University College London

PHASM201: Project Report

DEVELOPING ACCELERATORS 1000 TIMES MORE POWERFUL THAN CURRENT MACHINES

Author:
Danial Dervovic

1st Supervisor:
Prof. Matthew Wing
2nd Supervisor:
Dr. Simon Jolly

Submitted in partial fulfilment for the degree of **MSci Physics**

March 2015

I, Danial Dervovic, confirm that the work presented in this report is my own. Where information has been derived from other sources, I confirm that this has been indicated in the report.

_____

Signature

_____

Date

Abstract

Department of Physics and Astronomy
University College London

MSci Physics

Developing Accelerators 1000 Times More Powerful than Current Machines

by Danial Dervovic

danial.dervovic.11@ucl.ac.uk

This report documents the PHASM201 project undertaken by the author in Proton Driven Plasma Wakefield Acceleration (PDPWA), specifically the recently launched AWAKE experiment at CERN. The main project aim was to extend simulations of the SPS proton beam interacting in a Rb plasma cell to incorporate a vacuum drift space and an additional plasma cell. This was acheived, with accelerating fields of 0.7GV/m to 0.25GV/m observed over a combined 9m of plasma and drift space, at the point in the wakefield at which electrons are injected. Studies of the beam profile and shape in phase space, along with an overview of PDPWA and outlook of the project are included in the text also.

Contents

Acknowledgements

Thank you to my supervisor Matthew Wing, and to the AWAKE group at UCL, Simon Jolly, Lawrence Deacon, Scott Mandry, Peter Sherwood and Fearghus Keeble. Your help and discussions have been vital to the completion of this project, which has proven to be a rewarding and educational experience.

Physical Constants

Speed of Light	c	=	$2.997\ 924\ 58 \times 10^{8}$ ms^{-2} (exact)
Charge of the electron	$-e$	=	$1.602\ 176\ 565 \times 10^{-19}$ C
Mass of the electron	m_e	=	$9.109\ 382\ 91 \times 10^{-31}$ kg
Permittivity of the vacuum	ϵ_0	=	$8.854\ 187\ 817 \times 10^{-12}$ F/m
Mass of the proton	m_p	=	$1.672\ 621\ 777 \times 10^{-27}$ kg

Constants are taken from CODATA recommended values (2010) [1].

Symbols

n_p	plasma density	m^{-3}
ω_p	plasma frequency	rad s^{-1}
λ_p	plasma wavelength	m
n_b	proton bunch density	m^{-3}
σ_z	proton beam length	m
σ_r	proton beam width	m
ξ_b	distance behind proton beam head	m

Chapter 1

Introduction

1.1 Opening Statement

This document describes the PHASM201 project undertaken by the author, with a summary of the physics underlying the project, an outline of work done, results obtained and references to the relevant literature.

Chapter 1 details the relevant physics, motivating factors and the aims and objectives of the project. Chapter 2 describes the work that has been done to acheive the aims of the project. Chapter 3 presents the results of this work, and Chapter 4 provides a discussion of these results, putting them in context.

The project aims to contribute to the development of new methods of accelerating subatomic particles in High-Energy Physics (HEP) experiments. It forms a study of the as-yet untested technology of proton driven plasma wakefield acceleration (PDPWA), primarily in the context of the AWAKE experiment recently launched at CERN. The study is focused on assessing the viability of the staging of PDPWA accelerator cells using computer simulations, with the end goal of strengthening the case for PDPWA as the future of HEP experiments.

1.2 Motivation

The scientific utility of particle accelerators is well known, with the most recent high-profile discovery being the Higgs Boson [2, 3], at the Large Hadron Collider (LHC) in Geneva, Switzerland. This result was obtained in proton-proton collisions at a centre-of-mass energy of 7 TeV. The prevailing view in the particle physics community is that the next large particle accelerator should be a lepton collider, operating at the TeV scale.

A TeV-scale lepton collider is desirable for high-energy physics experiments as leptons are point particles[1]. The modern view of a proton is that of a 'soup' of quarks and gluons, more fundamental particles. This makes the analysis of proton-proton collision events difficult and means that the beam energy is distributed between many constituent particles. Using leptons would alleviate these issues.

A proposal with this goal in mind is the International Linear Collider (ILC)[4], a 500GeV centre-of-mass energy linear electron-positron collider, which can be upgraded to 1TeV. Whilst the technical design for the ILC is extensive, the cost of this project is holding back progress.

At the energy scales ($\sim$TeV) of acccelerators such as the LHC and the ILC, the amount by which radio-frequency (RF) cavities can accelerate particles is reaching its peak. The maximum electric field admitted by today's accelerating magnets is of the order 100 MV/m [5], due to RF breakdown on the walls of the accelerating structures used. For a linear accelerator using this acceleration mechanism, this corresponds to a length of $\sim$10 km.

Circular colliders such as the LHC are unsuitable for a lepton collider, as the effects of synchrotron radiation [6] become more pronounced. For a circular accelerator, the power radiated away due to synchrotron radiation scales as $(E/m)^{-4}$. An electron is $\sim$2000 times lighter than a proton. This means that an electron/positron beam will radiate away $\sim 10^{13}$ times more energy than an equivalent proton beam, travelling around a circular collider. This high level of energy loss makes TeV-scale circular lepton colliders unfeasible. An alternative to using electrons/positrons is using muons, which are $\sim$200 times heavier. However, their short lifetime ($\sim 2\mu$s) means any beam manipulation carried out before colliding the beams would have to take place on a timescale shorter than this, which is difficult experimentally.

Thus, in order to reach the TeV scale with current technologies, it is necessary to construct a many km-scale linear accelerator, the size and cost of which will become untenable in a generation's time.

Therefore, it is of utmost importance to the particle physics community that a medium with higher accelerating gradient be employed to create the next generation of accelerator technologies.

[1] At least, according to the Standard Model, a TeV scale lepton collider might challenge this view!

1.3 Plasmas as an Acceleration Medium

One such medium is a plasma. Radio-Frequency (RF) breakdown in accelerating structures is partly caused by ionisation of the walls leading to the structure being 'shorted', with the RF power partly reflected and partly absorbed in the process, thus reducing the power available to drive particles [7]. Ionised plasmas circumvent this problem, as they are already strongly ionised. Plasmas can sustain electron plasma waves with electric fields in excess of [5]:

$$E_0(\mathrm{V/m}) \simeq 96\sqrt{n_p(\mathrm{cm}^{-3})} \tag{1.1}$$

where n_p is the ambient electron number density. For instance, a plasma density of $n_p = 10^{18}\mathrm{cm}^{-3}$ supports a gradient of 96 GV/m, three orders of magnitude higher than that of today's accelerators.

1.3.1 Plasma Wakefield Acceleration

The idea of using plasma as an acceleration medium was first proposed by Tajima and Dawson in the late 1970s [8]. The mechanism by which a particle can be accelerated through a plasma is by a short ($\sim\mu$m) driver beam (of particles or radiation) being fired into the plasma.

The beam displaces the plasma electrons transversely, leaving a positive ion channel behind the driver. The plasma ions are taken to be effectively stationary as they are much heavier than the electrons, and the timescales considered are small. The positive ion channel exerts a restoring force on the electrons, which then oscillate transversely about the beam axis. This results in an alternating pattern of high density regions of positive and negative charges behind the driver beam, moving at the speed of the driver. The electric field due to this pattern forms a wake, which supports a strong electric field along the beam axis [9], as well as a strong focusing force.

Particles (know as a 'witness' beam) can then be accelerated by the wakefield of the driver beam, provided the witness beam is injected at an appropriate position.

1.3.2 Types of Driver Beam

There have been a number of different types of driver beam used in experiments, such as lasers in the scheme known as Laser Wakefield Acceleration [10], and electrons [11].

In the Laser Wakefield regime, laser pulses have accelerated electrons to 2 GeV energy gain within ∼cm long plasmas [10, 12], corresponding to accelerating fields of 100 GV/m.

An electron driver beam produced an accelerating field of 52 GV/m. In this experiment, the bunch tail was accelerated from 40 to 80GeV over 85 cm by the bunch head.

There are, however, limitations to acceleration in these methods, which pose problems when the end goal is to produce a new particle accelerator.

For instance, in the laser wakefield scheme, the maximum power short pulse (100-femtosecond) lasers are terawatt scale. The short pulse is required to excite the wakefield. We can say that, nominally, the energy available from this laser pulse is $100 \times 10^{-15}\text{s} \times 10^{12}\text{W} = 10^{-1}\text{J}$. We can then calculate roughly the energy required by a 1TeV electron bunch. Assume that the bunch has 10^{11} electrons, which is fairly typical.

$$10^{10}\text{electrons} \times 10^{12}\text{eV} \times e\ \text{J/eV} \sim 10^{3}\text{J} \tag{1.2}$$

Whilst not a rigorous calculation, the difference in orders of magnitude suggests that Laser wakefield acceleration is not currently suitable for a TeV scale lepton collider, as laser pulses of sufficient power at the pulse lengths required are not yet available.

In the electron driven case, the energy gain of a witness bunch is limited by the formula [13].

$$R = \frac{\Delta T^{\text{witness}}}{\Delta T^{\text{drive}}} \leq 2 \tag{1.3}$$

where T is the kinetic energy and R is the transformer ratio. Equation 1.3 states that a witness particle's energy can, at most, double with an electron driver.

Both of these schemes therefore require chaining multiple smaller accelerators together [9, 14] to produce 1TeV particles. Staging accelerators reduces the effective field gradient due to the drift between plasma cells. As in, an accclerator with plasma cells having an electric field gradient of 1GV/m, with one metre of drift for every metre of plasma, has an effective gradient of 0.5GV/m.

A more recent development has been the suggestion of using protons as the beam driver [15], overcoming the inherent setbacks found when using a laser or electron beam as a driver. This project focuses on this method of accelerating particles.

1.4 Proton Driven Plasma Wakefield Acceleration

This part of the text will describe the physics of PDPWA in detail. A schematic of an accelerating structure for PDPWA is shown in Figure 1.1. PDPWA works in much the same way as plasma wakefield acceleration, described in Section 1.3.1. The exception

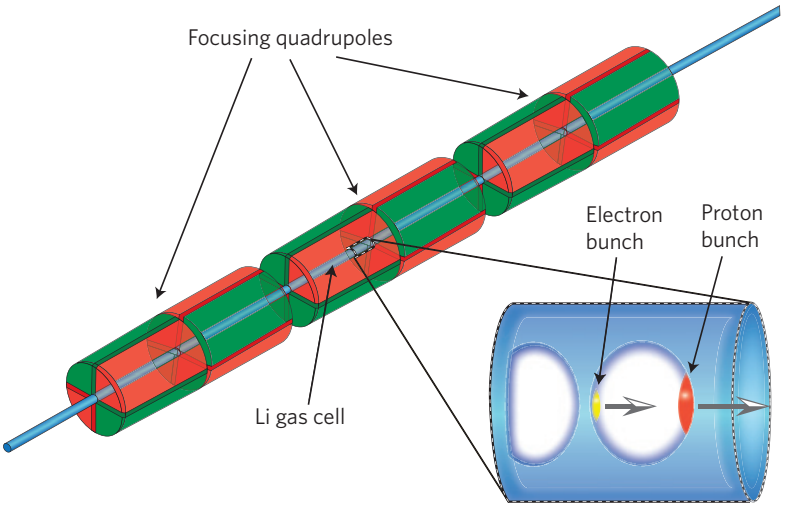


Figure 1.1: A diagram with an example of a PDPWA structure from [15]. A thin tube containing Li gas is surrounded by quadrupole magnets with alternating polarity. We see the plasma cavity from the proton driver in red. The electron bunch (yellow) being accelerated is located at the back of the bubble. Dimensions not to scale.

being that instead of the electrons being pushed out radially, then oscillating, the electrons are pulled towards the beam axis, then oscillate. The moving cavity structure of the wake is then π out of phase with the electron/laser case, or advanced by $\lambda_p/2$.

The driver beam makes the plasma electrons oscillate at frequency ω_p, given by

$$\omega_p = \sqrt{\frac{n_p e^2}{\epsilon_0 m_e}} \tag{1.4}$$

where n_p is the density of plasma electrons, ϵ_0 is the permittivity of free space, e is the electric charge and m_e is the mass of the electron.

By timing the injection of the witness bunch into a region of high electric field, electrons can be accelerated to high energies. There is also a radial force provided by the plasma which focuses the witness bunch, along with the tail of the drive bunch.

1.4.1 Length of driver beam

The maximum longitudinal electric field $E_{z,\max}$, admitted by a wakefield (in the linear regime[2]) is given by Equation 1.5 [16]:

$$E_{z,\max} = 2 \text{ GV/m} \cdot \left(\frac{N_b}{10^{10}}\right) \cdot \left(\frac{100\mu\text{m}}{\sigma_z}\right)^2 \tag{1.5}$$

where N_b is the number of protons in the bunch and σ_z is the length of the bunch.

[2]The linear regime is defined as $n_b \ll n_p$, with n_b the proton bunch density, and $eE/m_e\omega_p c \ll 1$.

What this tells us is that the acceleration gradient of a wakefield accelerator scales as $(1/\sigma_z)^2$, i.e. the shorter the bunch, the stronger the acceleration.

From Equation 1.5, we see that bunch lengths of $\sim$100μm are required for GV/m scale E-fields. Typical proton bunch lengths are on the order of $\sim$10cm. This clearly presents a problem for PDPWA, as 10cm gives a field gradient 10^6 times less than that of a 100μm bunch, which is currently unavailable. Fortuitously, there is a physical mechanism inherent to proton beam drivers, which can overcome this limitation, discussed in Section 1.4.5.

1.4.2 Momentum Spread of Bunch

The momentum spread of a moving proton beam causes the length of the bunch to increase over time. For the vacuum case, this spread is given by Equation 1.6 [15].

$$d = \left(\frac{\sigma_p}{p}\right)\frac{m_p^2c^4}{p^2c^2}L \tag{1.6}$$

where d is the spatial spread in the bunch and p, σ_p are the momentum and spread in the momentum.

For a 400GeV beam, a 10% momentum spread leads to lengthening of the bunch of $\sim$0.5μm per metre of travel. We can calculate what effect this has on the maximum sustainable E-field, using Equation 1.5.

$$\begin{aligned}
E_{\max} \propto \frac{1}{\sigma_z^2} &= \frac{C}{\sigma_z^2} && C \text{ constant} \\
\frac{\partial E_{\max}}{\partial \sigma_z} &= \frac{-2C}{\sigma_z^3} \\
\Delta E_{\max} &= \frac{\partial E_{\max}}{\partial \sigma_z}\Delta\sigma_z \\
\Delta E_{\max} &= \frac{2C}{\sigma_z^3}\Delta\sigma_z \\
\Delta E_{\max} &= \frac{2E}{\sigma_z}\Delta\sigma_z \\
\frac{\Delta E_{\max}}{E_{\max}} &= 2\frac{\Delta\sigma_z}{\sigma_z}
\end{aligned} \tag{1.7}$$

Taking the length of the proton bunch[3] to be 12cm, we get $\frac{\Delta\sigma_z}{\sigma_z} = \frac{0.5\mu\text{m}}{12\text{cm}} = 4.2 \times 10^{-6}$. So the change in the maximum admitted accelerating field changes by $\approx 0.001\%$ per

[3] Assuming SPS proton beam used in AWAKE.

metre of travel due to growth of the bunch. This is an insignificant change, meaning that PDPWA is relatively insensitive to the momentum spread of the driver beam.

1.4.3 Phase Slippage Between Driver and Witness Bunches

As the proton beam travels through the plasma, its high mass relative to the electron bunch means that over time the proton bunch slows down relative to the electrons. This phase change is given by Equation 1.8 [16].

$$\delta = \frac{\pi L}{\lambda_p}\left(\frac{m_p^2 c^4}{p_f p_i c^2}\right) \tag{1.8}$$

where L is the distance travelled, m_p is the proton mass, and p_i, p_f being initial and final momenta respectively.

For a phase change of 2π, taking $\lambda_p = 1$mm and a 400GeV beam[4] which loses half its energy, we get a dephasing length of $\delta \approx 50$m.

1.4.4 Interaction Between Protons in the Plasma

The mean free path of high-energy protons in the gas that makes up the plasma is on the order of kilometres [15], so it is not expected that protons interacting with each other, or with the plasma ions would present any serious effect inhibiting PDPWA.

1.4.5 Self-Modulation Instability

In order for the proton bunch to drive the plasma electrons to oscillate, the driver beam must contain a Fourier component close to the plasma frequency. From Equation 1.1, we see that plasma densities of at least $n_p \approx 10^{14}\,\mathrm{cm}^{-3}$ are required to reach accelerating gradients of GeV/m and above. The corresponding plasma wavelength is

$$\lambda_p = \frac{2\pi c}{\omega_p} \approx \sqrt{\frac{10^{15}}{n_p[\mathrm{cm}^{-3}]}}\ \mathrm{mm} \tag{1.9}$$

Proton beams produced today have a length σ_z = 3 - 12 cm, with a nearly Gaussian shape. The plasma wavelength at $n_p \approx 10^{14}$ cm^{-3} is $\sim$1 mm from Equation 1.9. Thus the proton bunch will not excite the plasma directly.

[4]Beam energy used in the SPS beam, for the AWAKE experiment, introduced in Section 1.5.

However, there is a mechanism which splits the proton into smaller micro-bunches [17]. The long ($L > \lambda_p$) proton bunch's wake also modulates the bunch itself, leading to an unstable modulation of the whole bunch along the beam axis. This self modulation splits the beam in micro-bunches of length λ_p, which resonantly excite the plasma wake. Its amplitude grows exponentially from the front to the back of the driver bunch and along the direction of propagation.

The self-modulation instability needs to grow from a seed perturbation. This is usually generated by a laser, co-propagating with the driver beam. The maximum longitudinal field of a maximally self-modulated proton beam has an inverse-square dependence on the width of a microbunch, rather than the length of the whole bunch, given by Equation 1.10 [18]:

$$E_{z,\max} \approx 0.12\,\mathrm{GV/m} \cdot \left(\frac{N_b}{10^{10}}\right) \cdot \left(\frac{100\mu\mathrm{m}}{\sigma_r}\right)^2 \tag{1.10}$$

For a 200μm wide beam of 10^{11} particles[5], Equation 1.10 gives a maximum supported field of 0.35GV/m.

1.4.6 Plasma Density Uniformity Condition

To accelerate an electron bunch in a proton wakefield, there is a restriction on the uniformity of the plasma [19]. Once the electron bunch has been injected into the proton wake, the maximum phase shift possible for the bunch to remain in the focusing/accelerating part of the wake is $\lambda_p/8$ [20]; we also have $\lambda_p \propto 1/\sqrt{n_p}$. therefore we can write the relation between the change in plasma wavelength and density as

$$\frac{\mathrm{d}\lambda_p}{\lambda_p} = -\frac{\mathrm{d}n_p}{2n_p} \tag{1.11}$$

The total allowed phase shift at the point of injection if the witness bunch is N wavelengths behind the starting point of the wakefield is

$$N \cdot \lambda_p \cdot \frac{\Delta n_{\max}}{2n_p} = \frac{\lambda_p}{8} \tag{1.12}$$

where $\Delta n_{\max}$ is the maximum allowed density perturbation. For realistic proton bunches this leads to

$$\frac{\Delta n_{\max}}{n_p} \leq 0.5\% \tag{1.13}$$

which needs to be maintained over many metres.

[5] Using AWAKE parameters.

To achieve this level of density uniformity, it is best to fill an evacuated chamber with neutral gas and ionise it with a laser. The laser pulse must be shorter than λ_p. If the laser pulse travels with the proton bunch, fast ionization creates the plasma in which the proton self-modulation can be reliably seeded.

1.4.7 Injection of Witness Bunch

Simulations have shown that on-axis injection of the witness bunch is feasible [21]. Out of the witness bunch, there are two sets of particles, trapped and untrapped. Whether or not a particle is trapped is determined by the particle's location in the initial wake-field. This separation occurs before the self-modulation has developed. An electron is trapped if it remains within $r < 3c/\omega_p$ after 1 metre of propagation in the plasma, where r is the particle's perpendicular distance from the beam axis. Figure 1.2 shows

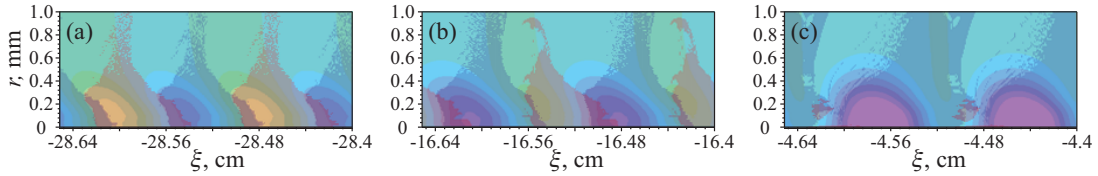


Figure 1.2: Acceptance of plasma wave for particles plotted over the potential map at three locations along the proton bunch. Yellow dots correspond to electrons. ξ is the distance from the driver bunch head [21]. Note that trapping of electrons only occurrs toward the tail of the proton beam.

the initial locations of trapped witness bunch particles, obtained from simulations, at different stages of the proton wake. Figure 1.2(a) shows that towards the tail of the proton wake, where the modulation instability has firmly established itself, the amount of trapped electrons is greatest. We define the trapping fraction as the ratio of trapped to untrapped electrons. Figure 1.3 shows the energy dependence of the trapping fraction at the location in the beam corresponding to Figure 1.2(a).

