else // Y/2 polarisation dispersion operator.
    return(dcomplex(-alpha*LD/2.0,-
(DELTA*radfreq*To)+0.5*sgn(beta2)*pow(radfreq*To,2.0))+dcomplex(0.0,-
Delta*pow(radfreq*To,3.0)));
  }




/**********************************************************************
*************************
 Name: propsolitonps()
Function:  Co-propagates two orthogonal normalized soliton pulses in a
optical fiber by Symmetrized Split Step Fourier Method.  Zero
attenuation is assumed.
Author: P.C. Chimfwembe
Date: 03/19/98
Modified: 04/14/98
**********************************************************************
************************
*/
```

```
void propsolitonps(dcomplex *bprofilex,dcomplex *bprofiley,double
pulse_FWHM,
                         double zstep, /*Fraction of soliton period  */
                         double zlength,double windowsize){
  FILE *dfpx,*dfpy,*dfpRx,*dfpRy,*dfpFx,*dfpFy; //Data file pointers.
  dcomplex *Ux,*aUx2,*DaUx2t,*DaUx2Uxt,*Nz1x;//Data computational
scratchpad array pointers for pulse x.
  dcomplex *Uy,*aUy2,*DaUy2t,*DaUy2Uyt,*Nz1y;//Data computational
scratchpad array pointers for pulse y.
  dcomplex *DaUx2Uyt,*DaUy2Uxt;//Data computational scratchpad array
pointers for cross terms of pulses x and y.
  double Ts=windowsize/bpsize; /* Sample interval and normalized sample
interval */
  double beta2=GVD(swlength); /* Fiber effective Beta2 (GVDx=GVDy) */
  double beta3=6.2e-42;//x and y 3rd order dispersions (s^3/m)
(TODx=TODy). Fiber core radius=2.27e-6 m; Core:13.5m/o GeO2,85.5m/o
SiO2;Cladding:SiO2
  double pulse_HW=pulse_FWHM/(2.0*log(1.0+sqrt(2.0)));//To:Half width
at 1/e intensity point.
  double wo=2*PI*C/swlength; // Carrier radian frequency.
  double delta,s,nX3rt;
  double LD,LNLx,LNLy,nzstep,nzlength;// Dispersion length,Nonlinear
length and normalized zstep.
  double nz0=PI/2.0,z0;  // Normalized soliton period and soliton
period.
  unsigned int j,k,toprint=1;
  double cnplength=0.0,snplength=0.0; // Current and sub- normalized
propagation length.
  double ANGLE=(45.0/180.0)*PI;//(84.0/180.0)*PI; // Splitting soliton
amplitude angle.
  double
Nx=1.24*cos(ANGLE)*0.0,Ny=1.24*sin(ANGLE);//(3)30;(2)60;(1)84(1.24SDG);
45 (STG) (SDG 45<ANGLE>45)cos(ANGLE), sin(ANGLE) Soliton order for x
and y axes.
  double OFF=0.0; // Effect switch.
  double  *Wx,*Wy,*Fx,*Fy,fpsdxmax=0.0,fpsdymax=0.0; // Time and
frequency intensities.
  double shots=10.0; // No. of snap shots.
  SAVE Rx,Ry; // Recovery data arrays.
  if((Ux=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer Ux\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((aUx2=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer aUx2\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((DaUx2t=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer DaUx2t\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((DaUx2Uxt=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer DaUx2Uxt\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((Nz1x=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer Nz1x\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((Uy=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer Uy\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((aUy2=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer aUy2\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((DaUy2t=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer DaUy2t\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((DaUy2Uyt=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer DU2Uty\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((Nz1y=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer Nz1y\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((DaUx2Uyt=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer DU2Uty\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((DaUy2Uxt=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer DU2Uty\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((Wx=(double *)calloc(bpsize,sizeof(double)))==NULL){
    printf("Not enough memory to allocate buffer Wx\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((Wy=(double *)calloc(bpsize,sizeof(double)))==NULL){
    printf("Not enough memory to allocate buffer Wy\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((Fx=(double *)calloc(bpsize,sizeof(double)))==NULL){
    printf("Not enough memory to allocate buffer Fx\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((Fy=(double *)calloc(bpsize,sizeof(double)))==NULL){
    printf("Not enough memory to allocate buffer Fy\n");
    exit(1);  /* terminate program if out of memory */
    }
  beta3=6.2e-42;// Fiber core radius=2.27e-6 m; Core:13.5m/o
GeO2,85.5m/o SiO2;Cladding:SiO2
  delta=beta3/(6.0*fabs(beta2)*pulse_HW);
  s=2.0/(wo*pulse_HW);
  nX3rt=X3RT/pulse_HW; // Normalized Chi-3 response time.
  LD=pow(pulse_HW,2.0)/fabs(beta2);// Dispersion length.
  z0=LD*PI/2.0;  // Soliton period in y polarization as reference
plane.
  nzstep=zstep*nz0;// Normalized z step.  Fraction of a soliton period.
  nzlength=zlength/LD;// Normalized z length referenced to y
polarization plane.
  Px=pow(Nx,2.0)*fabs(beta2)/(gamma(swlength)*pow(pulse_HW,2.0));
  Py=pow(Ny,2.0)*fabs(beta2)/(gamma(swlength)*pow(pulse_HW,2.0));
  if(Px!=0.0)
    LNLx=1.0/(gamma(swlength)*Px);// Non-linear length of signal.
  if(Py!=0.0)
    LNLy=1.0/(gamma(swlength)*Py);// Non-linear length of pump.
  printf("Peak
Powerx=%.8le;Beta2=%.8le;Gamma=%.8le\n",Px,beta2,gamma(swlength));
  printf("Peak Powery=%.8le\n",Py);
  printf("Soliton Period(z0)=%.8le;Dispersion
Length(LD)=%.8le\n",z0,LD);
  if(Px!=0.0)
    printf("Non-Linear Lengthx(LNLx)=%.8le\n",LNLx);
  else
    printf("Non-Linear Lengthx(LNLx)=infinity\n");
  if(Py!=0.0)
    printf("Non-Linear Lengthy(LNLy)=%.8le\n",LNLy);
  else
    printf("Non-Linear Lengthy(LNLy)=infinity\n");
  printf("s=%.8le;nX3rt=%.8le\nDelta=%.8le\n",s,nX3rt,delta);
  printf("Beta3=%.8le\nNx=%.8le;Ny=%.8le\n",beta3,Nx,Ny);
  printf("Total Iterations=%.8le\n",ceil(nzlength/nzstep));
  gf1solitonp(bprofilex,pulse_FWHM,windowsize);// Initialise soliton
beam profile x.
  gf1solitonp(bprofiley,pulse_FWHM,windowsize);// Initialise soliton
beam profile y.
  for(j=0;j<bpsize;j++){
    bprofilex[j]=bprofilex[j]*Nx;
    bprofiley[j]=bprofiley[j]*Ny;
    }
  dfft(bprofilex,bpsize,1.0);//Generate FFT of beam profile x.
  dfft(bprofiley,bpsize,1.0);//Generate FFT of beam profile y.
  fpsdxmax=pow(abs(bprofilex[0]),2.0)/pow(bpsize,2.0);// Power spectral
density for frequency zero - maximum power component.
  fpsdymax=pow(abs(bprofiley[0]),2.0)/pow(bpsize,2.0);
  dfft(bprofilex,bpsize,-1.0);//Generate IFFT for profile x.
  dfft(bprofiley,bpsize,-1.0);//Generate IFFT for profile y.
  for(j=0;j<bpsize;j++){ // Normalise IFFT for profiles x and y.
    bprofilex[j]=bprofilex[j]/((double)(bpsize));
    bprofiley[j]=bprofiley[j]/((double)(bpsize));
    }
  if((dfpRx=fopen("ppdatarx.m","a+b"))==NULL){ // Profile complex data
file for recovery.
    printf("Unable to open output file ppdatarx.m\n");
    return;
    }
  if((dfpRy=fopen("ppdatary.m","a+b"))==NULL){ // Profile complex data
file for recovery.
    printf("Unable to open output file ppdatary.m\n");
    return;
    }
  fread(&Rx,sizeof(SAVE),1,dfpRx); // Detect end of file with fread()
else ftell() or rewind has no effect on current file pointer position.
  fread(&Ry,sizeof(SAVE),1,dfpRy);
  if(ftell(dfpRx)&&ftell(dfpRy)){ //If file contains past beam profile
snap shots reload to continue.
    toprint=0; // Reset print flag.
    printf("Recovering from power failure termination ...\n");
    rewind(dfpRx);rewind(dfpRy);
    fread(&Rx,sizeof(SAVE),1,dfpRx);fread(&Ry,sizeof(SAVE),1,dfpRy); //
Initialise Rx and Ry context and profile.
    fclose(dfpRx);fclose(dfpRy);
    if(ceil(Rx.cnplength/nzstep)>=ceil(Rx.nzlength/nzstep)){ // x
reference plane.
      printf("At end of propagation length.\n");
      printf("Please re-initialize total propagation length (m):");
      scanf("%le",&zlength);
      zlength=fabs(zlength);
      nzlength=zlength/LD;// Normalized z length in x reference plane.
      Rx.nzlength=nzlength;Rx.cnplength=cnplength=0.0;
      if(!Rx.nzlength)//If zero or non-real character was entered, then
terminate.
        exit(1); // Exit.
      }
    else{
      nzlength=Rx.nzlength;cnplength=Rx.cnplength;
      }
    printf("nzlength/nz0=%.8le;
cnplength/nz0=%.8le\n",Rx.nzlength/nz0,Rx.cnplength/nz0);
    printf("Iterations left=%.8le\n",ceil((Rx.nzlength-
Rx.cnplength)/nzstep));
    for(k=0;k<bpsize;k++){// Load last saved profilex and profiley.
      bprofilex[k]=Rx.bprofile[k];
      bprofiley[k]=Ry.bprofile[k];
      }
    }
  else{
    fclose(dfpRx);fclose(dfpRy);
    }
  while(ceil(cnplength/nzstep)<=ceil(nzlength/nzstep)){

if((snplength>=nzlength/shots)||(ceil(cnplength/nzstep)==ceil(nzlength/
nzstep)))
      toprint=1;
    if(toprint){
      printf("At z/z0 Position:%le\n",cnplength/nz0);
      for(j=0;j<bpsize;j++){
        Wx[j]=pow(abs(bprofilex[j]),2.0); //Returns the squared
magnitude of bprofilex[j] and bprofiley[j].
        Wy[j]=pow(abs(bprofiley[j]),2.0);
           }
      if((dfpx=fopen("ppdatax.m","a+b"))==NULL){// Profile Matlab data
plot file.
        printf("Unable to open output file ppdatax.m\n");
        return;
        }
      if((dfpy=fopen("ppdatay.m","a+b"))==NULL){// Profile Matlab data
plot file.
        printf("Unable to open output file ppdatay.m\n");
        return;
        }
      if((dfpFx=fopen("ppdatafx.m","a+b"))==NULL){ // Frequency power
spectral density data file.
        printf("Unable to open output file ppdatafx.m\n");
        exit(1);
        }
      if((dfpFy=fopen("ppdatafy.m","a+b"))==NULL){ // Frequency power
spectral density data file.
        printf("Unable to open output file ppdatafy.m\n");
        exit(1);
        }
      if((dfpRx=fopen("ppdatarx.m","w+b"))==NULL){ // Profile complex
data file for recovery.
        printf("Unable to open output file ppdatarx.m\n");
        return;
        }
      if((dfpRy=fopen("ppdatary.m","w+b"))==NULL){ // Profile complex
data file for recovery.
        printf("Unable to open output file ppdatary.m\n");
        return;
        }
      fwrite(Wx,sizeof(double),bpsize,dfpx);// Save profile Matlab data
plot files.
      fwrite(Wy,sizeof(double),bpsize,dfpy);
      Rx.nzlength=nzlength;Rx.cnplength=cnplength;// Save context.
      for(j=0;j<bpsize;j++){
        Rx.bprofile[j]=bprofilex[j];
        Ry.bprofile[j]=bprofiley[j];
        }

fwrite(&Rx,sizeof(SAVE),1,dfpRx);fwrite(&Ry,sizeof(SAVE),1,dfpRy);  //
Save profile complex data files for recovery.
      dfft(bprofilex,bpsize,1.0);//Generate FFT of beam profile x.
      dfft(bprofiley,bpsize,1.0);//Generate FFT of beam profile y.
      for(j=0;j<bpsize/2;j++){//Re-organise discrete frequencies from -
N/2 - 0 - N/2.  FFT has it inside out.
        if(Px!=0.0)

Fx[j+bpsize/2]=pow(abs(bprofilex[j]),2.0)/(pow(bpsize,2.0)*fpsdxmax);//
Frequency power spectral density.
```

```
        else
          Fx[j+bpsize/2]=0.0;
        if(Py!=0.0)

Fy[j+bpsize/2]=pow(abs(bprofiley[j]),2.0)/(pow(bpsize,2.0)*fpsdymax);
        else
          Fy[j+bpsize/2]=0.0;
          }
      for(j=bpsize/2;j<bpsize;j++){
        if(Px!=0.0)
          Fx[j-
bpsize/2]=pow(abs(bprofilex[j]),2.0)/(pow(bpsize,2.0)*fpsdxmax);//Frequ
ency power spectral density.
        else
          Fx[j-bpsize/2]=0.0;
        if(Py!=0.0)
          Fy[j-
bpsize/2]=pow(abs(bprofiley[j]),2.0)/(pow(bpsize,2.0)*fpsdymax);
        else
          Fy[j-bpsize/2]=0.0;
          }
      dfft(bprofilex,bpsize,-1.0);//Generate IFFT for profile x.
      dfft(bprofiley,bpsize,-1.0);//Generate IFFT for profile y.
      for(j=0;j<bpsize;j++){ // Normalise IFFT for profiles x and y.
        bprofilex[j]=bprofilex[j]/((double)(bpsize));
        bprofiley[j]=bprofiley[j]/((double)(bpsize));
          }
      fwrite(Fx,sizeof(double),bpsize,dfpFx); // Save frequency power
spectral density data file.
      fwrite(Fy,sizeof(double),bpsize,dfpFy);
      fclose(dfpx);fclose(dfpy); // Close all data files.
      fclose(dfpRx);fclose(dfpRy);
      fclose(dfpFx);fclose(dfpFy);
      snplength=0.0;toprint=0;
      }
    for(k=0;k<bpsize;k++){
      Ux[k]=bprofilex[k];//Store Ux.
      Uy[k]=bprofiley[k];//Store Uy.
      DaUx2t[k]=aUx2[k]=dcomplex(pow(abs(bprofilex[k]),2.0),0.0);//
Calculate and store abs(Ux)^2.
      DaUy2t[k]=aUy2[k]=dcomplex(pow(abs(bprofiley[k]),2.0),0.0);//
Calculate and store abs(Uy)^2.
      DaUx2Uxt[k]=pow(abs(bprofilex[k]),2.0)*bprofilex[k];// Calculate
(abs(Ux)^2)*Ux.
      DaUy2Uyt[k]=pow(abs(bprofiley[k]),2.0)*bprofiley[k];// Calculate
(abs(Uy)^2)*Uy.
      DaUx2Uyt[k]=pow(abs(bprofilex[k]),2.0)*bprofiley[k];// Calculate
(abs(Ux)^2)*Uy.
      DaUy2Uxt[k]=pow(abs(bprofiley[k]),2.0)*bprofilex[k];// Calculate
(abs(Uy)^2)*Ux.
      } // Time derivatives.
    L5pDt(DaUx2t,bpsize,windowsize,pulse_HW);//Determine DaUx2t and
store.
```

