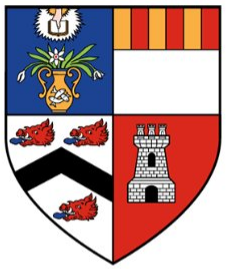


UNIVERSITY OF
ABERDEEN

Optical pulse compression using dispersion decreasing fibres

By Jan Polivka

Master of Engineering Individual Project Thesis

Student I.D. : 51553399

Email : j.polivka.15@aberdeen.ac.uk

Github repository: jan-polivka/optical-pulse-compression

Supervisor : Dr. Nakkeeran Kaliyaperumal

22 April 2019

Abstract

Due to increasing demand for faster communication rates, there is a need to develop better techniques for data transmission. While there are different ways that would satisfy such a need, this paper discusses dispersion decreasing fibres. This project sets out to study the basics of nonlinear fibre optics with focus on group-velocity dispersion and self-phase modulation. A model based on the split-step Fourier method is developed for pulses wider than $5ps$, compression of pulses using linear and exponential profiles is investigated and then applied to a second-order soliton breather. Finally, a method on how to optimise dispersion decreasing fibre for real world use is described and analysed. In terms of results, it was determined that a Cooley-Tukey algorithm performed on a CPU is the fastest. A second-order soliton breather was chosen as the best option for compression and lastly, an adaptive midpoint rule was deemed to be the best solution to creating segmented fibres.

Acknowledgements

I wish to thank my supervisor Dr. Nakkeeran Kaliyaperumal for his guidance and supervision. I would also like to acknowledge the support and patience of my friends and family during this journey.

Contents

1 Introduction and Literature review

With the ever-increasing demand for more data, there is also the corresponding demand for faster transmission speed. Currently, practical rates of 10 to 40 Gbit/s have been achieved but there is a demand for higher rates, which can only be satisfied using pulses that have their lengths in picoseconds [1].

There are several ways to produce these pulses, such as dispersion decreasing fibres (DDF), fibre lasers and fibre amplifiers. Among them, fibre lasers suffer from limitations of maximum output power and both fibre lasers and amplifiers are complex devices that are not economically viable [2][3][4][5].

The most common mechanisms that dispersion decreasing fibres use are soliton pulse compression and adiabatic compression. Both of these have their own disadvantages; soliton pulse compression creates pedestal which lead to nonlinear interactions between neighbouring solitons, while adiabatic compression provides high quality compression with minimal pedestals, it is, however, difficult to maintain the adiabatic conditions [6][7][8]. One of the novel approaches is to use chirped optical solitons with constant and approximately exponentially decreasing dispersion [9].

This project sets out to study basics of nonlinear fibre optics, including the Split-Step Fourier Method and develop C code to apply it, to investigate optical pulse compression in dispersion decreasing fibres and its optimisation.

This report has three main parts. In section two, the basics of nonlinear fibre optics are outlined with focus a on group-velocity dispersion and self-phase modulation. In section three, the development of Split-step Fourier Methods and required subroutines are presented and finally, in section four, compression is investigated along with a method explaining how to optimise dispersion decreasing fibre for real world use.

2 Basics of Nonlinear Fibre Optics

The centre point of this essay is an electromagnetic pulse in an optical fibre. While the pulse is described in the next subsection, it is also useful to describe what an optical fibre is.

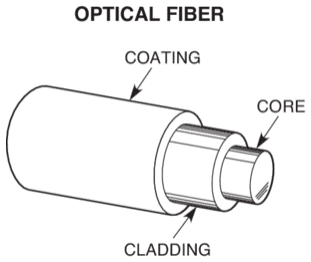


Figure 1: Diagram of a basic optical fibre. Taken from [10].

A simple fibre is made of three parts, coating, cladding and a core, through which the pulse propagates. There are two key parameters, the core cladding index difference,

$$\Delta = \frac{n_1 - n_2}{n_1}, \tag{2.0.1}$$

and a V parameter that is related to the number of modes supported by the fibre [11]

$$V = k_0 a (n_1^2 - n_2^2)^1/2, \tag{2.0.2}$$

with $k_0 = 2\pi/\lambda$, a is the core radius and λ is the wavelength.
There are also two important characteristics through which the medium influences the pulse. Firstly, the fibre losses which influence the transmitted power through the function

$$P_T = P_0 exp(-\alpha L), \tag{2.0.3}$$

where P_0 is the initial power, L is the fiber length and α is the attenuation constant, that is expressed in dB/km using the relation [11]

$$\alpha_{dB} = -\frac{10}{L} log \frac{P_T}{P_0} = 4.343\alpha. \tag{2.0.4}$$

The mechanisms behind the losses are material absorption and Rayleigh scattering. Both effects arise from material imperfections, with material absorption being due to impurities

and Rayleigh scattering due to density fluctuations in the fused silica. This causes the light to scatter in all directions.

The second influential characteristic is that of Chromatic dispersion. This property is very much dependant on the optical frequency ω and can be described as the result of an electromagnetic wave interacting with bound electrons in a dielectric medium. Fundamentally, the radiation is absorbed by the medium and the effect can be accounted for by the mode propagation constant β defined as [11]

$$\beta(\omega) \approx \beta_0 + \beta_1(\omega - \omega_0) + \frac{1}{2}\beta_2(\omega - \omega_0)^2\frac{1}{6} + \beta_3(\omega - \omega_0)^3 + ... \tag{2.0.5}$$

The most important parameter in this expansion is the β_2 which represents the dispersion of the group velocity and causes pulse broadening. This phenomenon is called group-velocity dispersion (GVD) and is discussed in later sections. This parameter is also exploited for the construction of dispersion decreasing fibres, where the β_2 term is tapered off either at a constant or exponential rate to achieve compression.

While Chromatic dispersion is caused by medium acting on the pulse, most fibre nonlinearities are caused by the anharmonic motion electrons when exposed to an applied field. This creates a nonlinear polarisation in the electric field governed by the relation

$$\boldsymbol{P} = \varepsilon_0(\chi^{(1)} \cdot \boldsymbol{E} + \chi^{(2)} : \boldsymbol{EE} + \chi^{(3)} \vdots \boldsymbol{EEE} + ...),$$

where ε_0 is the vacuum permittivity and χ is the susceptibility. The susceptibility $\chi^{(1)}$ is the linear term within the relation and its effect is included in the refractive index n and the attenuation α. The second order susceptibility is responsible for some nonlinearities, however, since SiO_2 is a symmetric molecule, the term vanishes. Finally, the third order susceptibility is responsible for most nonlinearities. These effects originate from nonlinear refraction, which is caused by the intensity dependence of the refractive index. This is directly tied to self-phase modulation, which is explored later.

2.1 Nonlinear Schrödinger Equation

In order to model the evolution of an optical pulse, it is important to first derive the equation which governs the behaviour. The starting points are the Maxwell's equations which are used to derive a wave equation and finally ending with a system that resembles the Nonlinear Schrödinger Equation (NLSE). While there are other ways to derive the NLSE [12], a simplified approach, that appears in [11], is shown as it widely used in other research.

$$\nabla \times \mathbf{E} = -\frac{\delta \mathbf{B}}{\delta t} \tag{2.1.1}$$

$$\nabla \times \mathbf{H} = \mathbf{J} + \frac{\delta \mathbf{D}}{\delta t} \tag{2.1.2}$$

$$\nabla \cdot \mathbf{D} = \rho_f \tag{2.1.3}$$

$$\nabla \cdot \mathbf{B} = 0 \tag{2.1.4}$$

The flux densities **D** and **B** are related to the electric and magnetic fields **E** and **H** as

$$\mathbf{D} = \varepsilon_0 \mathbf{E} + \mathbf{P} \tag{2.1.5}$$

$$\mathbf{B} = \mu_0 \mathbf{H} + \mathbf{M}. \tag{2.1.6}$$

By taking the curl of (2.1.1) and using (2.1.2), (2.1.5) and (2.1.6), the following is obtained

$$\nabla \times \nabla \times \mathbf{E} = -\frac{1}{c^2}\frac{\delta^2 \mathbf{E}}{\delta t^2} - \mu_0 \frac{\delta^2 \mathbf{P}}{\delta t^2}. \tag{2.1.7}$$

As explained in the previous section, the polarisation $\boldsymbol{P}$ can be split into linear and nonlinear parts

$$\mathbf{P}(\mathbf{r},t) = \mathbf{P}_L(\mathbf{r},t) + \mathbf{P}_{NL}(\mathbf{r},t). \tag{2.1.8}$$

Therefore the equation 2.1.7 can be rewritten as

$$\nabla^2 \mathbf{E} - \frac{1}{c^2}\frac{\delta^2 \mathbf{E}}{\delta t^2} = \mu_0 \frac{\delta^2 \mathbf{P}_L}{\delta t^2} + \mu_0 \frac{\delta^2 \mathbf{P}_{NL}}{\delta t^2}. \tag{2.1.9}$$

Since it is much easier to work in frequency domain, the previous equation is transformed into

$$\nabla^2\tilde{E} + \varepsilon(\omega)k_0^2\tilde{E} = 0, \tag{2.1.10}$$

where $k_0 = \omega/c$ and ε is a dielectric constant of the medium, which is approximated as

$$\varepsilon \approx n^2 + 2n\Delta n, \tag{2.1.11}$$

where Δn is a small perturbation given by

$$\Delta n = n_2|E|^2 + \frac{i\bar{\alpha}}{ik_0}. \tag{2.1.12}$$

The solution to the wave equation in frequency domain is assumed to be

$$\tilde{E}(\mathbf{r}, \omega - \omega_0) = F(x,y)\tilde{A}(\omega - \omega_0)exp(i\beta_0 z), \tag{2.1.13}$$

where F is the modal distribution and A is a slowly varying function. Using separation of variables, an eigenvalue equation and an equation describing the evolution of the wave are obtained

$$\frac{\delta^2 F}{\delta x^2} + \frac{\delta^2 F}{\delta y^2} + [\varepsilon(\omega k_0^2 - \tilde{\beta}^2]F = 0, \tag{2.1.14}$$

$$2i\beta_0\frac{\delta\tilde{A}}{\delta z} + (\tilde{\beta}^2 - \beta_0^2)\tilde{A} = 0. \tag{2.1.15}$$

By defining

$$\tilde{n} = n + n_2|E|^2, \quad \tilde{\alpha} = \alpha + \alpha_2|E|^2, \quad \tilde{\beta}(\omega) = \beta(\omega) + \Delta\beta, \tag{2.1.16}$$

where

$$\Delta\beta = \frac{k_0 \int\int_{-\infty}^{\infty} \Delta n|F(x,y)|^2 dxdy}{\int\int_{-\infty}^{\infty} |F(x,y)|^2 dxdy}, \tag{2.1.17}$$

the formal solution of (2.1.9) is completed.