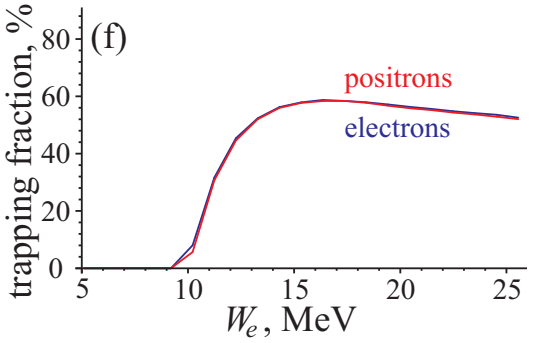


Figure 1.3: Dependence of trapping fraction on energy of witness particles [21].

Having discussed the physics behind PDPWA, we now move on to an introduction to AWAKE, the experiment which will set out to test the idea.

1.5 The AWAKE Experiment

The AWAKE experiment [22] at CERN will be the first ever physical realisation of PDPWA. Figure 1.4 shows the basic layout of the AWAKE experiment. The proton

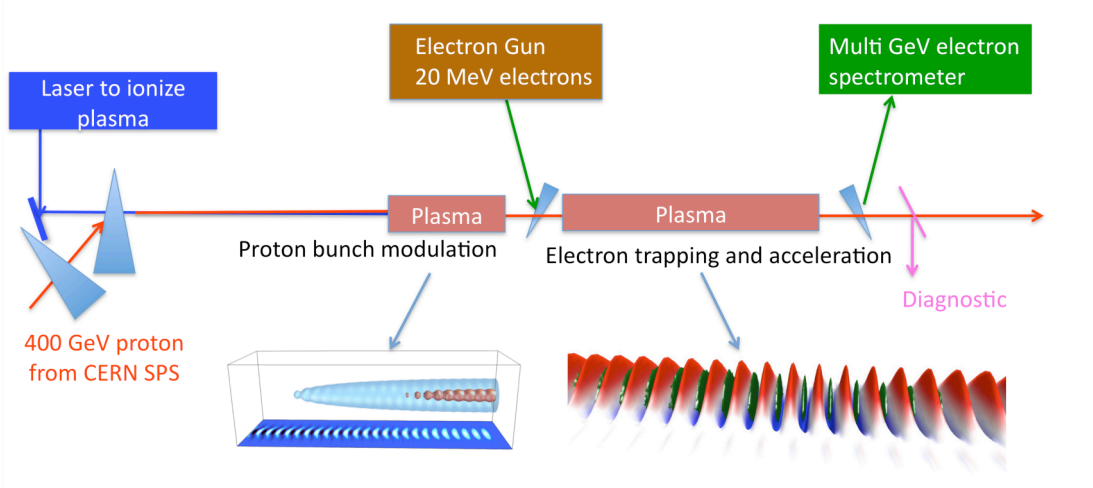


Figure 1.4: Overall design of the AWAKE experiment, showing major sections and effects [22].

driver is taken from the 400 GeV SPS beam. Typical bunches contain 3×10^{11} protons, with the bunches being length 12 cm. Bunches will be extracted every 30s from the SPS and sent along >800m of beam pipe towards the AWAKE experimental area, upstream of the former CNGS facility. The focused transverse width of the driver at the plasma entrance will be $\sigma_r = 0.2$ mm and the transverse normalised emittance[6] is $\epsilon_N = 3.5$mm mrad. The laser beam is combined with the proton beam 20 m upstream of the plasma entrance. Beam diagnostics and an electron spectrometer lie downstream of the plasma cell. The proton beam continues on to the existing CNGS hadron stop ~1 km downstream of the spectrometer, to avoid backscattering of radiation and particles into the experimental area.

The plasma source which best fulfils the requirements on the density uniformity of the plasma is an alkali metal vapour source. They have been used in previous experiments [23] and have low ionization potentials (e.g. 4.2 eV for the first electron of rubidium). The vapor ionizes with a laser threshold intensity of 1.7×10^{12} W/cm^2. In

[6] Normalised emittance is a measure of the spread of particle co-ordinates of a beam of particles in position and momentum phase-space; small emittances are better for the experiment.

the AWAKE experiment, a rubidium vapor will be used. Its large ion mass means that the plasma is less sensitive to ion motion [24].

The witness beam will be made of 1.5×10^9 electrons of energy 16 MeV, with bunch length 1.2 mm. The beam will be injected 13cm after the proton beam, for optimal acceleration, and has a normalised emittance of 2mm mrad.

The first protons are expected at the end of 2016, followed by a program of four two-week periods of data acquistion over the subsequent 3-4 years.

1.6 Project Outline

1.6.1 Project Aim

The main aim of this project is to aid in the preparation of the AWAKE experiment, by simulating the interactions of the proton bunch and plasma. This will be done to firmly establish the parameters to use and change for the experimental setup, alongside learning more about the physics inside the plasma cell.

1.6.2 Project Work

The main task of the project will be to adapt the existing simulation infrastructure to accommodate multiple plasma cells. The experimental setup to be simulated is shown in Figure 1.5. The idea behind using two stages for acceleration is that the first plasma

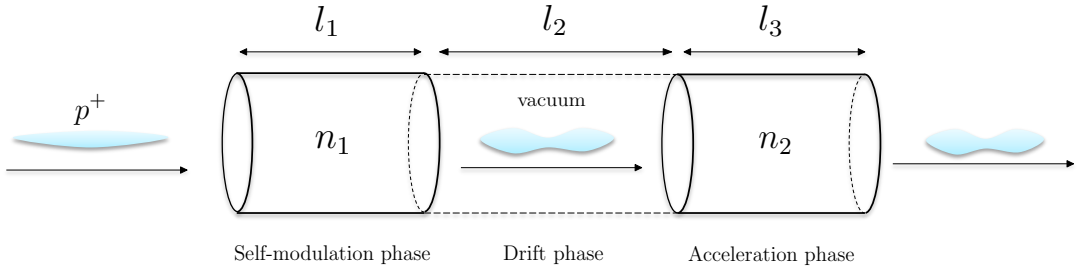


Figure 1.5: Schematic of the project simulation setup. The SPS proton beam enters the first plasma cell, where it self-modulates, then into an evacuated chamber. Then, the beam enters the second plasma cell, where it can then accelerate witness particles. n_1 and n_2 are the plasma densities of the two cells with the $\{l_i\}$ being the lengths of the different stages.

stage can have characteristics amenable to stable proton beam self-modulation, with the second stage having characteristics more suited to acceleration of witness particles. Namely, well-defined microbunches of equal length, separated by λ_p. This has the benefit

of making the stringent plasma-uniformity condition (see Section 1.4.6) more achievable experimentally.

Figure 1.5 indicates tuneable parameters (n_1, n_2 and $\{l_i\}$) that could be adjusted to achieve optimum acceleration. This would involve investigating the beam modulation and acceleration characteristics, along with the beam's propagation in vacuum, as a function of these and other parameters. The AWAKE experiment will seek to investigate novel plasma sources [25] and these could be incorporated into the analysis also.

There is also the potential of extending this scheme to N plasma cells, which could speculatively inform the design of a future, larger accelerator.

Chapter 2

Project Work

2.1 Introduction

This section of the report describes the work has that been carried out over the duration of the project. Section 2.2 gives a short overview of the simulation structure used by the AWAKE collaboration at the start of the project. Section 2.3 discusses tests of this code set against established results in the literature. Section 2.4 describes how a vacuum was introduced into the simulation structure. Section 2.5 describes integrating a second plasma into the simulation.

2.2 Existing Simulation Infrastructure

2.2.1 LCODE/lSim

For simulating the plasma in the experiment, a vital piece of software used by the AWAKE team is LCODE [26]. LCODE simulates particle-beam interactons within a plasma cell. This particle-in-cell (PIC) code operates by moving the window of simulation with the proton bunch, and recording data at discrete timesteps, simulating the dynamics in the plasma cell. The geometry of a typical simulation is shown in Figure 2.1. The code tracks large numbers ($\leq$40000) of plasma particles, electrons, and the beam self-modulation.

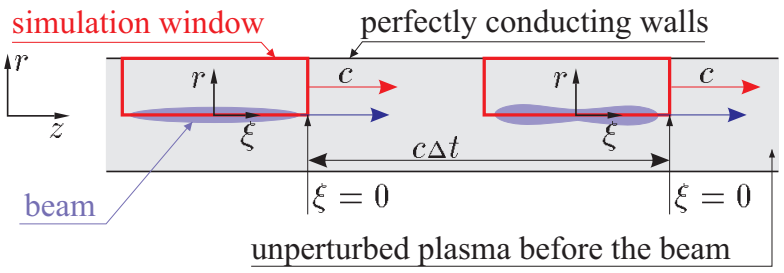


Figure 2.1: Schematic of the geometry of an LCODE simulation [27].

The code has a large number of adjustable parameters and geometries, and there exists the possibility of adding the capability of simulating a comoving laser pulse with the proton beam.

S. Mandry of the AWAKE collaboration has developed a software suite, which expands on the capabilities of LCODE, provisionally called lSim. This includes but is not limited to integrating a plotting and analysis environment with the code, to parse through the large amounts of data (∼500 GB per simulation run) and get to the physics of the accelerator quicker. LCODE uses exclusively plasma units for input, calculations and output, for speed of computatation. However, for interpreting results, these units are fairly unintuitive, so lSim is designed for the user to input simulation parameters in S.I. units, and for the output to be in S.I. units.

As the code set is still work-in-progress, a large part of early project work was to debug certain areas of the code and making the code set more robust, in addition to extending the scope of the simulations later on. Due to the nature of how the code is organised and its size, when something went wrong and an error was returned, often the source of the error was unclear. Efforts have been made to remedy this.

2.2.2 Collective Plasma Behaviour

Before describing the work using lSim, it is worth justifying why for simulating proton beam self-modulation, simulations with such large quantities of data, taking $\sim\frac{1}{2}$ day/metre of plasma are required.

Consider the effect of two slightly charged regions of plasma separated by a distance r, (See Figure 2.2). The Coulomb force between A and B diminishes as $1/r^2$. However, for a given solid angle, i.e. $\Delta r/r$ = constant, the volume of plasma in B that exerts an influence on A increases as r^3. So elements of plasma in one region affect other regions at large distances.

It is this long range, collective behaviour of plasmas that necessitates numerical studies over analytical methods, to study beam-plasma interactions to the required level of detail.

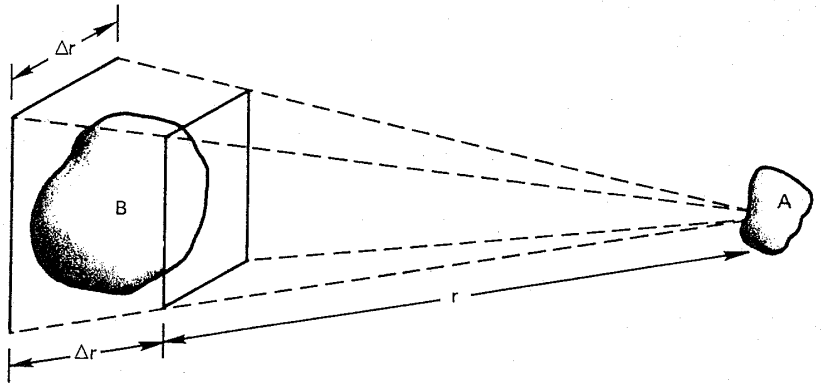


Figure 2.2: Illustration of how one region of plasma affects another distant region of plasma [28].

2.3 One Plasma Cell Simulations

2.3.1 Motivation

Before working on the code for a multiple plasma simulation and producing results, it was important to check that the one plasma cell setup was working correctly. This involved running simulations of setups using PDPWA in the published literature using the same simulation parameters and comparing.

The PDPWA setups simulated were those of a 1TeV proton beam propagating in a plasma made of Li vapour [15] and the 400GeV SPS beam in Rb vapour used in AWAKE [29].

2.3.2 PDPWA Proposal Paper

In [15], a hypothetical proton beam of length 100μm is simulated. This paper was published before the idea of self-modulation was invented, and thus relies on the assumption that bunch compression schemes can be made sufficiently powerful in the future to facilitate the thousand-fold decrease in bunch length required. As this beam was not self-modulating, the region immediately behind the bunch-head ($\xi_b < 5$cm) was observed, as in the paper.

The simulation parameters used are given in Table 2.1. A Gaussian beam profile was used, as in the paper.

Parameter	Symbol	Value	Units
Protons in drive bunch	N_b	10^{11}	
Proton energy	E_p	1	TeV
Initial proton momentum spread	σ_p/p	0.1	
Initial proton bunch longitudinal size	σ_z	100	μm
Initial proton bunch angular spread	σ_θ	0.03	mrad
Initial proton bunch transverse size	σ_r	0.43	mm
Free electron density	n_p	6×10^{20}	m^{-3}
Plasma wavelength	λ_p	1.35	mm
Atomic weight of plasma ions	M_i	6.941	m_p
Length of Plasma	l_1	4	m

Table 2.1: Parameters used in the simulation, for comparison with [15].

Accelerating fields of 2 GV/m are reported after 4m of plasma. The simulation gave accelerating fields of 1 GV/m. An explanation for this discrepancy is that the simulations in the paper include the plasma being embedded in a quadrupole field, to focus the proton bunch. This is to combat dispersive effects. With this field, the bunch density is higher and thus gives a stronger accelerating field. At the moment, the code set doesn't have the functionality to input an external field so this field was not simulated.

2.3.3 AWAKE Baseline

The next one-plasma simulation tested was the AWAKE baseline, the parameters of which are given in Table 2.2. The beam was simulated using a half-cut Pseudo-Gaussian[1] profile.

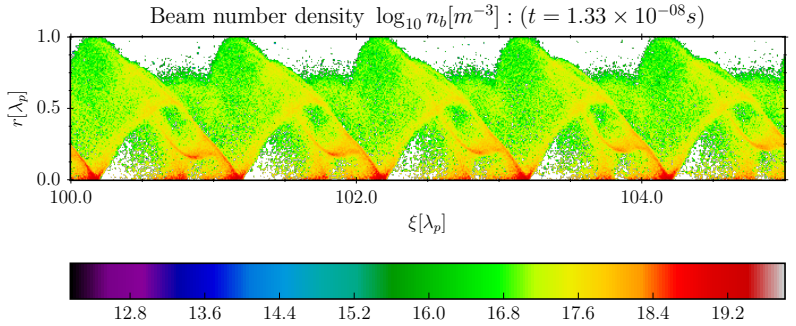
This beam profile is used, as for the self-modulation to be seeded, there needs to be high frequency (namely ω_p) harmonics in the beam profile to excite the plasma electron oscillations at ω_p. The discontinuity introduced by cutting off the front half of the beam generates these harmonics, thereby seeding the modulation. Physically, this beam-shape can be generated using a laser, co-propagating with the centre of the beam [24].

For the AWAKE baseline, the proton bunch evolution was simulated over 4m of plasma. 4m was chosen as after 4m is where phase slippage between the bunch-head and the wakefield stops, and the fields stabilise. The maximum accelerating field was found to be 0.7 GV/m. In the AWAKE results reported in [25], the E-field is reported as 1GV/m. the discrepancy in these values could potentially be due to numerical noise, which can add 20-30% to the maximum field observed in simulations [30]. Figure 2.3a shows a plot of the bunch density after 4m of travel in the plasma. It shows self-modulation and evolution of the wakefield, as expected. Figures 2.3b, 2.3c show the radial and

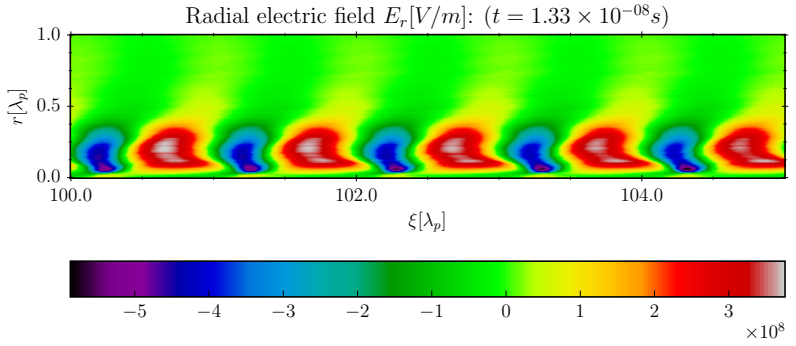
[1] A Gaussian distribution with a zero cutoff, so as to not introduce bugs in the simulation, resulting from the non-zero values at the tails of the distribution.

Parameter	Symbol	Value	Units
Protons in drive bunch	N_b	3×10^{11}	
Proton energy	E_p	400	GeV
Initial proton momentum spread	σ_p/p	0.0035	
Initial proton bunch longitudinal size	σ_z	12	cm
Initial proton bunch angular spread	σ_θ	0.045	mrad
Initial proton bunch transverse size	σ_r	0.02	cm
Free electron density	n_p	7×10^{20}	m^{-3}
Plasma wavelength	λ_p	1.458	mm
Atomic weight of plasma ions	M_i	85.4678	m_p
Length of Plasma	l_1	4	m

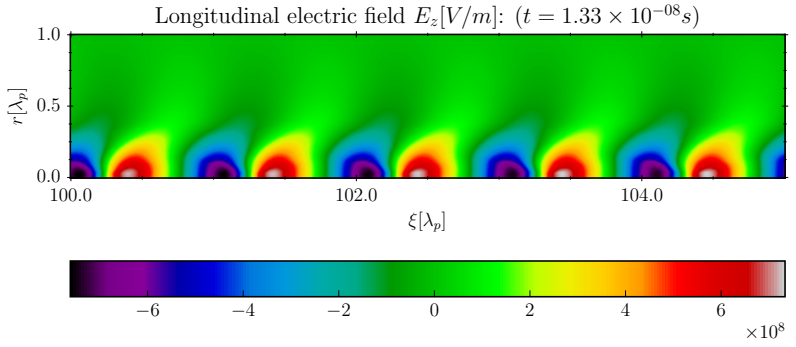
Table 2.2: Parameters used in the simulation, for the AWAKE baseline [29].



(a) Plot of the proton bunch density, note that the density is on a log scale.



(b) Plot of the radial electric field.



(c) Plot of the longitudinal electric field.

Figure 2.3: Beam density and wakefield plots after 4m of travel in the plasma. λ_p = 1.26mm, so the plot area is 12.6cm - 13.2cm behind the bunch head. The beam is travelling to the left.

longitudinal electric fields respectively. Note the cavity structure. The skewness of the fields, especially in the radial case, is a consequence of phase slippage, i.e. the bunch (so therefore the simulation window) moving faster than the the wake. The plot area is taken at 100-105λ_p behind the bunch-head, the reasons for this are discussed in Section 3.1.2.

With the functionality of the one-plasma cell code set established, the next step was to simulate the vacuum.

2.4 Vacuum Simulations

2.4.1 Simulating the Vacuum

In order to simulate the drift space, it was necessary to use different software to LCODE, as LCODE requires large plasma densities to run. It was decided that the code BDSIM [31], was to be used, which simulates particle transport through beamlines, and the relevant physics, such as ionisation, multiple scattering etc.

2.4.2 Beam Data Conversions

At the end of every run in LCODE, data for each macro-particle[2] is printed to an output file as a collection of floating point numbers, converted to binary, in order to save space. The relevant variables that are stored are the particle's positions and momenta. The momenta are given as a longitudinal and transverse component, plus the angular momentum around the beam axis. The particle positions are given as a distance from the axis, and from the bunch head, but with no polar angle, as the simulation is cylindrically symmetric. These quantities are all given in plasma units.

The input data for BDSIM is a specification of each particle of interest's position in space, relative to the bunch-head, its 3-momentum components and its energy. The space and momentum components are specified in terms of Cartesian co-ordinates.

LCODE and BDSIM specify beam data in different ways, so effective conversion between the two formats was needed for reliable physics.

2.4.2.1 LCODE ⇒ BDSIM

For the conversion from LCODE to BDSIM, the first task was to make the association between the macro-particles, and individual protons. The macro-particles are a ring of

[2]Cylindrically symmetric "ring" of protons.

charge, but since each ring contains the same amount of charge and mass, each macro-particle[3] was put into BDSIM as one proton. Cylindrical symmetry and the sheer number of macroparticles meant that the beam dynamics wouldn't change significantly under this approximation. What would change is the density, by the factor of number of protons/macro-particle, and the E-fields made/felt by the protons as a result. However, these changes are insignificant in beam-vacuum dynamics[4] due to the relativistic energies of the protons, as discussed in Section 4.1.4.1.