```
    L5pDt(DaUy2t,bpsize,windowsize,pulse_HW);//Determine DaUy2t and
store.
    L5pDt(DaUx2Uxt,bpsize,windowsize,pulse_HW);//Determine DaUx2Uxt and
store.
    L5pDt(DaUy2Uyt,bpsize,windowsize,pulse_HW);//Determine DaUy2Uyt and
store.
    L5pDt(DaUx2Uyt,bpsize,windowsize,pulse_HW);//Determine DaUx2Uyt and
store.
    L5pDt(DaUy2Uxt,bpsize,windowsize,pulse_HW);//Determine DaUy2Uxt and
store.
    for(j=0;j<bpsize;j++){//  Compute nonlinear operator.
      if(Px!=0.0)
        Nz1x[j]=dcomplex(0.0,1.0)*(aUx2[j]+((2.0/3.0)*aUy2[j])

+((dcomplex(0.0,s)/Ux[j])*(DaUx2Uxt[j]+((2.0/3.0)*DaUy2Uxt[j])))
               -(nX3rt*(DaUx2t[j]+((2.0/3.0)*DaUy2t[j])))); // x
polarized pulse.
      else
        Nz1x[j]=dcomplex(0.0,0.0);
      if(Py!=0.0)
        Nz1y[j]=dcomplex(0.0,1.0)*(aUy2[j]+((2.0/3.0)*aUx2[j])

+((dcomplex(0.0,s)/Uy[j])*(DaUy2Uyt[j]+((2.0/3.0)*DaUx2Uyt[j])))
               -(nX3rt*(DaUy2t[j]+((2.0/3.0)*DaUx2t[j]))));  // y
polarized pulse.
      else
        Nz1y[j]=dcomplex(0.0,0.0);
      }
    cnplength+=nzstep;// Increment current normalized propagation
length.
    snplength+=nzstep;
    dfft(bprofilex,bpsize,1.0);//Generate FFT of beam profile x.
    dfft(bprofiley,bpsize,1.0);//Generate FFT of beam profile y.
    for(j=0;j<bpsize;j++){ // Dispersion step at nz=nzstep/2.
      if((j>=(bpsize/2)-1-7680)&&(j<=(bpsize/2)-1+7680)){ //(Was at
7168 now 7680) Filter very high frequency round-off and
        bprofilex[j]=dcomplex(0.0,0.0);// trancation noise with high
pass filter cutoff at fk=+/-1k/N.
        bprofiley[j]=dcomplex(0.0,0.0);//7168(7k);8192(8k)
        }
      else{
        if(pow(abs(bprofilex[j]),2.0)>=1.0e-6)

bprofilex[j]=exp((nzstep/2.0)*dispersion_operatorxy(XAXIS,j,pulse_HW,Ts
,delta,beta2,bpsize,LD))*bprofilex[j];
        if(pow(abs(bprofiley[j]),2.0)>=1.0e-6)

bprofiley[j]=exp((nzstep/2.0)*dispersion_operatorxy(YAXIS,j,pulse_HW,Ts
,delta,beta2,bpsize,LD))*bprofiley[j];
        }
      }
    dfft(bprofilex,bpsize,-1.0);//Generate IFFT for profile x.
    dfft(bprofiley,bpsize,-1.0);//Generate IFFT for profile y.
    for(j=0;j<bpsize;j++){ // Normalise IFFT for profiles x and y.
      bprofilex[j]=bprofilex[j]/((double)(bpsize));
      bprofiley[j]=bprofiley[j]/((double)(bpsize));
      }
    for(j=0;j<bpsize;j++){// Nonlinear step at nz=nzstep.
      bprofilex[j]=exp(nzstep*Nz1x[j])*bprofilex[j];
      bprofiley[j]=exp(nzstep*Nz1y[j])*bprofiley[j];
      }
    dfft(bprofilex,bpsize,1.0);//Generate FFT of beam profile x.
    dfft(bprofiley,bpsize,1.0);//Generate FFT of beam profile y.
    for(j=0;j<bpsize;j++){ // Dispersion step at nz=nzstep/2.
      if((j>=(bpsize/2)-1-7680)&&(j<=(bpsize/2)-1+7680)){ //(Was at
7168 now 7680) Filter very high frequency round-off and
        bprofilex[j]=dcomplex(0.0,0.0);// trancation noise with high
pass filter cutoff at fk=+/-1k/N.
        bprofiley[j]=dcomplex(0.0,0.0);//7168(7k);8192(8k)
        }
      else{
        if(pow(abs(bprofilex[j]),2.0)>=1.0e-6)

bprofilex[j]=exp((nzstep/2.0)*dispersion_operatorxy(XAXIS,j,pulse_HW,Ts
,delta,beta2,bpsize,LD))*bprofilex[j];
        if(pow(abs(bprofiley[j]),2.0)>=1.0e-6)

bprofiley[j]=exp((nzstep/2.0)*dispersion_operatorxy(YAXIS,j,pulse_HW,Ts
,delta,beta2,bpsize,LD))*bprofiley[j];
        }
      }
    dfft(bprofilex,bpsize,-1.0);//Generate IFFT for profile x.
    dfft(bprofiley,bpsize,-1.0);//Generate IFFT for profile y.
    for(j=0;j<bpsize;j++){ // Normalise IFFT for profiles x and y.
      bprofilex[j]=bprofilex[j]/((double)(bpsize));
      bprofiley[j]=bprofiley[j]/((double)(bpsize));
      }
    }
  free(Ux);
  free(aUx2);
  free(DaUx2t);
  free(DaUx2Uxt);
  free(Nz1x);
  free(Uy);
  free(aUy2);
  free(DaUy2t);
  free(DaUy2Uyt);
  free(Nz1y);
  free(DaUx2Uyt);
  free(DaUy2Uxt);
  free(Wx);
  free(Wy);
  free(Fx);
  free(Fy);
  }
```

```
/***********************************************************************
*************************
 Name: nolmpropsolitonps()
Function:  Co-propagates two orthogonal normalized soliton pulses in an
optical fiber NOLM by Symmetrized Split Step Fourier Method.  Zero
attenuation is assumed.
Author: P.C. Chimfwembe
Date: 01/17/99
Modified: 02/03/99
***********************************************************************
************************
*/
void nolmpropsolitonps(dcomplex *bprofilex,dcomplex *bprofiley,double
pulse_FWHM,
                         double zstep, /*Fraction of soliton period  */
                         double zlength,double windowsize){
  dcomplex *Ux,*aUx2,*DaUx2t,*DaUx2Uxt,*Nz1x;//Data computational
scratchpad array pointers for pulse x.
  dcomplex *Uy,*aUy2,*DaUy2t,*DaUy2Uyt,*Nz1y;//Data computational
scratchpad array pointers for pulse y.
  dcomplex *DaUx2Uyt,*DaUy2Uxt;//Data computational scratchpad array
pointers for cross terms of pulses x and y.
  double Ts=windowsize/bpsize; /* Sample interval and normalized sample
interval */
  double beta2=GVD(swlength); /* Fiber effective Beta2 (GVDx=GVDy) */
  double beta3=6.2e-42;//x and y 3rd order dispersions (s^3/m)
(TODx=TODy). Fiber core radius=2.27e-6 m; Core:13.5m/o GeO2,85.5m/o
SiO2;Cladding:SiO2
  double pulse_HW=pulse_FWHM/(2.0*log(1.0+sqrt(2.0)));//To:Half width
at 1/e intensity point.
  double wo=2*PI*C/swlength; // Carrier radian frequency.
  double delta,s,nX3rt;
  double LD,LNLx,LNLy,nzstep,nzlength;// Dispersion length,Nonlinear
length and normalized zstep.
  double nz0=PI/2.0,z0;  // Normalized soliton period and soliton
period.
  unsigned int j,k,toprint=1;
  double cnplength=0.0,snplength=0.0; // Current and sub-normalized
propagation length.
  double OFF=0.0; // Effect switch.
  double shots=1.0; // No. of snap shots.
  if((Ux=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer Ux\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((aUx2=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer aUx2\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((DaUx2t=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer DaUx2t\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((DaUx2Uxt=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer DaUx2Uxt\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((Nz1x=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer Nz1x\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((Uy=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer Uy\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((aUy2=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer aUy2\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((DaUy2t=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer DaUy2t\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((DaUy2Uyt=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer DU2Uty\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((Nz1y=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer Nz1y\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((DaUx2Uyt=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer DU2Uty\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((DaUy2Uxt=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer DU2Uty\n");
    exit(1);  /* terminate program if out of memory */
    }
  beta3=6.2e-42;// Fiber core radius=2.27e-6 m; Core:13.5m/o
GeO2,85.5m/o SiO2;Cladding:SiO2
  delta=beta3/(6.0*fabs(beta2)*pulse_HW);
  s=2.0/(wo*pulse_HW);
  nX3rt=X3RT/pulse_HW; // Normalized Chi-3 response time.
  LD=pow(pulse_HW,2.0)/fabs(beta2);// Dispersion length.
  z0=LD*PI/2.0;  // Soliton period in y polarization as reference
plane.
  nzstep=zstep*nz0;// Normalized z step.  Fraction of a soliton period.
  nzlength=zlength/LD;// Normalized z length referenced to y
polarization plane.
  while(ceil(cnplength/nzstep)<=ceil(nzlength/nzstep)){
    for(k=0;k<bpsize;k++){
      Ux[k]=bprofilex[k];//Store Ux.
      Uy[k]=bprofiley[k];//Store Uy.
      DaUx2t[k]=aUx2[k]=dcomplex(pow(abs(bprofilex[k]),2.0),0.0);//
Calculate and store abs(Ux)^2.
      DaUy2t[k]=aUy2[k]=dcomplex(pow(abs(bprofiley[k]),2.0),0.0);//
Calculate and store abs(Uy)^2.
      DaUx2Uxt[k]=pow(abs(bprofilex[k]),2.0)*bprofilex[k];// Calculate
(abs(Ux)^2)*Ux.
      DaUy2Uyt[k]=pow(abs(bprofiley[k]),2.0)*bprofiley[k];// Calculate
(abs(Uy)^2)*Uy.
      DaUx2Uyt[k]=pow(abs(bprofilex[k]),2.0)*bprofiley[k];// Calculate
(abs(Ux)^2)*Uy.
      DaUy2Uxt[k]=pow(abs(bprofiley[k]),2.0)*bprofilex[k];// Calculate
(abs(Uy)^2)*Ux.
      } // Time derivatives.
    L5pDt(DaUx2t,bpsize,windowsize,pulse_HW);//Determine DaUx2t and
store.
    L5pDt(DaUy2t,bpsize,windowsize,pulse_HW);//Determine DaUy2t and
store.
    L5pDt(DaUx2Uxt,bpsize,windowsize,pulse_HW);//Determine DaUx2Uxt and
store.
    L5pDt(DaUy2Uyt,bpsize,windowsize,pulse_HW);//Determine DaUy2Uyt and
store.
    L5pDt(DaUx2Uyt,bpsize,windowsize,pulse_HW);//Determine DaUx2Uyt and
store.
    L5pDt(DaUy2Uxt,bpsize,windowsize,pulse_HW);//Determine DaUy2Uxt and
store.
    for(j=0;j<bpsize;j++){//  Compute nonlinear operator.
      if(Px!=0.0)
        Nz1x[j]=dcomplex(0.0,1.0)*(aUx2[j]+((2.0/3.0)*aUy2[j])

+((dcomplex(0.0,s)/Ux[j])*(DaUx2Uxt[j]+((2.0/3.0)*DaUy2Uxt[j])))
               -(nX3rt*(DaUx2t[j]+((2.0/3.0)*DaUy2t[j])))); // x
polarized pulse.
      else
        Nz1x[j]=dcomplex(0.0,0.0);
      if(Py!=0.0)
        Nz1y[j]=dcomplex(0.0,1.0)*(aUy2[j]+((2.0/3.0)*aUx2[j])

+((dcomplex(0.0,s)/Uy[j])*(DaUy2Uyt[j]+((2.0/3.0)*DaUx2Uyt[j])))
               -(nX3rt*(DaUy2t[j]+((2.0/3.0)*DaUx2t[j]))));  // y
polarized pulse.
      else
        Nz1y[j]=dcomplex(0.0,0.0);
      }
    cnplength+=nzstep;// Increment current normalized propagation
length.
    snplength+=nzstep;
    dfft(bprofilex,bpsize,1.0);//Generate FFT of beam profile x.
    dfft(bprofiley,bpsize,1.0);//Generate FFT of beam profile y.
    for(j=0;j<bpsize;j++){ // Dispersion step at nz=nzstep/2.
      if((j>=(bpsize/2)-1-7680)&&(j<=(bpsize/2)-1+7680)){ //(Was at
7168 now 7680) Filter very high frequency round-off and
        bprofilex[j]=dcomplex(0.0,0.0);// trancation noise with high
pass filter cutoff at fk=+/-1k/N.
```