The object of interest is the slowly varying pulse envelope $A(z,t)$, therefore further investigation needs to be done. To do so, the equation (2.1.15) is rewritten as

$$\frac{\delta\tilde{A}}{\delta z} = i[\beta(\omega) + \Delta\beta - \beta_0]\tilde{A}. \tag{2.1.18}$$

While it possible to go directly back to time domain with the previous equation, the exact form of $\beta(\omega)$ is unknown and needs to be expanded using the Taylor series

$$\beta(\omega) \approx \beta_0 + \beta_1(\omega - \omega_0) + \frac{1}{2}\beta_2(\omega - \omega_0)^2 + \frac{1}{6}\beta_3(\omega - \omega_0)^3 + ... \tag{2.1.19}$$

Substituting (2.1.19) into (2.1.18), following inverse Fourier Transform is used

$$A(z,t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \tilde{A}(z, \omega - \omega_0) exp[-i(\omega - \omega_0)t] d\omega, \tag{2.1.20}$$

which is then, with the use of the eqs. (2.1.12) and (2.1.17), made in to the Nonlinear Schrödinger equation

$$\frac{\delta A}{\delta z} + \beta_1 \frac{\delta A}{\delta t} + \frac{i\beta_2}{2}\frac{\delta^2 A}{\delta t^2} + \frac{\alpha}{2}A = i\gamma|A|^2 A. \tag{2.1.21}$$

This equation is suitable for pulses with a width bigger than $T_0 > 5$, for which nonlinear effects of third order and higher can be ignored. Furthermore, using the transformation $\tau = t - \beta_1 z$, the previous equation is made in to

$$i\frac{\delta A}{\delta z} + \frac{i\alpha}{2}A - \frac{\beta_2}{2}\frac{\delta^2 A}{\delta T^2} + \gamma|A|^2 A = 0. \tag{2.1.22}$$

To make it easier to analytically investigate the eq.(2.1.22) in the following sections, it is useful to normalise the equation using $\tau = T/T_0$ and $A(z,T) = \sqrt{P_0} exp(-\alpha z/2) U(z,t)$ to obtain

$$i\frac{\delta U}{\delta z} = \frac{sgn(\beta_2)}{2L_D}\delta^2 U \delta\tau^2 - \frac{e^{-\alpha z}}{L_{NL}}|U|^2 U \tag{2.1.23}$$

and

$$L_D = \frac{T_0^2}{|\beta_2|}, \quad L_{NL} = \frac{1}{\gamma P_0}. \tag{2.1.24}$$

β_2 is the dispersion parameter in the unit of $ps^2 km^{-1}$, γ is the nonlinear term in the units of $W^{-1}km^{-1}$, T_0 is width of the pulse in picoseconds, L_D is called the dispersive length, L_{NL} is nonlinear length and L is the fibre length in kilometers. Using these definitions, it is possible to define the behaviour of the propagation through different lengths. When $L << L_{NL}$ but $L \sim L_D$, the pulse evolution is governed by GVD and the nonlinear effects are negligible. Furthermore, the attenutation losses α can be omitted if needed as they are negligible in this model. When $L << L_D$ but $L \sim L_{NL}$, the pulse evolution is governed by nonlinear effects and

GVD is negligible. Finally, when L is longer or comparable to both L_D and L_{NL}, the pulse is affected by both linear and nonlinear effects and the behaviour is very much different from the separate effects.

2.2 Group-Velocity Dispersion

As mentioned previously, the effect of group-velocity dispersion derives from the chromatic dispersion where the electromagnetic wave interacts with electrons within the fibre and the cladding.

$$\beta(\omega) \approx \beta_0 + \beta_1(\omega - \omega_0) + \frac{1}{2}\beta_2(\omega - \omega_0)^2 + \frac{1}{6}\beta_3(\omega - \omega_0)^3 + ... \tag{2.2.1}$$

Particularly, the β_2 coefficient within the propagation constant $\beta(\omega)$ is responsible for this behaviour, while β_1 is the actual group velocity and β_3, the third-order dispersion parameter, further distorts the optical pulse.

To demonstrate the effects of GVD alone, it is useful to have an analytical model. Starting from the normalised NLSE (2.1.23), the nonlinear term γ is set to zero to obtain [11]

$$i\frac{\delta U}{\delta z} = \frac{\beta_2}{2}\frac{\delta^2 U}{\delta T^2} \tag{2.2.2}$$

which can be solved using a Fourier transform

$$U(z,T) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \tilde{U}(z,\omega) exp(-i\omega T) d\omega, \tag{2.2.3}$$

which satisfies the ordinary differential equation

$$i\frac{\delta \tilde{U}}{\delta z} = -\frac{1}{2}\beta_2 \omega^2 \tilde{U} \tag{2.2.4}$$

with a solution

$$\tilde{U}(z,\omega) = \tilde{U}(0,\omega) exp(\frac{i}{2}\beta_2 \omega^2 z). \tag{2.2.5}$$

By substituting 3.2.4 into 3.2.2, general solution for 3.2.1 is obtained.

$$U(z,T) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \tilde{U}(0,\omega) exp(\frac{i}{2}\beta_2 \omega^2 z - i\omega T) d\omega \tag{2.2.6}$$

where U is obtained using this

$$\tilde{U}(0,\omega) = \int_{-\infty}^{\infty} U(0,T) exp(i\omega T) dT. \tag{2.2.7}$$

Considering a simple Gaussian pulse [marcuse 1980]

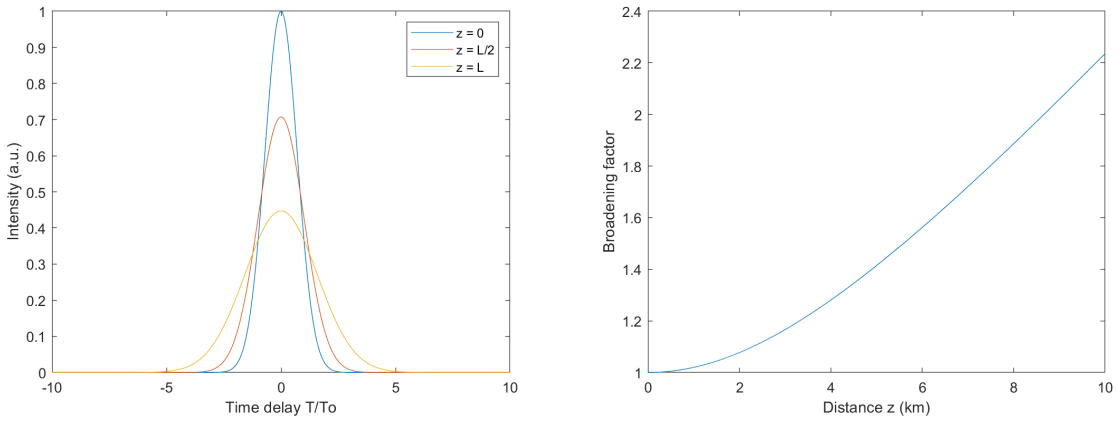
$$U(0,T) = P_0 exp(-\frac{T^2}{2T_0^2}) \tag{2.2.8}$$

following analytical model is obtained

$$U(z,T) = \frac{T_0}{(T_0^2 - i\beta_2 z)^{1/2}} exp(-\frac{T^2}{2T_0^2}) \tag{2.2.9}$$

$$T_1(z) = T_0[1 + (z/L_D)^2]^{1/2} \tag{2.2.10}$$

From the equation (2.2.7), it is apparent that GVD changes the phase of each spectral component based on the propagated distance, as well as the particular frequency. While this does not affect the frequency spectrum, it has an effect on the pulse shape. This is explained by (2.2.10), which shows the width of the pulse increasing with propagated distance.



(a) (b)

Figure 2: Simulation of a Gaussian pulse using the equation (2.2.9) with $\beta_2 = -20\ ps^2 km^{-1}$ and $L = 5\ km$. As explained by theory, the pulse gradually broadens and lowers in amplitude.

2.3 Self-Phase Modulation

Self-phase modulation (SPM) is a nonlinear effect that was discovered in 1967 and was explained by Shimizu as a phase modulation as a consequence of intensity dependent refractive index [13][14][15]. As mentioned previously, this effect occurs when the fibre length L is

comparable to the nonlinear length L_{NL}, with dispersion length L_D being much bigger than both.

As with GVD, SPM needs to be investigated separately and is done so by setting the GVD parameter, β_2 to zero. The normalised NLS equation (2.1.23) therefore becomes [11]

$$\frac{\delta U}{\delta z} = \frac{ie^{-\alpha z}}{L_{NL}}|U|^2 U \tag{2.3.1}$$

Substituting in $U = Vexp(i\phi_{NL})$ and equating the real and imaginary parts, we obtain

$$\frac{\delta V}{\delta z} = 0; \quad \frac{\delta \phi_{NL}}{\delta z} = \frac{e^{-\alpha z}}{L_{NL}}V^2 \tag{2.3.2}$$

The phase equation can be integrated analytically to obtain the solution

$$U(L,T) = U(0,T)exp[i\phi_{NL}(L,T)], \tag{2.3.3}$$

with the function ϕ_{NL} defined as

$$\phi_{NL}(L,T) = |U(0,T)|^2(L_{eff}/L_{NL}), \tag{2.3.4}$$

where

$$L_{eff} = [1 - exp(-\alpha L)]/\alpha. \tag{2.3.5}$$

The equation 2.3.3 shows that SPM adds a phase shift to the frequency based on intensity but does not change the shape of the pulse itself. As ϕ_{NL} is a function of distance L, the phase shift increases with distance. However, the spectral broadening is based on the time-dependence of ϕ_{NL} and therefore the instantaneous optical frequency changes across the pulse. This means that new frequency components are generated at all times along the length of the fibre.