Having made and justified the mapping between macro-particles and protons, the protons needed to be assigned transverse positions, as LCODE only specifies the distance from the axis and from the bunch-head, due to its cylindrical symmetry. In the standard accelerator co-ordinate system, where the z-axis is down the beamline, and the x, y axes are two mutually orthogonal axes, orthogonal[5] to z, this meant assigning x and y co-ordinates to the particles. An angle, θ, was generated such that $\theta = 2\pi \times \text{Uni}\{[0, 1)\}$.

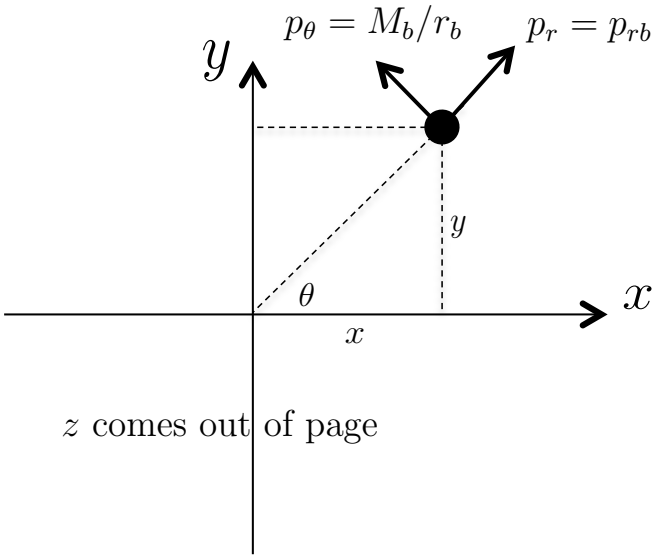


Figure 2.4: LCODE to BDSIM variable transformations. x, y, p_x and p_y are BDSIM parameters. r_b, p_{rb} and M_b are LCODE parameters. p_θ is tangential momentum, p_r is radial momentum.

Due to such large numbers of particles, it was important that the uniformly distributed numbers generated were as perfectly random as possible, so as not to introduce correlations in the beam data that weren't truly there. A good trade-off between speed and quality was using the Mersenne Twister algorithm [32], which has a period of $2^{19937} - 1$. This θ was then converted to x and y coordinates via the standard transformations:

$$x = r_b \cos\theta, \;\; y = r_b \sin\theta \tag{2.1}$$

[3] A macro-particle corresponds to 25000 protons in this simulation.
[4] As opposed to beam-plasma dynamics.
[5] where $\{x, y, z\}$ form a right-handed set.

where r_b is the distance from the axis. The momenta were transformed using

$$p_x = p_r \cos\theta - p_\theta \sin\theta, \;\; p_y = p_r \sin\theta + p_\theta \cos\theta \tag{2.2}$$

where p_r and p_θ are the radial and tangential momenta, as given by LCODE and p_x, p_y are Cartesian components of momentum. Figure 2.4 illustrates these transformations between the variables.

2.4.2.2 BDSIM ⇒ LCODE

For the conversion from BDSIM to LCODE, each proton was taken to be one macro-particle, with the macro-particle specific data from LCODE stored from the LCODE ⇒ BDSIM conversion and reappended. The data was then sorted by ξ_b and r_b for compatibility reasons, and printed to a binary file for use by LCODE.

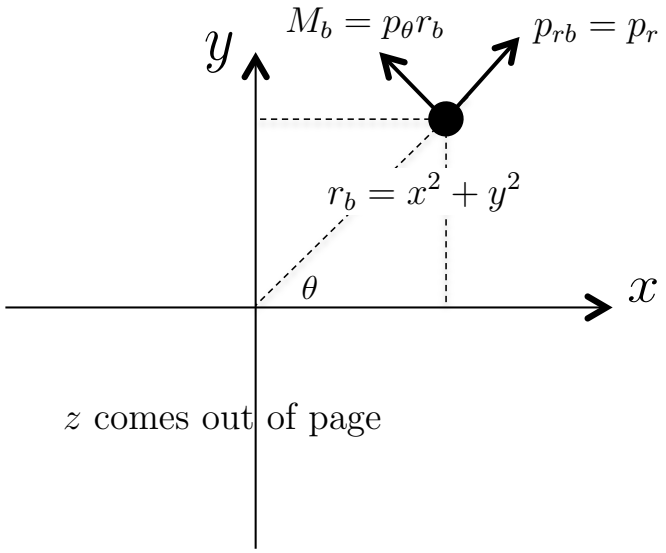


Figure 2.5: BDSIM to LCODE variable transformations. x, y, p_x and p_y are BDSIM parameters. r_b, p_{rb} and M_b are LCODE parameters. p_θ is tangential momentum, p_r is radial momentum.

The relevant transformations are illustrated in Figure 2.5. We have $\theta = \tan^{-1}\left(\frac{y}{x}\right)$, r_b as in Figure 2.5. The momenta transformations went as

$$p_r = p_x \cos\theta + p_y \sin\theta, \;\; p_\theta = -p_x \sin\theta + p_y \cos\theta \tag{2.3}$$

2.4.3 Proton Beam Study

When BDSIM is ran, along with the output, it prints (in a similar format) a "primaries" file. This gives the beam data at the start of the run, i.e. just after the beam exits the

plasma. It was deemed to be useful to study this data to see how the self modulation had manifested itself in detail. The shape of the beam in phase space influences how focusing can be applied.

Beam data for x, y, z, p_x, p_y and E were binned and plotted as histograms, for various z ranges behind the beam, with the parameters used in Section 2.3.3, i.e. the AWAKE baseline, after 4m of plasma. The results of this are shown in Section 3.1. The histograms were least squares fitted to Gaussian and Cauchy distributions, along with linear combinations of these distributions, to see if the beam data could be approximated to these well-known distributions. The goodness of fit was tested using the χ^2 test statistic, with an associated p-value calculated from the χ^2 distribution. The χ^2 test statistic is given by Equation 2.4 [33]. The code for carrying out this analysis is given in Appendix B.3.

$$\chi^2_{\text{test}} = \sum_i^{n_{\text{bins}}} \frac{(\text{observed counts}_i - \text{predicted counts}_i)^2}{\text{predicted counts}_i} \tag{2.4}$$

where the predicted counts were given by the difference between the cumulative distribution functions at the bin edges, normalised to the total number of counts. The p-value for the fits was taken as

$$1 - F_{\chi^2}(\chi^2, \nu) \tag{2.5}$$

where F_{χ^2} is the cumulative distribution function for the χ^2 distribution with ν as the number of degrees of freedom. ν was taken as the number of bins, n_{bins}, minus the number of fitted parameters.

Also, 2D histograms were plotted between these sets of data, to see how they are correlated together. These plots are shown in Section 3.1. The Pearson sample correlation coefficient, ρ, for these data sets was calculated using Equation 2.6 [33]

$$\rho_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{2.6}$$

where the bar denotes the mean of the data set and n is the number of data points.

2.4.4 Vacuum Runs

BDSIM allows the specification of numerous beamline parameters. These include the vacuum pressure, with the default composition being 48.2% H, 22.1%C and 29.7% O, at a temperature of 300K. Parameters such as the beam pipe radius, material and surrounding tunnel material, along with the placement and strength of quadrupole focusing magnets can also be fully specified.

Unfortunately, due to time constraints, these options were not explored fully. It was decided that running simulations of the beam entering the second plasma would take precedence. This resulted in the vacuum simulation being run over 1m, to provide space for the eventual addition of the electron, while preventing adverse effects of beam dispersion (in the vacuum region the beam would not be self-focusing as it wasn't in plasma). The vacuum was taken to be effectively perfect (10^{-20} bar) with only beam transportation enabled in BDSIM.

Vacuum simulations were ran for an AWAKE baseline parameter set after 4m of plasma.

2.5 Two Plasma Cell Simulations

Once the beam was propagated through the vacuum, the beam data was converted back to the format readable by LCODE, as discussed in Section 2.4.2.2. The beam was sent through an additional 4m of plasma, at the same plasma density, so that comparisons could be made between the wakefields of pre and post-vacuum beams.

A parameter scan of different plasma densities would have been beneficial to produce. However, with computing resources limited and every 4m run taking a week to complete, this was not acheivable within the time-frame of the project.

A more detailed presentation of how the simulations were run is given in Appendix A, with choice pieces of code given in Appendix B.

The results obtained from the work described in this Chapter are now presented in Chapter 3.

Chapter 3

Results

This part of the report outlines the main results obtained in the project. Section 3.1 looks at the AWAKE modulated beam after 4m of plasma, Section 3.2 concerns the changes brought about by the 1m drift stage, and Section 3.3 shows how the second plasma cell affected the beam and wakefield.

3.1 Modulated Proton Beam Study

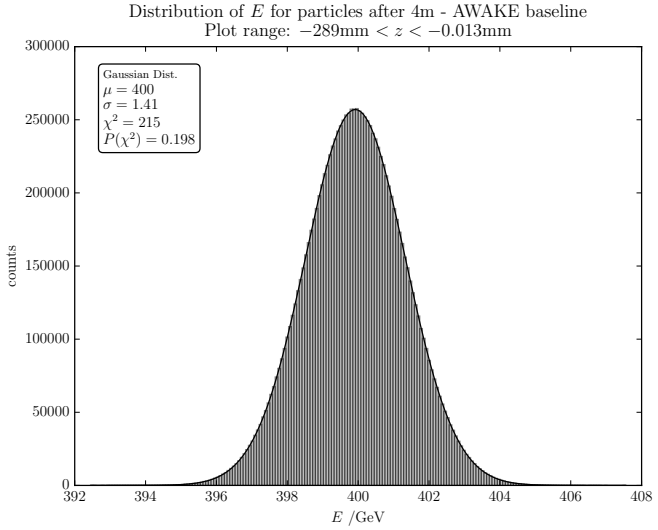
This section presents the data for the AWAKE baseline after 4m of plasma, analysed as described in Section 2.4.3.

3.1.1 Whole Beam

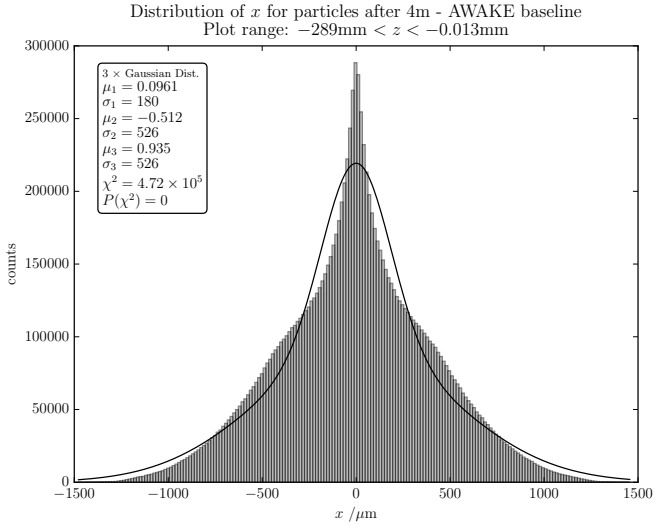
Figure 3.1 shows histograms of properties of the beam particles for the AWAKE baseline, with 200 bins, covering the whole length of the beam simulation window, 30cm. Each count represents one particle in a particular bin range.

Figure 3.1a shows the energies of the particles, with a Gaussian fit. The beam energy distribution started before the plasma as a Gaussian with $\mu = 400\text{GeV}$, $\sigma = 1.4\text{GeV}$, and largely remained distributed as such, as shown by the p-value of 0.198. Figure 3.1b shows the distribution in one transverse direction, x, of the particles[1]. An attempt was made to fit three Gaussians to this curve, but looking at the figure, we see it is a bad fit, the p-value of 0 confirming this. Looking at the figure, qualitatively we see a sharp peak in the middle, corresponding to the focused, bunched particles driving the wakefield. The large bulge represents particles which are being ejected from the beam.

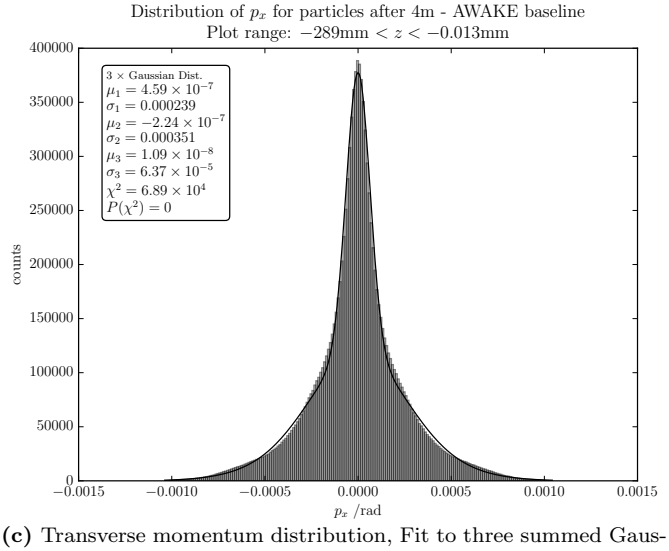
[1]The y direction plot is essentially the same, and is omitted for brevity



(a) Energy distribution, Gaussian Fit.



(b) Transverse position distribution, Fit to three summed Gaussians.



(c) Transverse momentum distribution, Fit to three summed Gaussians.

Figure 3.1: Histograms for beam variables plotted over the whole beam.

Figure 3.1c shows the distribution[2] of p_x, with a three Gaussian fitted curve. By eye, the fit appears to be better than in the spatial co-ordinate case, but closer inspection of the figure and the p-value show that the p_x distribution looks much like the x distribution, but a lot more tightly distributed. This makes sense as the beam is travelling at relativistic speeds in the z-direction, so transverse components of momenta will be small, along with the effect of the self-focusing of the beam in the plasma.

2D histograms were also made, to see how the different variables correlated with one another. Figure 3.2 shows these histograms, with 200 bins in each axis. We see in Figures 3.2a and 3.2b how x and p_x vary along the length of the beam. In the x case, particles have been ejected radially, but the density is still highest on the beam axis. For the momentum, p_x, there is a similar shape of distribution but far less spread around the mean value of 0.

Figures showing histograms between x, y and p_x, p_y are not included for brevity, and as they were generated in the beam conversion as independent variables, there is no correlation between them. Figures 3.2c and 3.2d show how the momenta and position variables are correlated with one another.

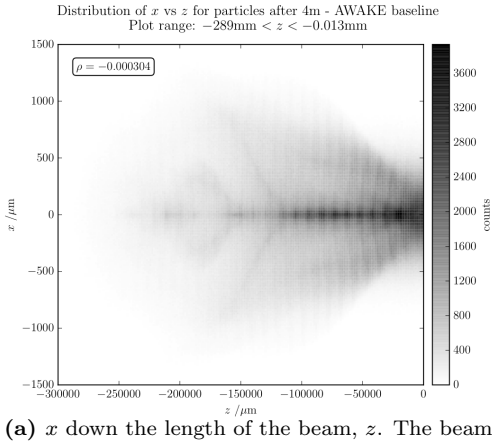
We see in the p_x vs x case that they are strongly correlated ($\rho = 0.721$), and there appear to be three distributions between them, from the three "arms" in the plot. This was the motivation for trying to fit three Gaussians to the histograms in Figure 3.1.

For p_y vs x, we see that while the variables aren't correlated in a linear way, there is a non-linear relation between p_y and x. This is seen from the horizontal and vertical arms.

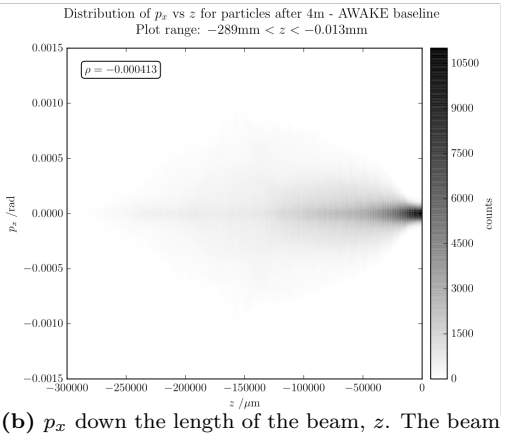
It is unfortunate that these plots show that the beam is not distributed in a familar way, such as a Gaussian, or sum of Gaussians[3]. However, this is to be expected as the interaction between the beam and plasma is highly complex. This prevented a straightforward implementation of quadrupole focusing, so this work was put on hold, in order to continue on to implementing the second plasma cell.

It was also considered that the beam profile followed familar distributions, but at different points along the beam. From Figures 3.2a and 3.2b we see different distributions in the vertical direction along the beam at different z co-ordinates. However, it was found that the beam variables' distributions were qualitatively distributed in the same way as shown in Figures 3.1 and 3.2, for 2mm sections taken at 5cm intervals along the plasma.

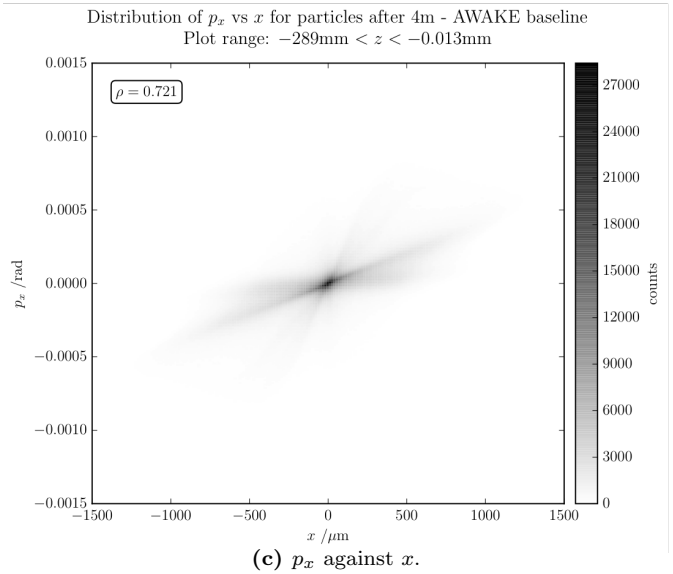
[2]Similarly to the space co-ordinates, the p_x and p_y momentum distributions are the same.
[3]Cauchy distributions were also combined for the fit as well, but returned nothing of note.



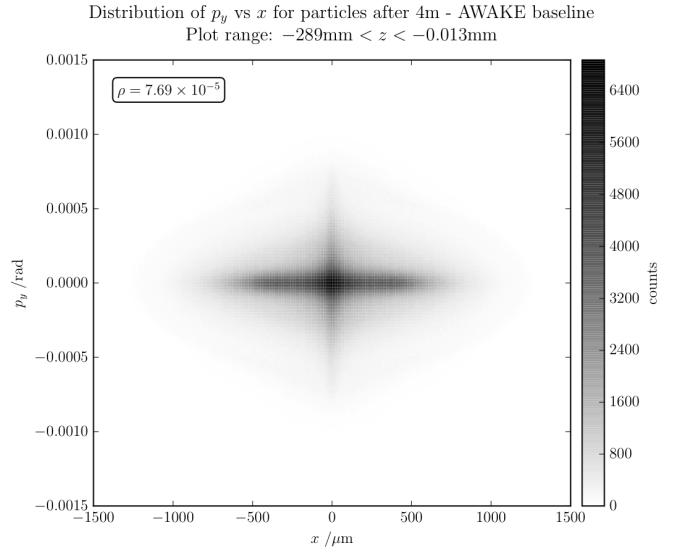
(a) x down the length of the beam, z. The beam is travelling to the right.



(b) p_x down the length of the beam, z. The beam is travelling to the right.



(c) p_x against x.



(d) p_y against x.

Figure 3.2: 2D Histograms for beam variables plotted over the whole beam. ρ is the calculated Pearson correlation coefficient.

3.1.2 Injection Point

In Section 2.3.3, and in all of the work described afterwards, the part of the beam of interest is taken as at 100-105λ_p. this is the part of the beam where the wakefield is most coherent. We can see why this is from Figure 3.3. We see in the figure the beam profile at various points along the beam. Figures 3.3a and 3.3d show that at the start and end of the beam, there is no microbunching, hence no appropriate accelerating wakefield. Figure 3.3b, 5cm behind the bunch-head, shows some structure, but the microbunches are wide relative to a plasma wavelength, so wouldn't resonantly excite a strong wakefield. Figure 3.3c, 12cm behind the bunch-head, has small microbunches relative to λ_p, but they are not equidistant, so wouldn't excite a strong wakefield.

In Figure 3.3e, we see the beam profile close to the proposed injection point, at $\sim$100λ_p. The microbunches are both small and spaced apart by a plasma wavelength, which would excite strong wakefields, thereby making this part of the beam amenable to bunch acceleration.