```
        bprofiley[j]=dcomplex(0.0,0.0);//7168(7k);8192(8k)
        }
      else{
        if(pow(abs(bprofilex[j]),2.0)>=1.0e-6)

bprofilex[j]=exp((nzstep/2.0)*dispersion_operatorxy(XAXIS,j,pulse_HW,Ts
,delta,beta2,bpsize,LD))*bprofilex[j];
        if(pow(abs(bprofiley[j]),2.0)>=1.0e-6)

bprofiley[j]=exp((nzstep/2.0)*dispersion_operatorxy(YAXIS,j,pulse_HW,Ts
,delta,beta2,bpsize,LD))*bprofiley[j];
        }
      }
    dfft(bprofilex,bpsize,-1.0);//Generate IFFT for profile x.
    dfft(bprofiley,bpsize,-1.0);//Generate IFFT for profile y.
    for(j=0;j<bpsize;j++){ // Normalise IFFT for profiles x and y.
      bprofilex[j]=bprofilex[j]/((double)(bpsize));
      bprofiley[j]=bprofiley[j]/((double)(bpsize));
      }
    for(j=0;j<bpsize;j++){// Nonlinear step at nz=nzstep.
      bprofilex[j]=exp(nzstep*Nz1x[j])*bprofilex[j];
      bprofiley[j]=exp(nzstep*Nz1y[j])*bprofiley[j];
      }
    dfft(bprofilex,bpsize,1.0);//Generate FFT of beam profile x.
    dfft(bprofiley,bpsize,1.0);//Generate FFT of beam profile y.
    for(j=0;j<bpsize;j++){ // Dispersion step at nz=nzstep/2.
      if((j>=(bpsize/2)-1-7680)&&(j<=(bpsize/2)-1+7680)){ //(Was at
7168 now 7680) Filter very high frequency round-off and
        bprofilex[j]=dcomplex(0.0,0.0);// trancation noise with high
pass filter cutoff at fk=+/-1k/N.
        bprofiley[j]=dcomplex(0.0,0.0);//7168(7k);8192(8k)
        }
      else{
        if(pow(abs(bprofilex[j]),2.0)>=1.0e-6)

bprofilex[j]=exp((nzstep/2.0)*dispersion_operatorxy(XAXIS,j,pulse_HW,Ts
,delta,beta2,bpsize,LD))*bprofilex[j];
        if(pow(abs(bprofiley[j]),2.0)>=1.0e-6)

bprofiley[j]=exp((nzstep/2.0)*dispersion_operatorxy(YAXIS,j,pulse_HW,Ts
,delta,beta2,bpsize,LD))*bprofiley[j];
        }
      }
    dfft(bprofilex,bpsize,-1.0);//Generate IFFT for profile x.
    dfft(bprofiley,bpsize,-1.0);//Generate IFFT for profile y.
    for(j=0;j<bpsize;j++){ // Normalise IFFT for profiles x and y.
      bprofilex[j]=bprofilex[j]/((double)(bpsize));
      bprofiley[j]=bprofiley[j]/((double)(bpsize));
      }
    }
  free(Ux);
  free(aUx2);
  free(DaUx2t);
  free(DaUx2Uxt);
  free(Nz1x);
  free(Uy);
  free(aUy2);
  free(DaUy2t);
  free(DaUy2Uyt);
  free(Nz1y);
  free(DaUx2Uyt);
  free(DaUy2Uxt);
  }


/*********************************************************************
*************************
 Name: stgpsinolm()
Function: Implements a STG-NOLM in an optical fiber by Symmetrized
Split Step Fourier Method.
          Zero attenuation is assumed.
Author: P.C. Chimfwembe
Date: 01/16/99
Modified: 02/03/99
*********************************************************************
************************
*/
void stgpsinolm(dcomplex *inputprofile,double pulse_FWHM,
                   double zstep, /*Fraction of soliton period */
                   double zlength,double windowsize){
  FILE *dfpR;// Data recovery file "last profiles (Single/Co-
propagation) recorded" data file.
  FILE *dfpSPORT1,*dfpSPORT2,*dfpSPORT1F,*dfpSPORT2F;// STG-NOLM switch
file.
  FILE *dfpx1,*dfpy1,*dfpy2,*dfpFx1,*dfpFy1,*dfpFy2,*dfpPCWtCCW; //Data
file pointers.
  dcomplex
*ein1profile,*ein2profile,*eout1profile,*eout2profile,*sprofile;
  double *phasediff,phaseCW,phaseCCW; // Output port phase difference
ie PhaseCW-PhaseCCW
  double nz0=PI/2.0,z0,LD;  // Normalized soliton period, soliton
period and dispersion length.
  double beta2=GVD(swlength); /* Fiber effective Beta2 (GVDx=GVDy) */
  double beta3=6.2e-42;//x and y 3rd order dispersions (s^3/m)
(TODx=TODy). Fiber core radius=2.27e-6 m; Core:13.5m/o GeO2,85.5m/o
SiO2;Cladding:SiO2
  double pulse_HW=pulse_FWHM/(2.0*log(1.0+sqrt(2.0)));//To:Half width
at 1/e intensity point.
  double wo=2*PI*C/swlength; // Carrier radian frequency.
  double delta,s,nX3rt;
  double EiLNL,EoLNL;// Nonlinear length and normalized zstep
  double Nx=NxSTGNOLM,Ny=NySTGNOLM;// Soliton order for x and y axes.
  double power=0.0; // Power buffer.
  double walkofflength=(pulse_FWHM*C)/dn; // Walk-off length of
orthogonal pulses.
  double segmentlength=walkofflength*1.0;  // Optimized segment length.
  double timeshift=pulse_HW*5.0;//Time shift of signal with respect to
control at coupler input to NOLM loop.
  unsigned int SIGN=NEGATIVE; // Timeshift variable.
  double
*Wx1,*Wy1,*Wy2,*Fx1,*Fy1,*Fy2,fpsdx1max=0.0,fpsdy1max=0.0,fpsdy2max=0.0
,fpsdyimax=0.0; // Time and frequency intensities.
  unsigned int segmentcounter=0; // Number of flipped axes (x<->y)
segments.
  unsigned int splicenumber=0; // Splice position locator.
  unsigned int togglecounter=0; // Number of axis switches.
  int AXISTOGGLE=0;  // Axis switch flag.
  int START=1; // Signal time shift flag.
  unsigned int i,j; //Iteration variables.
  SAVESWITCH R; // Recovery data structure for single/co-propagation.
  if((ein1profile=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer ein1profile\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((ein2profile=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer ein2profile\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((eout1profile=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer eout1profile\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((eout2profile=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer eout2profile\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((sprofile=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer sprofile\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((Wx1=(double *)calloc(bpsize,sizeof(double)))==NULL){
    printf("Not enough memory to allocate buffer Wx1\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((Wy1=(double *)calloc(bpsize,sizeof(double)))==NULL){
    printf("Not enough memory to allocate buffer Wy1\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((Wy2=(double *)calloc(bpsize,sizeof(double)))==NULL){
    printf("Not enough memory to allocate buffer Wy2\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((Fx1=(double *)calloc(bpsize,sizeof(double)))==NULL){
    printf("Not enough memory to allocate buffer Wx1\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((Fy1=(double *)calloc(bpsize,sizeof(double)))==NULL){
    printf("Not enough memory to allocate buffer Wy1\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((Fy2=(double *)calloc(bpsize,sizeof(double)))==NULL){
    printf("Not enough memory to allocate buffer Wy2\n");
    exit(1);  /* terminate program if out of memory */
    }
   if((phasediff=(double *)calloc(bpsize,sizeof(double)))==NULL){
    printf("Not enough memory to allocate buffer phasediff\n");
    exit(1);  /* terminate program if out of memory */
    }
  //Setting of delta,s, and nX3rt.
  delta=beta3/(6.0*fabs(beta2)*pulse_HW);
  s=2.0/(wo*pulse_HW);
  nX3rt=X3RT/pulse_HW; // Normalized Chi-3 response time.
  LD=pow(pulse_HW,2.0)/fabs(beta2);// Dispersion length.
  z0=LD*nz0;  // Soliton period in y polarization as reference plane.
  togglecounter=segmentcounter=floor(zlength/segmentlength); // Number
of flipped axes segments. Optimize with floor or ceil.

EiPy=pow(EiNySTGNOLM,2.0)*fabs(beta2)/(gamma(swlength)*pow(pulse_HW,2.0
));
  EiLNL=1.0/(gamma(swlength)*EiPy);// Non-linear length
  gf1solitonp(inputprofile,pulse_FWHM,windowsize);// Initialise input
soliton control beam profile (y-axis or fast axis).
  for(j=0;j<bpsize;j++){
    inputprofile[j]=inputprofile[j]*EiNySTGNOLM;
    }
  printf("NOLM characteristics:\n");
  printf("Walk-off length (lwo)=%.8le; Segment length
(Ls)=%.8le\n",walkofflength,segmentlength);
  printf("Number of segments in STG-NOLM=%ld\n",segmentcounter);
  printf("11 segment length(z)=%.8le\n",segmentlength*11.0);
  printf("Input Control Pulse (Ein1):\n");
  printf("Control
Power=%.8le;Beta2=%.8le;Gamma=%.8le\n",EiPy,beta2,gamma(swlength));
  printf("Soliton Period(z0)=%.8le;Dispersion
Length(LD)=%.8le\n",z0,LD);
  printf("Non-Linear Length(EiLNL)=%.8le\n",EiLNL);
  printf("Delta=%.8le; Beta3=%.8le\n",delta,beta3);
  printf("nX3rt=%.8le; s=%.8le; Nyicp=%.8le\n",nX3rt,s,EiNySTGNOLM);

Py=pow(NySTGNOLM,2.0)*fabs(beta2)/(gamma(swlength)*pow(pulse_HW,2.0));

Px=pow(NxSTGNOLM,2.0)*fabs(beta2)/(gamma(swlength)*pow(pulse_HW,2.0));
  EoLNL=1.0/(gamma(swlength)*Py);// Non-linear length
  printf("Initial CW and CCW NOLM Pulses:\n");

printf("Power=%.8le;Beta2=%.8le;Gamma=%.8le\n",Py,beta2,gamma(swlength)
);
  printf("Soliton Period(z0)=%.8le;Dispersion
Length(LD)=%.8le\n",z0,LD);
  printf("Non-Linear Length(EoLNL)=%.8le\n",EoLNL);
  printf("Delta=%.8le; Beta3=%.8le\n",delta,beta3);
  printf("nX3rt=%.8le; s=%.8le; Nyiscp=%.8le\n",nX3rt,s,NySTGNOLM);
  //getchar();
  //exit(1);
  for(j=0;j<bpsize;j++)// Initialise all STG profiles.

ein1profile[j]=ein2profile[j]=eout1profile[j]=eout2profile[j]=sprofile[
j]=dcomplex(0.0,0.0);

nolmswitch(XCALPHA,inputprofile,ein2profile,eout1profile,eout2profile);
// STG-NOLM y-axis (fast axis) input.
  for(j=0;j<bpsize;j++)// Initialise STG signal pulse profile.
    sprofile[j]=eout1profile[j]; // Signal pulse (co-propagating with
control pulse) for STG only.
  dfft(eout1profile,bpsize,1.0);//Generate FFT.
  dfft(eout2profile,bpsize,1.0);//Generate FFT.
  dfft(inputprofile,bpsize,1.0); // Generate FFT.
  fpsdx1max=fpsdy1max=pow(abs(eout1profile[0]),2.0)/pow(bpsize,2.0);//
Power spectral density for frequency zero - maximum power component.
  fpsdy2max=pow(abs(eout2profile[0]),2.0)/pow(bpsize,2.0);// Power
spectral density for frequency zero - maximum power component.
  fpsdyimax=pow(abs(eout2profile[0]),2.0)/pow(bpsize,2.0);// Power
spectral density for frequency zero - maximum power component.
  dfft(eout1profile,bpsize,-1.0);//Generate IFFT
  dfft(eout2profile,bpsize,-1.0);//Generate IFFT
  dfft(inputprofile,bpsize,-1.0);//Generate IFFT
  for(j=0;j<bpsize;j++){ // Normalise IFFT
    eout1profile[j]=eout1profile[j]/((double)(bpsize));// Clockwise
(cw) control pulse (co-propagating with signal pulse).
    eout2profile[j]=eout2profile[j]/((double)(bpsize)); // Counter-
clockwise (ccw) control pulse (propagating alone).
    inputprofile[j]=inputprofile[j]/((double)(bpsize)); // Counter-
clockwise input port control pulse (propagating alone).
    }
  if((dfpR=fopen("srp.m","a+b"))==NULL){ // Switch raw profile complex
data file for recovery.
    printf("Unable to open output file srp.m\n");
    exit(1);
    }
  fread(&R,sizeof(SAVE),1,dfpR); // Detect end of file with fread()
else ftell() or rewind has no effect on current file pointer position.
  if(ftell(dfpR)){ //If file contains past beam profile snap reload to
continue.
    printf("Recovering from power failure termination ...\n");
//Recovering from power failure termination of program.
    rewind(dfpR);
    fread(&R,sizeof(SAVE),1,dfpR);  // Initialise R context.
    fclose(dfpR);
    if(R.segmentcounter==0){
      printf("At end of propagation length.\n");
      printf("Please re-initialize total number of segment length count
(integer):");
      scanf("%ld",&segmentcounter);
      if(!segmentcounter)//If zero or non-real character was entered,
then terminate.
```

```
          exit(1); // Exit.
        }
      else{
        segmentcounter=R.segmentcounter;// Load last segmentcounter.
        }
      printf("Number of segment length counts
left=%ld\n",segmentcounter);
      for(j=0;j<bpsize;j++){ // Load saved single/co-propagation
profiles.
        sprofile[j]=R.bprofilex1[j]; // Signal pulse (co-propagating with
control pulse).
        eout1profile[j]=R.bprofiley1[j];// Clockwise (cw) control pulse
(co-propagating with signal pulse).
        eout2profile[j]=R.bprofiley2[j];// Counter-clockwise (ccw)
control pulse (propagating alone).
        }
      fpsdx1max=R.fpsdx1max;fpsdy1max=R.fpsdy1max;fpsdy2max=R.fpsdy2max;
// Reload power spectral densities.
      }
    else{
      fclose(dfpR);
      if((dfpSPORT1=fopen("sport1.m","a+b"))==NULL){ // STG-NOLM switch
time domain data file.
        printf("Unable to open output file sport1.m\n");
        exit(1);
        }
      if((dfpSPORT2=fopen("sport2.m","a+b"))==NULL){ // STG-NOLM switch
time domain data file.
        printf("Unable to open output file sport2.m\n");
        exit(1);
        }
      if((dfpPCWtCCW=fopen("sportpd.m","a+b"))==NULL){ // STG-NOLM switch
port1-port2 phase difference data file.
        printf("Unable to open output file sportpd.m\n");
        exit(1);
        }
      for(j=0;j<bpsize;j++){
        Wy1[j]=norm(inputprofile[j]); //Returns the squared magnitude
(norm) of input port control profile.
        Wy2[j]=norm(ein2profile[j]); //Returns the squared magnitude
(norm) of output port control profile.
        }
      fwrite(Wy1,sizeof(double),bpsize,dfpSPORT1); // Save initial input
pulse temporal profile of STG-NOLM on y-axis (fast axis) at input port.
      fwrite(Wy2,sizeof(double),bpsize,dfpSPORT2); // Save initial input
pulse temporal profile of STG-NOLM on y-axis (fast axis) at outputport.
      fclose(dfpSPORT1);
      fclose(dfpSPORT2);
      if((dfpSPORT1F=fopen("sport1f.m","a+b"))==NULL){ // STG-NOLM switch
frequency domain data file.
        printf("Unable to open output file sport1f.m\n");
        exit(1);
        }
    if((dfpSPORT2F=fopen("sport2f.m","a+b"))==NULL){ // STG-NOLM switch
frequency domain data file.
      printf("Unable to open output file sport2f.m\n");
      exit(1);
      }
    dfft(inputprofile,bpsize,1.0);//Generate FFT of control (fast=y
axis) on input port.
    dfft(ein2profile,bpsize,1.0);//Generate FFT of control (fast=y
axis) on output port.
    for(j=0;j<bpsize/2;j++){//Re-organise discrete frequencies from -
N/2 - 0 - N/2.  FFT has it inside out.
      if(EiPy!=0.0){

Fy1[j+bpsize/2]=pow(abs(inputprofile[j]),2.0)/(pow(bpsize,2.0)*fpsdyima
x);// Control input port frequency power spectral density.