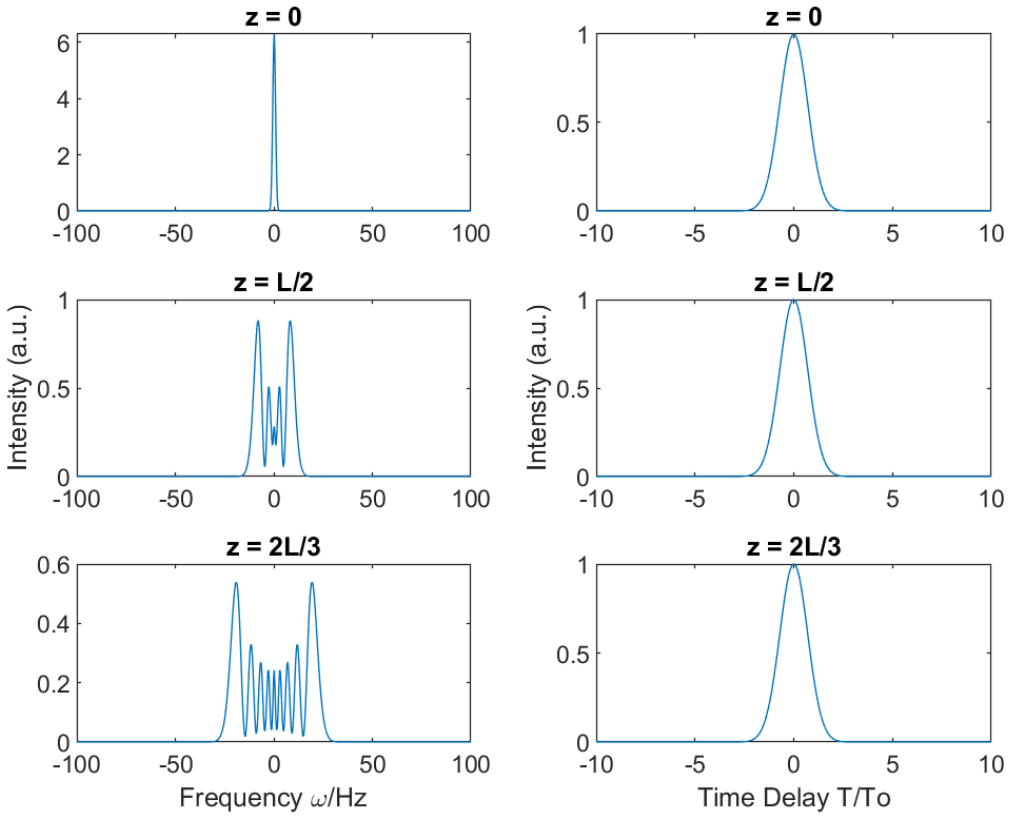


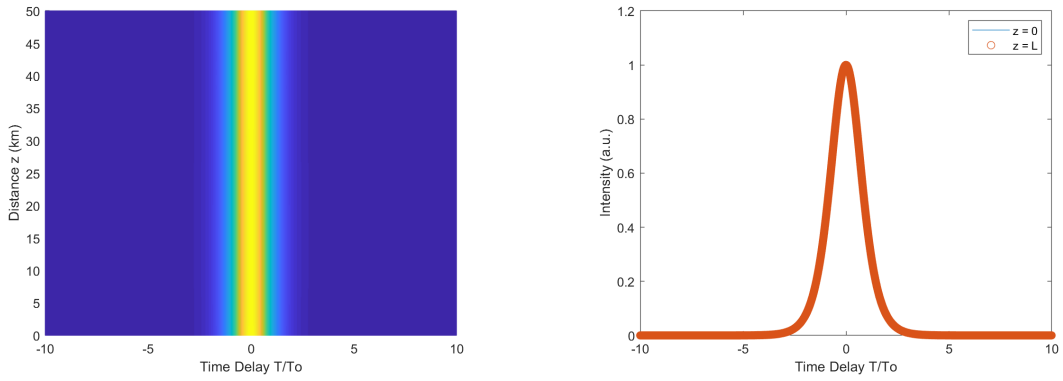
Figure 3: Simulation of a Gaussian pulse using the equation (2.3.3) with $\gamma = 2\ W^{-1}km^{-1}$ and $L = 5\ km$. What can be observed in the graph is that while the frequency spectrum wildly changes, the pulse shape is completely uneffected.

2.4 Solitons

Solitons were first observed back in 1844 propagating as a wave in a canal. They also have been observed in many other branches of physics and are the main tool of nonlinear fibre optics.

2.4.1 Soliton Behaviour

An optical soliton arises as an interplay between GVD and SPM, when the lengths L_D and L_{NL} are approximately equal. In this project, so called light solitons are studied, which require the GVD coefficient to be in the anomalous dispersion regime where $\beta_2 < 0$.



(a) (b)

Figure 4: Simulation of a sech pulse $U(0,T) = Nsech(\tau)$ [11] as a first order soliton. First figure presents behaviour over distance, the second presents the initial and final waveform, showing that they are the same.

As it turns out, there are more types of light solitons and they are classified by their order N which has the following relation

$$N^2 = \frac{L_D}{L_{NL}} = \frac{\gamma P_0 T_0^2}{|\beta_2|}. \tag{2.4.1}$$

With $N = 1$ describing the fundamental soliton as shown in the previous figure. The behaviour of higher order solitons is, however, more varied.

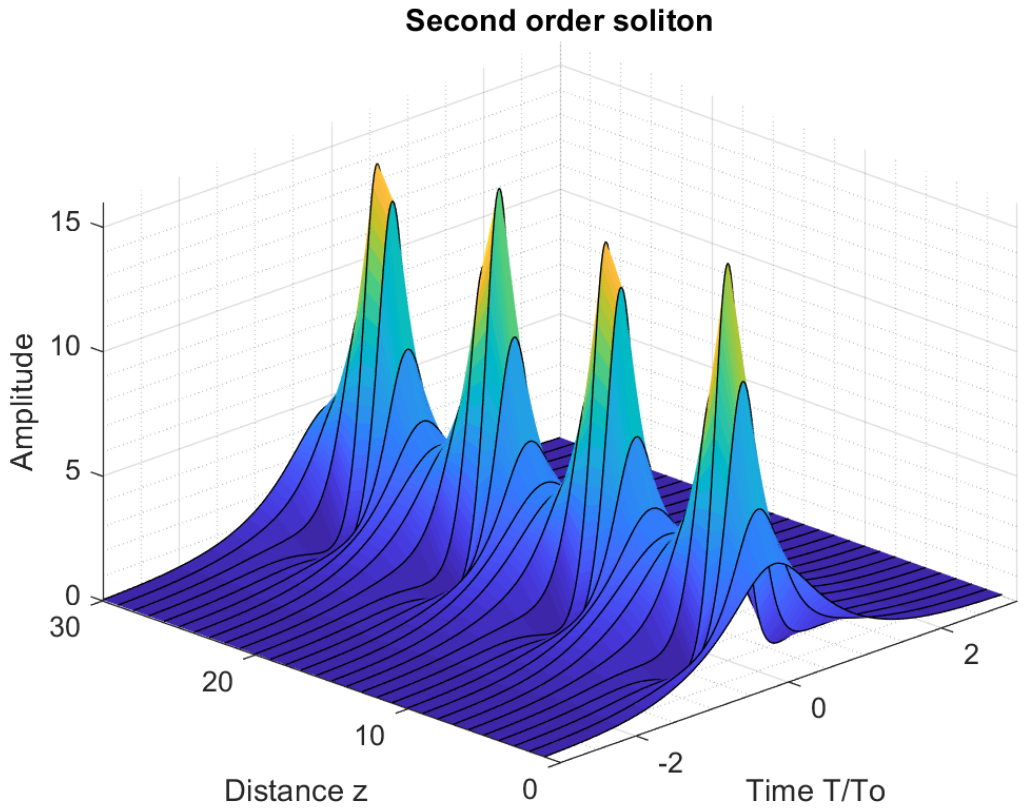


Figure 5: Second-order soliton with periodic peaks.

Alongside group-velocity dispersion and self-phase modulation, it is important to investigate the chirp property, which is defined as the linear increase of instantaneous frequency either from leading $(C > 0)$ or trailing $(C < 0)$ edge. This is referred to as positive or negative chirp respectively. This is also how GVD and SPM cooperate, that is, both create an opposite chirp to preserve the shape and spectrum when the order $N = 1$.

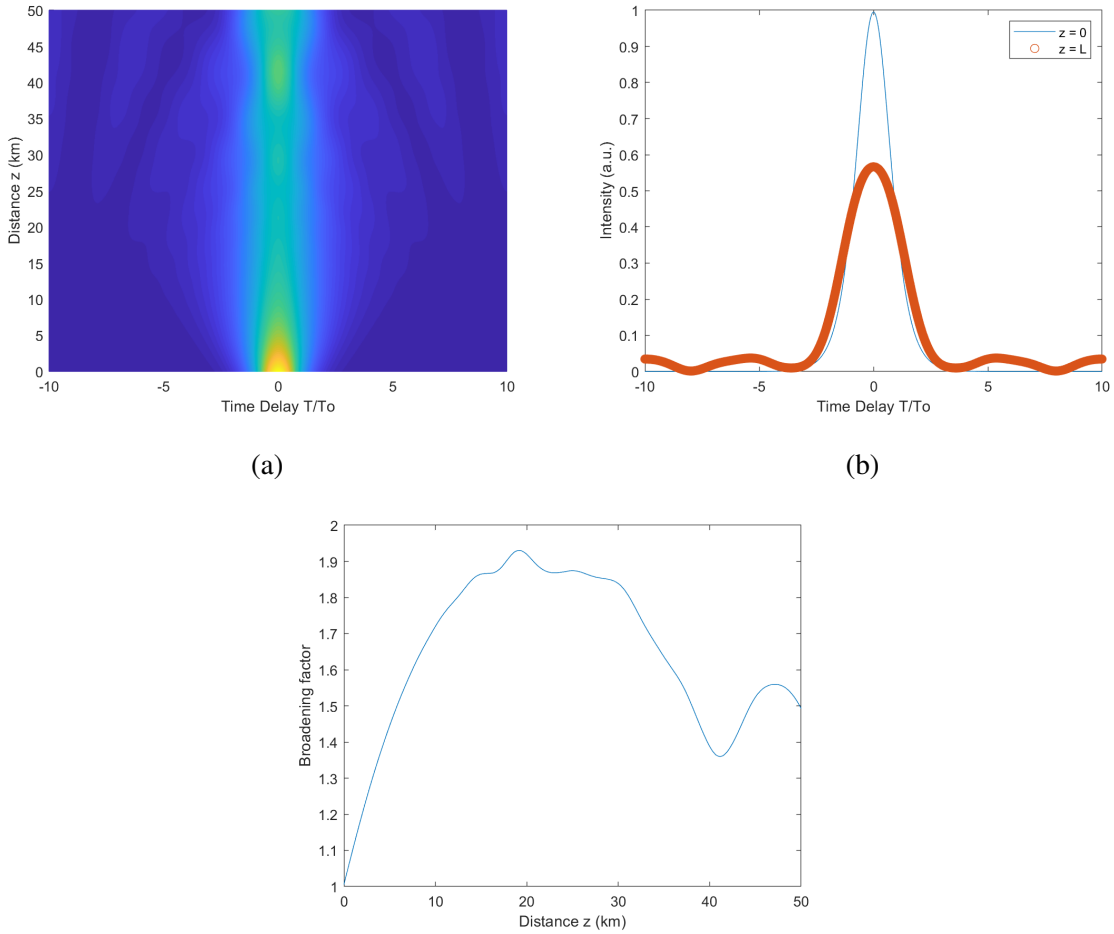


Figure 6: Soliton evolution for the case $N = 1$ with an initial chirp $C = 0.5$ The pulse initially broadens but quickly compresses again, without recovery, shedding energy as a pedestal.

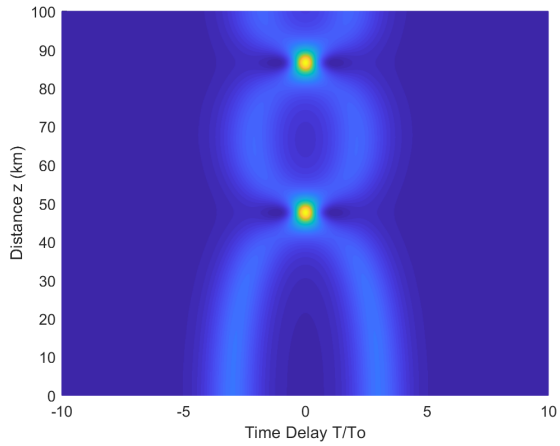
As shown in the Figure 6, solitons are resistant to some chirp, however, it is important to point out that with a large enough magnitude of chirp, $|C|$, the soliton is destroyed. Similar energy shed happens when input parameters are substantially different from their ideal values. However, this general resiliency is a key feature that, along with chirp, is still useful, as will be shown in later chapters.