3.2 Vacuum Propagation

After the beam had travelled through 1m of vacuum drift, the change in width of the beam was tracked, with the code for this analysis shown in Appendix B.4. Errors were calculated using standard formulae from [33].

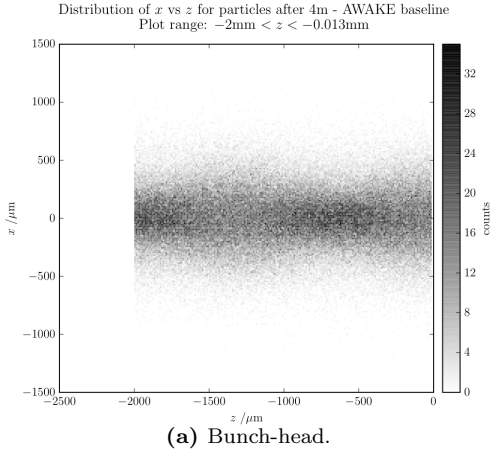
It was found that over the whole beam, the average transverse width of the beam grew from $486.72 \pm 0.08\mu$m to $755.3 \pm 0.1\mu$m, where the uncertainty is the standard error on the mean[4].

For the area around the injection point, -13.2cm$< z < -$13cm, the beam width grew from $638 \pm 1\mu$m to $1094 \pm 2\mu$m. This represents a 1.713 ± 0.004-fold increase in beam width in this region. From Equation 1.10, the maximum longitudinal field for a self-modulated beam is proportional to $1/\sigma_r^2$. Thus, naïvely we would expect the electric field in this region would reduce from 0.7GV/m$\rightarrow$ 0.239 ± 0.001GV/m in the second plasma.

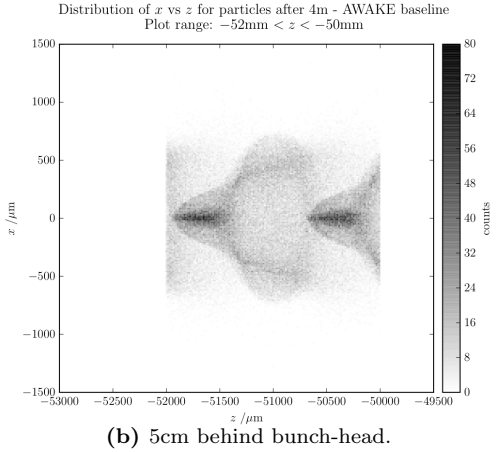
3.3 Two Plasma Cells

After the 1m drift space, the modulated proton beam was sent into 4m of a second plasma, with the same density as the first, $n_b = 7 \times 10^{20}\text{m}^{-3}$.

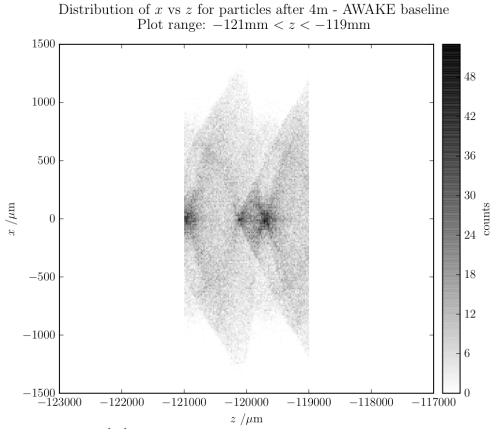
[4]Defined as: $\frac{\sigma}{\sqrt{N}}$, where σ is the standard deviation.



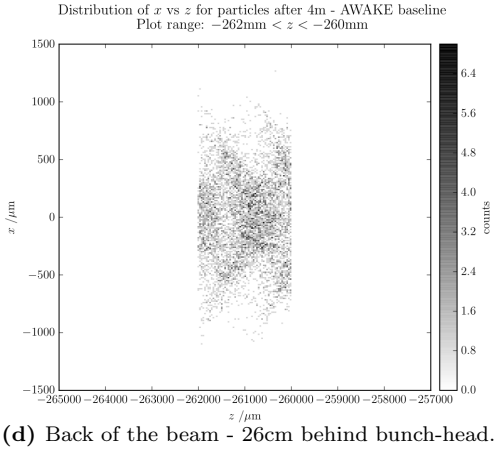
(a) Bunch-head.



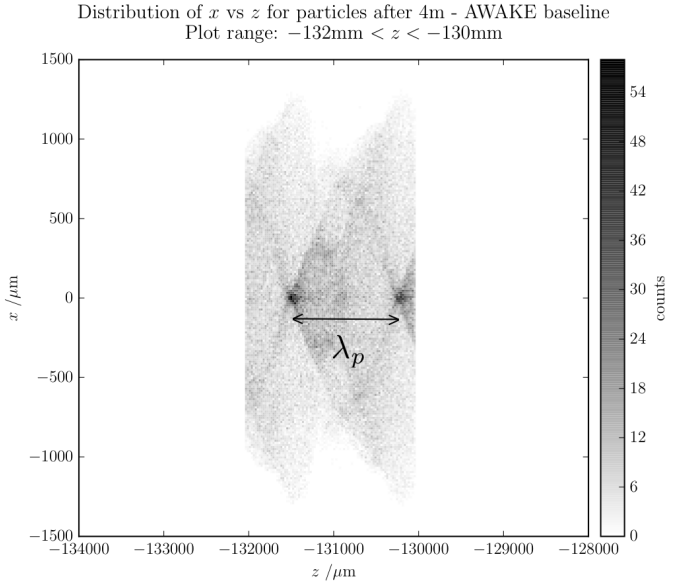
(b) 5cm behind bunch-head.



(c) 12cm behind bunch-head.



(d) Back of the beam - 26cm behind bunch-head.

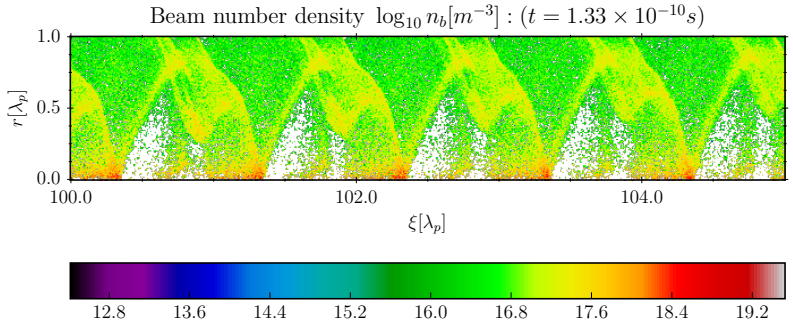


(e) Injection point - 13cm behind bunch-head

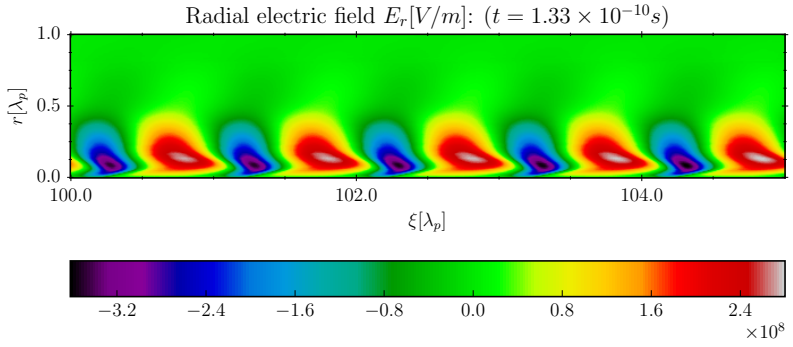
Figure 3.3: Plots of x against z across the proton beam, covering 2mm of beam. We see in (e) the most coherent, tightly packed microbunches, separated by λ_p, ideal for exciting a wakefield. The beam is travelling to the right in these plots.

3.3.1 Start of Plasma Cell

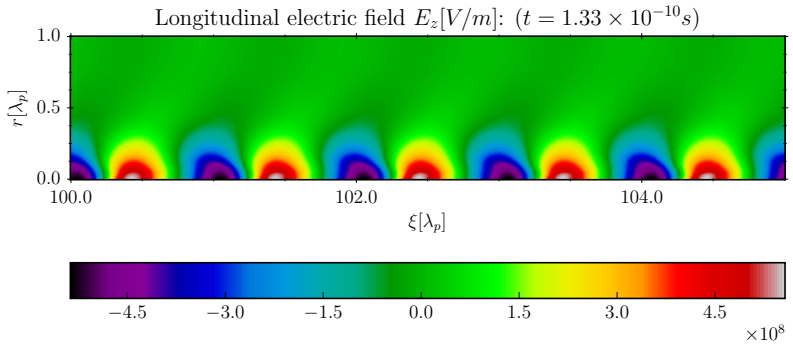
We see in Figure 3.4 the density and electric fields after the beam enters the plasma.



(a) Plot of the proton bunch density, note that the density is on a log scale. The white regions of the plot correspond to a density of ≈ 0.



(b) Plot of radial electric field.

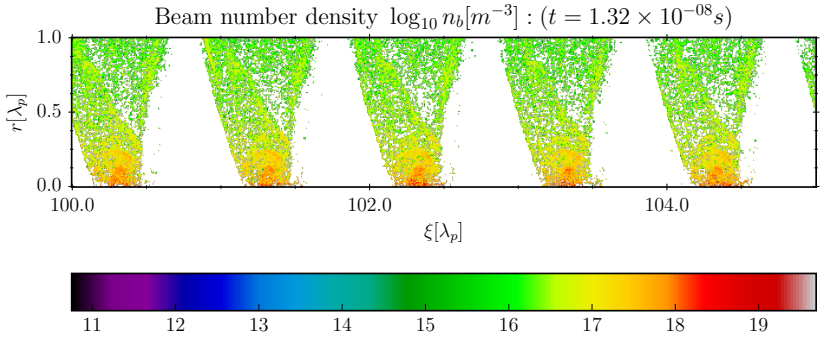


(c) Plot of longitudinal electric field.

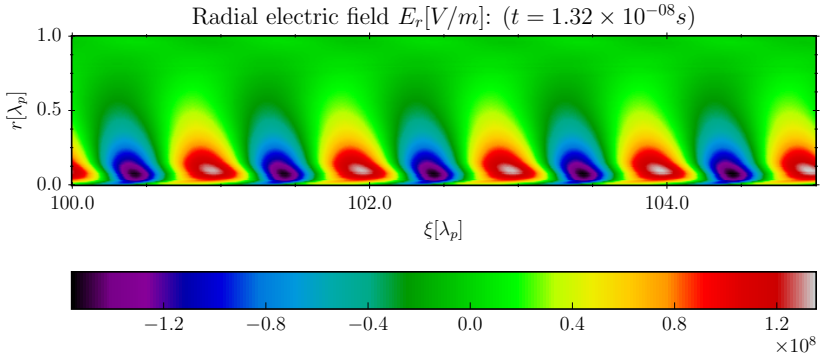
Figure 3.4: Beam density and wakefield plots at the start of the second plasma cell. $\lambda_p = 1.26$mm, so the plot area is 12.6cm - 13.2cm behind the bunch head. The beam is travelling to the left.

From the density plot in Figure 3.4a densities, the microbunches are less dense than before the beam entered the vacuum[5]. The density of the microbunches decreased by a factor of 10 from $\sim 10^{19}\text{m}^{-3}$ to $\sim 10^{18}\text{m}^{-3}$. Also, the secondary microbunches seen in the first plasma cell were ejected, so only microbunches on the axis, spaced apart at λ_p were left.

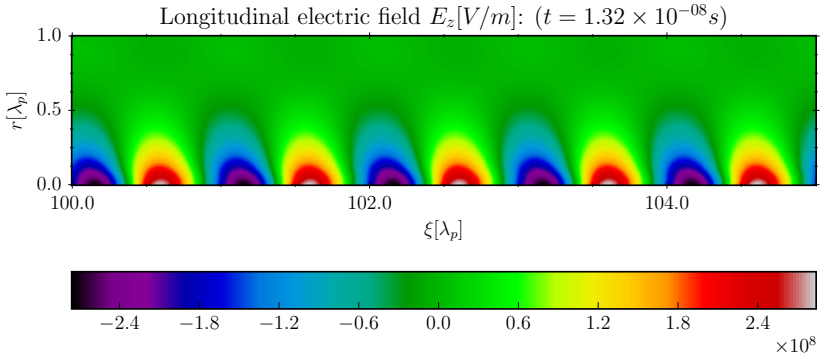
[5] see Figure 2.3a.



(a) Plot of the proton bunch density, note that the density is on a log scale. The white regions of the plot correspond to a density of ≈ 0.



(b) Plot of radial electric field.



(c) Plot of longitudinal electric field.

Figure 3.5: Beam density and wakefield plots after 4m of travel in the second plasma cell. $\lambda_p = 1.26$mm, so the plot area is 12.6cm - 13.2cm behind the bunch head. The beam is travelling to the left.

The radial electric fields, comparing Figure 3.4b to Figure 2.3b), decreased from 0.4GV/m to 0.25GV/m, but the size of the beam focusing region remained the same (the red region of the plots, i.e. positive E_r.).

The longitudinal electric field, comparing Figure 3.4c with Figure 2.3b reduced from 0.7GV/m to 0.5GV/m. This reduction is less than predicted in Section 3.2 from the increase in width of the beam. This can be explained as the beam particles close to the axis will have been self-focused in the first plasma cell more than particles off-axis, so enter the drift with less transverse momentum. So, despite the beam in the region of

interest overall getting 1.7 times wider over the drift, on axis the beam wouldn't have separated as much. From the change in field, $\frac{0.7}{0.5}$, we can naïvely infer that the on-axis part of the beam's transverse size increased by a factor of 1.2 over the drift stage, using Equation 1.10.

3.3.2 End of Plasma Cell

Figure 3.5 shows the density of the beam and the electric fields after 4m of plasma.

From Figure 3.5a, we see that over the length of the cell that the microbunch density stayed the same at around 10^{18}m^{-3}. However, the white regions of this plot indicate that much of the beam was ejected transversely. In fact, over the whole beam, it was found that 92% of the beam particles that entered the second plasma cell had been lost over the 4m cell. The figure shows that the beam had now been split into a bunch train. Due to the fact that these bunches are spaced at λ_p, see that even with significant particle loss in the beam, the accelerating fields are not significantly affected.

In Figure 3.5b we see the radial electric fields reduced from 0.25GV/m to 0.125GV/m over the cell, and the size of the beam defocusing region (blue = negative E_r) increased at the expense of the size of the focusing region decreasing.

Figure 3.5 shows the longitudinal field. The accelerating field reduced from 0.5GV/m to 0.25GV/m over the length of the plasma. It is interesting to note that the field remained on the GV/m-scale, despite the loss of so many beam particles.

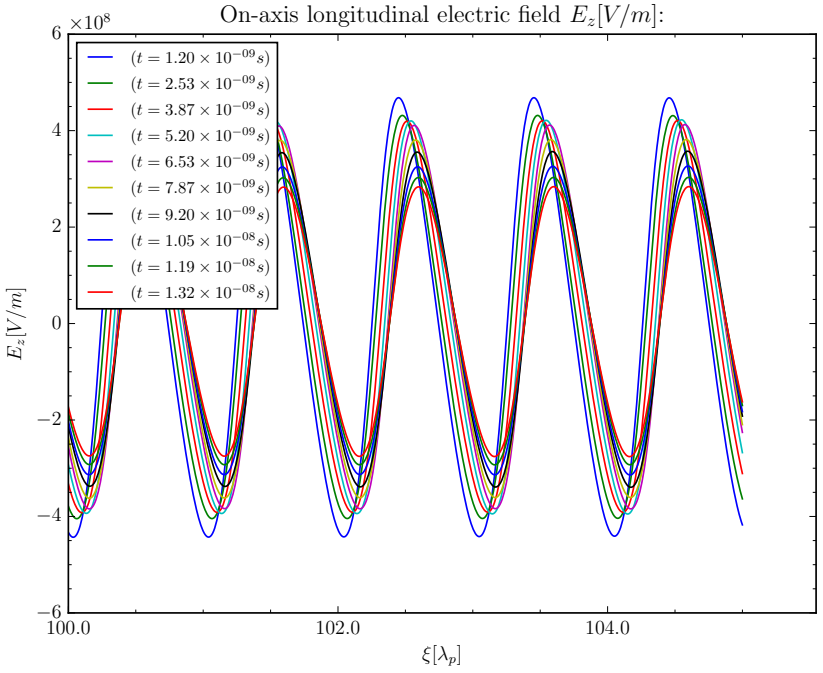


Figure 3.6: Plot of on-axis longitudinal E-fields at different points in the plasma. Multiply t by c to get the distance.

3.3.3 Whole Plasma

Figure 3.6 shows a plot of the longitudinal field, on-axis, plotted at different lengths along the plasma, displayed equivalently as time spent in the plasma[6]. We see the reduction in field gradient, from the red curve to the blue curve, and also the phase slippage between the wakefield and the bunch-head, due to the bunch-head travelling faster than the cavity structure of the wakefield. This phase slippage over the 4 metres is $\sim\lambda_p/8$.

The next section of this report moves on to a discussion of the results, as well as the project as a whole.

[6]Length in plasma $= ct$ where t is the time spent in the plasma.

Chapter 4

Discussion

In this final part of the report, the results obtained over the project will be discussed in Section 4.1, followed by suggestions of how to further extend the work done in the project into the future in Section 4.2.

4.1 Analysis of Results

4.1.1 First Plasma Cell

The one plasma-cell simulation provided a useful test of the simulation infrastructure before significant modifications needed to be made to extend it. It also provided benchmark data for comparison with the results from the second plasma-cell, to see how much the acceleration characteristics were affected by the introduction of a drift space and second plasma.

4.1.2 Vacuum Drift

In terms of the physics of the vacuum simulations, the study showed that after 4m in one plasma-cell with the AWAKE baseline setup, introducing a small drift space wouldn't adversely affect the integrity of the beam, meaning when it continued into the second cell, it still had the properties needed for self-modulation to take hold. This is important for future work in the AWAKE experiment, when the electron witness bunch will be introduced into the wake to be accelerated. This tentatively shows that the gap in plasma needed to include the electron injection hardware won't be detrimental to the proton beam's acceleration properties.

In terms of the simulation infrastructure, using BDSIM to simulate the vacuum has many benefits, due to it being a mature code set which has many physics options. Unfortunately, due to time constraints, these parameters couldn't be explored to the fullest, suggestions to this end are discussed in Section 4.2.1.

4.1.3 Second Plasma Cell

Introduction of the second cell saw the beam-plasma interactions yield a bunch train of protons, which was exciting the wakefield. While the accelerating field was weaker than in the first plasma cell, it was still at GV/m-scale, orders of magnitude above what is currently possible. This is a good check, confirming the viability of the AWAKE experiment, and therefore that of PDPWA.

Also observed in the second cell was a slight dephasing over the 4m length of the second plasma cell. In terms of electron injection, further into the AWAKE experiment this slow dephasing is good as it means that additional difficulty in timing the injection of the witness bunch won't be added by dephasing of the wake with the bunch-head. In Section 1.4.6, we see that the allowed phase shift once the electron bunch is injected is $\lambda_p/8$. This phase shift is seen in the second plasma cell over the 4m, implying that an electron bunch could be injected in the drift stage, and accelerated over this distance. At the start of the plasma cell, the accelerating field is 0.5GV/m, and at the end it is 0.25GV/m. If we assume a linear decrease in field strength, the accelerating field is given by Equation 4.1.

$$E_z[\mathrm{GV/m}] \approx \frac{z[\mathrm{m}]}{16} + 0.5 \tag{4.1}$$

where z is the distance travelled in the plasma. The energy gained by an electron accelerated by this field over the plasma cell would then be $\int_0^{4\mathrm{m}} E_z \,\mathrm{d}z = 2.5\mathrm{GeV}$. While this is not at the energy frontier, for such a short acceleration distance this is impressive, and is on the same scale as the results quoted in Section 1.3.2. The calculation is by no means rigorous, but provides an order-of-magnitude estimate of what is to be expected. A full simulation of the electron bunch injection would be required to substantiate this.

Time constraints prevented longer distances in the plasma being simulated. This would have been useful, to see how far the beam would need to travel through the plasma, before the cavity structure in the wakefield disintegrated to the point of not producing an accelerating field.

4.1.4 Limitations

There are a number of limitations of the LCODE → BDSIM → LCODE scheme for simulating staged plasmas, which will now be discussed.

4.1.4.1 Space Charge Effects

The first consideration is space charge effects. The proton beam is travelling in vacuum, a dielectric, and so, each particle of the beam will feel a force as if the other particles in the beam form a continuum of positive charge in the region of space which the beam occupies. BDSIM does not take space charge effects into account.

It remains to see if space charge effects are significant over the length of drift simulated in the project (1m). The transverse force on a beam particle due to the space charge of a beam is proportional to $1/\gamma^2$ [34], where $\gamma is the relativistic Lorentz factor$. As the proton beam is highly relativistic (400GeV proton $\Rightarrow \gamma \approx 430$), over short distances, space charge effects can be safely neglected.