Fy2[j+bpsize/2]=pow(abs(ein2profile[j]),2.0)/(pow(bpsize,2.0)*fpsdyimax
);// Control output port frequency power spectral density.
        }
      else{
        Fy1[j+bpsize/2]=0.0;
        Fy2[j+bpsize/2]=0.0;
        }
      }
    for(j=bpsize/2;j<bpsize;j++){
      if(EiPy!=0.0){
        Fy1[j-
bpsize/2]=pow(abs(inputprofile[j]),2.0)/(pow(bpsize,2.0)*fpsdyimax); //
Control input port frequency power spectral density.
        Fy2[j-
bpsize/2]=pow(abs(ein2profile[j]),2.0)/(pow(bpsize,2.0)*fpsdyimax); //
Control output port frequency power spectral density.
        }
      else{
        Fy1[j-bpsize/2]=0.0;
        Fy2[j-bpsize/2]=0.0;
        }
      }

nolmswitch(XCALPHA,eout2profile,eout1profile,ein1profile,ein2profile);
    dfft(inputprofile,bpsize,-1.0);//Generate IFFT for control (fast=y
axis) on input port.
    for(j=0;j<bpsize;j++){ // Normalise IFFT for control (fast=y axis)
input and output profiles.
      inputprofile[j]=inputprofile[j]/((double)(bpsize));
      phasediff[j]=arg(ein1profile[j])-arg(ein2profile[j]);
      } // Save frequency power spectral density data files.
    fwrite(Fy1,sizeof(double),bpsize,dfpSPORT1F); // Save initial input
pulse spectral profile of STG-NOLM on y-axis (fast axis) at input port.
    fwrite(Fy2,sizeof(double),bpsize,dfpSPORT2F); // Save initial input
pulse spectral profile of STG-NOLM on y-axis (fast axis) at outputport.
```

```
    fwrite(phasediff,sizeof(double),bpsize,dfpPCWtCCW); // Save initial
input pulse spectral profile of STG-NOLM on y-axis (fast axis) at
outputport.
    fclose(dfpSPORT1F);
    fclose(dfpSPORT2F);
    fclose(dfpPCWtCCW);
    }
  while(segmentcounter>0){// Propagate the 3 STG pulses in NOLM over
constant length.
    printf("At segment count: - %ld\n",segmentcounter);
    if(START){
      profiletshift(SIGN,timeshift,windowsize,sprofile);
      START=0;
      }
    if(!(togglecounter%2)){// Even number of segments or odd number of
splices or axis switches.
      printf("Even number of segments or odd number of splices or axis
switches.\n");
      if(AXISTOGGLE==0){// Signal and control on slow and fast axis
respectively.
        nolmpropsolitonps(sprofile,eout1profile,spwidth,1.0e-
2,segmentlength,2000.0e-15);// Single and cw contol pulses co-
propagating.
        nolmpropsolitonp(XAXIS,eout2profile,spwidth,1.0e-
2,segmentlength,2000.0e-15);// CCW pulse propagating alone.

nolmswitch(XCALPHA,eout2profile,eout1profile,ein1profile,ein2profile);
        AXISTOGGLE=1;
        }
      else{ // Signal and control on fast and slow axis respectively.
        nolmpropsolitonps(eout1profile,sprofile,spwidth,1.0e-
2,segmentlength,2000.0e-15);// Single and cw contol pulses co-
propagating.
        nolmpropsolitonp(YAXIS,eout2profile,spwidth,1.0e-
2,segmentlength,2000.0e-15);// CCW pulse propagating alone.

nolmswitch(XCALPHA,eout2profile,eout1profile,ein1profile,ein2profile);
        AXISTOGGLE=0;
        }
      }
    else{ // Odd number of segments or even number of splices or axis
switches.
      printf("Odd number of segments or even number of splices or axis
switches.\n");
      if(AXISTOGGLE==0){// Signal and control on slow and fast axis
respectively.
        nolmpropsolitonps(sprofile,eout1profile,spwidth,1.0e-
2,segmentlength,2000.0e-15);// Single and cw contol pulses co-
propagating.
        nolmpropsolitonp(YAXIS,eout2profile,spwidth,1.0e-
2,segmentlength,2000.0e-15);// CCW pulse propagating alone.

nolmswitch(XCALPHA,eout2profile,eout1profile,ein1profile,ein2profile);
```

```
        AXISTOGGLE=1;
        }
      else{ // Signal and control on fast and slow axis respectively.
        nolmpropsolitonps(eout1profile,sprofile,spwidth,1.0e-
2,segmentlength,2000.0e-15);// Single and cw contol pulses co-
propagating.
        nolmpropsolitonp(XAXIS,eout2profile,spwidth,1.0e-
2,segmentlength,2000.0e-15);// CCW pulse propagating alone.

nolmswitch(XCALPHA,eout2profile,eout1profile,ein1profile,ein2profile);
        AXISTOGGLE=0;
        }
      }
    segmentcounter--;
    splicenumber++;

if((!(togglecounter%2)&&(!(splicenumber%2)))||(togglecounter%2)&&(splic
enumber%2)){// Even segments or odd axis switches.
      printf("Number of axis switches=%ld; Segment count=%ld; At splice
#=%ld\n",togglecounter-1,segmentcounter,splicenumber);
      for(j=0;j<bpsize;j++){
        Wx1[j]=norm(sprofile[j]); //Returns the squared magnitude
(norm) of signal, cw and ccw control profiles.
        Wy1[j]=norm(eout1profile[j]);
        Wy2[j]=norm(eout2profile[j]);
        }
      if((dfpx1=fopen("ppdatax1.m","a+b"))==NULL){// Profile Matlab
data plot file.
        printf("Unable to open output file ppdatax1.m\n");
        return;
        }
      if((dfpy1=fopen("ppdatay1.m","a+b"))==NULL){// Profile Matlab
data plot file.
        printf("Unable to open output file ppdatay1.m\n");
        return;
        }
      if((dfpy2=fopen("ppdatay2.m","a+b"))==NULL){// Profile Matlab
data plot file.
        printf("Unable to open output file ppdatay2.m\n");
        return;
        }
      if((dfpFx1=fopen("ppdatafx1.m","a+b"))==NULL){ // Frequency power
spectral density data file.
        printf("Unable to open output file ppdatafx1.m\n");
        exit(1);
        }
      if((dfpFy1=fopen("ppdatafy1.m","a+b"))==NULL){ // Frequency power
spectral density data file.
        printf("Unable to open output file ppdatafy1.m\n");
        exit(1);
        }
      if((dfpFy2=fopen("ppdatafy2.m","a+b"))==NULL){ // Frequency power
spectral density data file.
```

```
        printf("Unable to open output file ppdatafy2.m\n");
        exit(1);
        }
      if((dfpR=fopen("srp.m","w+b"))==NULL){ // Switch raw profile
complex data file for recovery.
        printf("Unable to open output file srp.m\n");
        exit(1);
        }
      fwrite(Wx1,sizeof(double),bpsize,dfpx1);// Save profile Matlab
data plot files.
      fwrite(Wy1,sizeof(double),bpsize,dfpy1);
      fwrite(Wy2,sizeof(double),bpsize,dfpy2);
      R.segmentcounter=segmentcounter;  // Save context.
      for(j=0;j<bpsize;j++){ // Save single/co-propagation profiles.
      R.bprofilex1[j]=sprofile[j]; // Signal pulse (co-propagating with
control pulse).
      R.bprofiley1[j]=eout1profile[j];// Clockwise (cw) control pulse
(co-propagating with signal pulse).
      R.bprofiley2[j]=eout2profile[j];// Counter-clockwise (ccw)
control pulse (propagating alone).
      }

R.fpsdx1max=fpsdx1max;R.fpsdy1max=fpsdy1max;R.fpsdy2max=fpsdy2max; //
Power spectral densities.
      fwrite(&R,sizeof(SAVE),1,dfpR);  // Save profiles complex data
file for recovery.
      dfft(sprofile,bpsize,1.0);//Generate FFT of signal profile on
slow (x) axis.
      dfft(eout1profile,bpsize,1.0);//Generate FFT of cw control
profile on fast (y) axis.
      dfft(eout2profile,bpsize,1.0);//Generate FFT of ccw control
profile on fast (y) axis.
      for(j=0;j<bpsize/2;j++){//Re-organise discrete frequencies from -
N/2 - 0 - N/2.  FFT has it inside out.
        if(EiPy!=0.0){

Fx1[j+bpsize/2]=pow(abs(sprofile[j]),2.0)/(pow(bpsize,2.0)*fpsdx1max);/
/ Signal frequency power spectral density.

Fy1[j+bpsize/2]=pow(abs(eout1profile[j]),2.0)/(pow(bpsize,2.0)*fpsdy1ma
x);//CW control frequency power spectral density.

Fy2[j+bpsize/2]=pow(abs(eout2profile[j]),2.0)/(pow(bpsize,2.0)*fpsdy2ma
x);//CCW control frequency power spectral density.
          }
        else{
          Fx1[j+bpsize/2]=0.0;
          Fy1[j+bpsize/2]=0.0;
          Fy2[j+bpsize/2]=0.0;
          }
        }
      for(j=bpsize/2;j<bpsize;j++){
        if(EiPy!=0.0){
          Fx1[j-
bpsize/2]=pow(abs(sprofile[j]),2.0)/(pow(bpsize,2.0)*fpsdx1max);//Signa
l frequency power spectral density.
          Fy1[j-
bpsize/2]=pow(abs(eout1profile[j]),2.0)/(pow(bpsize,2.0)*fpsdy1max);//
CW control requency power spectral density.
          Fy2[j-
bpsize/2]=pow(abs(eout2profile[j]),2.0)/(pow(bpsize,2.0)*fpsdy2max);//
CCW control frequency power spectral density.
          }
        else{
          Fx1[j-bpsize/2]=0.0;
          Fy1[j-bpsize/2]=0.0;
          Fy2[j-bpsize/2]=0.0;
          }
        }
      dfft(sprofile,bpsize,-1.0);//Generate IFFT for signal profile on
slow (x) axis.
      dfft(eout1profile,bpsize,-1.0);//Generate IFFT for cw control
profile on fast (y) axis.
      dfft(eout2profile,bpsize,-1.0);//Generate IFFT for ccw control
profile on fast (y) axis.
      for(j=0;j<bpsize;j++){ // Normalise IFFT for profiles x and y.
        sprofile[j]=sprofile[j]/((double)(bpsize));
        eout1profile[j]=eout1profile[j]/((double)(bpsize));
        eout2profile[j]=eout2profile[j]/((double)(bpsize));
        }  // Save frequency power spectral density data files.
      fwrite(Fx1,sizeof(double),bpsize,dfpFx1); // Save signal data
file.
      fwrite(Fy1,sizeof(double),bpsize,dfpFy1);  // Save cw control
data file.
      fwrite(Fy2,sizeof(double),bpsize,dfpFy2);  // Save ccw control
data file.
      fclose(dfpx1);fclose(dfpy1);fclose(dfpy2); // Close all data
files.
      fclose(dfpR);
      fclose(dfpFx1);fclose(dfpFy1);fclose(dfpFy1);
      if((dfpSPORT1=fopen("sport1.m","a+b"))==NULL){ // STG-NOLM switch
time domain data file.
        printf("Unable to open output file sport1.m\n");
        exit(1);
        }
      if((dfpSPORT2=fopen("sport2.m","a+b"))==NULL){ // STG-NOLM switch
time domain data file.
        printf("Unable to open output file sport2.m\n");
        exit(1);
        }
      if((dfpPCWtCCW=fopen("sportpd.m","a+b"))==NULL){ // STG-NOLM
switch frequency domain data file.
        printf("Unable to open output file sportpd.m\n");
        exit(1);
        }
      for(j=0;j<bpsize;j++){
        Wy1[j]=norm(ein1profile[j]); //Returns the squared magnitude
(norm) of input port control profile.
        Wy2[j]=norm(ein2profile[j]); //Returns the squared magnitude
(norm) of output port control profile.
        phasediff[j]=arg(ein1profile[j])-arg(ein2profile[j]);
        }
      fwrite(Wy1,sizeof(double),bpsize,dfpSPORT1); // Save output port1
pulse temporal profile of STG-NOLM on y-axis (fast axis) at input port.
      fwrite(Wy2,sizeof(double),bpsize,dfpSPORT2); // Save output port2
pulse temporal profile of STG-NOLM on y-axis (fast axis) at outputport.
      fwrite(phasediff,sizeof(double),bpsize,dfpPCWtCCW); // Save port1
and port2 pulse phase difference profile of STG-NOLM on y-axis (fast
axis) at outputport.
      fclose(dfpSPORT1);
      fclose(dfpSPORT2);
      fclose(dfpPCWtCCW);
      if((dfpSPORT1F=fopen("sport1f.m","a+b"))==NULL){ // STG-NOLM
switch frequency domain data file.
        printf("Unable to open output file sport1f.m\n");
        exit(1);
        }
      if((dfpSPORT2F=fopen("sport2f.m","a+b"))==NULL){ // STG-NOLM
switch frequency domain data file.
        printf("Unable to open output file sport2f.m\n");
        exit(1);
        }
      dfft(ein1profile,bpsize,1.0);//Generate FFT of control (fast=y
axis) on input port.
      dfft(ein2profile,bpsize,1.0);//Generate FFT of control (fast=y
axis) on output port.
      for(j=0;j<bpsize/2;j++){//Re-organise discrete frequencies from -
N/2 - 0 - N/2.  FFT has it inside out.
        if(EiPy!=0.0){

Fy1[j+bpsize/2]=pow(abs(ein1profile[j]),2.0)/(pow(bpsize,2.0)*fpsdyimax
);// Control input port frequency power spectral density.