2.4.2 Soliton Interaction

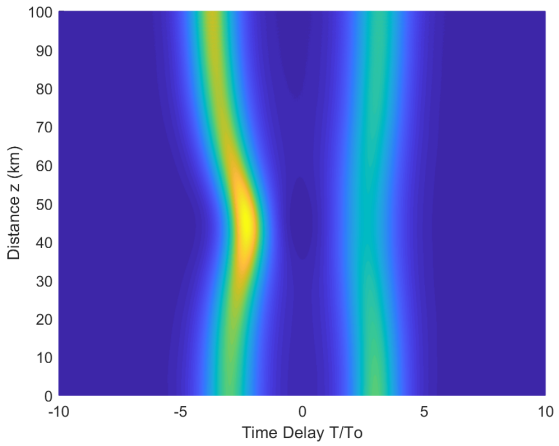
What is also vital to propagation of solitons is how they interact with each other in a pulse train. Time interval T_B between each pulse is directly tied to the bit rate $B = 1/T_B$. To show the interaction, the following initial waveform is used

$$u(0,\tau) = sech(\tau - q_0) + rsech[r(\tau + q_0)]e^{i\theta},$$

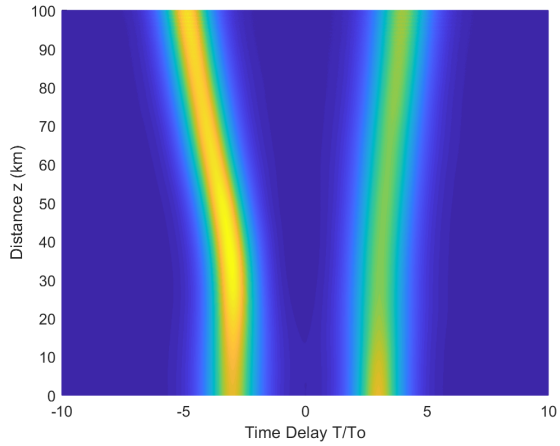
where q_0 is the initial separation, r is the relative amplitude and θ is the initial phase difference.



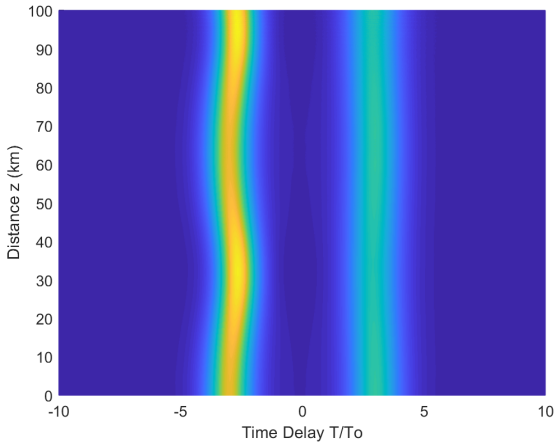
(a) Collision of two solitons, $r = 1, \theta = 0$



(b) Attraction of two solitons, $r = 1, \theta = \pi/4$



(c) Repulsion of two solitons, $r = 1, \theta = \pi/2$



(d) Coexistence of two solitons, $r = 1.2, \theta = 0$

Figure 7

It should be obvious that these behaviours are undesirable in communication systems as they can destroy the pulse train. To avoid this, certain precautions are required, mainly for the case of collision. The collision distance is calculated with the following formula

$$L_{col} = \frac{\pi}{2} L_D exp(q_0) \equiv z_0 exp(q_0).$$

When the separation coefficient q_0 is set high enough, the collision can be avoided alltogether as L_{col} will be much higher than the transmission distance, meaning that the solitons will never collide.

2.5 Split-Step Fourier Method

As the NLS equation (2.1.22) does not lend itself to an analytical solution in the general case, it is necessary to use a numerical analysis. The two main strategies to solving this problem are finite difference methods and pseudospectral methods, Split-step Fourier method (SSFM) is one of them. Of the two, the pseudospectral method prove to be faster than the finite difference methods and therefore they are the weapon of choice for this problem [16].

The SSFM works by partitioning the fibre into segments, where nonlinear (self-phase modulation) and linear (group-velocity dispersion) segments alternate. This operates based on the assumption that the length of the segments is short enough, such that the effects can act independently [17][11].

$$\frac{\delta A}{\delta z} = (\hat{D} + \hat{N})A \tag{2.5.1}$$

$$\hat{D} = -\frac{i\beta_2}{2}\frac{\delta^2}{\delta T^2} - \frac{\alpha}{2}; \quad \hat{N} = i\gamma|A|^2 \tag{2.5.2}$$

$$A(z+h,T) \approx exp[h(\hat{D}+\hat{N})]A(z,T) \tag{2.5.3}$$

[11]

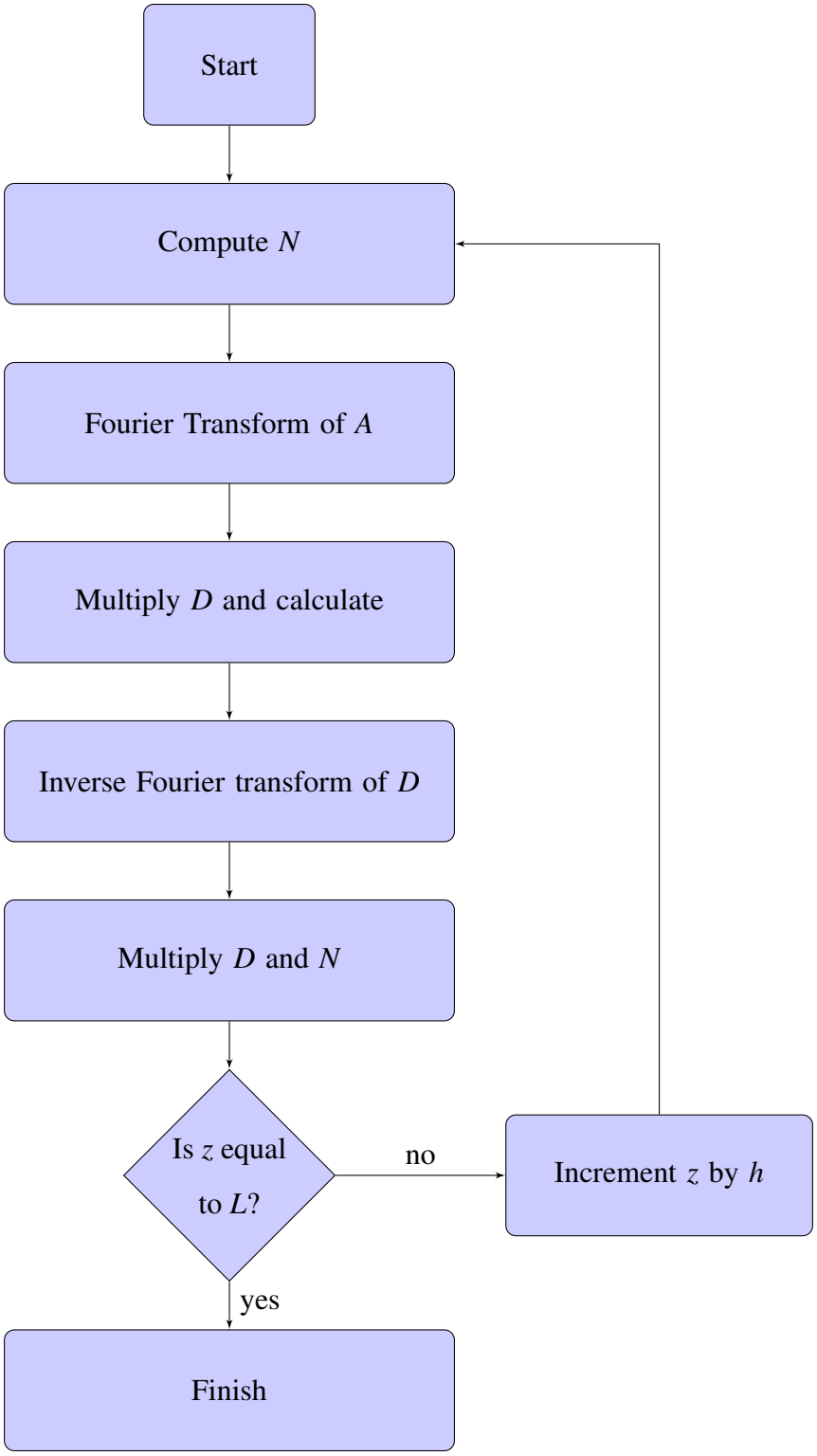


Figure 8: Split-Step Fourier Method flowchart

3 Simulation of Propagation in C

The C language dates back to 1972, when Dennis Ritchie first started the development. His co-worker, Ken Thompson, required a low-level language for utilities within Unix and there was no suitable language to do so and therefore C was designed for this task. Eventually, the Unix kernel itself was re-implemented in C, marking the first time Unix was implemented in anything other than Assembly.

This decision came about as the new language was low-level enough, and therefore efficient, to implement an operating system. The language is still used today because of its relative ease of use while still maintaining portability and efficiency [18].

3.1 Application of the Split-Step Fourier Method

The code shown in this chapter is an implementation of a first-order soliton without any compression. The C language standard is C99 as it contains the complex.h library, which is vital for this project. Alongside this, stidio.h, stdlib.h math.h and time.h, for execution speed measurement, are used as well. Two constants are used - M_PI which is π and NSAMPLES which is the number of samples.

```c
double b2 = -10;
double gamma = 0.1;
double To = 10;
double T = 20 * To;
double C = 0;

double N_order = 1;
double Po = pow(N_order, 2);
double Ld = pow(To, 2) / abs(b2);
double Lnl = 1/(Po*gamma);

double dz = Ld/100;;
double L = 10*Ld;


int z_size = L / dz;
double z_vector[z_size];
for(int i = 0; i < z_size; i++)
{
    z_vector[i] = i*dz;
}
```

(a) Initial conditions setup

```c
double Fs = (NSAMPLES - 1) / T;
double df = 2*M_PI / T;
double t[NSAMPLES];
double f[NSAMPLES];
for (int i = 0; i < NSAMPLES; i++)
{
    t[i] = ((-NSAMPLES/2) + i)*(1/Fs);
    f[i] = ((-NSAMPLES/2) + i)*df;
}
//Frequency vector shift
for(int k = 0; k < NSAMPLES/2; k++)
{
    double temp[NSAMPLES];

    temp[NSAMPLES-1] = f[0];
    for(int i = 0; i < NSAMPLES-1; i++)
        temp[i] = f[i+1];

    for(int i = 0; i < NSAMPLES; i++)
        f[i] = temp[i];
}
```

(b) Time and frequency vector setup

Figure 9: Standard setup of initial conditions. The frequency spectrum is shifted to enable discrete Fourier transform, an equivalent to MATLAB's fftshift().

```c
double complex * simul_wave = (double complex *)calloc(NSAMPLES, sizeof(double complex));
double complex * simul_wave_storage = (double complex *)calloc(z_size*NSAMPLES, sizeof(double complex));
double complex * A = (double complex *)calloc(NSAMPLES, sizeof(double complex));

for(int j = 0; j < NSAMPLES; j++)
{
    A[j] = N_order*(1/ccosh(t[j]/To));
    simul_wave[j] = cpow(cabs(A[j]),2);
}
```