4.1.4.2 Plasma Cell - Vacuum Interface

The main limitation of this simulation scheme is how it deals with the vacuum-plasma interface. Figure 4.1 shows in a qualitative way how the plasma density varies across a PDPWA accelerator. The proton beam enters the plasma from the beam pipe (vacuum),

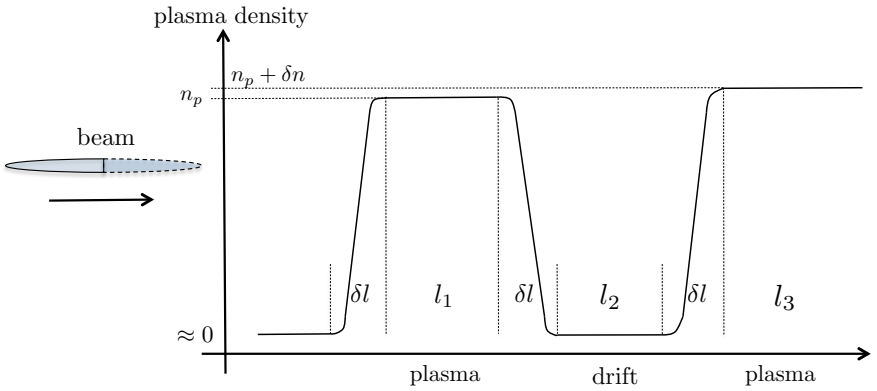


Figure 4.1: Idealised plasma density variation across a staged PDPWA accelerator. The density in the second cell can be the same as in the first, n_p, or be changed by some small δn, which can be either positive or negative. The dashed lines of the beam represent the fact that the beam is half cut.

enters the first cell, encounters a drift space, then enters the second cell. After exiting the cell (not shown in figure), it will then enter another vacuum and be subject to beam

optics to assess the quality of the beam. Note that at the interfaces between vacuum and plasma, the density continuously changes over some short distance δl.

The simulation scheme used in the project doesn't follow this prescription. Modelling the continuous change of plasma to vacuum was not implemented in the code set before, and implementing this variation would require significant work. the differences between how beam-plasma and beam-vacuum dynamics are modelled are briefly discussed in Section 2.4.1.

The scheme used in the project is having the beam travelling through each medium separately, and then being "teleported" into the next stage, as shown in Figure 4.2. While this model doesn't account for the interface physics, it is thought that the fact that the beam will only self-modulate in the plasma, along with space charge effects being negligible in the drift stages, makes this approximation reasonable.

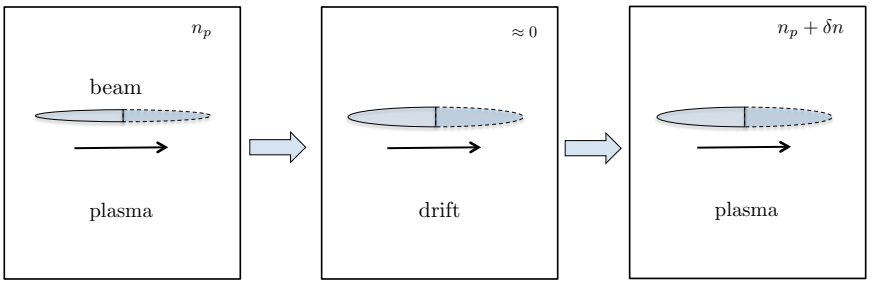


Figure 4.2: Plasma density variation across a staged PDPWA accelerator, in the project simulations.

4.2 Future Work

In this section, potential extensions to the work carried out are suggested, the outlook of the work is discussed and a concluding statement is given.

4.2.1 Quadrupole Study

It has been mentioned previously in the text that BDSIM includes the modelling of quadrupole magnets in the drift space. In Section 3.1, it is shown that the transverse beam profile after vacuum does not follow any well known distribution, thus preventing the straightforward determination of optimal quadrupole parameters. Introducing quadrupole focusing in the drift space would decrease the transverse size of the beam; Equation 1.10 implies that this focusing would increase the accelerating field in the second plasma cell. A study of the effect of varying quadrupole parameters on the size and

shape of the beam would prove beneficial to the AWAKE experiment, to further enhance the energy gains of an electron bunch in the accelerator. Implementing this study in the code set would be straightforward, but time-consuming, as many vacuum runs would need to be undertaken, along with the subsequent analysis.

4.2.2 Plasma Density, n_p

In Section 4.1.3, the detrimental effect on the acceleration characteristics in the simulations of dephasing of the wake with the bunch-head is discussed. In [35], a solution to the dephasing issue in PDPWA is proposed. This solution is to include a density step in the plasma, as in Figure 4.3. The idea is that the the density step allows the

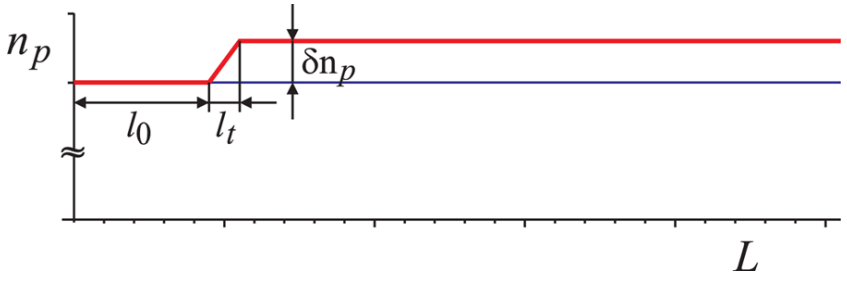


Figure 4.3: Illustration of density step in plasma for controlled modulation [18].

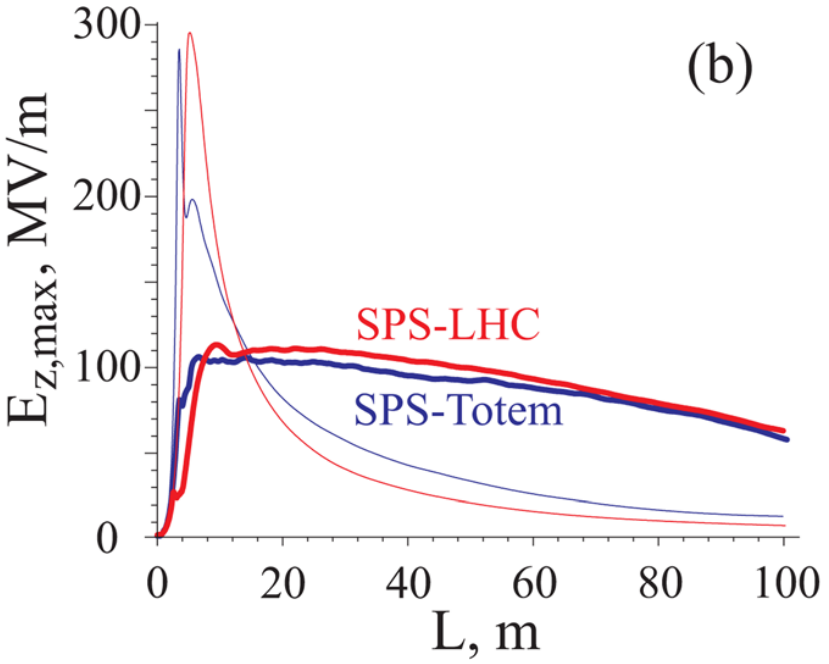


Figure 4.4: Accelerating field strengths over distance resulting from density step. The thin lines are the field strengths with no density step in the plasma, the thick lines with a density step. SPS-TOTEM and SPS-LHC are configurations of the SPS beam [18].

wake to follow the bunch in phase, as the increase in density causes a shortening of the plasma wavelength, thus a forward shift of the wake, relative to the bunch-head, which compensates for the lag occuring due to the self-modulation.

Figure 4.4 shows the effect of the density step on the accelerating field in simulations using AWAKE parameters.

Note that the maximum field overall is decreased with the density step, but stays at a maximum over much longer distances. In [18], the parameters of the step are $l_0 = 2.5$m, $l_t = 1.5$m and $\delta n/n_p = 3\%$.

A potential issue with this method for taming the self-modulation instability is the experimental difficulty in producing the density step in the plasma. It would be useful to investigate introducing a higher density in the second plasma cell into the simulation structure made for this project, to see if this could combat dephasing. While the step would be discontinuous as opposed to a smoothly varying profile, were this to work, it would be very useful for constructing a long distance PDPWA accelerator.

4.2.3 *N* Plasma Cells

The simulation structure made in this project also allows the simulation of an arbitrary number of plasma cells. In Section 1.4.6, the fact that PDPWA requires a very uniform plasma is discussed. Producing long, uniform plasma cells is experimentally difficult, with research ongoing. Knowing whether PDPWA can be sustained over a large number of cells would be useful in the long-term for designing a large PDPWA accelerator.

4.2.4 Electron Injection

The simulation structure used in the project currently doesn't allow for multiple beams in the plasma. Simulating the electron bunch being injected into the cell would be desirable to directly see by how much a bunch would be accelerated, and the quality of the bunch after acceleration. Introducing this into the simulation would require considerable modification, but is in principle possible by low-level modification of the LCODE beam-file.

4.3 Concluding Statement

Overall, the project has succeeded in that a simulation scheme for staging multiple plasma cells in the AWAKE experiment has been created. This scheme is also highly extensible and allows for further investigation into PDPWA, as discussed in Section 4.2.

It has also been demonstrated that introducing a drift space and second stage doesn't destroy the wakefield generated by a proton beam travelling through the plasma, with the wakefield having characteristics suitable for accelerating an electron bunch.

It remains to be seen in late 2016, when the AWAKE experiment goes live, whether proton beam self-modulation is observed, and whether the strong theoretical underpinnings of PDPWA can be realised experimentally. The author is confident that this will be the case.

References

1. Mohr, P. J., Taylor, B. N. & Newell, D. B. CODATA recommended values of the fundamental physical constants: 2010*. *Rev. Mod. Phys.* **84,** 1527–1605 (2012).
2. ATLAS Collaboration. Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. *Phys. Lett.* **B 716,** 1–29 (2012).
3. CMS Collaboration. Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC. *Phys. Lett.* **B 716,** 30–61 (2012).
4. Behnke, T. *et al.* The International Linear Collider Technical Design Report - Volume 1: Executive Summary. arXiv: `1306.6327 [physics.acc-ph]` (2013).
5. Esarey, E., Schroeder, C. B. & Leemans, W. P. Physics of laser-driven plasma-based electron accelerators. *Rev. Mod. Phys.* **81,** 1229–1285 (2009).
6. Martin, B. *Nuclear and Particle Physics: An Introduction* (Wiley, 2006).
7. Palaia, A., Jacewicz, M., Ruber, R., Ziemann, V. & Farabolini, W. Effects of rf breakdown on the beam in the Compact Linear Collider prototype accelerator structure. *Phys. Rev. ST Accel. Beams* **16,** 081004 (2013).
8. Tajima, T. & Dawson, J. M. Laser Electron Accelerator. *Phys. Rev. Lett.* **43,** 267–270 (1979).
9. Leemans, W. & Esarey, E. Laser-driven plasma-wave electron accelerators. *Phys. Today* **62,** 44–49 (2009).
10. Leemans, W. P. *et al.* GeV electron beams from a centimetre-scale accelerator. *Nat. Phys.* **2,** 696–699 (2006).
11. Blumenfeld, I. *et al.* Energy doubling of 42 GeV electrons in a metre-scale plasma wakefield accelerator. *Nature* **445,** 741–744 (2007).
12. Wang, X. *et al.* Quasi-monoenergetic laser-plasma acceleration of electrons to 2 GeV. *Nat. Commun.* **4,** 1988 (2013).
13. Ruth, R. D., Chao, A. W., Morton, P. L. & Wilson, P. B. A plasma wake-field accelerator. *Part. Accel.* **17,** 171–189 (1985).

14. Jones, R. M., Adolphsen, C., Wang, J. & Li, Z. Wakefield damping in a pair of X-band accelerators for linear colliders. *Phys.Rev.ST Accel.Beams* **9,** 102001 (2006).

15. Caldwell, A., Lotov, K., Pukhov, A. & Simon, F. Proton-driven plasma-wakefield acceleration. *Nat. Phys.* **5,** 363–367 (2009).

16. Lee, S., Katsouleas, T., Hemker, R. G., Dodd, E. S. & Mori, W. B. Plasma-wakefield acceleration of a positron beam. *Phys. Rev. E* **64,** 045501 (2001).

17. Kumar, N., Pukhov, A. & Lotov, K. Self-Modulation Instability of a Long Proton Bunch in Plasmas. *Phys. Rev. Lett.* **104,** 255003 (2010).

18. Caldwell, A. & Lotov, K. V. Plasma wakefield acceleration with a modulated proton bunch. *Phys. Plasmas* **18,** 103101 (2011).

19. Öz, E. & Muggli, P. A novel Rb vapor plasma source for plasma wakefield accelerators. *Nuc. Inst. Meth.* **A 740,** 197–202 (2014).

20. Lotov, K., Pukhov, A. & Caldwell, A. Effect of plasma inhomogeneity on plasma wakefield acceleration driven by long bunches. *Phys. Plasmas* **20,** 013102 (2013).

21. Lotov, K. V. *et al.* Electron trapping and acceleration by the plasma wakefield of a self-modulating proton beam. arXiv: `1408.4448 [physics.plasm-ph]` (2014).

22. AWAKE Collaboration, Assmann, R. *et al.* Proton-driven plasma wakefield acceleration: a path to the future of high-energy particle physics. *Plasma Phys. Contr. Fusion* **56,** 084013 (2014).

23. Muggli, P. *et al.* Photo-ionized lithium source for plasma accelerator applications. *IEEE Trans. Plasma Sci.* **27,** 791–799 (1999).

24. Vieira, J., Fonseca, R. A., Mori, W. B. & Silva, L. O. Ion Motion in Self-Modulated Plasma Wakefield Accelerators. *Phys. Rev. Lett.* **109,** 145005 (2012).

25. Caldwell, A *et al. AWAKE Design Report, A Proton-Driven Plasma Wakefield Acceleration Experiment at CERN* tech. rep. (CERN-SPSC-2013-013, CERN, Geneva, 2013).

26. Lotov, K. & Sosedkin, A. *LCODE source* `http://www.inp.nsk.su/~lotov/lcode/`. (2013).

27. Lotov, K. & Sosedkin, A. *LCODE User Manual* `http://www.inp.nsk.su/~lotov/autobuilds-lcode/lcode-stable/manual-r406/manual.pdf`. (2013).

28. Chen, F. *Introduction to Plasma Physics and Controlled Fusion* 2nd Ed. (Plenum Press, 1984).

29. Lotov, K. V., Minakov, V. A. & Sosedkin, A. P. Parameter sensitivity of plasma wakefields driven by self-modulating proton beams. *Phys. Plasmas* **21,** 083107 (2014).

30. Mandry, S. Noise in E-field (Ez) and potential (Φ), and EzAbsMax / xi(at EzAbsMax) plots. *UCL AWAKE meeting, minutes of* `https://www.hep.ucl.ac.uk/elog/PDPWA/140918_103349/UCL_MPI_2014_09_18.pdf` (2014).

31. Agapov, I. *et al. BDSIM source* `https://twiki.ph.rhul.ac.uk/twiki/bin/view/PP/JAI/BDsimDownload`. (2014).

32. Matsumoto, M. & Nishimura, T. Mersenne Twister: A 623-dimensionally Equidistributed Uniform Pseudo-random Number Generator. *ACM Trans. Model. Comput. Simul.* **8,** 3–30 (1998).

33. G.Cowan. *Statistical Data Analysis* 1st Ed. (Clarendon Press, 1998).

34. Mandry, S. Notes on Relativistic Bunch Space Charge Effects. *UCL AWAKE meeting, minutes of* `https://www.hep.ucl.ac.uk/elog/PDPWA/150226_112018/spaceCharge.pdf` (2015).

35. Lotov, K. V. Controlled self-modulation of high energy beams in a plasma. *Phys. Plasmas* **18,** 024501 (2011).

36. Agapov, I. *et al. BDSIM User's Manual* `https://twiki.ph.rhul.ac.uk/twiki/bin/view/PP/JAI/BDsimDocumentation`. (2014).

Appendix A

Simulation Details

This part of the Appendix gives a quick overwiew on how to get access to and run the simulations.

A.1 Initial Setup

All work is stored on the UCL HEP CVS server. To view/continue work on the simulation, a user must get a UCL HEP account as well as access to the CVS server, accessed via `ssh` thtough a UNIX terminal. After this, they execute
`git clone $USER@cvs.hep.ucl.ac.uk:/home/git/pdpwa/lcodeSimulations.git`
in a directory of choice. This creates a directory named `lcodeSimulations`. The relevant branch to `git checkout` to in this directory is `origin/DD_staged`. A full list of branches can be viewed with the command, `git branch -a`, once inside the new directory (default name `lcodeSimulations`). To see what precise code has been added/-modified, execute `gitk --date-order --all` (requires X11) when inside the simulation directory, to see all the additions made over the project.

The (known) software dependencies are LCODE, BDSIM, Python 2.7 (with Anaconda[1]). The author has had success running the plotting and analysis on `pc194`, and BDSIM on `pc178` on the UCL HEP system.

[1] `http://continuum.io/downloads`

A.2 Running the Simulation

A.2.1 LCODE - First Plasma Cell

The simulation directory is, from now on, referred to as `lSimDir`.

The physics of the first cell is specified in `lSimDir/settings/physics/sim.phys`. Here the physics and computing settings are specified for the first plasma, the comments in the code outline what everything does.

Once the user is satisfied with the settings, they must change directory to `lSimDir/scripts/runScripts`, then execute:

```
nohup ./run.sh -l &
```

where the `nohup` and the `&` mean LCODE runs in the background without stopping, when the user logs out of UCL HEP.

Once the run is done (this can take a number of days depending on the length of plasma and grid size), the user must then go the directory `lSimDir/scripts/postProcessing` and execute `./convertOutput.sh`, which makes the output files suitable for plotting.

After this, the user then goes to `lSimDir/scripts/plotScripts`, where there are a number of scripts which will plot various data of interest.

All of the LCODE output is stored in `lSimDir/output/`.

A.2.2 Beam Conversion

After the first plasma, the user must first copy the file `lSimDir/output/beamfile.bin` to the directory `lSimDir/binaryDecode`. Then the file must be converted to a format readable by BDSIM by executing

```
./binaryDecode.py -l2b < beamfile.bin > p_beam.txt
```

Note: after this conversion is complete, the file `LCODE_data.dump`, which is created *must not be deleted*. The code for `binaryDecode.py` can be found in Appendix B.2.

A.2.3 BDSIM - Drift space

At this point, the user decides what kind of vacuum they want to run in BDSIM. The relevant files to edit in `lSimDir/binaryDecode` are `p_beam.gmad`, `drift.gmad` and `options.gmad`. A point of reference for what to include in these files is the BDSIM User's Manual [36]. In `p_beam.gmad`, the number of particles being simulated needs to be specified.

Once the parameters are decided upon, BDSIM is ran by executing

```
bdsim --file=drift.gmad --batch
```

A.2.4 Beam Analysis

The scripts, `analyseBeamSize.py` and `bdPrimariesPlots.py` can be ran at this stage in order to look at the beam in more detail. Instructions for their usage is in the source code, found in Appendix B. The BDSIM parameters can then be adjusted accordingly and the analyses re-ran before the next plasma cell.

A.2.5 Second Beam Conversion

Once the user is satisfied with the vacuum run, they execute

```
./binaryDecode.py < output.txt > newBeamfile.bin
```

This produces the post vacuum output file for input into LCODE.

A.2.6 LCODE - Second Plasma Cell

For the second plasma cell, a new simulation folder must be created, as outlined in Section A.1. Then, the new beamfile in the old simulation folder, `newBeamfile.bin`, must be copied over to the new `lSimDir/output`, and re-named to `beamfile.bin`. Then, the file `beamfile.bit` must be copied over from the old output directory to the new one.

For the physics settings in `sim.phys`, care must be taken for the maximum time to be higher than the maximum time set for the first plasma cell, by the amount Δt such that $c\Delta t$ is the length of the second plasma cell.

The second plasma cell run is then initiated by going to the directory `lSimDir/scripts/runScripts`, as before, then executing

```
nohup ./run.sh -l -c &
```

The plotting and analysis for the second cell is then carried out in the same fashion as for the first cell.

Appendix B

Code

In this Appendix, the most relevant code used in the project is shown. A full list of code made in the project can be seen by following the instructions in Appendix A.1.