Fy2[j+bpsize/2]=pow(abs(ein2profile[j]),2.0)/(pow(bpsize,2.0)*fpsdyimax
);// Control output port frequency power spectral density.
          }
        else{
          Fy1[j+bpsize/2]=0.0;
          Fy2[j+bpsize/2]=0.0;
          }
        }
      for(j=bpsize/2;j<bpsize;j++){
        if(EiPy!=0.0){
          Fy1[j-
bpsize/2]=pow(abs(ein1profile[j]),2.0)/(pow(bpsize,2.0)*fpsdyimax); //
Control input port frequency power spectral density.
          Fy2[j-
bpsize/2]=pow(abs(ein2profile[j]),2.0)/(pow(bpsize,2.0)*fpsdyimax); //
Control output port frequency power spectral density.
```

```
          }
        else{
          Fy1[j-bpsize/2]=0.0;
          Fy2[j-bpsize/2]=0.0;
          }
        }// Save frequency power spectral density data files.
      fwrite(Fy1,sizeof(double),bpsize,dfpSPORT1F); // Save initial
input pulse spectral profile of STG-NOLM on y-axis (fast axis) at input
port.
      fwrite(Fy2,sizeof(double),bpsize,dfpSPORT2F); // Save initial
input pulse spectral profile of STG-NOLM on y-axis (fast axis) at
outputport.
      fclose(dfpSPORT1F);
      fclose(dfpSPORT2F);
      }
    }
  free(ein1profile);
  free(ein2profile);
  free(eout1profile);
  free(eout2profile);
  free(sprofile);
  }


/***********************************************************************
**************************
 Name: stgnolm()
Function: Implements a SDG-NOLM in an optical fiber by Symmetrized
Split Step Fourier Method.
          Zero attenuation is assumed.
Author: P.C. Chimfwembe
Date: 02/05/99
Modified: 03/02/99
************************************************************************
*************************
*/
void stgnolm(dcomplex *inputprofile,double pulse_FWHM,
                   double zstep, /*Fraction of soliton period */
                   double zlength,double windowsize){
  FILE *dfpR;// Data recovery file "last profiles (Single/Co-
propagation) recorded" data file.
  FILE *dfpSPORT1,*dfpSPORT2,*dfpSPORT1F,*dfpSPORT2F;// STG-NOLM switch
file.
  FILE *dfpx1,*dfpy1,*dfpy2,*dfpFx1,*dfpFy1,*dfpFy2,*dfpPCWtCCW; //Data
file pointers.
  dcomplex
*ein1profile,*ein2profile,*eout1profile,*eout2profile,*sprofile;
  double *phasediff,phaseCW,phaseCCW; // Output port phase difference
ie PhaseCW-PhaseCCW
  double nz0=PI/2.0,z0,LD;  // Normalized soliton period, soliton
period and dispersion length.
  double beta2=GVD(swlength); /* Fiber effective Beta2 (GVDx=GVDy) */
  double beta3=6.2e-42;//x and y 3rd order dispersions (s^3/m)
(TODx=TODy). Fiber core radius=2.27e-6 m; Core:13.5m/o GeO2,85.5m/o
SiO2;Cladding:SiO2
  double pulse_HW=pulse_FWHM/(2.0*log(1.0+sqrt(2.0)));//To:Half width
at 1/e intensity point.
  double wo=2*PI*C/swlength; // Carrier radian frequency.
  double delta,s,nX3rt;
  double EiLNL,EoLNL;// Nonlinear length and normalized zstep
  double Nx=NxSTGNOLM,Ny=NySTGNOLM;// Soliton order for x and y axes.
  double power=0.0; // Power buffer.
  double walkofflength=(pulse_FWHM*C)/dn; // Walk-off length of
orthogonal pulses.
  double segmentlength=walkofflength*1.0;  // Optimized segment length.
  double timeshift=pulse_HW*4.0;//Time shift of signal with respect to
control at coupler input to NOLM loop.
  unsigned int SIGN=POSITIVE; // Timeshift variable.
  double
*Wx1,*Wy1,*Wy2,*Fx1,*Fy1,*Fy2,fpsdx1max=0.0,fpsdy1max=0.0,fpsdy2max=0.0
,fpsdyimax=0.0; // Time and frequency intensities.
  unsigned int segmentcounter=0; // Number of walk-off lengths.
  int START=1; // Signal time shift flag.
  unsigned int i,j; //Iteration variables.
  SAVESWITCH R; // Recovery data structure for single/co-propagation.
  if((ein1profile=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer ein1profile\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((ein2profile=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer ein2profile\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((eout1profile=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer eout1profile\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((eout2profile=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer eout2profile\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((sprofile=(dcomplex *)calloc(bpsize,sizeof(dcomplex)))==NULL){
    printf("Not enough memory to allocate buffer sprofile\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((Wx1=(double *)calloc(bpsize,sizeof(double)))==NULL){
    printf("Not enough memory to allocate buffer Wx1\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((Wy1=(double *)calloc(bpsize,sizeof(double)))==NULL){
    printf("Not enough memory to allocate buffer Wy1\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((Wy2=(double *)calloc(bpsize,sizeof(double)))==NULL){
    printf("Not enough memory to allocate buffer Wy2\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((Fx1=(double *)calloc(bpsize,sizeof(double)))==NULL){
    printf("Not enough memory to allocate buffer Wx1\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((Fy1=(double *)calloc(bpsize,sizeof(double)))==NULL){
    printf("Not enough memory to allocate buffer Wy1\n");
    exit(1);  /* terminate program if out of memory */
    }
  if((Fy2=(double *)calloc(bpsize,sizeof(double)))==NULL){
    printf("Not enough memory to allocate buffer Wy2\n");
    exit(1);  /* terminate program if out of memory */
    }
   if((phasediff=(double *)calloc(bpsize,sizeof(double)))==NULL){
    printf("Not enough memory to allocate buffer phasediff\n");
    exit(1);  /* terminate program if out of memory */
    }
  //Setting of delta,s, and nX3rt.
  delta=beta3/(6.0*fabs(beta2)*pulse_HW);
  s=2.0/(wo*pulse_HW);
  nX3rt=X3RT/pulse_HW; // Normalized Chi-3 response time.
  LD=pow(pulse_HW,2.0)/fabs(beta2);// Dispersion length.
  z0=LD*nz0;  // Soliton period in y polarization as reference plane.
  segmentcounter=floor(zlength/segmentlength); // Number of walk-off
lengths. Optimize with floor or ceil.

EiPy=pow(EiNySTGNOLM,2.0)*fabs(beta2)/(gamma(swlength)*pow(pulse_HW,2.0
));
  EiLNL=1.0/(gamma(swlength)*EiPy);// Non-linear length
  gf1solitonp(inputprofile,pulse_FWHM,windowsize);// Initialise input
soliton control beam profile (y-axis or fast axis).
  for(j=0;j<bpsize;j++){
    inputprofile[j]=inputprofile[j]*EiNySTGNOLM;
    }
  printf("NOLM characteristics:\n");
  printf("Walk-off length (lwo)=%.8le; Segment length
(Ls)=%.8le\n",walkofflength,segmentlength);
  printf("Number of segments in STG-NOLM=%ld\n",segmentcounter);
  printf("11 segment length(z)=%.8le\n",segmentlength*11.0);
  printf("Input Control Pulse (Ein1):\n");
  printf("Control
Power=%.8le;Beta2=%.8le;Gamma=%.8le\n",EiPy,beta2,gamma(swlength));
  printf("Soliton Period(z0)=%.8le;Dispersion
Length(LD)=%.8le\n",z0,LD);
  printf("Non-Linear Length(EiLNL)=%.8le\n",EiLNL);
  printf("Delta=%.8le; Beta3=%.8le\n",delta,beta3);
  printf("nX3rt=%.8le; s=%.8le; Nyicp=%.8le\n",nX3rt,s,EiNySTGNOLM);

Py=pow(NySTGNOLM,2.0)*fabs(beta2)/(gamma(swlength)*pow(pulse_HW,2.0));

Px=pow(NxSTGNOLM,2.0)*fabs(beta2)/(gamma(swlength)*pow(pulse_HW,2.0));
  EoLNL=1.0/(gamma(swlength)*Py);// Non-linear length
  printf("Initial CW and CCW NOLM Pulses:\n");

printf("Power=%.8le;Beta2=%.8le;Gamma=%.8le\n",Py,beta2,gamma(swlength)
);
  printf("Soliton Period(z0)=%.8le;Dispersion
Length(LD)=%.8le\n",z0,LD);
  printf("Non-Linear Length(EoLNL)=%.8le\n",EoLNL);
  printf("Delta=%.8le; Beta3=%.8le\n",delta,beta3);
  printf("nX3rt=%.8le; s=%.8le; Nyiscp=%.8le\n",nX3rt,s,NySTGNOLM);
  //getchar();
  //exit(1);
  for(j=0;j<bpsize;j++)// Initialise all STG profiles.

ein1profile[j]=ein2profile[j]=eout1profile[j]=eout2profile[j]=sprofile[
j]=dcomplex(0.0,0.0);

nolmswitch(XCALPHA,inputprofile,ein2profile,eout1profile,eout2profile);
// STG-NOLM y-axis (fast axis) input.
  for(j=0;j<bpsize;j++)// Initialise STG signal pulse profile.
    sprofile[j]=eout1profile[j]; // Signal pulse (co-propagating with
control pulse) for STG only.
  dfft(eout1profile,bpsize,1.0);//Generate FFT.
  dfft(eout2profile,bpsize,1.0);//Generate FFT.
  dfft(inputprofile,bpsize,1.0); // Generate FFT.
  fpsdx1max=fpsdy1max=pow(abs(eout1profile[0]),2.0)/pow(bpsize,2.0);//
Power spectral density for frequency zero - maximum power component.
  fpsdy2max=pow(abs(eout2profile[0]),2.0)/pow(bpsize,2.0);// Power
spectral density for frequency zero - maximum power component.
  fpsdyimax=pow(abs(eout2profile[0]),2.0)/pow(bpsize,2.0);// Power
spectral density for frequency zero - maximum power component.
  dfft(eout1profile,bpsize,-1.0);//Generate IFFT
  dfft(eout2profile,bpsize,-1.0);//Generate IFFT
  dfft(inputprofile,bpsize,-1.0);//Generate IFFT
  for(j=0;j<bpsize;j++){ // Normalise IFFT
    eout1profile[j]=eout1profile[j]/((double)(bpsize));// Clockwise
(cw) control pulse (co-propagating with signal pulse).
    eout2profile[j]=eout2profile[j]/((double)(bpsize)); // Counter-
clockwise (ccw) control pulse (propagating alone).
    inputprofile[j]=inputprofile[j]/((double)(bpsize)); // Counter-
clockwise input port control pulse (propagating alone).
    }
  if((dfpR=fopen("srp.m","a+b"))==NULL){ // Switch raw profile complex
data file for recovery.
    printf("Unable to open output file srp.m\n");
    exit(1);
    }
  fread(&R,sizeof(SAVE),1,dfpR); // Detect end of file with fread()
else ftell() or rewind has no effect on current file pointer position.
  if(ftell(dfpR)){ //If file contains past beam profile snap reload to
continue.
    printf("Recovering from power failure termination ...\n");
//Recovering from power failure termination of program.
    rewind(dfpR);
    fread(&R,sizeof(SAVE),1,dfpR);  // Initialise R context.
    fclose(dfpR);
    if(R.segmentcounter==0){
      printf("At end of propagation length.\n");
      printf("Please re-initialize total number of segment length count
(integer):");
      scanf("%ld",&segmentcounter);
      if(!segmentcounter)//If zero or non-real character was entered,
then terminate.
        exit(1); // Exit.
      }
    else{
      segmentcounter=R.segmentcounter;// Load last segmentcounter.
      }
    printf("Number of segment length counts
left=%ld\n",segmentcounter);
    for(j=0;j<bpsize;j++){ // Load saved single/co-propagation
profiles.
      sprofile[j]=R.bprofilex1[j]; // Signal pulse (co-propagating with
control pulse).
      eout1profile[j]=R.bprofiley1[j];// Clockwise (cw) control pulse
(co-propagating with signal pulse).
      eout2profile[j]=R.bprofiley2[j];// Counter-clockwise (ccw)
control pulse (propagating alone).
      }
    fpsdx1max=R.fpsdx1max;fpsdy1max=R.fpsdy1max;fpsdy2max=R.fpsdy2max;
// Reload power spectral densities.
    }
  else{
    fclose(dfpR);
    if((dfpSPORT1=fopen("sport1.m","a+b"))==NULL){ // STG-NOLM switch
time domain data file.
      printf("Unable to open output file sport1.m\n");
      exit(1);
      }
    if((dfpSPORT2=fopen("sport2.m","a+b"))==NULL){ // STG-NOLM switch
time domain data file.
      printf("Unable to open output file sport2.m\n");
      exit(1);
      }
    if((dfpPCWtCCW=fopen("sportpd.m","a+b"))==NULL){ // STG-NOLM switch
port1-port2 phase difference data file.
      printf("Unable to open output file sportpd.m\n");
      exit(1);
      }
    for(j=0;j<bpsize;j++){
      Wy1[j]=norm(inputprofile[j]); //Returns the squared magnitude
(norm) of input port control profile.
      Wy2[j]=norm(ein2profile[j]); //Returns the squared magnitude
(norm) of output port control profile.
      }
    fwrite(Wy1,sizeof(double),bpsize,dfpSPORT1); // Save initial input
pulse temporal profile of STG-NOLM on y-axis (fast axis) at input port.
```

```
    fwrite(Wy2,sizeof(double),bpsize,dfpSPORT2); // Save initial input
pulse temporal profile of STG-NOLM on y-axis (fast axis) at outputport.
    fclose(dfpSPORT1);
    fclose(dfpSPORT2);
    if((dfpSPORT1F=fopen("sport1f.m","a+b"))==NULL){ // STG-NOLM switch
frequency domain data file.
      printf("Unable to open output file sport1f.m\n");
      exit(1);
      }
    if((dfpSPORT2F=fopen("sport2f.m","a+b"))==NULL){ // STG-NOLM switch
frequency domain data file.
      printf("Unable to open output file sport2f.m\n");
      exit(1);
      }
    dfft(inputprofile,bpsize,1.0);//Generate FFT of control (fast=y
axis) on input port.
    dfft(ein2profile,bpsize,1.0);//Generate FFT of control (fast=y
axis) on output port.
    for(j=0;j<bpsize/2;j++){//Re-organise discrete frequencies from -
N/2 - 0 - N/2.  FFT has it inside out.
      if(EiPy!=0.0){

Fy1[j+bpsize/2]=pow(abs(inputprofile[j]),2.0)/(pow(bpsize,2.0)*fpsdyima
x);// Control input port frequency power spectral density.