Figure 10: Setup of the initial wave. The vectors holding the wave are declared as double complex pointers and have their memory allocated based on the number of samples. While the arrays "simul_wave" and "simul_wave_storage" do not need to be complex arrays, they are not so vital to require optimisation in terms of data types.

```c
double complex * D = (double complex *)calloc(NSAMPLES, sizeof(double complex));
int total_track = 0;
for(int i = 0; i < z_size; i++)
{
    //Compute N
    double complex N[NSAMPLES];
    for(int j = 0; j < NSAMPLES; j++)
    {
        N[j] = cexp(I*gamma*dz*cpow(cabs(A[j]),2));
    }

    //Fourier transform of A
    A = fft_wrapper(A, NSAMPLES);

    //Compute D
    for(int j = 0; j < NSAMPLES; j++)
    {
        D[j] = cexp(I*(dz/2)*b2*cpow(f[j],2));
        D[j] = D[j]*A[j];
    }

    //Inverse Fourier Transform of D
    D = ifft_wrapper(D, NSAMPLES);

    //A = D*N
    for(int j = 0; j < NSAMPLES; j++)
    {
        A[j] = D[j]*N[j];
        simul_wave_storage[total_track] = cpow(cabs(A[j]),2);
        total_track++;
    }
}
```

Figure 11: Split-step Fourier Method implementation. The algorithm works in six steps as shown previously on the flowchart. Notably, $\hat{D} = ih\frac{\beta_2}{2}\omega^2$ as a derivative is simple multiplication in the frequency domain. The fft_wrapper and ifft_wrapper functions are supplied by the fft-ifft-recursive.h library, which is explained in the following section.

```c
char * fs = fopen("D:\\Uni\\4thYear\\EG4013\\thesis-work\\MATLAB\\SSFM\\vals.csv", "w");
for(int j = 0; j < NSAMPLES; j++)
{
    fprintf(fs, "%f%+fi,", creal(simul_wave[j]), cimag(simul_wave[j]));
}
fprintf(fs, "\n");

for(int i = 0; i < NSAMPLES*z_size; i++)
{
        if(i % NSAMPLES == 0)
            fprintf(fs, "\n");
        fprintf(fs, "%f%+fi,", creal(simul_wave_storage[i]), cimag(simul_wave_storage[i]));
}

free(D);
free(simul_wave_storage);
fclose(fs);
```

Figure 12: Saving the waveform as a csv file and clean up.

3.2 Discrete Fourier Transform

Apart from the SSFM, it is also important to explain the implementation of a Fourier Transform. Simply put, as computers do not have infinite memory, some discretisation is required, hence Discrete Fourier Transforms (DFT) with a finite number of samples with equal spacing are used. They can be implemented mostly in two ways. The first is the most obvious one and that is the naive DFT.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk} \qquad k = 0, \ldots, N-1 \tag{3.2.1}$$

```
double complex * fft_conquer_func(double complex * original_vector, int num_samples)
{
    double complex * fft_vector = (double complex *)calloc(num_samples, sizeof(double complex));
    double complex wn = cexp(-2*M_PI*I/num_samples);
    int exponent;
    for(int k = 0; k < num_samples; k++)
    {
        for(int j = 0; j < num_samples; j++)
        {
            exponent = j*k;
            fft_vector[k] = fft_vector[k] + original_vector[j]*cpow(wn,exponent);
        }

    }
    //Copy the FFT vector to the passed pointer
    for(int j = 0; j < num_samples; j++)
    {
        *original_vector = *fft_vector;
        original_vector++;
        fft_vector++;
    }
    original_vector = original_vector - num_samples;
    fft_vector = fft_vector - num_samples;
    free(fft_vector);
    return original_vector;
}
```

Figure 13: Naive implementation of a discrete Fourier transform. The fft vector is copied to the passed pointer instead of returning it. This is because the pointer is declared within the scope of the function and it would not be returned properly.

The second one is the most common and is called Cooley-Tukey algorithm [19], originally invented by Carl Friedrich Gauss. The fast Fourier transform (FFT) splits the entire array into two based on parity and does so recursively, this approach is otherwise known as “Divide and Conquer”. The radix-2 version implemented requires the array to have a length of power of two.

$$X_k = \sum_{n=0}^{N/2-1} x_{2n} e^{-\frac{2\pi i}{N/2}nk} + e^{-\frac{2\pi i}{N}k} \sum_{m=0}^{N/2-1} x_{2n+1} e^{-\frac{2\pi i}{N/2}nk} \tag{3.2.2}$$

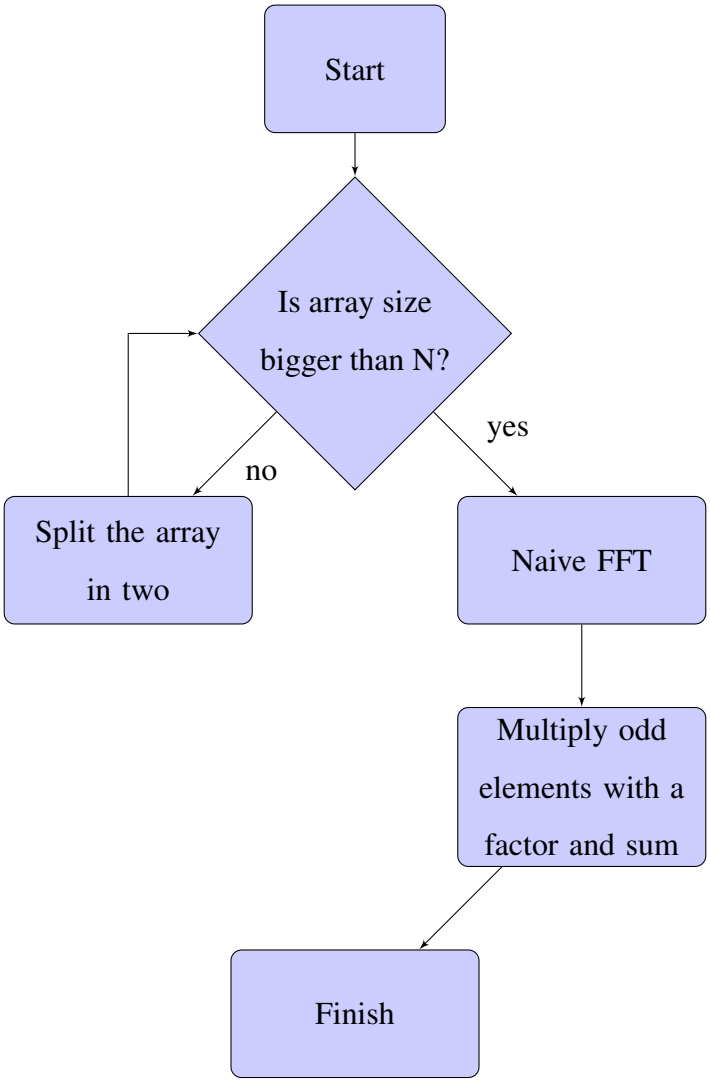


Figure 14: Cooley-Tukey algorithm

```c
double complex * fft_divide_func(double complex * original_vector, int num_samples)
{
    if(num_samples <= 2)
        return fft_conquer_func(original_vector, num_samples);
    else
    {
        double complex factor[num_samples];
        double complex * odd = (double complex *)calloc(num_samples/2, sizeof(double complex));
        double complex * even = (double complex *)calloc(num_samples/2, sizeof(double complex));
        double complex comb[num_samples];
        //Split the original vector
        int k = 0;
        for(int j = 0; j < num_samples; j++)
        {
            if(j % 2 == 0)
                even[k] = original_vector[j];
            else
            {
                odd[k] = original_vector[j];
                k++;
            }
        }
        even = fft_divide_func(even, num_samples/2);
        odd = fft_divide_func(odd, num_samples/2);

        //Calculate the factor
        for(int k = 0; k < num_samples; k++)
            factor[k] = cexp(-2*M_PI*I*k/num_samples);

        //Multiply out
        k = 0;
        while(k < num_samples)
        {
            for(int j = 0; j < num_samples/2; j++)
            {
                original_vector[k] = even[j] + odd[j]*factor[k];
                k++;
            }
        }
        free(odd);
        free(even);
        return original_vector;
    }
}
```

Figure 15: The division part of the “Divide and Conquer” FFT algorithm. If the passed pointer is larger than 2, which can be optimised, it is divided and then recursively called again. Once it is equal to 2, the function from Figure 13 is called.

The FFT algorithm easily exploits parallel execution as it executes one simple instruction many times over. The graphics processing unit (GPU) is a perfect device for this as this is what it is doing when rendering an image. NVidia provides the CUDA application programming interface (API) to directly access the NVidia GPU. This parallel scheme can also take advantage of the Cooley-Tukey scheme [20].

```
cuDoubleComplex * fft_conquer(cuDoubleComplex * original_vector, int num_samples, cuDoubleComplex * wn_vector)
{
    cuDoubleComplex * d_wn_vector;
    cuDoubleComplex * d_original_vector;
    cuDoubleComplex * d_fft_vector;
    cudaMalloc(&d_wn_vector, num_samples * num_samples * sizeof(cuDoubleComplex));
    cudaMalloc(&d_original_vector, num_samples * num_samples * sizeof(cuDoubleComplex));
    cudaMalloc(&d_fft_vector, num_samples * sizeof(cuDoubleComplex));

    cudaMemcpy(d_wn_vector, wn_vector, num_samples*num_samples * sizeof(cuDoubleComplex), cudaMemcpyHostToDevice);
    for(int i = 0; i < num_samples; i++)
        cudaMemcpy(&d_original_vector[i*num_samples], original_vector, num_samples * sizeof(cuDoubleComplex), cudaMemcpyHostToDevice);

    fftCUDA << <num_samples, num_samples >> > (d_original_vector, d_wn_vector, d_fft_vector, num_samples);

    cudaMemcpy(original_vector, d_fft_vector, num_samples*sizeof(cuDoubleComplex), cudaMemcpyDeviceToHost);

    cudaFree(d_wn_vector);
    cudaFree(d_original_vector);
    cudaFree(d_fft_vector);

    return original_vector;

}
```

Figure 16: Conquer function of the CUDA implementation. The data is copied to the device using cudaMemcpy, executed using fftCUDA and copied back.

```
__global__ void fftCUDA(cuDoubleComplex * sum_vector, cuDoubleComplex * wn_vector, cuDoubleComplex * fft_vector, int num_samples)
{
    int tid = threadIdx.x;
    int number_of_blocks = blockDim.x / 2;
    int step = 1;
    int target, source;
    sum_vector[blockIdx.x*num_samples + tid] = cuCmul(sum_vector[blockIdx.x*num_samples + tid], wn_vector[blockIdx.x*num_samples+tid]);
    while (number_of_blocks > 0)
    {
        if (tid < number_of_blocks)
        {
            target = blockIdx.x*num_samples + tid * step * 2;
            source = target + step;
            sum_vector[target] = cuCadd(sum_vector[target], sum_vector[source]);

        }

        step = step * 2;
        number_of_blocks = number_of_blocks / 2;

    }
    if (number_of_blocks == 0 && tid == 0)
    {
        fft_vector[blockIdx.x] = sum_vector[blockIdx.x*num_samples + tid];
    }
}
```

Figure 17: Computation of odd and even arrays with length of 32 samples. While the intent is the same as in the Figure 13, the actual implementation is different. Multiplication works by executing the operation at once across the entire array, the summation of the vector is done through parallel reduction, where multiple pairs within the same array are added together in the fashion of an inverse binary tree.