B.1 `run.sh`

```
#!/bin/bash
# Written by Scott Mandry, amended by Danial Dervovic

scriptName=$(basename "$0")
errMsgHdr="Error ($scriptName):"

# Paths
OLDDIR=$PWD
RUNSCRIPTDIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"

continuation='F'
moveData='T'

case $@ in
  -l) moveData='F' ;;
  -c) continuation='T' ;;
  '-l -c') moveData='F'
           continuation='T' ;;
  '-c -l') moveData='F'
           continuation='T' ;;
  *) echo $errMsgHdr 1>&2
     echo "   input argument not recognised. Exiting..." 1>&2
     exit 1
esac
```

```
cd $RUNSCRIPTDIR/../..
if [ -z $LSIM_isSetup ] ; then
  source scripts/.environment.sh
fi
source scripts/.macros.sh
simName=${PWD##*/}
cd $LSIM_outputDir

if [ $continuation = 'F' ]; then
  ./flush.sh
fi

eval "lcode --help" 2>&1 | head -n 1 >> $LSIM_versionFile
echo "Git repository commit code:"   >> $LSIM_versionFile
LSgitg | head -n 1 | cut -d " " -f 2 >> $LSIM_versionFile
cp ./$LSIM_physicsDir/$LSIM_physicsFile $LSIM_physicsSettings

./$LSIM_subScriptsDir/configPhysics.py $LSIM_prototypeConfigFile <
    $LSIM_physicsSettings
if [ $? -ne 0 ] ; then
  echo $errMsgHdr 1>&2
  echo "   Configuration error: problem with the physics file
    $physicsFile!  Exiting..." 1>&2
  exit $?
fi
cp -a ./$LSIM_configDir/$LSIM_configFile .
cp -a $LSIM_configFile $simName.cfg
configFile=$simName.cfg

echo "Recovering simulation parameters..."

awkCmd='{print $3}'
LCODE_PLOTHEIGHT=$(grep -w LCODE_PLOTHEIGHT $LSIM_lcodeGlobalsFile | awk
    "$awkCmd")
LCODE_PLOTLENGTH=$(grep -w LCODE_PLOTLENGTH $LSIM_lcodeGlobalsFile | awk
    "$awkCmd")
LCODE_LAMBDAP=$(   grep -w LCODE_LAMBDAP    $LSIM_lcodeGlobalsFile | awk
    "$awkCmd")
LCODE_PDENSITY=$(  grep -w LCODE_PDENSITY   $LSIM_lcodeGlobalsFile | awk
    "$awkCmd")
LCODE_LASERON=$(   grep -w LCODE_LASERON    $LSIM_lcodeGlobalsFile | awk
    "$awkCmd")
if [ "$LCODE_LASERON" == 'True' ] ; then
  echo "Formatting $LSIM_laserFile into LCODE units."
  if [ ! -f ./$LSIM_physicsDir/$LSIM_laserFile ] ; then
    echo "BASH Error: $LSIM_laserFile not found!" 1>&2
  else
    cp -a ./$LSIM_physicsDir/$LSIM_laserFile $LSIM_laserSettings
```

```
    ./$LSIM_subScriptsDir/configLaser.py $LCODE_LAMBDAP <
    $LSIM_laserSettings > laser.dat
  fi
fi
#echo "LCODE_PLOTHEIGHT = " $LCODE_PLOTHEIGHT
#echo "LCODE_PLOTLENGTH = " $LCODE_PLOTLENGTH
echo  "LCODE_LAMBDAP    = " $LCODE_LAMBDAP
echo  "LCODE_PDENSITY   = " $LCODE_PDENSITY
#echo >> $LSIM_physicsSettings
#echo "LCODE_PLOTHEIGHT = "  $LCODE_PLOTHEIGHT >> $LSIM_physicsSettings
#echo "LCODE_PLOTLENGTH = "  $LCODE_PLOTLENGTH >> $LSIM_physicsSettings
#echo  "LCODE_LAMBDAP    = " $LCODE_LAMBDAP    >> $LSIM_physicsSettings
#echo  "LCODE_PDENSITY   = " $LCODE_PDENSITY   >> $LSIM_physicsSettings

numberOfSimulations=`./$LSIM_subScriptsDir/getVar.sh N_species
    $LSIM_physicsSettings`

echo
echo "Running LCODE simulations..."
echo "PID = "$$ | tee -a $LSIM_runPIDfile

simCount=0
configFileStr="autoConfig"
while [ $simCount -lt $numberOfSimulations ]
  do
    LSIM_configFile=$configFileStr$simCount".cfg"
    echo "Running LCODE simulation "$simCount"..."
    lcode $LSIM_configFile 2>&1 | tee -a $LSIM_lcodeStdoutFile
    (( simCount++ ))
    echo "Lcode exit status:" $?
    echo
    echo "LCODE simulation complete."
done

cd $OLDDIR
```

B.2 binaryDecode.py

```
#!/usr/bin/env python

# Script which converts an LCODE beamfile for processing by BDSIM and
    vice versa

# Usage: ./binaryDecode.py -direction < $INPUTBEAMFILE > $OUTPUTBEAMFILE
# direction is either -b2l (BDSIM to LCODE) or -l2b (LCODE to BDSIM)
```

```
# Written by Danial Dervovic 28-01-15

#  Form of LCODE beam data from LCODE manual:
#  This file (beamfile.bin) contains information about beam macro-
    particles in the binary form. For each particle, 8 values are are
    written, 8 bytes each (type double). These values are:
#  [0] Longitudinal position, xi_b.
#  [1] Transverse position, rb (cylindrical geometry) or xb (plane
    geometry).
#  [2] Longitudinal momentum, pbz.
#  [3] Transverse momentum, pbr (cylindrical geometry) or pbx (plane
    geometry).
#  [4] Angular momentum Mb (cylindrical geometry) or third momentum
    component pby (plane geometry).
#  [5] Absolute value of the charge-to-mass ratio, compared to electron.
#  [6] Current carried by the macro-particle, in units of mc3/e.
#  [7] Ordinal number of the macro-particle.
#
#  Form of data to be inputted to BDSim, as per p_beam.txt
#   [0] x[mm] x-position relative to bunch head
#   [1] xp[mrad] x-momentum in mrad (~ component for small angles)
#   [2] y[mm] y-postion
#   [3] yp[mrad] y-momentum
#   [4] z[mm] z-position
#   [5] E[MeV] energy of particle

#  Form of data outputted by BDSim, as per p_beam.txt
#   [0] x[mum] x-position relative to bunch head
#   [1] xp[rad] x-momentum in mrad (~ component for small angles)
#   [2] y[mum] y-postion
#   [3] yp[rad] y-momentum
#   [4] z[mum] z-position
#   [5] E[GeV] energy of particle



import struct
import phys  # unit conversions
import physEV
import physCGS
import sys # for stdin
import re
import math  # For use of sqrt function
import os # For environment variables
import decimal # high precision
import random

## SETUP - Defining variables and lists etc
```

```
51
dumpFileName = 'LCODE_data.dump'
decimal.getcontext().prec=50 #set precision of decimal calcs
random.seed(1435431)

56 N_species = 2 # will need to get this from environment eventually
#plasmaDensity1 = os.environ.get('LCODE_PDENSITY') will want from globals
    file
#plasmaDensity2 = os.environ.get('LCODE_PDENSITY')
plasmaDensity1 = 7.0e20 # For now, manually put in densities
plasmaDensity2 = 7.0e20
61 deltaT = 1.333333333e-10 # deltaT = os.environ.get('LCODE_DELTA_T')
proton_mass = 1.67262178e-27 # proton mass in kg


# Plasma Units to be defined
66 plasmaDensity = 0.0
lambda_p = 0.0
lcodeLengthUnit = 0.0
lcodeTimeUnit = 0.0
lcodeCurrentUnit = 0.0
71 lcodeMomentumUnit = 0.0
lcodeAngMomentumUnit = 0.0
lcodeChargeToMassUnit = 0.0

# LCODE beam file structure setup
76 macroParticlesList = []
spareChars = '' # for lines which overrun
eightByteChars = 64 # to split lines into 8 byte chunks
oneByteChars = 8 # to split chunks into bytes

81 # BDSIM beam file structure setup
bdParticlesList = []


## PLASMA UNIT CONVERSIONS
86

def definePlasmaUnits(plasmaDensityOfInterest):
  global plasmaDensity, lambda_p, lcodeLengthUnit, lcodeTimeUnit,
    lcodeCurrentUnit, lcodeMomentumUnit, lcodeAngMomentumUnit,
    lcodeChargeToMassUnit # make global so can change in function
  plasmaDensity     = plasmaDensityOfInterest
  lambda_p = math.sqrt(phys.m_e() * phys.e0() / plasmaDensity) \
91       * phys.twoPi() * phys.c() / phys.eV()
  lcodeLengthUnit   = lambda_p / phys.twoPi()
  lcodeTimeUnit     = lcodeLengthUnit / phys.c()
  lcodeCurrentUnit  = physCGS.m_e() * physCGS.c()**3 / physCGS.e()
  lcodeMomentumUnit = phys.m_e() * phys.c()
```

```
  lcodeAngMomentumUnit = phys.m_e() * phys.c()**2 * lcodeTimeUnit
  lcodeChargeToMassUnit = physCGS.e() / physCGS.m_e()

## COMPARISON FUNCTION (for sorting LCODE beamfile)

def compareField(index, order):
  '''Function for use with sort() method orders elements of nested list
    by some parameter within the inner list'''
  # index, index of inner list by which to order, must be integer from 0
    to size(innerList)
  # order, either 'ascending' or 'descending'
  orderValue = None
  if order == 'ascending':
    orderValue = 1
  elif order == 'descending':
    orderValue = -1
  else:
    sys.stderr.write('Need to specify direction of sort, either ascending
     or descending. \n')
    exit(2)
  def compareInner(orderVal):
    def compareList(list1, list2):
      if len(list1) != len(list2):
        sys.stderr.write('Lists inside main List not equal size! \n')
      if index >= len(list1) or index < 0:
        sys.stderr.write('compareField Error: index out of bounds \n')
        exit(2)
      returnVal = None
      compVal = list1[index] - list2[index]
      if compVal > 0:
        returnVal = orderVal
      elif compVal < 0:
        returnVal = orderVal * (-1)
      elif compVal == 0:
        returnVal = 0
      return returnVal
    return compareList
  return compareInner(orderValue)

## LCODE TO BDSIM FILE CONVERSION

def lcodeToBdsim(beamfile):
  global macroParticlesList, spareChars, eightByteChars, oneByteChars #
    declare variables global for use in function
  ## Import LCODE beam data, convert from binary to doubles and store
  numberOfLines = len(beamfile)
  lineCounter = 0
  for line in beamfile:
```

```
    lineCounter += 1
    sys.stderr.write('Importing data for line '+str(lineCounter)+' of ' +
     str(numberOfLines) + ' in beamfile. \r')
    line = spareChars + line # Add data from previous overrun line
    splitList = [line[i:i+eightByteChars] for i in range(0, len(line),
    eightByteChars)] # Split into 8-byte chunks
    for onePartData in splitList: # Check if length 64, if not add to
    start of next line
      if len(onePartData) != eightByteChars:
        spareChars = onePartData
        continue
      # split into individual bytes
      particleDataListBinary = [onePartData[j:j+oneByteChars] for j in
    range(0, len(onePartData), oneByteChars)]
      particleDataListDouble = []
      # now loop through bytes
      for binary in particleDataListBinary:
        num = struct.unpack('d',binary)[0] # Convert to double
        particleDataListDouble.append(num) # put double into new array
      macroParticlesList.append(particleDataListDouble)
  macroParticlesList.pop() #remove file marker
  sys.stderr.write('\n')

## Dump parameters not used by BDSim
  dumpFile = open(dumpFileName, 'w')
  for i in range(0, len(macroParticlesList)):
    sys.stderr.write('Dumping data for particle '+str(i+1)+' of ' + str(
    len(macroParticlesList)) + '. \r')
    particleDataList = macroParticlesList[i]
    stringToWrite = ''
    for j in [5,6,7]: # last 3 parameters
      stringToWrite += (str(particleDataList[j]) + '\t')
    dumpFile.write(stringToWrite + '\n')
  dumpFile.close()
  sys.stderr.write('\n')

## Add reference particle
  bdParticlesList.append([0, 0, 0, 0, 0, 400000.0])
## Unit Conversions
  definePlasmaUnits(plasmaDensity1)
  # loop through beam data
  for i in range(0, len(macroParticlesList)):
    sys.stderr.write('Converting data for particle '+str(i+1)+' of ' +
    str(len(macroParticlesList)) + '. \r')
    particleDataList = macroParticlesList[i]
    # now convert units into SI
    particleDataList[0] = particleDataList[0] * lcodeLengthUnit
    particleDataList[1] = particleDataList[1] * lcodeLengthUnit
```

```
    particleDataList[2] = particleDataList[2] * lcodeMomentumUnit
    particleDataList[3] = particleDataList[3] * lcodeMomentumUnit
    particleDataList[4] = particleDataList[4] * lcodeAngMomentumUnit
    particleDataList[5] = particleDataList[5] * lcodeChargeToMassUnit
    particleDataList[6] = particleDataList[6] * lcodeCurrentUnit

    # now into format readable by BDSim
     # postions
    r = particleDataList[1]
    theta = 2 * phys.pi() * random.random()
    x = r * math.cos(theta) * 1000
    y = r * math.sin(theta) * 1000
    z = particleDataList[0] * 1000

    costheta = decimal.Decimal(str(math.cos(theta)))
    sintheta = decimal.Decimal(str(math.sin(theta)))

    # momenta
    p_theta = particleDataList[4] / r
    p_r = particleDataList[3]
    p_z = particleDataList[2]
    p = math.sqrt(p_r**2 + p_z**2 + (p_theta * r)**2)
    prsintheta = decimal.Decimal(str(p_r)) * sintheta
    prcostheta = decimal.Decimal(str(p_r)) * costheta
    pthetasintheta = decimal.Decimal(str(p_theta)) * sintheta
    pthetacostheta = decimal.Decimal(str(p_theta)) * costheta
    px = float(prcostheta - pthetasintheta)/p * 1000 # mrad
    py = float(prsintheta + pthetacostheta)/p * 1000 # mrad
    E = math.sqrt((p * phys.c())**2 + (proton_mass**2 * phys.c()**4)) #
   joules
    E = E * 6.241509e12 #convert to MeV

    # put particle into BDSim List
    bdParticlesList.append([x, px, y, py, z, E])
  sys.stderr.write('\n')

  ## Write to BDSim beamfile
  for i in range(0, len(bdParticlesList)):
    sys.stderr.write('Writing data for particle '+str(i+1)+' of ' + str(
   len(bdParticlesList)) + ' to beamfile. \r')
    particleDataList = bdParticlesList[i]
    print ' '.join(str(num) for num in particleDataList)

  sys.stderr.write('\n')


## BDSIM TO LCODE FILE CONVERSION
```

```
def bdsimToLcode(beamfile):
  global bdParticlesList
  numberOfLines = len(beamfile)
  lineCounter = 0
  ## Unpack BDSim beam data, and put into bdParticlesList
  for line in beamfile:
    lineCounter += 1
    sys.stderr.write('Importing data for line '+str(lineCounter)+' of ' +
     str(numberOfLines) + ' in beamfile. \r')
    if line[0].isdigit() == False: # leave lines with no data
      continue
    line = line.split() # split line into List
    particleData = [] # new data List
    # put in data from beamfile and make SI units
    particleData.append(float(line[5])/1000000)
    particleData.append(float(line[15]))
    particleData.append(float(line[6])/1000000)
    particleData.append(float(line[16]))
    particleData.append(float(line[7])/1000000)
    particleData.append(float(line[1]) * 1.602177e-10)

    bdParticlesList.append(particleData)

  # Remove reference particle
  bdParticlesList.pop(0)
  sys.stderr.write('\n')

  # Define Plasma Units
  definePlasmaUnits(plasmaDensity2)

  ## Physics Conversions
  for i in range(len(bdParticlesList)):
    sys.stderr.write('Converting data for particle '+str(i+1)+' of ' +
    str(len(bdParticlesList)) + '. \r')
    particleDataList = bdParticlesList[i]
    xi_b = particleDataList[4] / lcodeLengthUnit
    x = particleDataList[0]
    y = particleDataList[2]
    r_bSI = math.sqrt(x**2 + y**2)
    r_b = r_bSI / lcodeLengthUnit
    # momenta
    p = math.sqrt((particleDataList[5]**2 - (proton_mass**2 * phys.c()
    **4))/(phys.c()**2))
    px = particleDataList[1] * p
    py = particleDataList[3] * p
    pz = math.sqrt(p**2 - px**2 - py**2)
    theta = math.atan2(y, x)
    costheta = decimal.Decimal(str(math.cos(theta)))
```

```
    sintheta = decimal.Decimal(str(math.sin(theta)))
    pxDec = decimal.Decimal(str(px))
    pyDec = decimal.Decimal(str(py))
    pxsintheta = pxDec * sintheta
    pxcostheta = pxDec * costheta
    pysintheta = pyDec * sintheta
    pycostheta = pyDec * costheta

    p_r = pxcostheta + pysintheta
    p_r = float(p_r)
    p_theta = -pxsintheta + pycostheta
    p_theta = float(p_theta)
    M_b = (p_theta * math.fabs(r_bSI)) / lcodeAngMomentumUnit
    p_br = p_r / lcodeMomentumUnit
    p_bz = pz / lcodeMomentumUnit
    # add particles to file
    bdParticlesList[i] = [xi_b, r_b, p_bz, p_br, M_b]
sys.stderr.write('\n')

## Merge in LCODE dumped data
dumpFile = open(dumpFileName, 'r')
lcodeDumpedData = dumpFile.readlines()
dumpList = []
# load in data dump
for line in lcodeDumpedData:
  line = line.split()
  for index, item in enumerate(line):
    line[index] = float(item)
  dumpList.append(line)
# Add in LCODE dump data with BDSim data together
for i in range(len(bdParticlesList)):
  sys.stderr.write('Merging dumped data for particle '+str(i+1)+' of '
 + str(len(bdParticlesList)) + '. \r')
  macroParticlesList.append((bdParticlesList[i] + dumpList[i]))

# Sort Array by r, then xi. So beamfile in descending xi order. For
  same xi, sorted by ascending r

sys.stderr.write('\n Sorting beamfile ...')
macroParticlesList.sort(compareField(1,'ascending'))
macroParticlesList.sort(compareField(0,'descending'))

# End of file marker
macroParticlesList[len(macroParticlesList)-1][0] = -100000.0

# print to file
for i in range(0, len(macroParticlesList)):
```

```
    sys.stderr.write('Writing data for particle '+str(i+1)+' of ' + str(
    len(macroParticlesList)) + ' to beamfile. \r')
    particleDataList = macroParticlesList[i]
    sys.stdout.write(''.join(struct.pack('d', float(num)) for num in
    particleDataList)) # print data for one particle converted to binary
321
  sys.stderr.write('\n')

## MAIN METHOD - Checks which way to convert files then executes relevant
    commands

326 def main(argv):
  usageMessage = sys.argv[0] + " USAGE: \n    " + sys.argv[0] + " -
    direction < $INPUTBEAMFILE > $OUTPUTBEAMFILE \n    -direction is
    either: \n     -l2b -> LCODE binary to BDSim input \n     -b2l ->
    BDSim output to LCODE binary \n"
  if len(sys.argv) != 2: # check direction of file conversion is
    specified
    sys.stderr.write(usageMessage)
    sys.exit(2)
331 BEAMFILE = sys.stdin.readlines()
  if len(BEAMFILE) == 0:
    sys.stderr.write(sys.argv[0] + ' error: Input Beamfile is empty \n')
    sys.exit(2)
  if sys.argv[1] == '-l2b':
336   sys.stderr.write('Parsing LCODE beamfile as input... \n')
    lcodeToBdsim(BEAMFILE)
  elif sys.argv[1] == '-b2l':
    sys.stderr.write('Parsing BDSIM beamfile as input... \n')
    bdsimToLcode(BEAMFILE)
341 else:
    sys.stderr.write(usageMessage)
    sys.exit(2)

## RUN THE MAIN METHOD
346
if __name__ == "__main__":
   main(sys.argv[1:])
```