Fy2[j+bpsize/2]=pow(abs(ein2profile[j]),2.0)/(pow(bpsize,2.0)*fpsdyimax
);// Control output port frequency power spectral density.
        }
      else{
        Fy1[j+bpsize/2]=0.0;
        Fy2[j+bpsize/2]=0.0;
        }
      }
    for(j=bpsize/2;j<bpsize;j++){
      if(EiPy!=0.0){
        Fy1[j-
bpsize/2]=pow(abs(inputprofile[j]),2.0)/(pow(bpsize,2.0)*fpsdyimax); //
Control input port frequency power spectral density.
        Fy2[j-
bpsize/2]=pow(abs(ein2profile[j]),2.0)/(pow(bpsize,2.0)*fpsdyimax); //
Control output port frequency power spectral density.
        }
      else{
        Fy1[j-bpsize/2]=0.0;
        Fy2[j-bpsize/2]=0.0;
        }
      }

nolmswitch(XCALPHA,eout2profile,eout1profile,ein1profile,ein2profile);
    dfft(inputprofile,bpsize,-1.0);//Generate IFFT for control (fast=y
axis) on input port.
    for(j=0;j<bpsize;j++){ // Normalise IFFT for control (fast=y axis)
input and output profiles.
```

```
      inputprofile[j]=inputprofile[j]/((double)(bpsize));
      phasediff[j]=arg(ein1profile[j])-arg(ein2profile[j]);
      }  // Save frequency power spectral density data files.
    fwrite(Fy1,sizeof(double),bpsize,dfpSPORT1F); // Save initial input
pulse spectral profile of STG-NOLM on y-axis (fast axis) at input port.
    fwrite(Fy2,sizeof(double),bpsize,dfpSPORT2F); // Save initial input
pulse spectral profile of STG-NOLM on y-axis (fast axis) at outputport.
    fwrite(phasediff,sizeof(double),bpsize,dfpPCWtCCW); // Save initial
input pulse spectral profile of STG-NOLM on y-axis (fast axis) at
outputport.
    fclose(dfpSPORT1F);
    fclose(dfpSPORT2F);
    fclose(dfpPCWtCCW);
    }
  while(segmentcounter>0){// Propagate the 3 STG pulses in NOLM over
constant length.
    printf("At segment count: - %ld\n",segmentcounter);
    if(START){
      profiletshift(SIGN,timeshift,windowsize,sprofile);
      START=0;
      }
    nolmpropsolitonps(sprofile,eout1profile,spwidth,1.0e-
2,segmentlength,2000.0e-15);// Single and cw contol pulses co-
propagating.
    nolmpropsolitonp(YAXIS,eout2profile,spwidth,1.0e-
2,segmentlength,2000.0e-15);// CCW pulse propagating alone.

nolmswitch(XCALPHA,eout2profile,eout1profile,ein1profile,ein2profile);
    segmentcounter--;
    for(j=0;j<bpsize;j++){
      Wx1[j]=norm(sprofile[j]); //Returns the squared magnitude (norm)
of signal, cw and ccw control profiles.
      Wy1[j]=norm(eout1profile[j]);
      Wy2[j]=norm(eout2profile[j]);
      }
    if((dfpx1=fopen("ppdatax1.m","a+b"))==NULL){// Profile Matlab data
plot file.
      printf("Unable to open output file ppdatax1.m\n");
      return;
      }
    if((dfpy1=fopen("ppdatay1.m","a+b"))==NULL){// Profile Matlab data
plot file.
      printf("Unable to open output file ppdatay1.m\n");
      return;
      }
    if((dfpy2=fopen("ppdatay2.m","a+b"))==NULL){// Profile Matlab data
plot file.
      printf("Unable to open output file ppdatay2.m\n");
      return;
      }
    if((dfpFx1=fopen("ppdatafx1.m","a+b"))==NULL){ // Frequency power
spectral density data file.
      printf("Unable to open output file ppdatafx1.m\n");
```

```
        exit(1);
        }
    if((dfpFy1=fopen("ppdatafy1.m","a+b"))==NULL){ // Frequency power
spectral density data file.
        printf("Unable to open output file ppdatafy1.m\n");
        exit(1);
        }
    if((dfpFy2=fopen("ppdatafy2.m","a+b"))==NULL){ // Frequency power
spectral density data file.
        printf("Unable to open output file ppdatafy2.m\n");
        exit(1);
        }
    if((dfpR=fopen("srp.m","w+b"))==NULL){ // Switch raw profile
complex data file for recovery.
        printf("Unable to open output file srp.m\n");
        exit(1);
        }
    fwrite(Wx1,sizeof(double),bpsize,dfpx1);// Save profile Matlab data
plot files.
    fwrite(Wy1,sizeof(double),bpsize,dfpy1);
    fwrite(Wy2,sizeof(double),bpsize,dfpy2);
    R.segmentcounter=segmentcounter;  // Save context.
    for(j=0;j<bpsize;j++){ // Save single/co-propagation profiles.
        R.bprofilex1[j]=sprofile[j]; // Signal pulse (co-propagating with
control pulse).
        R.bprofiley1[j]=eout1profile[j];// Clockwise (cw) control pulse
(co-propagating with signal pulse).
        R.bprofiley2[j]=eout2profile[j];// Counter-clockwise (ccw)
control pulse (propagating alone).
        }
    R.fpsdx1max=fpsdx1max;R.fpsdy1max=fpsdy1max;R.fpsdy2max=fpsdy2max;
// Power spectral densities.
    fwrite(&R,sizeof(SAVE),1,dfpR);  // Save profiles complex data file
for recovery.
    dfft(sprofile,bpsize,1.0);//Generate FFT of signal profile on slow
(x) axis.
    dfft(eout1profile,bpsize,1.0);//Generate FFT of cw control profile
on fast (y) axis.
    dfft(eout2profile,bpsize,1.0);//Generate FFT of ccw control profile
on fast (y) axis.
    for(j=0;j<bpsize/2;j++){//Re-organise discrete frequencies from -
N/2 - 0 - N/2.  FFT has it inside out.
        if(EiPy!=0.0){

Fx1[j+bpsize/2]=pow(abs(sprofile[j]),2.0)/(pow(bpsize,2.0)*fpsdx1max);/
/ Signal frequency power spectral density.

Fy1[j+bpsize/2]=pow(abs(eout1profile[j]),2.0)/(pow(bpsize,2.0)*fpsdy1ma
x);//CW control frequency power spectral density.

Fy2[j+bpsize/2]=pow(abs(eout2profile[j]),2.0)/(pow(bpsize,2.0)*fpsdy2ma
x);//CCW control frequency power spectral density.
          }
```

```
      else{
        Fx1[j+bpsize/2]=0.0;
        Fy1[j+bpsize/2]=0.0;
        Fy2[j+bpsize/2]=0.0;
        }
      }
    for(j=bpsize/2;j<bpsize;j++){
      if(EiPy!=0.0){
        Fx1[j-
bpsize/2]=pow(abs(sprofile[j]),2.0)/(pow(bpsize,2.0)*fpsdx1max);//Signa
l frequency power spectral density.
        Fy1[j-
bpsize/2]=pow(abs(eout1profile[j]),2.0)/(pow(bpsize,2.0)*fpsdy1max);//
CW control requency power spectral density.
        Fy2[j-
bpsize/2]=pow(abs(eout2profile[j]),2.0)/(pow(bpsize,2.0)*fpsdy2max);//
CCW control frequency power spectral density.
        }
      else{
        Fx1[j-bpsize/2]=0.0;
        Fy1[j-bpsize/2]=0.0;
        Fy2[j-bpsize/2]=0.0;
        }
      }
    dfft(sprofile,bpsize,-1.0);//Generate IFFT for signal profile on
slow (x) axis.
    dfft(eout1profile,bpsize,-1.0);//Generate IFFT for cw control
profile on fast (y) axis.
    dfft(eout2profile,bpsize,-1.0);//Generate IFFT for ccw control
profile on fast (y) axis.
    for(j=0;j<bpsize;j++){ // Normalise IFFT for profiles x and y.
      sprofile[j]=sprofile[j]/((double)(bpsize));
      eout1profile[j]=eout1profile[j]/((double)(bpsize));
      eout2profile[j]=eout2profile[j]/((double)(bpsize));
      }  // Save frequency power spectral density data files.
    fwrite(Fx1,sizeof(double),bpsize,dfpFx1); // Save signal data file.
    fwrite(Fy1,sizeof(double),bpsize,dfpFy1);  // Save cw control data
file.
    fwrite(Fy2,sizeof(double),bpsize,dfpFy2);  // Save ccw control data
file.
    fclose(dfpx1);fclose(dfpy1);fclose(dfpy2); // Close all data files.
    fclose(dfpR);
    fclose(dfpFx1);fclose(dfpFy1);fclose(dfpFy1);
    if((dfpSPORT1=fopen("sport1.m","a+b"))==NULL){ // STG-NOLM switch
time domain data file.
      printf("Unable to open output file sport1.m\n");
      exit(1);
      }
    if((dfpSPORT2=fopen("sport2.m","a+b"))==NULL){ // STG-NOLM switch
time domain data file.
      printf("Unable to open output file sport2.m\n");
      exit(1);
      }
    if((dfpPCWtCCW=fopen("sportpd.m","a+b"))==NULL){ // STG-NOLM switch
frequency domain data file.
      printf("Unable to open output file sportpd.m\n");
      exit(1);
      }
    for(j=0;j<bpsize;j++){
      Wy1[j]=norm(ein1profile[j]); //Returns the squared magnitude
(norm) of input port control profile.
      Wy2[j]=norm(ein2profile[j]); //Returns the squared magnitude
(norm) of output port control profile.
      phasediff[j]=arg(ein1profile[j])-arg(ein2profile[j]);
      }
    fwrite(Wy1,sizeof(double),bpsize,dfpSPORT1); // Save output port1
pulse temporal profile of STG-NOLM on y-axis (fast axis) at input port.
    fwrite(Wy2,sizeof(double),bpsize,dfpSPORT2); // Save output port2
pulse temporal profile of STG-NOLM on y-axis (fast axis) at outputport.
    fwrite(phasediff,sizeof(double),bpsize,dfpPCWtCCW); // Save port1
and port2 pulse phase difference profile of STG-NOLM on y-axis (fast
axis) at outputport.
    fclose(dfpSPORT1);
    fclose(dfpSPORT2);
    fclose(dfpPCWtCCW);
    if((dfpSPORT1F=fopen("sport1f.m","a+b"))==NULL){ // STG-NOLM switch
frequency domain data file.
      printf("Unable to open output file sport1f.m\n");
      exit(1);
      }
    if((dfpSPORT2F=fopen("sport2f.m","a+b"))==NULL){ // STG-NOLM switch
frequency domain data file.
      printf("Unable to open output file sport2f.m\n");
      exit(1);
      }
    dfft(ein1profile,bpsize,1.0);//Generate FFT of control (fast=y
axis) on input port.
    dfft(ein2profile,bpsize,1.0);//Generate FFT of control (fast=y
axis) on output port.
    for(j=0;j<bpsize/2;j++){//Re-organise discrete frequencies from -
N/2 - 0 - N/2.  FFT has it inside out.
      if(EiPy!=0.0){

Fy1[j+bpsize/2]=pow(abs(ein1profile[j]),2.0)/(pow(bpsize,2.0)*fpsdyimax
);// Control input port frequency power spectral density.

Fy2[j+bpsize/2]=pow(abs(ein2profile[j]),2.0)/(pow(bpsize,2.0)*fpsdyimax
);// Control output port frequency power spectral density.
        }
      else{
        Fy1[j+bpsize/2]=0.0;
        Fy2[j+bpsize/2]=0.0;
        }
      }
    for(j=bpsize/2;j<bpsize;j++){
      if(EiPy!=0.0){
        Fy1[j-
bpsize/2]=pow(abs(ein1profile[j]),2.0)/(pow(bpsize,2.0)*fpsdyimax); //
Control input port frequency power spectral density.
        Fy2[j-
bpsize/2]=pow(abs(ein2profile[j]),2.0)/(pow(bpsize,2.0)*fpsdyimax); //
Control output port frequency power spectral density.
        }
      else{
        Fy1[j-bpsize/2]=0.0;
        Fy2[j-bpsize/2]=0.0;
        }
      }// Save frequency power spectral density data files.
    fwrite(Fy1,sizeof(double),bpsize,dfpSPORT1F); // Save initial input
pulse spectral profile of STG-NOLM on y-axis (fast axis) at input port.
    fwrite(Fy2,sizeof(double),bpsize,dfpSPORT2F); // Save initial input
pulse spectral profile of STG-NOLM on y-axis (fast axis) at outputport.
    fclose(dfpSPORT1F);
    fclose(dfpSPORT2F);
    }
  free(ein1profile);
  free(ein2profile);
  free(eout1profile);
  free(eout2profile);
  free(sprofile);
  }


/**********************************************************************
*************************
 Name: nolmswitch()
Function: Implements an NOLM switch in an optical fiber.
Author: P.C. Chimfwembe
Date: 01/16/99
Modified: 01/16/99
**********************************************************************
************************
*/
void nolmswitch(double xcalpha,dcomplex *Ein1,dcomplex *Ein2,dcomplex
*Eout1,dcomplex *Eout2){
  unsigned int i,j; //Iteration variables.
  for(j=0;j<bpsize;j++){
    Eout1[j]=((sqrt(xcalpha)*Ein1[j])+(dcomplex(0.0,1.0)*sqrt(1.0-
xcalpha)*Ein2[j]));
    Eout2[j]=((dcomplex(0.0,1.0)*sqrt(1.0-
xcalpha)*Ein1[j])+(sqrt(xcalpha)*Ein2[j]));
    }
  }
```

```
/***********************************************************************
*************************
 Name: profiletshift()
Function: Generates a positive or negative time shift in pulse time
domain profile. End padded.
Author: P.C. Chimfwembe
Date: 02/16/99
Modified: 02/16/99
***********************************************************************
*************************
*/
void profiletshift(unsigned int type,double timeshift,double
cwindow,dcomplex *tprofile){
  unsigned int i,j; //Iteration variables.
  unsigned int dtimeshift=ceil(timeshift/(cwindow/((double)bpsize)));
// Discrete time shift required in profile.
  if(type){// Positive time shift of profile ie the profile is delayed.
    printf("Signal delayed by +%ld\n",dtimeshift);
    for(j=0;j<=bpsize-1-dtimeshift;j++)
      tprofile[bpsize-1-j]=tprofile[bpsize-1-dtimeshift-j]; // Delay
profile.
    for(j=0;j<=dtimeshift;j++)
      tprofile[j]=tprofile[dtimeshift]; // End pad.
    }
  else{// Negative time shift of profile ie the profile is sped up.
    printf("Signal sped up by -%ld\n",dtimeshift);
    for(j=dtimeshift;j<bpsize;j++)
      tprofile[j-dtimeshift]=tprofile[j]; // Delay profile.
    for(j=bpsize-1-dtimeshift;j<bpsize;j++)
      tprofile[j]=tprofile[bpsize-1-dtimeshift]; // End pad.
    }
  }

/*
***********************************************************************
*************
File: nlbfunc.cpp
   Author: P.C. Chimfwembe
   Created: 08/01/96
   Modified: 08/16/97

***********************************************************************
**********************
   Function:  Provides basic nonlinear functions
*/
# include <math.h>
# include <stdio.h>
# include <complex>
# include "wlglobal.h"
# include "wlength.h"
# include "nlbfunc.h"
# include "wgbfunc.h"
# include "admathf.h"



/* Nonlinear coefficient gamma */

double gamma(double wavelength){
  double radw=2.0*PI*C/wavelength;
  double u,w,wo;
  u=U(V(wavelength,a));
  w=W(u);
  wo=p1espotsize(u,w);
  //printf("wo=%.8le;V=%.8le;a=%.8le\n",wo,V(wavelength,a),a);
  return ((nri*radw)/(C*PI*pow(wo,2.0)));
  }

/* n-soliton peak power */

double nsolitonp(double wavelength,double sorder,double pwidth){
  return
(fabs(GVD(wavelength))*pow(sorder,2.0))/(gamma(wavelength)*pow(pwidth,2
.0));
  }
/* Peterman I effective spot size */

double p1espotsize(double u,double w){
  double J0U,J1U;
  J0U=bessj0(u);
  J1U=bessj1(u);
  return(a*sqrt(((J0U/(u*J1U))+(1.0/2.0)+(1.0/pow(w,2.0))-
(1.0/pow(u,2.0)))*2.0/3.0));// Peterman I effective spot size (beam
radius)
  }