The inverse of a DFT is fairly trivial and has nearly the same complexity

$$X_k = \frac{1}{N}\sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N}-nk} \qquad k = 0,\ldots,N-1 \tag{3.2.3}$$

3.2.1 Algorithm Comparison

	1024 Samples	2048 Samples
Method	Execution speed	Execution speed
Naive DFT	0.220 s	0.687 s
CPU FFT	0.001 s	0.002 s
CUDA FFT	0.320 s	0.545 s

Table 1: Comparison of DFT methods at 1024 and 2048 samples. The CPU used was Intel i7-4710MQ, GPU NVIDIA K1100M. Central processing unit (CPU) FFT emerges superior across samples with the naive DFT and CUDA FFT switching places.

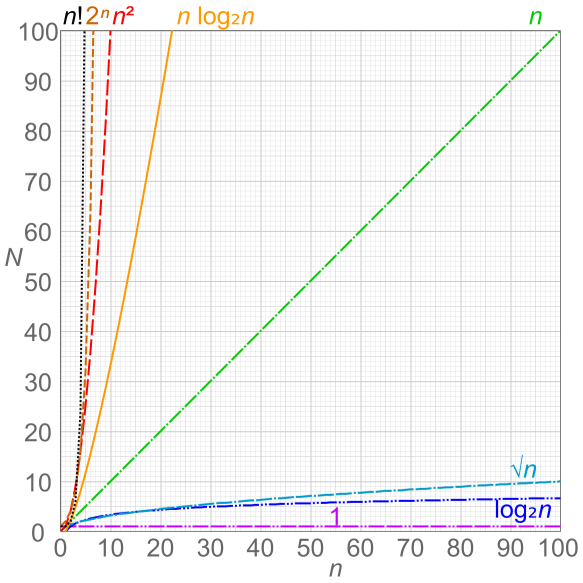


Figure 18: Comparison of various computational complexities. Taken from [21].

While it is clear why the recursive implementation is faster than the naive when the time complexity for naive is $O(N^2)$ and recursive is $O(5Nlog_2(N))$ [22][23], it is not so clear why the CUDA implementation takes much longer to complete. The reason, despite the algorithm being nearly identical to the one used on a CPU, is that the data transfers from the GPU constitute the longest part of the execution. However, much of this can be optimised to achieve performances of a CPU FFT and higher [24].

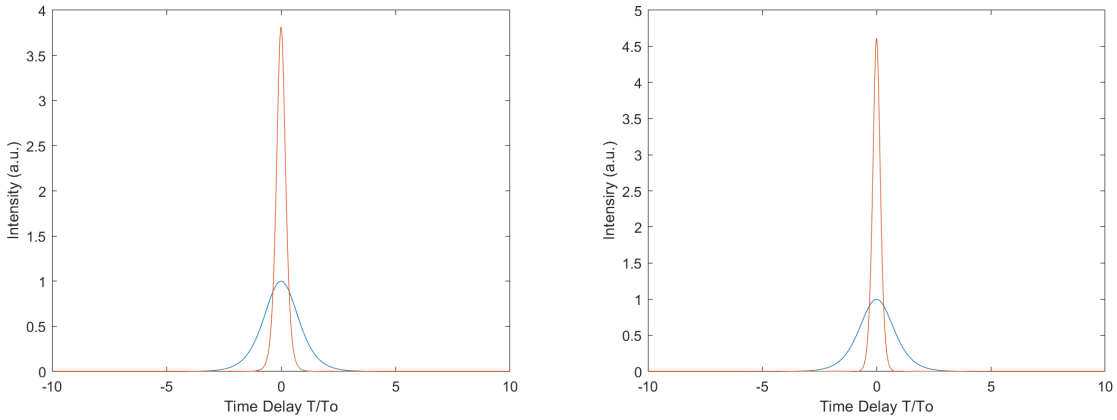
In this chapter, the implementation of the Split-step Fourier method was demonstrated using the C99 standard. Afterwards, three different approaches to discrete Fourier transforms were displayed with a Cooley-Tukey radix-2 algorithm implemented on a CPU having the best performance, followed by the same algorithm implemented on a GPU.

4 Pulse Compression

As mentioned before, pulse compression is extremely vital for communication systems, particularly in time division multiplexing. In this chapter, explanation of compression and the comparison between linear and exponential profiles are done, followed by a demonstration of second order solitons and their compression. Lastly, optimisation of real world profiles is shown.

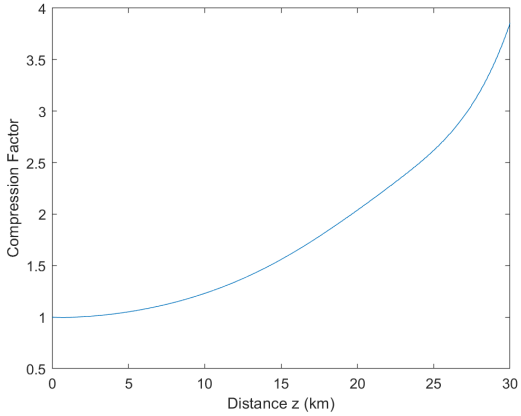
4.1 Linear and Exponential Profiles

As mentioned before, the object of interest in this thesis is the compression of pulses using dispersion decreasing fibres. It is therefore paramount to demonstrate what that means and then compare the two most basic methods, compression using linear and exponential profiles.

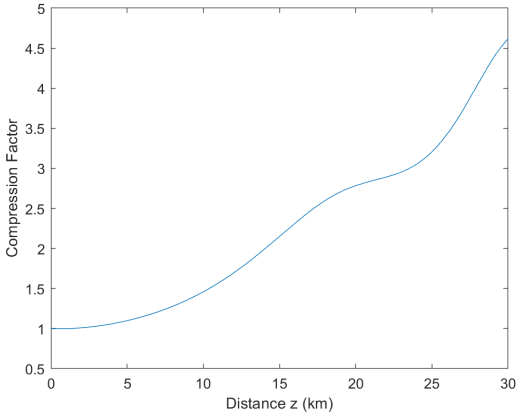


(a)

(b)



(c)



(d)

Figure 19: Initial and final waveforms for linear and exponential profiles in Figures 19a and 19b respectively. The fibre parameters for the linear compression are the same with $\beta_2 = -33\ ps^2km^{-1}, \gamma = 0.33\ W^{-1}km^{-1}$ and $L = 30\ km$ but for linear $\sigma = 0.025\ km^{-1}$ and exponential $\sigma = 0.05\ km^{-1}$. Compression factors on 19c and 19d respectively. The construction of each profile is straighforward, both being driven by the decay constant σ. For linear, it is simply $\beta_2 = \beta_{20} + \sigma z$ and for exponential, it is $\beta_2 = \beta_{20} exp(-\sigma z)$, where β_{2fo0} is the initial dispersion and $\sigma = C_0\beta_{20}$.

The compression itself relies on the interaction between initial chirp, decreasing dispersion and constant self-phase modulation. As the the chirp is embedded in the decay constant σ, it interacts with the soliton at all times by helping to maintain the balance between chirps generated by linear and nonlinear effects, specifically it adds to the linear chirp.

To analyse the results, it is important to look at two parameters. First is the compression factor, which is simply the ratio of the full-width-half-maximum length of the initial wave form and the target wave, as seen in Figures 19c and 19d. The second parameter is the pedestal energy, which is defined as [25]

$$PE(\%) = \frac{|E_{total} - E_{sech}|}{E_{total}} \times 100\%, \quad E_{sech} = 2P_{peak}\frac{T_{FWHM}}{1.763}.$$

The linearly decreasing dispersion produces a compression of roughly 4 times the original with an insignificant pedestal of 0.8%, the exponentially decreasing dispersion produces compression of around 4.6 times the original width with an even smaller pedestal energy of 0.25% and is therefore the preferred option for any further compressions.

4.2 Advanced Compression Techniques

There is also the possibility of compression using higher order solitons, as demonstrated by Senthilanathan et al. (2012), that is either using a standard two soliton or a second-order soliton breather, which has the initial function

$$A = \frac{2}{T_0}\sqrt{\frac{\beta_0}{\gamma}}sech(\frac{t}{T_0})exp[i(\frac{\alpha_0}{2}t^2 + \frac{\pi}{8})],$$

where $\alpha = -4/(T_0^2\pi)$ and is twice the amplitude of a fundamental soliton.

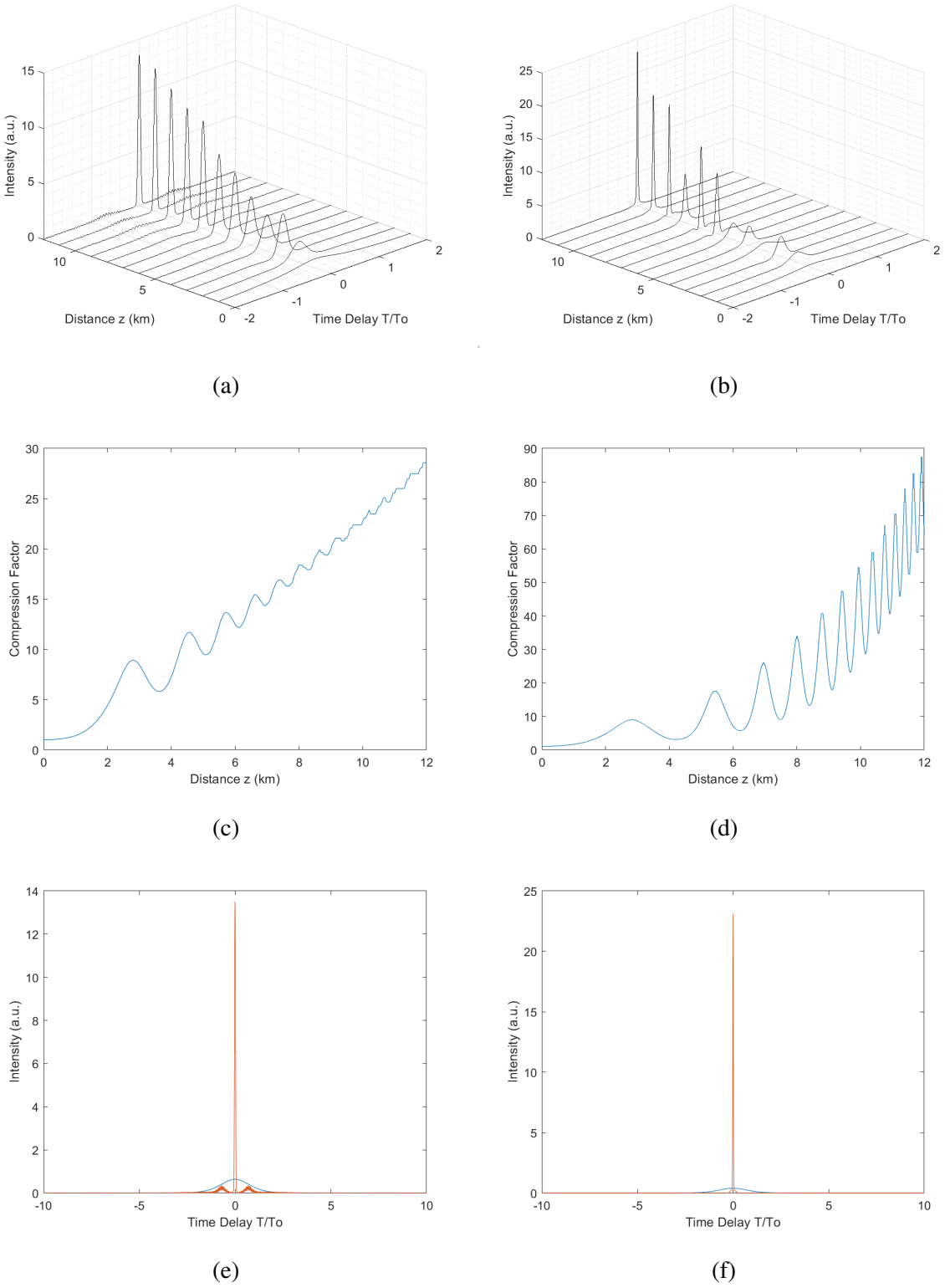


Figure 20: Compression of standard second order soliton on the left and a second order soliton breather on the right with their compression factors and amplitudes of the first and last pulse. The parameters for both are $\beta_2 = -20\ ps^2 km^{-1}, \gamma = 2\ W^{-1} km^{-1}$ and $z = 12\ km$

From the Figure 20, it can be seen that the standard second order soliton experiences compression of the factor thirty, this is accompanied by a pedestal energy of 26%. The second order soliton breather has a compression of up to 87 times with energy loss of 4%.