B.3 bdPrimariesPlots.py

```
1 #!/home/dervovic/anaconda/bin/python
#-*- coding: utf-8 -*-

# Script to take BDSim output.primaries.txt file, make histograms from it
    and fit curves to data.
```

```
# Fit parameters are printed to STDOUT. In main method at the bottom of
    the script is where plots
# are declared (line 646).

# Usage ./bdPrimariesPlots.py < $PRIMARIESFILE > $FITDATADUMP

# Written by Danial Dervovic, 3 March 2015

## SETUP

import sys
import numpy as np
import matplotlib as mpl
mpl.use('Agg') # Don't use X11 backend for plotting
import matplotlib.pyplot as plt
from matplotlib import rc #latex
import matplotlib.mlab as mlab
from scipy import stats
from scipy import optimize
import math

# empty Lists
bdParticlesList =[]
bdE = []
bdX = []
bdY = []
bdZ = []
bdPx = []
bdPy = []

# plot region

minimum_z = -sys.float_info.max
maximum_z = 0

# LateX fonts in plots
plt.rc('font', **{'family': 'serif', 'serif': ['Computer Modern']})
plt.rc('text', usetex=True)

#plt.rc('font', family='serif')
#plt.rc('font', weight='normal')

## USEFUL FUNCTIONS

# Format scientific float into LaTeX expression
def tex_float(f):
  float_str = "{0:.3g}".format(f) # change the number before g for sig
    figs
```

```
51    if "e" in float_str:
        base, exponent = float_str.split("e")
        return r"{0} \times 10^{{{1}}}".format(base, int(exponent))
      else:
        return float_str
56
  # Define bin edges for 2D Histogram
  def makeBins(list, nbins):
      if min(list) < 0:
        list_min = 1.01 * min(list) # so all values binned
61    else:
        list_min = 0.99 * min(list)
      if max(list) < 0:
        list_max = 0.99 * max(list)
      else:
66      list_max = 1.01 * max(list)
      subStep = (list_max - list_min) / nbins
      list_ticks = []
      for i in range(0,(nbins+1)):
        tick = list_min + i * subStep
71      list_ticks.append(tick)
      return list_ticks

  # Normal distribution wrapper
  def normdist(x, mu, sigma, cdf=False):
76    def normpdf(x1, mu1, sigma1):
        u = (x1-mu1)/math.fabs(sigma1)
        y = (1/(np.sqrt(2*math.pi)*math.fabs(sigma1)))*np.exp(-u*u/2)
        return y
      def normcdf(x1, mu1, sigma1):
81      t = x1-mu1;
        y = 0.5*math.erfc(-t/(sigma1*math.sqrt(2.0)));
        if y>1.0:
          y = 1.0;
        return y
86    if cdf == True :
        y = normcdf(x,mu,sigma)
      elif cdf == False:
        y = normpdf(x,mu,sigma)
      else:
91      sys.stderr.write('normdist error: Specify if cdf with True or pdf
       with False')
        exit(2)
      return y

  # Cauchy/Lorentzian distribution wrapper
96 def cauchydist(x, x0, gamma, cdf=False):
      def cauchypdf(x1, x01, gamma1):
```

```
    u = (x1-x01)/math.fabs(gamma1)
    denominator = (math.pi * gamma1)* (1 + u**2)
    y = 1 / denominator
    return y
  def cauchycdf(x1, x01, gamma1):
    u = (x1-x01)/math.fabs(gamma1)
    y = (math.atan(u)/(math.pi)) + 0.5
    return y
  if cdf == True :
    y = cauchycdf(x,x0,gamma)
  elif cdf == False:
    y = cauchypdf(x,x0,gamma)
  else:
    sys.stderr.write('cauchydist error: Specify if cdf with True or pdf
    with False')
    exit(2)
  return y

## IMPORT DATA

def importData(beamfile, zMax=0, zMin=-sys.float_info.max):
  """ Import data into appropriate data structures """
  global bdParticlesList, bdE, bdX, bdY, bdZ, bdPx, bdPy, minimum_z,
    maximum_z
  numberOfLines = len(beamfile)
  lineCounter = 0
  # clear data arrays
  bdParticlesList =[]
  bdE = []
  bdX = []
  bdY = []
  bdZ = []
  bdPx = []
  bdPy = []
  if zMax==0 and zMin==-sys.float_info.max:
    importDataStr = 'Processing data for whole beam...\n'
  else:
    importDataStr = 'Processing data for beam from z = '+str(zMin)+' to '
    +str(zMax)+' mum... \n'
  sys.stderr.write(importDataStr)
  ## Unpack BDSim beam data, and put into bdParticlesList and individual
    Lists
  for line in beamfile:
    lineCounter += 1
    if line[0].isdigit() == False: # leave lines with no data
      continue
    if lineCounter <= 4: # remove reference particles
      continue
```

```
    sys.stderr.write('Parsing data for line '+str(lineCounter)+' of ' +
    str(numberOfLines) + ' in beamfile. \r')
    newline = line.split() # split line into List
    # Check if particle in range
    if float(newline[4]) < zMax and float(newline[4]) > zMin:
      particleData = [] # new data List
      # put in data from beamfile
      particleData.append(float(newline[1])) # Energy in GeV
      bdE.append(float(newline[1])) # Energy in GeV
      particleData.append(float(newline[2])) # x in mum
      bdX.append(float(newline[2])) # x in mum
      particleData.append(float(newline[3])) # y in mum
      bdY.append(float(newline[3])) # y in mum
      particleData.append(float(newline[4])) # z in mum
      bdZ.append(float(newline[4])) # z in mum
      particleData.append(float(newline[7])) # xp in rad
      bdPx.append(float(newline[7])) # xp in rad
      particleData.append(float(newline[8])) # yp in rad
      bdPy.append(float(newline[8])) # xp in rad
      bdParticlesList.append(particleData)
    else:
      continue
  minimum_z = float(min(bdZ)) / 1000 # mm
  maximum_z = float(max(bdZ)) / 1000
  sys.stderr.write('\n')

## PLOT HISTOGRAM OF INDIVIDUAL QUANTITIES AND FIT CURVE TO DATA
def plotHist(dataList, plotName, xLabel, nbins, fitFunc):

  """

  dataList - python List containg values for a particular variable for
    every particle.
  plotName - base name to give to plot.
  xLabel - x-axis label, can use LateX characters such as \mu etc.
  nbins - number of bins in histogram.
  fitFunc - distribution to fit to data, either 'Gaussian', 'Cauchy', '
    TwoGauss, or 'None' for no fitting.

  """
  # plot setup
  plt.clf()
  sys.stderr.write('Plotting Data for ' + plotName + '... \n')
  fig, ax = plt.subplots(1)
  if minimum_z == -sys.float_info.max and maximum_z == 0:
    plotRange = r'Whole beam: $-$300mm $ < z < $ 0mm'
    fileName = plotName + str(nbins)+'bins_wholebeam.pdf'
  else:
```

```
    plotRange = r'Plot range: $'+tex_float(minimum_z)+r'$mm $< z <'+
    tex_float(maximum_z)+'$mm'
    fileName = plotName + str(nbins)+'bins'+str(minimum_z)+'-'+str(
    maximum_z)+'mm.pdf'
  plotTitle = 'Distribution of $'+plotName+'$ for particles after 4m -
    AWAKE baseline\n'+plotRange
  ax.set_xlabel(xLabel)
  ax.set_ylabel('counts')
  plt.title(plotTitle, y=1.01)

  # plot histogram
  n, bins, patches = ax.hist(dataList, bins=nbins, normed=False, alpha
    =0.5, color='0.5')

  # quantities used for fitting/plotting distributions
  binCentres = bins + 0.5 * (bins[1] - bins[0])
  binCentres = np.delete(binCentres,-1)
  totalN = sum(n)
  normalise = totalN * (bins[1] - bins[0])

  if fitFunc == 'None': # no fitting
    plt.savefig('NoFit'+fileName, bbox_inches='tight') # print to file

  elif fitFunc == 'Gaussian': # gaussian fit
    mu = np.mean(dataList) # Estimators for distribution
    sigma = np.std(dataList)
    # plot distribution over histogram
    distVals = np.linspace(bins[0], bins[-1] ,300 ,endpoint=True)
    y = [(normalise * normdist(distVals[i], mu, sigma, False)) for i in
    range(0, len(distVals))] # distribution scaled to counts
    ax.plot(distVals, y, 'k', linewidth=1) # plot distribution over
    histogram

    ## chi2 goodness of fit test
    # number of counts predicted by Gaussian distribution
    predictedCounts = [(totalN * (normdist(bins[(i+1)], mu, sigma, True)
    - normdist(bins[i], mu, sigma, True))) for i in range(0, nbins)]
    # residuals
    residuals = [((predictedCounts[i] - n[i])/math.sqrt(predictedCounts[i
    ])) for i in range(0, len(n))]
    #chi2
    chi2 = sum([residuals[i]**2 for i in range(0, len(residuals))])
    # p-value, (n - 2) degrees of freedom
    pchi2 = 1 - stats.chi2.cdf(chi2, (nbins-2))
    # Finish and save plot
    pchi2Str = tex_float(pchi2)
    chi2Str = tex_float(chi2)
    muStr = tex_float(mu)
```

```
    sigmaStr = tex_float(sigma)
    textStr = '\small Gaussian Dist.\n'+r'$\mu='+muStr+'$\n$\sigma='+
    sigmaStr+'$\n$\chi^2 ='+chi2Str+'$\n$ P(\chi^2) ='+pchi2Str +'$'
    textStr2 = '\small Gaussian Dist.\n'+r'$\mu='+muStr+'$\n$\sigma='+
    sigmaStr+'$'
    # these are matplotlib.patch.Patch properties
    props = dict(boxstyle='round', facecolor='None')
    # place a text box in upper left in axes coords
    ax.text(0.05, 0.95, textStr2, transform=ax.transAxes, fontsize=12,
    verticalalignment='top', bbox=props)
    txt = plt.savefig('Gauss'+fileName, bbox_inches='tight') # print to
    file

    # remake plot without statistic labels
    plt.clf()
    fig, ax = plt.subplots(1)
    ax.hist(dataList, bins=nbins, normed=False, alpha=0.5, color='0.5') #
     histogram
    ax.plot(distVals, y, 'k', linewidth=1) # gaussian overlaid
    ax.set_xlabel(xLabel)
    ax.set_ylabel('counts')
    plt.title(plotTitle,y=1.01)
    # fit values
    ax.text(0.05, 0.95, textStr, transform=ax.transAxes, fontsize=12,
    verticalalignment='top', bbox=props)
    plt.savefig('chi2Gauss'+fileName, bbox_inches='tight') # print to
    file

    # print fit data
    dataStr = plotName+', Gaussian fit, '+str(nbins)+' bins, z from '+str
    (minimum_z)+' to '+str(maximum_z)+', (mm) :\nmu = '+str(mu)+', sigma =
     '+str(sigma)+'\nchi^2 = '+str(chi2)+', p(chi2) = '+str(pchi2)+'\n'
    print dataStr

  elif fitFunc == 'Cauchy': # Cauchy/Lorentzian fit
    # find fitted values
    normedN = [(n[i]/normalise) for i in range(0, nbins)] # normalise
    bins
    guess = [np.mean(dataList), np.std(dataList)] # guess values close to
     estimated
    try:
      popt, pcov = optimize.curve_fit(cauchydist, binCentres, ydata=
    normedN, p0=guess) # fit to cauchy
    except:
      print 'No suitable fit values found'
      return
    x0 = popt[0] # fitted parameters
    gamma = popt[1]
```

```
    # plot distribution
    distVals = np.linspace(bins[0], bins[-1] , 300, endpoint=True)
    y = [(normalise * cauchydist(distVals[i], x0, gamma, False)) for i in
     range(0, len(distVals))]
    ax.plot(distVals, y, 'k', linewidth=1) # distribution overlaid

    ## chi2 goodness of fit test
    # number of counts predicted by Cauchy distribution, note scaled to
    number of counts
    predictedCounts = [(totalN * (cauchydist(bins[(i+1)], x0, gamma, True
    ) - cauchydist(bins[i], x0, gamma, True))) for i in range(0, nbins)]
    # residuals
    residuals = [((predictedCounts[i] - n[i])/math.sqrt(predictedCounts[i
    ])) for i in range(0, len(n))]
    #chi2
    chi2 = sum([residuals[i]**2 for i in range(0, len(residuals))])
    # p-value, (n - 2) degrees of freedom
    pchi2 = 1 - stats.chi2.cdf(chi2, (nbins-2))

    # Finish and save plot
    pchi2Str = tex_float(pchi2)
    chi2Str = tex_float(chi2)
    x0Str = tex_float(x0)
    gammaStr = tex_float(gamma)
    textStr = '\small Cauchy Dist.\n'+r'$x_0='+x0Str+'$\n$\gamma='+
    gammaStr+'$\n$\chi^2 ='+chi2Str+'$\n$ P(\chi^2) ='+pchi2Str +'$'
    textStr2 = '\small Cauchy Dist.\n'+r'$x_0='+x0Str+'$\n$\gamma='+
    gammaStr+'$'
    # these are matplotlib.patch.Patch properties
    props = dict(boxstyle='round', facecolor='None')
    # place a text box in upper left in axes coords
    ax.text(0.05, 0.95, textStr2, transform=ax.transAxes, fontsize=12,
    verticalalignment='top', bbox=props)
    txt = plt.savefig('Cauchy'+fileName, bbox_inches='tight') # print to
    file

    # remake plot without chi2 label
    plt.clf()
    fig, ax = plt.subplots(1)
    ax.hist(dataList, bins=nbins, normed=False, alpha=0.5, color='0.5') #
     histogram
    ax.plot(distVals, y, 'k', linewidth=1) # distribution overlaid
    ax.set_xlabel(xLabel)
    ax.set_ylabel('counts')
    plt.title(plotTitle, y=1.01)
    # display fitted parameters
    ax.text(0.05, 0.95, textStr, transform=ax.transAxes, fontsize=12,
    verticalalignment='top', bbox=props)
```

```
301     plt.savefig('chi2Cauchy'+fileName, bbox_inches='tight') # print to
        file

        # print fit data
        dataStr = plotName+', Cauchy fit, '+str(nbins)+' bins, z from '+str(
        minimum_z)+' to '+str(maximum_z)+', (mm) :\nx0 = '+str(x0)+', gamma =
        '+str(gamma)+'\nchi^2 = '+str(chi2)+', p(chi2) = '+str(pchi2)+'\n'
        print dataStr
306

      elif fitFunc == 'TwoGauss': # Two Gaussians fit
        # find fitted values
        normedN = [(n[i]/normalise) for i in range(0, nbins)] # normalise
        bins
311     mu = np.mean(dataList)
        sigma = np.std(dataList)
        guess = [mu, mu, sigma, sigma] # guess values close to estimated

        xMin = min(dataList)
316     xMax = max(dataList)

        def twoNorms(x, mu1, mu2, sigma1, sigma2, cdf=False):
          if mu1 > xMin and mu1 < xMax and mu2 > xMin and mu2 < xMax:
            y = 0.5*normdist(x, mu1, sigma1, cdf) + 0.5*normdist(x, mu2,
        sigma2, cdf)
321       else:
            y = sys.float_info.max # return large value if parameters in
        unphysical region
          return y

        try:
326       popt, pcov = optimize.curve_fit(twoNorms, binCentres, ydata=normedN
        , p0=guess) # fit to 2 gaussians
        except:
          print 'No suitable fit values found...'
          return
        mu1 = popt[0] # fitted parameters
331     mu2 = popt[1]
        sigma1 = popt[2]
        sigma2 = popt[3]
        # plot distribution
        distVals = np.linspace(bins[0], bins[-1] , 300, endpoint=True)
336     y = [(normalise * twoNorms(distVals[i], mu1, mu2, sigma1, sigma2))
        for i in range(0, len(distVals))]
        ax.plot(distVals, y, 'k', linewidth=1) # distribution overlaid

        ## chi2 goodness of fit test
```

```
    # number of counts predicted by twoNorms distribution, note scaled to
     number of counts
    predictedCounts = [(totalN * (twoNorms(bins[(i+1)], mu1, mu2, sigma1,
     sigma2, True) - twoNorms(bins[i], mu1, mu2, sigma1, sigma2, True)))
for i in range(0, nbins)]
    # residuals
    residuals = []
    for i in range(0, len(n)):
      if predictedCounts[i] > 0:
        residuals.append(math.fabs(predictedCounts[i] - n[i])/math.sqrt(
predictedCounts[i]))
      else:
        residuals.append(math.fabs(predictedCounts[i] - n[i])) # since
zero predicted counts, we want to
    #chi2
    chi2 = sum([residuals[i]**2 for i in range(0, len(residuals))])
    # p-value, (n - 4) degrees of freedom
    pchi2 = 1 - stats.chi2.cdf(chi2, (nbins - 4))

    # Finish and save plot
    pchi2Str = tex_float(pchi2)
    chi2Str = tex_float(chi2)
    mu1Str = tex_float(mu1)
    mu2Str = tex_float(mu2)
    sigma1Str = tex_float(sigma1)
    sigma2Str = tex_float(sigma2)
    textStr = r'\small 2 $\times$ Gaussian Dist.'+'\n'+r'$\mu_1='+mu1Str+
'$\n$\sigma_1='+sigma1Str+'$\n$\mu_2='+mu2Str+'$\n$\sigma_2='+
sigma2Str+'$\n$\chi^2 ='+chi2Str+'$\n$ P(\chi^2) ='+pchi2Str +'$'
    textStr2 = r'\small 2 $\times$ Gaussian Dist.'+'\n'+r'$\mu_1='+mu1Str
+'$\n$\sigma_1='+sigma1Str+'$\n$\mu_2='+mu2Str+'$\n$\sigma_2='+
sigma2Str+'$'
    # these are matplotlib.patch.Patch properties
    props = dict(boxstyle='round', facecolor='None')
    # place a text box in upper left in axes coords
    ax.text(0.05, 0.95, textStr, transform=ax.transAxes, fontsize=12,
verticalalignment='top', bbox=props)
    txt = plt.savefig('chi2TwoGauss'+fileName, bbox_inches='tight') #
print to file

    # remake plot without chi2 label
    plt.clf()
    fig, ax = plt.subplots(1)
    ax.hist(dataList, bins=nbins, normed=False, alpha=0.5, color='0.5') #
     histogram
    ax.plot(distVals, y, 'k', linewidth=1) # distribution overlaid
    ax.set_xlabel(xLabel)
    ax.set_ylabel('counts')
```

```
376     plt.title(plotTitle, y=1.01)
        # display fitted parameters
        ax.text(0.05, 0.95, textStr2, transform=ax.transAxes, fontsize=12,
       verticalalignment='top', bbox=props)
        plt.savefig('TwoGauss'+fileName, bbox_inches='tight') # print to file

381     # print fit data
        dataStr = plotName+', Summed Gaussian fit, '+str(nbins)+' bins, z
       from '+str(minimum_z)+' to '+str(maximum_z)+', (mm) :\nmu1 = '+str(mu1
       )+', sigma1 = '+str(sigma1)+', mu2 = '+str(mu2)+', sigma2 = '+str(
       sigma2)+'\nchi^2 = '+str(chi2)+', p(chi2) = '+str(pchi2)+'\n'
        print dataStr

    elif fitFunc == 'GaussCauchy': # Gaussian + Cauchy fit
386     # find fitted values
        normedN = [(n[i]/normalise) for i in range(0, nbins)] # normalise
       bins
        mu = np.mean(dataList)
        sigma = np.std(dataList)
        guess = [mu, mu, sigma, sigma] # guess values close to estimated
391
        xMin = min(dataList)
        xMax = max(dataList)

        def GaussPlusCauchy(x, mu, x0, sigma, gamma, cdf=False):
396       if mu > xMin and mu < xMax and gamma > xMin and gamma < xMax:
            y = 0.5*normdist(x, mu, sigma, cdf) + 0.5*cauchydist(x, x0, gamma
       , cdf)
          else:
            y = sys.float_info.max # return large value if parameters in
       unphysical region
          return y
401
        try:
          popt, pcov = optimize.curve_fit(GaussPlusCauchy, binCentres, ydata=
       normedN, p0=guess) # fit to 2 gaussians
        except:
          print 'No suitable fit values found...\n'
406       return
        mu = popt[0] # fitted parameters
        x0 = popt[1]
        sigma = popt[2]
        gamma = popt[3]
411     # plot distribution
        distVals = np.linspace(bins[0], bins[-1] , 300, endpoint=True)
        y = [(normalise * GaussPlusCauchy(distVals[i], mu, x0, sigma, gamma))
         for i in range(0, len(distVals))]
        ax.plot(distVals, y, 'k', linewidth=1) # distribution overlaid
```

```
## chi2 goodness of fit test
# number of counts predicted by CauchyPlusGauss distribution, note
scaled to number of counts
predictedCounts = [(totalN * (GaussPlusCauchy(bins[(i+1)], mu, x0,
sigma, gamma, True) - GaussPlusCauchy(bins[i], mu, x0, sigma, gamma,
True))) for i in range(0, nbins)]
# residuals
residuals = []
for i in range(0, len(n)):
  if predictedCounts[i] > 0:
    residuals.append(math.fabs(predictedCounts[i] - n[i])/math.sqrt(
predictedCounts[i]))
  else:
    residuals.append(math.fabs(predictedCounts[i] - n[i])) # since
zero predicted counts, we want to
#chi2
chi2 = sum([residuals[i]**2 for i in range(0, len(residuals))])
# p-value, (n - 4) degrees of freedom
pchi2 = 1 - stats.chi2.cdf(chi2, (nbins - 4))

# Finish and save plot
pchi2Str = tex_float(pchi2)
chi2Str = tex_float(chi2)
muStr = tex_float(mu)
x0Str = tex_float(x0)
sigmaStr = tex_float(sigma)
gammaStr = tex_float(gamma)
textStr = r'\small Gauss. $+$ Cauchy Dist.'+'\n'+r'$\mu='+muStr+'$\n$
\sigma='+sigmaStr+'$\n$x_0='+x0Str+'$\n$\gamma='+gammaStr+'$\n$\chi^2
='+chi2Str+'$\n$ P(\chi^2) ='+pchi2Str +'$'
textStr2 = r'\small Gauss. $+$ Cauchy Dist.'+'\n'+r'$\mu='+muStr+'$\
n$\sigma='+sigmaStr+'$\n$x_0='+x0Str+'$\n$\gamma='+gammaStr+'$'
# these are matplotlib.patch.Patch properties
props = dict(boxstyle='round', facecolor='None')
# place a text box in upper left in axes coords
ax.text(0.05, 0.95, textStr, transform=ax.transAxes, fontsize=12,
verticalalignment='top', bbox=props)
txt = plt.savefig('chi2GaussCauchy'+fileName, bbox_inches='tight') #
print to file

# remake plot without chi2 label
plt.clf()
fig, ax = plt.subplots(1)
ax.hist(dataList, bins=nbins, normed=False, alpha=0.5, color='0.5') #
 histogram
ax.plot(distVals, y, 'k', linewidth=1) # distribution overlaid
ax.set_xlabel(xLabel)
```

```
    ax.set_ylabel('counts')
    plt.title(plotTitle, y=1.01)
    # display fitted parameters
    ax.text(0.05, 0.95, textStr2, transform=ax.transAxes, fontsize=12,
    verticalalignment='top', bbox=props)
    plt.savefig('GaussCauchy'+fileName, bbox_inches='tight') # print to
    file

    # print fit data
    dataStr = plotName+', Gaussian+Cauchy fit, '+str(nbins)+' bins, z
    from '+str(minimum_z)+' to '+str(maximum_z)+', (mm) :\nmu = '+str(mu)+
    ', sigma = '+str(sigma)+', x0 = '+str(x0)+', gamma = '+str(gamma)+'\
    nchi^2 = '+str(chi2)+', p(chi2) = '+str(pchi2)+'\n'
    print dataStr

  elif fitFunc == 'ThreeGauss': # Three Gaussians fit
    # find fitted values
    normedN = [(n[i]/normalise) for i in range(0, nbins)] # normalise
    bins
    mu = np.mean(dataList)
    sigma = np.std(dataList)
    guess = [mu, mu, mu, sigma, sigma, sigma] # guess values close to
    estimated

    xMin = min(dataList)
    xMax = max(dataList)

    def threeNorms(x, mu1, mu2, mu3, sigma1, sigma2, sigma3, cdf=False):
      if mu1 > xMin and mu1 < xMax and mu2 > xMin and mu2 < xMax and mu3
    > xMin and mu3 < xMax:
        y = (normdist(x, mu1, sigma1, cdf) + normdist(x, mu2, sigma2, cdf
    ) + normdist(x, mu3, sigma3, cdf)) / 3
      else:
        y = sys.float_info.max # return large value if parameters in
    unphysical region
      return y
    try:
      popt, pcov = optimize.curve_fit(threeNorms, binCentres, ydata=
    normedN, p0=guess) # fit to 2 gaussians
    except:
      print 'No suitable fit values found...'
      return
    mu1 = popt[0] # fitted parameters
    mu2 = popt[1]
    mu3 = popt[2]
    sigma1 = popt[3]
    sigma2 = popt[4]
    sigma3 = popt[5]
```

```
# plot distribution
distVals = np.linspace(bins[0], bins[-1] , 300, endpoint=True)
y = [(normalise * threeNorms(distVals[i], mu1, mu2, mu3, sigma1,
sigma2, sigma3)) for i in range(0, len(distVals))]
ax.plot(distVals, y, 'k', linewidth=1) # distribution overlaid

## chi2 goodness of fit test
# number of counts predicted by ThreeGauss distribution, note scaled
to number of counts
predictedCounts = [(totalN * (threeNorms(bins[(i+1)], mu1, mu2, mu3,
sigma1, sigma2, sigma3, True) - threeNorms(bins[i], mu1, mu2, mu3,
sigma1, sigma2, sigma3, True))) for i in range(0, nbins)]
# residuals
residuals = []
for i in range(0, len(n)):
  if predictedCounts[i] > 0:
    residuals.append(math.fabs(predictedCounts[i] - n[i])/math.sqrt(
predictedCounts[i]))
  else:
    residuals.append(math.fabs(predictedCounts[i] - n[i])) # since
zero predicted counts, we want to
#chi2
chi2 = sum([residuals[i]**2 for i in range(0, len(residuals))])
# p-value, (n - 4) degrees of freedom
pchi2 = 1 - stats.chi2.cdf(chi2, (nbins - 4))

# Finish and save plot
pchi2Str = tex_float(pchi2)
chi2Str = tex_float(chi2)
mu1Str = tex_float(mu1)
mu2Str = tex_float(mu2)
mu3Str = tex_float(mu3)
sigma1Str = tex_float(sigma1)
sigma2Str = tex_float(sigma2)
sigma3Str = tex_float(sigma3)

textStr = r'\small 3 $\times$ Gaussian Dist.'+'\n'+r'$\mu_1='+mu1Str+
'$\n$\sigma_1='+sigma1Str+'$\n$\mu_2='+mu2Str+'$\n$\sigma_2='+
sigma2Str+'$\n$\mu_3='+mu3Str+'$\n$\sigma_3='+sigma3Str+'$\n$\chi^2 ='
+chi2Str+'$\n$ P(\chi^2) ='+pchi2Str +'$'
textStr2 = r'\small 3 $\times$ Gaussian Dist.'+'\n'+r'$\mu_1='+mu1Str
+'$\n$\sigma_1='+sigma1Str+'$\n$\mu_2='+mu2Str+'$\n$\sigma_2='+
sigma2Str+'$\n$\mu_3='+mu3Str+'$\n$\sigma_3='+sigma3Str+'$'
# these are matplotlib.patch.Patch properties
props = dict(boxstyle='round', facecolor='None')
# place a text box in upper left in axes coords
ax.text(0.05, 0.95, textStr, transform=ax.transAxes, fontsize=12,
verticalalignment='top', bbox=props)
```

```
    txt = plt.savefig('chi2ThreeGauss'+fileName, bbox_inches='tight') #
    print to file

    # remake plot without chi2 label
    plt.clf()
    fig, ax = plt.subplots(1)
    ax.hist(dataList, bins=nbins, normed=False, alpha=0.5, color='0.5') #
     histogram
    ax.plot(distVals, y, 'k', linewidth=1) # distribution overlaid
    ax.set_xlabel(xLabel)
    ax.set_ylabel('counts')
    plt.title(plotTitle, y=1.01)
    # display fitted parameters
    ax.text(0.05, 0.95, textStr2, transform=ax.transAxes, fontsize=12,
    verticalalignment='top', bbox=props)
    plt.savefig('ThreeGauss'+fileName, bbox_inches='tight') # print to
    file

    # print fit data
    dataStr = plotName+', Summed Gaussian fit, '+str(nbins)+' bins, z
    from '+str(minimum_z)+' to '+str(maximum_z)+', (mm) :\nmu1 = '+str(mu1
    )+', sigma1 = '+str(sigma1)+', mu2 = '+str(mu2)+', sigma2 = '+str(
    sigma2)+', mu_3 = '+str(mu3)+', sigma3 = '+str(sigma3)+'\nchi^2 = '+
    str(chi2)+', p(chi2) = '+str(pchi2)+'\n'
    print dataStr

  else:
    sys.stderr.write('include fit function parameter')
    exit(2)

  plt.close('all')

## PLOT 2D HISTOGRAMS OF DATA WITH CORRELATION COEFFICIENT
def plotTwoVars(list1, list2, xName, yName, xLabel, yLabel, nbins,
    equalAxis):

  """
  list1, list2 - list1 is x-axis value to plot, list2 is y-axis.
  xName, yName - file name is given as <xName>vs<yName><nbins>bins.pdf,
  xLabel, yLabel - x and y axis plots, can use LaTeX characters.
  nbins - number of bins, same for both variables.
  equalAxis - True or False, sets scales the same, use when plotting x vs
     y or p_x vs p_y.

  """
  # plot setup
  plt.clf()
  fig, ax = plt.subplots()
```

```
sys.stderr.write('Plotting Data for ' + xName+ ' against ' + yName + '
  ... \n')
rho = np.corrcoef(list1, list2)[0, 1] # calculate correlation
  coefficient
rhoStr = tex_float(rho)
textStr = r'$\rho='+rhoStr+'$'                                          566
# these are matplotlib.patch.Patch properties
props = dict(boxstyle='round', facecolor='None')
plotName = '$'+yName+'$ vs $'+xName+'$'
if minimum_z == -sys.float_info.max and maximum_z == 0:
   plotRange = r'Whole beam: $-$30cm $ < z < $ 0cm'                     571
   fileName = yName+'vs'+xName+str(nbins)+'bins_wholebeam.pdf'
else:
   plotRange = r'Plot range: $'+tex_float(minimum_z)+r'$mm $< z <'+
  tex_float(maximum_z)+'$mm'
   fileName = yName+'vs'+xName+str(nbins)+'bins'+str(minimum_z)+'-'+str(
  maximum_z)+'mm.pdf'
plotTitle = 'Distribution of '+plotName+' for particles after 4m -      576
  AWAKE baseline\n'+plotRange
   # make bins
x_bins = makeBins(list1, nbins)
y_bins = makeBins(list2, nbins)
# make histogram data
try:                                                                    581
   H, xedges, yedges = np.histogram2d(list1, list2, bins=(x_bins, y_bins
  ))
except:
   sys.stderr.write('numpy error with histogram edges, check on other
  builds')
   return
H = np.transpose(H) # need to transpose data for correct plotting       586
  orientation
# plot
X, Y = np.meshgrid(xedges, yedges)
im = ax.pcolormesh(X, Y, H, cmap='binary')
# add colourbar for counts
cb = plt.colorbar(im, orientation='vertical', pad=0.02, shrink =1)      591
cb.set_label('counts')

plt.xlabel(xLabel)
plt.ylabel(yLabel)
plt.title(plotTitle, y=1.03, x = 0.52 )                                 596

if equalAxis == True: # make axes equally scaled if necessary
   plt.axis('equal')
plt.savefig(fileName, bbox_inches='tight')
                                                                        601
# add correlation coefficient to new plot
```

```
    plt.text(0.05, 0.95, textStr, transform=ax.transAxes, fontsize=12,
      verticalalignment='top', bbox=props)
    plt.savefig('corr'+fileName, bbox_inches='tight')
    plt.close('all')

    # print correlation coefficient to file
    dataStr = xName + ' vs '+yName+', '+str(nbins**2)+' bins, z from '+str(
      minimum_z)+' to '+str(maximum_z)+', (mm) :\nrho = '+str(rho)
    print dataStr

def histWrap1D(nbins, fitFunc): # wrapper function
    """ Specify how many bins and whether 'Gaussian' or 'Cauchy' fit-
      function,
    then relevant plots made. """
    plotHist(bdX, 'x', r'$x$ $/\mu$m', nbins, fitFunc)
    plotHist(bdY, 'y', r'$y$ $/\mu$m', nbins, fitFunc)
    plotHist(bdZ, 'z', r'$z$ $/\mu$m', nbins, 'None' )
    plotHist(bdE, 'E', '$E$ /GeV', nbins, fitFunc)
    plotHist(bdPx, 'p_x', '$p_x$ /rad', nbins, fitFunc)
    plotHist(bdPy, 'p_y', '$p_y$ /rad', nbins, fitFunc)

def histWrap2D(nbins): # wrapper function
    """ Specify number of bins and function makes relevant 2D histograms
      """
    plotTwoVars(bdX, bdY, 'x', 'y', r'$x$ $/\mu$m', r'$y$ $/\mu$m', nbins,
      True)
    plotTwoVars(bdPx, bdPy, 'p_x', 'p_y', '$p_x$ /rad', '$p_y$ /rad', nbins
      , True)
    plotTwoVars(bdX, bdPx, 'x', 'p_x', r'$x$ $/\mu$m', '$p_x$ /rad', nbins,
       False)
    plotTwoVars(bdY, bdPy, 'y', 'p_y', r'$y$ $/\mu$m', '$p_y$ /rad', nbins,
       False)
    plotTwoVars(bdX, bdPy, 'x', 'p_y', r'$x$ $/\mu$m', '$p_y$ /rad', nbins,
       False)
    plotTwoVars(bdY, bdPx, 'y', 'p_x', r'$y$ $/\mu$m', '$p_x$ /rad', nbins,
       False)
    plotTwoVars(bdZ, bdE, 'z', 'E', r'$z$ $/\mu$m', '$E$ /GeV', nbins,
      False)
    plotTwoVars(bdZ, bdX, 'z', 'x', r'$z$ $/\mu$m', '$x$ $/\mu$m', nbins,
      False)
    plotTwoVars(bdZ, bdY, 'z', 'y', r'$z$ $/\mu$m', '$y$ $/\mu$m', nbins,
      False)
    plotTwoVars(bdZ, bdPx, 'z', 'p_x', r'$z$ $/\mu$m', '$p_x$ /rad', nbins,
       False)
    plotTwoVars(bdZ, bdPy, 'z', 'p_y', r'$z$ $/\mu$m', '$p_y$ /rad', nbins,
       False)

## MAIN METHOD - SPECIFY WHAT TO PLOT HERE
```

```
def main(argv):
  # setup
  usageMessage = sys.argv[0] + " USAGE: \n     " + sys.argv[0] + "  <
    $PRIMARIESFILE  \n  "
  # store beamfile
  sys.stderr.write('Importing Beamfile...')
  BEAMFILE = sys.stdin.readlines()
  if len(BEAMFILE) == 0:
    sys.stderr.write(sys.argv[0] + ' error: Input Beamfile is empty \n')
    sys.exit(2)

  ### PLOTS/ANALYSIS

  # Array with z-ranges [zMax, zMin] in mum
  zVals = [[-i+2000,-i-2000] for i in range(2000, 300000, 20000)]


  # Loop over z-ranges of interest
  for i in range(0, len(zVals)):
    # Get zValues
    zRange = zVals[i]
    zMax = zRange[0]
    zMin = zRange[1]
    # import data,
    importData(BEAMFILE, zMax, zMin)

    # 1D Histograms
    for nbins in [100, 200]:
      histWrap1D(nbins, 'Gaussian')
      histWrap1D(nbins, 'Cauchy')
      histWrap1D(nbins, 'ThreeGauss')
      histWrap1D(nbins, 'TwoGauss')
      histWrap1D(nbins, 'GaussCauchy')
      histWrap1D(nbins, 'None')

    # 2D Histograms
    for nbins in [30, 50, 100, 200]:
      histWrap2D(nbins)


## RUN THE MAIN METHOD
if __name__ == "__main__":
   main(sys.argv[1:])
```