/*********************************************************************
************
    File: wgbfunc.cpp
   Author: P.C. Chimfwembe
   Created: 08/01/96
   Modified: 09/08/96

**********************************************************************
**********
   Function:
   Routines below are for linear core and cladding refractive index in
an optical fiber.  The
   eigenvalues U,W,b etc., are obtained via relatively accurate
approximate explicit empirical
   expressions, are solutions of the linear step-index optical fiber
eigenvalue equation.  For
   normalized frequency >=1.0.
**********************************************************************
************/
```

```
# include <stdio.h>
# include <math.h>
# include "wlglobal.h"
# include "wlength.h"
# include "wgbfunc.h"
# include "admathf.h"


/* V Number=Normalized frequency */
double V(double wavelength, double coreradius){
  double k=2*PI/wavelength;
  double numaper=sqrt(pow(nsellmeier(wavelength,core,corewl),2.0)
/* Numerical aperture */
                                     -
pow(nsellmeier(wavelength,cladding,claddingwl),2.0));
  return (coreradius*k*numaper);
  }

/* Single mode optical fiber step-index cladding eignvalue W
approximation
   by Rudolph and Neumann (N.B.:eigenvalue of core is U). Valid for
   2.5>V>1.5 with an error of 0.1% in this range. Error in the
determination of
   (bV)' and V(bV)" using Wrn() is very large and unacceptable. It can
ONLY be
   ACCURATELY used in the determination of W, b, and Beta-z, the
propagation
   constant.*/
double Wrn(double V){
   return (1.1428*V-0.9960);
   }

/* Cladding eigenvalue W. Based on approximate eigenvalue equation type
I (See
   D. Gloge). For LP01.*/
double W(double v){
  return(sqrt(pow(v,2.0)-pow(U(v),2.0)));
  }

/* Core eigenvalue U. Based on Rudolph and Neumann approximation.*/
double Urn(double v){
  return (sqrt(pow(v,2.0)-pow(Wrn(v),2.0)));
  }

/* Core eigenvalue U. Based on approximate eigenvalue equation type I
(See
   D. Gloge). */
double U(double v){
  double epsilon=1.0e-11,delta=1.0e-13;
  double u,ub,error,errorme,errorpe;
  double lower=0.0,upper=v;
  if((1.5<=v)&&(v<=2.5))
```

```
     u=Urn(v); /* Estimate u using Rudolph and Neumann approximation.
*/
  else
    u=v/2.0; /* Arbitrary estimate of u */
  error=(bessk0(sqrt(pow(v,2.0)-pow(u,2.0)))*bessj1(u)*u)
           -(bessk1(sqrt(pow(v,2.0)-
pow(u,2.0)))*bessj0(u)*sqrt(pow(v,2.0)-pow(u,2.0)));
  while(fabs(error)>epsilon){
        ub=u;u-=delta;
        errorme=(bessk0(sqrt(pow(v,2.0)-pow(u,2.0)))*bessj1(u)*u)
                       -(bessk1(sqrt(pow(v,2.0)-
pow(u,2.0)))*bessj0(u)*sqrt(pow(v,2.0)-pow(u,2.0)));
        u=ub;u+=delta;
        errorpe=(bessk0(sqrt(pow(v,2.0)-pow(u,2.0)))*bessj1(u)*u)
                      -(bessk1(sqrt(pow(v,2.0)-
pow(u,2.0)))*bessj0(u)*sqrt(pow(v,2.0)-pow(u,2.0)));
        u=ub;
      if(fabs(errorpe)<fabs(errorme)){
        if(fabs(errorpe)<fabs(error)){
          lower=u;u=(lower+upper)/2.0;
          }
        else
          delta/=2.0;
        }
      else if(fabs(errorme)<fabs(errorpe)){
        if(fabs(errorme)<fabs(error)){
          upper=u;u=(upper+lower)/2.0;
          }
        else
          delta/=2.0;
        }
      else
        delta/=2.0;
      error=(bessk0(sqrt(pow(v,2.0)-pow(u,2.0)))*bessj1(u)*u)
                -(bessk1(sqrt(pow(v,2.0)-
pow(u,2.0)))*bessj0(u)*sqrt(pow(v,2.0)-pow(u,2.0)));
    }
  return u;
  }


/* Normalized propagation constant b=1-(U^2/V^2)=(W/V)^2 approximation.
   Since V^2=U^2+W^2. U^2=V^2-W^2. Based on Rudolph and Neumann
approximation.*/
double brn(double v){
  return (pow(Wrn(v)/v,2.0));
   }
/* Normalized propagation constant based on approximate eigenvalue
equation type I (See
   D. Gloge). For LP01.*/
double b(double v){
  return (pow(W(v)/v,2.0));
   }
```

```
/* Mode Propagation constant (Total Beta) */
double Beta_z(double wavelength, double b){
  double k=2*PI/wavelength;
  double n1=nsellmeier(wavelength,core,corewl);
  double n2=nsellmeier(wavelength,cladding,claddingwl);
  return (k*sqrt(pow(n2,2.0)+b*(pow(n1,2.0)-pow(n2,2.0))));
  }

/* Effective refractive index */
double eri(double wavelength, double b){
  double n1=nsellmeier(wavelength,core,corewl);
  double n2=nsellmeier(wavelength,cladding,claddingwl);
  return sqrt(pow(n2,2.0)+(b*(pow(n1,2.0)-pow(n2,2.0))));
  }

/* Cut off wavelength for single mode fibers for operation in the range
1.2e-6 to
   2.0e-6 m for a specified core, cladding and core radius. */
double cow(double coreradius){
  double epsilon=1.0e-9;
  double error=1.0;
  double swavelength=1.2e-6; /* Start wavelength for low loss window */
  double fwavelength=2.0e-6; /* Finish wavelength for low loss window
*/
  double wavelength=swavelength;
  while(fabs(error)>epsilon){
    error=Vc-V(wavelength,coreradius);
    if(error<0.0){
      swavelength=wavelength;
      wavelength=(swavelength+fwavelength)/2.0;
      }
    else if(error>0.0){
      fwavelength=wavelength;
      wavelength=(fwavelength+swavelength)/2.0;
      }
     }
  return wavelength;
  }

/* Kappa 0 */
double kappa0(double w){
  double x=pow(bessk0(w)/bessk1(w),2.0);
  return (x);
  }


/* Power confinement factor (GAMMA) for single mode fiber fundamental
mode HE11.
   Based on approximate eigenvalue equation type I (See D. Gloge). For
LP01 or HE11. */
double PCFactor(double v){
  double w=W(v);
  double u=U(v);
  return (1-(pow(u,2.0)/pow(v,2.0))*(1-kappa0(w)));
  }


/* vD^2(bv) with respect to v dispersion term for HE11 or LP01 */
double VD2BV(double v){
  double w=W(v);
  double u=U(v);
  double temp1=2*pow(u,2.0)*kappa0(w)/pow(v*w,2.0);
  double temp2=3*pow(w,2.0)-2*kappa0(w)*(pow(w,2.0)-pow(u,2.0));
  double temp3=w*(pow(w,2.0)+pow(u,2.0)*kappa0(w))*(kappa0(w)-1);
  double temp4=2*bessk1(w)/bessk0(w);
  return (temp1*(temp2+temp3*temp4));
  }

/* D(bv) with respect to v dispersion term for HE11 or LP01 */
double DBV(double v){
  double w=W(v);
  double u=U(v);
  return (1.0-pow(u/v,2.0)*(1.0-2*kappa0(w)));
  }

/* Composite Material Dispersion. Based on approximate
   eigenvalue equation type I (See D. Gloge). For LP01 or HE11.*/
double cmd(double wavelength){
  double
n=eri(wavelength,b(V(wavelength,a))),pcf=PCFactor(V(wavelength,a));
  return((-wavelength/(C*n))*((nsellmeier(wavelength,core,corewl)

*pcf*D2lbdnsellmeier(wavelength,core,corewl))
                           +(nsellmeier(wavelength,cladding,claddingwl)
                           *(1-
pcf)*D2lbdnsellmeier(wavelength,cladding,claddingwl))));
  }

/* Waveguide Dispersion. Based on approximate
   eigenvalue equation type I (See D. Gloge). For LP01 or HE11.*/
double wgd(double wavelength){
  double m1=sgroupnwl(wavelength,core,corewl);
  double delta=DELTA(wavelength);
  double n=eri(wavelength,b(V(wavelength,a)));
  double v=V(wavelength,a);
  return (((-pow(m1,2.0)*delta)/(C*wavelength*n))*VD2BV(v));
  }

/* Composite Profile Dispersion */
double cpd(double wavelength){
  double m1=sgroupnwl(wavelength,core,corewl);
  double D1delta=D1DELTA(wavelength);
  double delta=DELTA(wavelength);
  double  n1=nsellmeier(wavelength,core,corewl);
  double n=eri(wavelength,b(V(wavelength,a)));
  double temp=0.0,v=V(wavelength,a);
  temp=(pow(n1,2.0)*D1delta/(C*n))*((wavelength*D1delta/(4*delta))-
(m1/n1));
  return  (-temp*((2*(PCFactor(v)-b(v)))+VD2BV(v)));
  }

 /* Dispersion Cross Products as by J.M. Adams */
 double dcp(double wavelength){
  double m1=sgroupnwl(wavelength,core,corewl);
  double m2=sgroupnwl(wavelength,cladding,claddingwl);
  double delta=DELTA(wavelength);
  double  n1=nsellmeier(wavelength,core,corewl);
  double  n2=nsellmeier(wavelength,cladding,claddingwl);
  double n=eri(wavelength,b(V(wavelength,a)));
  double v=V(wavelength,a);
  double nm,m;
  nm=n2*m2+(0.5*(b(v)+DBV(v))*(n1*m1-n2*m2));
  m=nm/n;
  return((-1.0/(wavelength*C*n))*(pow(m1,2.0)*2.0*delta*(PCFactor(v)-
b(v))-pow(m,2.0)
      +(pow(m1,2.0)*PCFactor(v))+(pow(m2,2.0)*(1.0-PCFactor(v)))));
  }

/* Effective Total Dispersion, D, coefficient as define by D. Gloge. */
double D(double wavelength){
  double CMD=cmd(wavelength);
  double WD=wgd(wavelength);
  double CPD=cpd(wavelength);
  double R=dcp(wavelength);
  return  (CMD+WD+CPD+R);
  }
 /* Effective Group Velocity Dispersion GVD (Beta2).  Includes
composite material dispersion
     waveguide dispersion , composite profile dispersion and cross-
product terms.*/
 double GVD(double wavelength){
   double DP= D(wavelength);
   return  (-pow(wavelength,2.0)*DP/(2*PI*C));
   }

 /* Effective group index. */
 double egi(double wavelength){
  double m1=sgroupnwl(wavelength,core,corewl);
  double m2=sgroupnwl(wavelength,cladding,claddingwl);
  double  n1=nsellmeier(wavelength,core,corewl);
  double  n2=nsellmeier(wavelength,cladding,claddingwl);
  double n=eri(wavelength,b(V(wavelength,a)));
  double v=V(wavelength,a);
  double nm;
  nm=n2*m2+(0.5*(b(v)+DBV(v))*(n1*m1-n2*m2));
  return (nm/n);
   }
```

```
/* Effective group velocity. */
double egv(double wavelength){
  return (C/egi(wavelength));
  }

/* Effective Beta1 */
double ebeta1(double wavelength){
  return (1.0/egv(wavelength));
  }

/*
***********************************************************************
**********************
   File: admathf.cpp
   Author(s): Press W.H., Flannery B.P., Teukolsky S.A. and Vetterling
W.T.
   Modified: P.C. Chimfwembe
   Created: 08/04/96
   Modified: 08/22/96


***********************************************************************
*************************
Function
Provide advanced mathematica function formulae.  Ordinary Bessel and
Modified Bessel function (of
integer order) from Numerical Recipes:  The Art of Scientific Computing
- Press W.H.,
Flannery B.P., Teukolsky S.A. and Vetterling W.T.. Modified by P.C.
Chimfwembe to enhance
precision=-log(max(error)) to about 23.22 giving a maximum error of
6.025595861x10^-24, using
approximating polynomial from Hart J.F., 1968, in Computer
Approximations, New York, Wiley.

Unlisted Source Code
See Numerical Recipes: The Art of Scientific Computing - Press W.H.,
Flannery B.P., Teukolsky S.A. and Vetterling W.T. and Hart J.F., 1968,
in Computer Approximations, New York, Wiley
***********************************************************************
***************/
# include <math.h>
# include <stdio.h>
# include <alloc.h>
# include <process.h>
# include <values.h>
# include "wlglobal.h"
# include "admathf.h"


# define pi 3.14159265358979323846 /* Constant pi */
```

```
/* Ordinary Bessel functions */
/* Bessel J0 */
double bessj0(double x){…}

/* Bessel Y0 */
double bessy0(double x){…}

/* Sign function */
double sign(double x){
  if(x>=0.0)
    return 1.0;
   else
    return -1.0;
   }

/* Bessel J1 */
double bessj1(double x){…}

/* Bessel Y1 */
double bessy1(double x){…}

/* Bess JN */
double bessj(int n, double x){…}

/* Bessel YN */
double bessy(int n, double x){…}

/* Modified Bessel functions */
/* Bessel I0 */
double bessi0(double x){…}

/* Bessel K0 */
double bessk0(double x){…}

/* Bessel I1 */
double bessi1(double x){…}

/* Bessel K1 */
double bessk1(double x){…}

/* Bessel IN */
double bessi(int n,double x){…}

/* Bessel KN */
double bessk(int n, double x){…}

/***********************************************************************
*************************
Name: L3pDt()
Function: Calculate the numerical first derivative of a boundary-zero
padded array using
        a 3-point Lagrangian interpolation formulae.
Author: P.C. Chimfwembe
```

```
Date: 06/06/97
Modified: 04/14/98
**********************************************************************
*************************
*/
void L3pDt(dcomplex *bprofile,unsigned int size,double
pulse_window,double pulse_HW){
  double Ts=pulse_window/size;
  double ts=Ts/pulse_HW;// Sample time normalization
  unsigned int i,j;
  dcomplex bminprofile;
  bminprofile=bprofile[bpsize-1];
  dcomplex scratch_pad[3]={bminprofile,bminprofile,bminprofile};
  dcomplex
df[3]={dcomplex(0.0,0.0),dcomplex(0.0,0.0),dcomplex(0.0,0.0)}; //3
constant end padding.
  for(i=0;i<size;i+=3){
    if(i+3<=size-1){
      df[0]=((-3.0*bprofile[i])+(4.0*bprofile[i+1])-
bprofile[i+2])/(2.0*ts);
      df[1]=(-bprofile[i]+bprofile[i+2])/(2.0*ts);
      df[2]=(bprofile[i]-
(4.0*bprofile[i+1])+(3.0*bprofile[i+2]))/(2.0*ts);
      bprofile[i]=df[0];bprofile[i+1]=df[1];bprofile[i+2]=df[2];
      }
    else{
      for(j=0;j<size-i;j++)// Copy bprofile array with remainder zero
buffered.
        scratch_pad[j]=bprofile[i+j];
      j=0;// Reset scratch pad indexing variable.
      df[0]=((-3.0*scratch_pad[i])+(4.0*scratch_pad[i+1])-
scratch_pad[i+2])/(2.0*ts);
      df[1]=(-scratch_pad[i]+scratch_pad[i+2])/(2.0*ts);
      df[2]=(scratch_pad[i]-
(4.0*scratch_pad[i+1])+(3.0*scratch_pad[i+2]))/(2.0*ts);
      for(j=0;j<size-i;j++)
           bprofile[i+j]=df[j];
      }
    }
  }