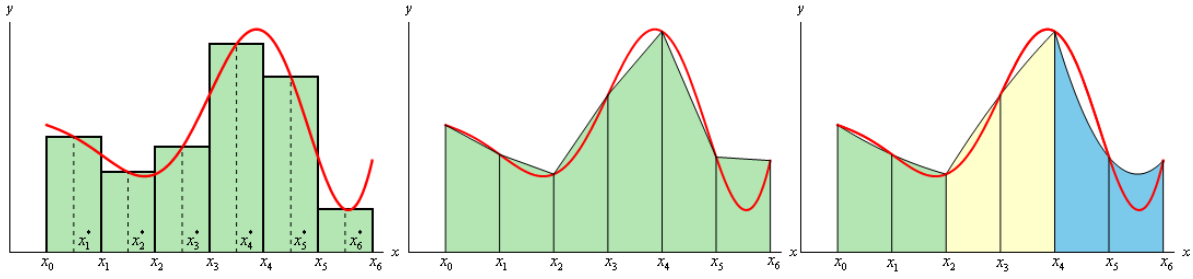
The compression of a soliton breather is clearly the superior method with an almost insignificant pedestal and higher compression.

4.3 Stepwise Approximation of Exponential Profiles

In real life, it is simply not possible to produce an optic fibre with a continuously changing profile and therefore it has to be approximated using steps; the more steps, the better. However, four-segment fibre is a reasonable candidate for a study [26].

4.3.1 Search for Optimum Profiles

To find the optimum profile, it is best to approach the issue as an integration problem. This is possible because ultimately, the goal is to create a profile that best estimates the original curve. To do this, there are three main options and that is the midpoint rule, trapezoidal rule and the Simpson's rule. Simply put, the midpoint rule approximate the curve with rectangles, trapezoidal rule uses trapezoids and Simpson's rule uses quadratic curves.



(a) Midpoint Rule (b) Trapezoid Rule (c) Simpson's Rule

Figure 21: Taken from [27].

In this case, the midpoint rule is of the biggest interest as the profile is made up for several segments with constant dispersion coefficient. The curve is segmented by rectangles with equal width and different height and then their area is summed. This rule can be enhanced by borrowing an error estimation from the Simpson's rule [28], which is:

$$\varepsilon = \frac{1}{15}|S(a,b) + S(a,c) - S(c,b)|$$

This combination of the midpoint rule and the error estimation can be readily implemented in C as a recursive function. The factor $1/15$ can be substituted for $1/10$ to decrease the chance infinite recursive loops.

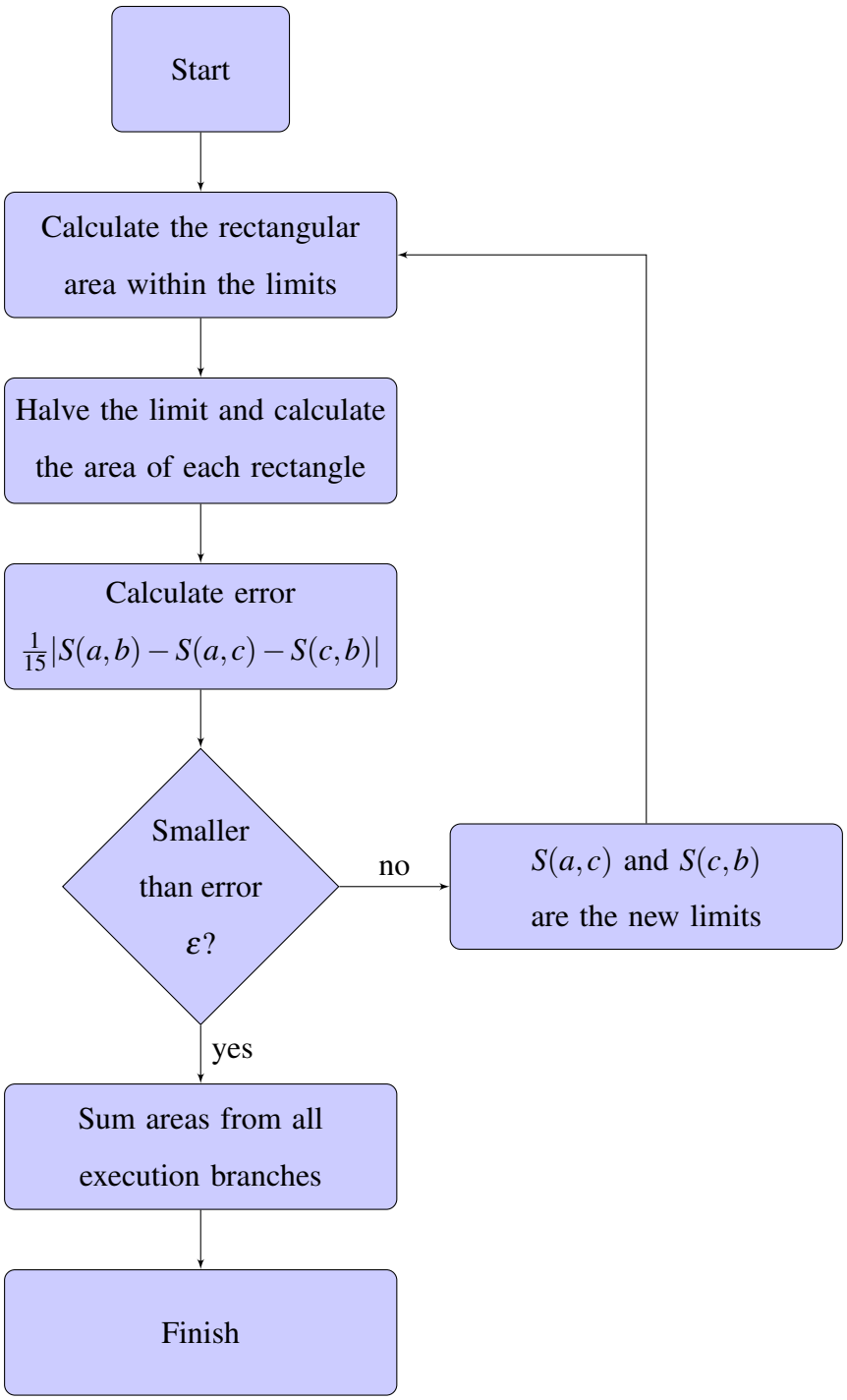
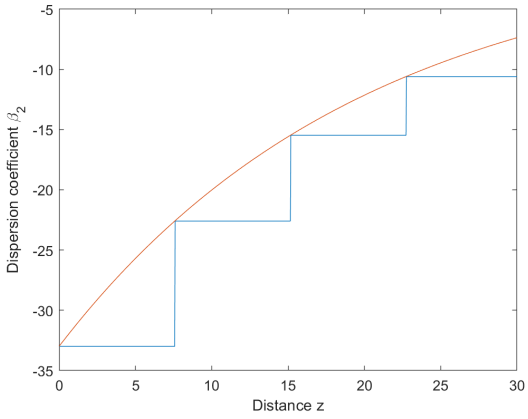
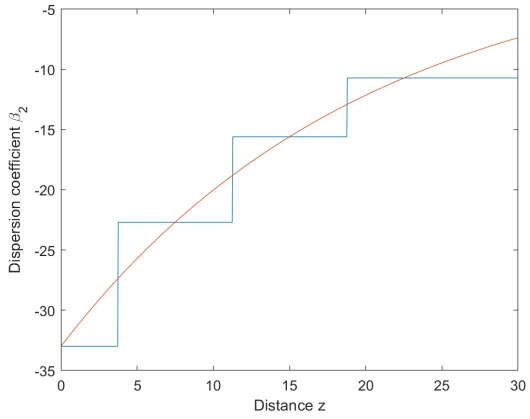


Figure 22: Adaptive midpoint rule method flowchart

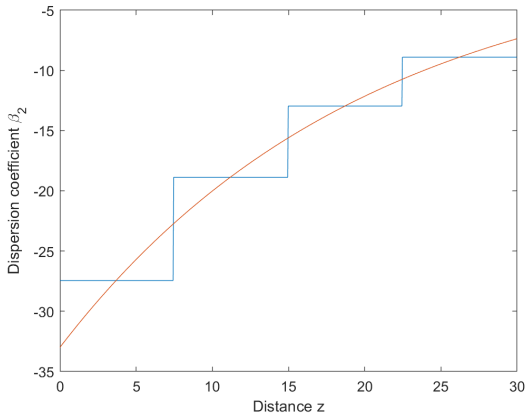
4.3.2 Profile Comparison and Analysis



(a) Simple sections



(b) Simple sections halved



(c) Sections based on the adaptive method

Figure 23: Stepwise dispersion profiles. Three profiles are compared, the most simplest one with the continuous profile divided into four segments, a profile with simple sections halved and finally a profile based on the adaptive method.

Construction Method	Maximum Compression	Pedestal Energy
Continuous Profile	4.5	0.03%
Simple Sections	3.12	2.3%
Simple Sections Halved	3.12	2.3%
Adaptive Method	4	0.8%

Table 2: Comparison of the three methods. While the first two methods compare exactly the same, the approximation based on an adaptive rule shown an large improvement in terms compression and pedestal energy, which is insignificant. The adaptive rule also trails very closely behind the continuous profile of the dispersion coefficient in terms of compression.

Out of the three methods, the one based on adaptive midpoint rule is clearly the best with maximum compression of 4 and insignificant pedestal energy, compared to the continuous profile with compression of 4.5 and an insignificant pedestal energy.

Profile	Compression Factor	Pedestal Energy
Continuous	87	4%
4 Segments	61	16%
5 Segments	61	16%
6 Segments	54	23%
10 Segments	74	4.8%
20 Segments	78	4.23%

Table 3: Comparison of several segmented profiles against a continuous profile applied to a second-order soliton breather. The quality of compression increases as the number of segments increases albeit not proportionally. However, the quality decreases between five and four segments.