B.4 analyseBeamSize.py

```
#!/home/dervovic/anaconda/bin/python
#-*- coding: utf-8 -*-

# Script to analyse growth of the beam over vacuum. Uses output.txt and
    output.primaries.txt
# USAGE: ./analyseBeamSize.py > $STATISTICS

# Written by Danial Dervovic, 17 March 2015

## SETUP

import sys
import numpy as np
import math

## IMPORT DATA

def importData(beamfile, fileType, zMax=0, zMin=-sys.float_info.max):
  """ Import data into appropriate data structures """
  numberOfLines = len(beamfile)
  lineCounter = 0
  # clear data arrays
  bdParticlesList =[]
  if zMax==0 and zMin==-sys.float_info.max:
    importDataStr = 'Processing data for whole beam...\n'
  else:
    importDataStr = 'Processing data for beam from z = '+str(zMin)+' to '
   +str(zMax)+' mum... \n'
  sys.stderr.write(importDataStr)
  ## Unpack BDSim beam data, and put into bdParticlesList and individual
    Lists
  for line in beamfile:
    lineCounter += 1
    if line[0].isdigit() == False: # leave lines with no data
      continue
    if lineCounter <= 4: # remove reference particles
      continue
    #sys.stderr.write('Parsing data for line '+str(lineCounter)+' of ' +
   str(numberOfLines) + ' in beamfile. \r')
    newline = line.split() # split line into List
    # Check if particle in range
    if fileType == 'primaries':
      zIndex = 4
      xpIndex = 7
      ypIndex = 7
    elif fileType == 'output':
      zIndex = 7
```

```
      xpIndex = 15
      ypIndex = 16
    else:
      sys.stderr.write('no fileType specified, use output or primaries.')
      exit(2)

    if float(newline[zIndex]) < zMax and float(newline[zIndex]) > zMin:
      particleData = [] # new data List
      # put in data from beamfile
      particleData.append(float(newline[1])) # Energy in GeV
      particleData.append(float(newline[2])) # x in mum
      particleData.append(float(newline[3])) # y in mum
      particleData.append(float(newline[4])) # z in mum
      particleData.append(float(newline[xpIndex])) # xp in rad
      particleData.append(float(newline[ypIndex])) # yp in rad
      bdParticlesList.append(particleData)
    else:
      continue
  sys.stderr.write('\n')
  return bdParticlesList

## ANALYSIS
def compareDataSets(primariesData, outputData):
  # get data

  primariesX = []
  primariesY = []
  for i in range(0, len(primariesData)):
    dataPoint = primariesData[i]
    primariesX.append(dataPoint[1])
    primariesY.append(dataPoint[2])

  outputX = []
  outputY = []
  for i in range(0, len(outputData)):
    dataPoint = outputData[i]
    outputX.append(dataPoint[1])
    outputY.append(dataPoint[2])

  primariesR = [math.sqrt(primariesX[i]**2 + primariesY[i]**2) for i in
    range(0, len(primariesX))]
  outputR = [math.sqrt(outputX[i]**2 + outputY[i]**2) for i in range(0,
    len(outputX))]

  # Averages

  primariesMeanR = np.mean(primariesR)
  primariesStdDevR = np.std(primariesR)
```

```
  primariesRError = primariesStdDevR / math.sqrt(len(primariesR))
  primariesFracError = primariesRError / primariesMeanR

  outputMeanR = np.mean(outputR)
  outputStdDevR = np.std(outputR)
  outputRError = outputStdDevR / math.sqrt(len(outputR))
  outputFracError = outputRError / outputMeanR

  beamSizeRatio = outputMeanR / primariesMeanR
  errorInRatio = math.sqrt(beamSizeRatio**2 * (primariesFracError**2 +
    outputFracError**2))

  primariesStatistics = [primariesMeanR, primariesStdDevR,
    primariesRError]
  outputStatistics = [outputMeanR, outputStdDevR, outputRError]
  combinedStatistics = [beamSizeRatio, errorInRatio]

  sys.stdout.write('primaries: [mean r, stddev r, error on mean r]\n'+str
    (primariesStatistics)+'\n')
  sys.stdout.write('output: [mean r, stddev r, error on mean r]\n'+str(
    outputStatistics)+'\n')
  sys.stdout.write('combined: [beamSizeRatio, errorInRatio]\n'+str(
    combinedStatistics)+'\n')


## MAIN METHOD - SPECIFY WHAT AREA OF BEAM TO ANALYSE HERE
def main(argv):
  # setup
  usageMessage = sys.argv[0] + " USAGE: \n    " + sys.argv[0] + " Need
    output.txt and output.primaries.txt from BDSIM in working directory  \
    n  "
  # store beamfile
  sys.stderr.write('Importing Beamfile 1...\n')
  try:
    outputFile = open('output.txt', 'rw+')
  except IOError:
    sys.stderr.write('Error: No file output.txt\n')
    exit(2)
  outputFileArray = outputFile.readlines()
  sys.stderr.write('Importing Beamfile 2...\n')
  try:
    primariesFile = open('output.primaries.txt', 'rw+')
  except IOError:
    sys.stderr.write('Error: No file output.primaries.txt\n')
    exit(2)
  primariesFileArray = primariesFile.readlines()

  if len(primariesFileArray) == 0 or len(outputFileArray) == 0:
```

```
    sys.stderr.write(usageMessage)
    sys.exit(2)

  ### PLOTS/ANALYSIS

  # Array with z-ranges [zMax, zMin] in mum
  zVals = [[0,-sys.float_info.max],[-130000,-132000]] # corresponds to
    whole beam and most stable region

  # Loop over z-ranges of interest
  for i in range(0, len(zVals)):
    # Get zValues
    zRange = zVals[i]
    zMax = zRange[0]
    zMin = zRange[1]
    # import data,
    sys.stdout.write('Data for '+str(zMin)+' < z < '+str(zMax)+'mum\n')
    outputData = importData(outputFileArray, 'output', zMax, zMin)
    sys.stdout.write(str(len(outputData))+'\n')
    primariesData = importData(primariesFileArray, 'primaries', zMax,
    zMin)
    compareDataSets(primariesData, outputData)

## RUN THE MAIN METHOD
if __name__ == "__main__":
   main(sys.argv[1:])
```