/*********************************************************************
*************************
Name: L5pDt()
Function: Calculate the numerical first derivative of a boundary-zero
padded array using
        a 5-point Lagrangian interpolation formulae.
Author: P.C. Chimfwembe
Date: 06/06/97
Modified: 04/14/98
**********************************************************************
*************************
*/
```

```
void L5pDt(dcomplex *bprofile,unsigned int size,double
pulse_window,double pulse_HW){
  double Ts=pulse_window/size;
  double ts=Ts/pulse_HW;// Sample time normalization
  unsigned int i,j;
  dcomplex bminprofile;
  bminprofile=bprofile[bpsize-1];
  dcomplex
scratch_pad[5]={bminprofile,bminprofile,bminprofile,bminprofile,bminpro
file}; //5 constant end padding.
  dcomplex
df[5]={dcomplex(0.0,0.0),dcomplex(0.0,0.0),dcomplex(0.0,0.0),dcomplex(0
.0,0.0)
                 ,dcomplex(0.0,0.0)};
  for(i=0;i<size;i+=5){
    if(i+5<=size-1){
      df[0]=((-25.0*bprofile[i])+(48.0*bprofile[i+1])-
(36.0*bprofile[i+2])+(16.0*bprofile[i+3])
            -(3.0*bprofile[i+4]))/(12.0*ts);
      df[1]=((-3.0*bprofile[i])-
(10.0*bprofile[i+1])+(18.0*bprofile[i+2])-(6.0*bprofile[i+3])
            +bprofile[i+4])/(12.0*ts);
      df[2]=(bprofile[i]-(8.0*bprofile[i+1])+(8.0*bprofile[i+3])-
bprofile[i+4])/(12.0*ts);
      df[3]=(-bprofile[i]+(6.0*bprofile[i+1])-
(18.0*bprofile[i+2])+(10.0*bprofile[i+3])
            +(3.0*bprofile[i+4]))/(12.0*ts);
      df[4]=((3.0*bprofile[i])-
(16.0*bprofile[i+1])+(36.0*bprofile[i+2])-(48.0*bprofile[i+3])
            +(25.0*bprofile[i+4]))/(12.0*ts);
      bprofile[i]=df[0];bprofile[i+1]=df[1];bprofile[i+2]=df[2];
      bprofile[i+3]=df[3];bprofile[i+4]=df[4];
      }
    else{
      for(j=0;j<size-i;j++)// Copy bprofile array with remainder zero
buffered.
        scratch_pad[j]=bprofile[i+j];
      j=0;// Reset scratch pad indexing variable.
      df[0]=((-25.0*scratch_pad[j])+(48.0*scratch_pad[j+1])-
(36.0*scratch_pad[j+2])+(16.0*scratch_pad[j+3])
            -(3.0*scratch_pad[j+4]))/(12.0*ts);
      df[1]=((-3.0*scratch_pad[j])-
(10.0*scratch_pad[j+1])+(18.0*scratch_pad[j+2])-(6.0*scratch_pad[j+3])
            +scratch_pad[j+4])/(12.0*ts);
      df[2]=(scratch_pad[j]-
(8.0*scratch_pad[j+1])+(8.0*scratch_pad[j+3])-
scratch_pad[j+4])/(12.0*ts);
      df[3]=(-scratch_pad[j]+(6.0*scratch_pad[j+1])-
(18.0*scratch_pad[j+2])+(10.0*scratch_pad[j+3])
            +(3.0*scratch_pad[j+4]))/(12.0*ts);
      df[4]=((3.0*scratch_pad[j])-
(16.0*scratch_pad[j+1])+(36.0*scratch_pad[j+2])-(48.0*scratch_pad[j+3])
            +(25.0*scratch_pad[j+4]))/(12.0*ts);
      for(j=0;j<size-i;j++)
           bprofile[i+j]=df[j];
      }
    }
  }

/***********************************************************************
*************************
Name: TrapIz()
Function: Calculate the numerical integration of an array using the
trapezoidal rule.
Author: P.C. Chimfwembe
Date: 06/07/97
Modified: 06/08/97
***********************************************************************
************************
*/
void TrapIz(dcomplex *bprofile1,dcomplex *bprofile2,unsigned int
size,double zstep_size){
  unsigned int i;
  for(i=0;i<size;i++)
    bprofile1[i]=(zstep_size/2.0)*(bprofile1[i]+bprofile2[i]);
  }

/***********************************************************************
*************************
Name: sgn()
Function: Determines the mathematical sign of argument
Author: P.C. Chimfwembe
Date: 06/26/97
Modified: 06/26/97
***********************************************************************
************************
*/
double sgn(double number){
  if(fabs(number)==number)
    return(1.0);
  else
    return(-1.0);
  }

/***********************************************************************
**************
File: wlglobal.h
***********************************************************************
*************/

# include <complex>

/* General Constants */
# define C  2.997925e8 /* Speed of light */
# define PI 3.14159265358979323846
//    M_PI 3.14159265358979323846
```

```
# define XAXIS         1
# define YAXIS         2
# define POSITIVE 1
# define NEGATIVE 0

/* Fiber Constants */
# define nri      3.2e-20 /* Nonlinear refractive index (m^2/W).Agrawal
p.40 */
# define a  (4.54e-6/2.0) /* @ Vc=2.405 with cut-off
wavelength=1.48090931e-6 m (shortest possible wavelength mode). Thus
1.55e-6 m is well coupled into the fiber.*/
# define FXSA     (PI*pow(a,2.0)) /* Fiber X-Sectional Area (m^2)  */
# define Vc 2.405 /* Step-index single mode fibers cut-off normalized
frequency (maximum normalized frequency possible) for linear refractive
index operation */
# define dn (1.0*1.1103e-4) // (Moores et. al used dn=5.4x10^-4 fiber)
x1 STG and SDG and x2 for STG/SDG-NOLM Modal birefringence=|nx-ny| with
ny (y fast axis)<nx(x slow axis);

/* Pulse Constants */

# define ppwidth  50.0e-15 /* Pump pulse width  (m) */
# define pwlength 1.55e-6 /* Pump wavelength (m) */
# define spwidth  50.0e-15 /* Signal pulse width */
# define swlength  1.55e-6 /* Signal wavelength 1.55e-6 m.  Zero Beta2
or  D at 1.4035e-6 m for a=4.54e-6m, core:13.56m/o GeO2,86.5m/o SiO2,
cladding:SiO2; Dipersion Shifted (DS) fiber.*/
                           //@ 1.55e-6 m Beta2=-10.1596441 ps^2/km
/* Step Index Single Mode Fiber Material Variables */

extern double
SiO2[],SiO2wl[],GeSiO135865[],GeSiO135865wl[],*core,*corewl,*cladding,*
claddingwl;
extern double Px,Py,EiPy;// Peak power for x and y axes (W).
extern double xintensity,yintensity;// Peak intensity for x and y axes
(W/m^2).


// Beam profile characteristics
# define bpsize  16384
/*1024(1k),2048(2k),4096(4k),16384(16k),32768(32k),262144(128k) large
array. Under DOS <=2048 (Else memory wraparoung). Under Win32 - RAM
size dependent. */
# define alpha   0.00          /* Fiber attenuation constant (per
meter) */
# define X3RT      3.0e-15         /* Fiber Chi-3 nonlinear response
time in seconds. */


// STG/SDG NOLM characteristics
# define  XCALPHA                                  0.5
// 50:50 cross-coupler. Was at 0.5
```

```
# define  STGNOLMANGLE                           ((45.0/180.0)*PI)
// Splitting soliton amplitude angle for STG-NOLM.
# define  SDGNOLMANGLE                           ((60.0/180.0)*PI)
# define     EiNySTGNOLM                               1.24
// (1.24)Input soliton control beam normalized soliton amplitude (y-
axis or fast axis).
# define     EiNySDGNOLM
      (1.24*sqrt(2.0)*sin(SDGNOLMANGLE))
# define  NySTGNOLM
      (1.24*sin(STGNOLMANGLE))  // Single/Co-propagation beams
normalized soliton amplitude (y-axis or fast axis).
# define  NxSTGNOLM
      (1.24*cos(STGNOLMANGLE))
# define  NySDGNOLM
      (1.24*sin(SDGNOLMANGLE))
# define     NxSDGNOLM                  (1.24*cos(SDGNOLMANGLE))

/*********************************************************************
***************
File: wlength.h
**********************************************************************
**************/

double nsellmeier(double wavelength, double B[3], double wlength[3]);
double D1nsellmeier(double wavelength, double B[3], double wlength[3]);
double D2nsellmeier(double wavelength, double B[3], double wlength[3]);
double sgroupnrf(double wavelength, double B[3], double wlength[3]);
double sgroupnwl(double wavelength, double B[3], double wlength[3]);
double sgvelocity(double wavelength, double B[3], double wlength[3]);
double sgvd(double wavelength, double B[3], double wlength[3]);
double SD(double wavelength, double B[3], double wlength[3]);
double D1lbdnsellmeier(double wavelength, double B[3], double
wlength[3]);
double D2lbdnsellmeier(double wavelength, double B[3], double
wlength[3]);
double DELTA(double wavelength);
double D1DELTA(double wavelength);
void fiberDdata(void);

/*********************************************************************
***************
File: bpm.h
**********************************************************************
**************/
# include "admathf.h"

void dfft(dcomplex *bprofile,unsigned int size,double type);
void gf1solitonp(dcomplex *bprofile,double pulse_FWHM,double
windowsize);
void propsolitonp(dcomplex *bprofile,double pulse_FWHM,double
zstep,double zlength,double windowsize);
void propsolitonps(dcomplex *bprofilex,dcomplex *bprofiley,double
pulse_FWHM,
                        double zstep, /*Fraction of soliton period  */
                        double zlength,double windowsize);
dcomplex dispersion_operator(unsigned int dfreq,double To,double
Ts,double Delta,double beta2,unsigned int size,double LD);
dcomplex dispersion_operatorxy(unsigned int axis,unsigned int
dfreq,double To,double Ts,
                              double Delta,double beta2,unsigned int
size,double LD);
void nolmswitch(double xcalpha,dcomplex *Ein1,dcomplex *Ein2,dcomplex
*Eout1,dcomplex *Eout2);
void nolmpropsolitonp(unsigned int axis,dcomplex *bprofile,double
pulse_FWHM,
              double zstep /*Fraction of soliton period  */
              ,double zlength,double windowsize);
void nolmpropsolitonps(dcomplex *bprofilex,dcomplex *bprofiley,double
pulse_FWHM,
                        double zstep, /*Fraction of soliton period  */
                        double zlength,double windowsize);
void stgpsinolm(dcomplex *inputprofile,double pulse_FWHM,
                  double zstep, /*Fraction of soliton period */
                  double zlength,double windowsize);
void stgnolm(dcomplex *inputprofile,double pulse_FWHM,
                  double zstep, /*Fraction of soliton period */
                  double zlength,double windowsize);
void nolmswitch(double xcalpha,dcomplex *Ein1,dcomplex *Ein2,dcomplex
*Eout1,dcomplex *Eout2);
void profiletshift(unsigned int type,double timeshift,double
cwindow,dcomplex *tprofile);

/*********************************************************************
***************
File: nlbfunc.h
*********************************************************************
**************/

double gamma(double wavelength);
double nsolitonp(double wavelength,double sorder,double pwidth);
double plespotsize(double u,double w);

/*********************************************************************
**************
File: wgbfunc.h
*********************************************************************
**************/

double V(double wavelength, double coreradius);
double Wrn(double v);
double W(double v);
double Urn(double v);
double U(double v);
double brn(double V);
double b(double v);
double Beta_z(double wavelength, double b);
double eri(double wavelength, double b);
double cow(double coreradius);
double kappa0(double w);
double PCFactor(double v);
double VD2BV(double v);
double DBV(double v);
double cmd(double wavelength);
double wgd(double wavelength);
double cpd(double wavelength);
double dcp(double wavelength);
double D(double wavelength);
double GVD(double wavelength);
double egi(double wavelength);
double egv(double wavelength);
double ebeta1(double wavelength);

/*********************************************************************
***************
File: admathf.h
*********************************************************************
**************/
# include <complex>
using namespace std;
typedef complex<double> dcomplex;

double bessj0(double x);
double bessy0(double x);
double sign(double x);
double bessj1(double x);
double bessy1(double x);
double bessj(int n, double x);
double bessy(int n, double x);

double bessi0(double x);
double bessk0(double x);
double bessi1(double x);
double bessk1(double x);
double bessi(int n, double x);
double bessk(int n, double x);

void L3pDt(dcomplex *bprofile,unsigned int size,double
pulse_window,double pulse_HW);
void L5pDt(dcomplex *bprofile,unsigned int size,double
pulse_window,double pulse_HW);
void TrapIz(dcomplex *bprofile1,dcomplex *bprofile2,unsigned int
size,double zstep_size);
double sgn(double number);
```

VITA 2

Patrick Chilufya Chimfwembe

Candidate for the Degree of

Doctor of Philosophy

Thesis: CROSS KERR & RAMAN EFFECTS ON THE SWITCHING EFFICIENCY OF SOLITON OPTICAL FIBER GATES

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Lusaka, Zambia, On May 9, 1960, the son of Elijah David Chimfwembe and Berita Mulenga Chimfwembe.

Education: Graduated from Canisius Secondary School, Chisekesi, Zambia in August 1978. Received Bachelor of Science (Hons.) in Mining Engineering from University College Cardiff, Wales, UK in July 1983, and Bachelor of Engineering in Electronics and Telecommunications Engineering from University of Zambia, Lusaka, Zambia in November 1988. Obtained Master of Science in Computer Engineering from Boston University, Massachusetts, USA in September 1990. Completed the requirements for the Doctor of Philosophy degree with a major in Electrical Engineering at Oklahoma State University in May, 1999.

Experience: Senior Mining Engineer at Konkola Mine, Chililabombwe, Zambia from 1983-1985. Networks and Applications Engineer at Computer Technologies, Lusaka, Zambia in 1989. Lecturer and Head of Department in Electrical and Electronics Engineering Department, University of Zambia, Lusaka, Zambia from 1990 to 1995.

Professional Memberships: Member and Registered Engineer of the Engineering Institute of Zambia. Member of the Institute of Electrical and Electronics Engineers, USA.