As expected, the compression factor does increase with the amount of segments and the pedestal also improves. The inconsistency between five and six segments is explained by the fact that a second order soliton oscillates and therefore the dispersion coefficient at that point, caused the pulse to have a smaller instensity than expected.

Overall, it has been determined that exponential profiles are superior to linear ones due to a much lower energy loss during compression, in this case 0.8% compared to 0.25%. Secondly, compression of a standard second order soliton and a second order soliton breather was done, with the breather being the better candidate compression with an 87 times smaller width and a nearly insignificant pedestal of 4%. Lastly, it was demonstrated that the adaptive midpoint rule produces very good approximation for a real world fibre, only slightly trailing the continous profile by 0.5 in terms of compression for a fibre made up of four segments.

5 Recommendations and Suggestions for Future Work

In this thesis, four areas were explored and that is background theory, model of wave propagation in an optic fibre, compression and alongside that, stepwise approximation. There are many possibilities for the future, however there are two most obvious ones.

First is further improvement of the model to allow for compression of pulses with a width smaller than 5 *ps*. This would allow more research into the combination and compression method that uses very short pulses and is affected by more nonlinear effects [29].

Second, the development of a Python wrapper of sorts that would implement the existing C but also take advantage of the better data visualisation of Python to work with data. Furthermore, the C could possibly be updated to include the C11 standard and make use of multithreading.

There are other minor possibilities, such as comparison between DDFs and Fibre Bragg Gratings or perhaps whether they can be combined.

6 Conclusion

In this report, the research into literature on the topic of dispersion decreasing fibres and their use was done. It was noted that pulses of lengths in picoseconds are in demand by the communication industry and one of the viable ways to generate them is compressing chirped solitons. The basics of nonlinear fibre optics have also been presented, with the emphasis on the interplay between group-velocity dispersion and self-phase modulation, particularly solitons.

The Split-Step Fourier Method was explained and applied to simulate various pulses. Additionally, three different Fourier transform algorithms were benchmarked, with a Cooley-Tukey radix-2 algorithm being faster than a naive Fourier Transform and Cooley-Tukey radix-2 algorithm applied through CUDA API.

Compression of solitons was investigated using both linear and exponential profiles, with the latter being superior with its higher compression and lower pedestal energy. This profile was then used along with a standard second order soliton and a soliton breather. It was

found that the soliton breather performed extremely well with a high compression rate and low energy loss. Furthermore, three different real world fibre profiles were compared to demonstrate the effectivness of the adaptive midpoint rule to design a stepwise profile.

Finally, several possibilities for the future of the project were explained, particularly extending the model to pulses with a width smaller than $5ps$ and developing a Python wrapper for better analysis.

References

[1] Mollenauer, L.F. and Mamyshev, P.V., (1998). Massive wavelength-division multiplexing with solitons. *IEEE journal of quantum electronics*, **34**(11), pp.2089-2102.

[2] Limpert, J., Roser, F., Schreiber, T. and Tunnermann, A., (2006). High-power ultrafast fiber laser systems. *IEEE Journal of selected topics in Quantum Electronics*, **12**(2), pp.233-244.

[3] Wise, F.W., Chong, A. and Renninger, W.H., (2008). High-energy femtosecond fiber lasers based on pulse propagation at normal dispersion. *Laser & Photonics Reviews*, **2**(1-2), pp.58-73.

[4] Chernikov, S.V., Dianov, E.M., Richardson, D.J. and Payne, D.N., (1993). Soliton pulse compression in dispersion-decreasing fiber. *Optics letters*, **18**(7), pp.476-478.

[5] Li, Q., Jian, Z., Lu, W., Nakkeeran, K., Senthilnathan, K., & Wai, P. K. A. (2018). Combination and Compression of Multiple Optical Pulses in Nonlinear Fibers with the Exponentially Decreasing Dispersion. *IEEE Journal of Quantum Electronics*, **54**(2), pp.1-10.

[6] Senthilnathan, K., Nakkeeran, K., Li, Q. and Wai, P.K.A., (2012). Pedestal free pulse compression of chirped optical solitons. *Optics Communications*, **285**(6), pp.1449-1455.

[7] Mamyshev, P.V., Chernikov, S.V. and Dianov, E.M., (1991). Generation of fundamental soliton trains for high-bit-rate optical fiber communication lines. *IEEE journal of quantum electronics*, **27**(10), pp.2347-2355.

[8] Chan, K.C. and Liu, H.F., (1995). Short pulse generation by higher order soliton-effect compression: effects of optical fiber characteristics. *IEEE journal of quantum electronics*, **31**(12), pp.2226-2235.

[9] Moores, J.D., (1996). Nonlinear compression of chirped solitary waves with and without phase modulation. *Optics letters*, **21**(8), pp.555-557.

[10] MKS Instruments, (2019). Fiber Optic Basics, Available at: https://www.newport.com/t/fiber-optic-basics (Accessed 12 January 2019)

[11] Agrawal, G.P., (2001). Nonlinear Fiber Optics, 3rd ed. London: Academic Press.

[12] Agrawal, G.P. and Potasek, M.J., (1986). Nonlinear pulse distortion in single-mode optical fibers at the zero-dispersion wavelength. *Physical Review A*, **33**(3), p.1765.

[13] Shimizu, F., (1967. Frequency broadening in liquids by a short light pulse. *Physical Review Letters*, **19**(19), p.1097.

[14] Stolen, R.H. and Lin, C., (1978). Self-phase-modulation in silica optical fibers. *Physical Review A*, **17**(4), p.1448.

[15] Hasegawa, A. and Tappert, F., (1973). Transmission of stationary nonlinear optical pulses in dispersive dielectric fibers. I. Anomalous dispersion. *Applied Physics Letters*, **23**(3), pp.142-144.

[16] Taha, T.R. and Ablowitz, M.I., (1984). Analytical and numerical aspects of certain nonlinear evolution equations. II. Numerical, nonlinear Schr$\acute{}$odinger equation. *Journal of Computational Physics*, **55**(2), pp.203-230.

[17] Fisher, R.A. and Bischel, W.K., (1975). Numerical studies of the interplay between self-phase modulation and dispersion for intense plane-wave laser pulses. *Journal of Applied Physics*, **46**(11), pp.4921-4934.

[18] Ritchie, D.M., Kernighan, B.W. and Lesk, M.E., (1975). The C programming language. Bell Laboratories.

[19] Cooley, J.W. and Tukey, J.W., (1965). An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation*, **19**(90), pp.297-301.

[20] Volkov, V. and Kazian, B., (2008). Fitting FFT onto the G80 architecture. *University of California, Berkeley*, **40**.

[21] Wikipedia, (2019). Time complexity, Available at: https://en.wikipedia.org/wiki/Time_complexity (Accessed 14 April 2019)

[22] Frigo, M. and Johnson, S.G., (2005). The design and implementation of FFTW3. *Proceedings of the IEEE*, **93**(2), pp.216-231.

[23] Soni, M. and Kunthe, P., (2011). A General comparison of FFT algorithms. *Pioneer Journal Of IT & Management*.

[24] Govindaraju, N.K., Lloyd, B., Dotsenko, Y., Smith, B. and Manferdelli, J., (2008), November. High performance discrete Fourier transforms on graphics processors. *In Proceedings of the 2008 ACM/IEEE conference on Supercomputing* (p. 2). IEEE Press.

[25] Cao, W.H. and Wai, P.K.A., (2005). Picosecond soliton transmission by use of concatenated gain-distributed nonlinear amplifying fiber loop mirrors. *Applied optics*, **44**(35), pp.7611-7620.

[26] Li, Q., Senthilnathan, K., Nakkeeran, K. and Wai, P.K.A., (2009). Nearly chirp-and pedestal-free pulse compression in nonlinear fiber Bragg gratings. *JOSA B*, **26**(3), pp.432-443.

[27] Paul's Online Notes, (2018). Calculus II - Approximating Definite Integrals, Available at: `http://tutorial.math.lamar.edu/Classes/CalcII/ApproximatingDefIntegrals.aspx` (Accessed 14 April 2019)

[28] Lyness, J.N., (1969). Notes on the adaptive Simpson quadrature routine. *Journal of the ACM (JACM)*, **16**(3), pp.483-495.

[29] Li, Q., Jian, Z., Lu, W., Nakkeeran, K., Senthilnathan, K. and Wai, P.K.A., (2018). Combination and compression of multiple optical pulses in nonlinear fibers with the exponentially decreasing dispersion. *IEEE Journal of Quantum Electronics*, **54**(2), pp.1-10.

A Additional code

```
double simpsonRule(double * exp_vec, double * dis_vec, double tar, int arr_size)
{
    int a = 0;
    int b = arr_size-1;

    double amp_diffAB = exp_vec[b] - (exp_vec[b] - exp_vec[a])/2;
    double dis_diffAB = dis_vec[b] - dis_vec[a];
    double areaAB = amp_diffAB*dis_diffAB;

    double amp_diffAC = exp_vec[b/2] - (exp_vec[b/2] - exp_vec[a])/2;
    double dis_diffAC = dis_vec[b/2] - dis_vec[a];
    double areaAC = amp_diffAC*dis_diffAC;

    double amp_diffCB = exp_vec[b] - (exp_vec[b] - exp_vec[b/2])/2;
    double dis_diffCB = dis_vec[b] - dis_vec[b/2];
    double areaCB = amp_diffCB*dis_diffCB;

    double error = 0.1*absDouble(areaAB-areaAC-areaCB);

    if(error < tar)
    {
        printf("a is %0.4f, b is %0.4f\n",dis_vec[a],dis_vec[b]);
        return absDouble(areaAB);
    }


    else
    {
        double * left_exp = exp_vec;
        double * left_dis = dis_vec;
        double * right_exp = &exp_vec[b/2];
        double * right_dis = &dis_vec[b/2];

        double left = simpsonRule(left_exp, left_dis, tar, arr_size/2);
        double right = simpsonRule(right_exp, right_dis, tar, arr_size/2);

        left_exp = left_dis = right_exp = right_dis = NULL;

        return (left+right);
    }
}
```

Figure 24: Code for the adaptive midpoint rule.

```c
double complex * fft_wrapper(double complex * original_vector, int num_samples)
{
    original_vector = fft_divide_func(original_vector, num_samples);
    return original_vector;
}


double complex * ifft_wrapper(double complex * original_vector, int num_samples)
{
    double complex * temp_vector = (double complex *)calloc(num_samples-1, sizeof(double complex));
    original_vector = ifft_divide_func(original_vector, num_samples);
    for(int k = 0; k < num_samples; k++)
    {
        original_vector[k] = original_vector[k] * (1.0/num_samples);
    }

    int i = 1;
    for(int k = 1; k < num_samples/2; k++)
    {
        temp_vector[k-1] = original_vector[k];
        original_vector[k] = original_vector[num_samples - i];
        original_vector[num_samples - i] = temp_vector[k-1];
        i++;
    }

    free(temp_vector);
    return original_vector;
}
```

Figure 25: Wrappers for FFT and IFFT functions. As noted previously, the IFFT wrapper divides the sum by the number of samples. Furthermore, the array is shifted circularly, an equivalent of MATLAB's ifftshift().