2-1-2016

Studies of Coherent Synchrotron Radiation with the Discontinuous Galerkin Method

David Bizzozero

David Bizzozero
Candidate

Mathematics and Statistics
Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

Stephen Lau, Chairperson

James Ellison

Daniel Appelö

Robert Warnock

Klaus Heinemann

Studies of Coherent Synchrotron Radiation with the Discontinuous Galerkin Method

by

David A. Bizzozero

M.S., Applied Mathematics
University of New Mexico, 2010

THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Mathematics

The University of New Mexico

Albuquerque, New Mexico

December, 2015

Dedication

To Oscar and Nora

Acknowledgments

Foremost, I would like to thank Jim Ellison for his invaluable dedication towards the progress of my graduate career. I would also like to thank Stephen Lau for his expert insight in advanced mathematical concepts, Robert Warnock for his vast experience in accelerators and beam dynamics, Daniel Appelö for his helpful discussions on discontinous Galerkin schemes, Klaus Heinemann for general comments on the thesis, and Dann Brewer for his assistance in the setup of our GPU-enabled computing desktop. This material is based upon work primarily supported by the U.S. Department of Energy, Office of Science, Office of High Energy Physics under Award Number DE-FG02-99ER41104 and supported in part by the National Science Foundation under Grant No. NSF PHY 0855678.

Studies of Coherent Synchrotron Radiation with the Discontinuous Galerkin Method

by

David A. Bizzozero

M.S., Applied Mathematics
University of New Mexico, 2010

Ph.D., Mathematics, University of New Mexico, 2015

Abstract

In this thesis, we present methods for integrating Maxwell's equations in Frenet-Serret coordinates in several settings using discontinuous Galerkin (DG) finite element method codes in 1D, 2D, and 3D. We apply these routines to the study of coherent synchrotron radiation, an important topic in accelerator physics. We build upon the published computational work of T. Agoh and D. Zhou in solving Maxwell's equations in the frequency-domain using a paraxial approximation which reduces Maxwell's equations to a Schrödinger-like system. We also evolve Maxwell's equations in the time-domain using a Fourier series decomposition with 2D DG motivated by an experiment performed at the Canadian Light Source. A comparison between theory and experiment has been published (Phys. Rev. Lett. **114**, 204801 (2015)). Lastly, we devise a novel approach to integrating Maxwell's equations with 3D DG using a Galilean transformation and demonstrate proof-of-concept. In the above studies, we examine the accuracy, efficiency, and convergence of DG.

Contents

List of Figures

Chapter 1

Introduction

1.1 Scope of Thesis

This thesis consists of a hybrid analysis of Coherent Synchrotron Radiation (CSR) in the context of accelerators along with a study of Discontinuous Galerkin (DG) schemes in these applications. We aim to provide a clear overview of the physical systems, their mathematical representations, and their numerical solutions arising from applications in beam dynamics.

1.1.1 Coherent Synchrotron Radiation Overview

Our starting point for the thesis work are the well-established Maxwell's Equations. Through the use of transformations and approximations, we construct various approaches to analyzing the electromagnetic fields produced, in a vacuum chamber of varying geometry, in both the frequency and time domain. We build upon the computational CSR analysis done by T. Agoh et al [1, 2], D. Zhou et al [22, 23], and G. Stupakov et al [20].

The mechanism for producing synchrotron radiation in this thesis are relativistic bunches of electrons whose trajectory is altered by an external magnetic field. The external magnetic fields we consider are assumed hard-edged and constant. This results in the electron bunch orbits to be comprised of straight sections, where no radiation is emitted, and arcs of circles where the radiation is produced. The coherency in CSR assumes that the electron bunch size is smaller than the wavelength of the radiation produced; thus the electrons radiate in phase providing constructive interference for the radiation.

While the production of CSR may be desirable in some experiments, CSR can cause difficulties in bunch compression and cooling of the external structures housing the electron beam. The study on how to mitigate CSR effects is of great importance in the operation of current facilities and construction of new facilities.

1.1.2 Discontinuous Galerkin Overview

In the numerical implementation used in this thesis, we adopt the discontinuous Galerkin method, a high-order method which shares features with both finite element and finite volume methods. It has seen rapid development with a myriad of applications over the past 15 years, however, we are not aware of its use in the beam physics community. We therefore give a short overview of our approaches, closely following the treatment of [11].

The DG formulations in this thesis involve decomposition of a computational domain into simplex elements, local representations of the relevant operators and solution on each element, and coupling of adjacent elements through flux terms along common boundaries. The element-wise local solutions are represented by polynomials, but these local solutions may be discontinuous across elements. See [7, 11, 13, 18] for more thorough and general treatments of DG formulations.

1.2 Summary of Chapters

For chapters 3–6, we introduce the necessary background material in chapter 2. In all formulations of Maxwell's equations, the electromagnetic fields and sources are functions of 4 variables; spatial, temporal, or parametric. We divide the study of CSR using DG into 4 parts: chapters 3 and 4 address Maxwell's equations in the frequency-domain while chapters 5 and 6 study the time-domain.

Additionally, we separate these 4 chapters by the dimension of PDE they address. In chapter 3 we evolve PDEs in space in the frequency domain with 1 independent spatial variable and 2 parameters arising in a 1D formulation. In chapter 4 we also evolve PDEs in space in the frequency domain, however, with 2 spatial independent variables and 1 parameter arising in a 2D formulation. In chapter 5 we evolve PDEs in the time-domain with 2 spatial independent variables and 1 parameter also arising in a 2D formulation. In chapter 6 we evolve PDEs in the time-domain with 3 spatial independent variables arising in a full 3D formulation. Lastly, chapter 7 is dedicated to discussions on the results and possible future work. We also discuss a few difficulties encountered in the thesis studies here.

Chapter 2 – To set the stage for the study of CSR using DG formulations, we begin with a thorough analysis of Maxwell's equations and their equivalent formulations. We also present a few coordinate transformations and integral transforms to solve and analyze the solutions of Maxwell's equations in various geometries. Additionally, we derive the paraxial approximation, useful for frequency-domain analysis.

Chapter 3 – This chapter is a typographically edited version of our submission [6] to the Free Electron Laser Conference 2014 in Basel, Switzerland. The document has been modified for clarity and consistency with the rest of the thesis. Our goal in this chapter is to examine CSR in the frequency domain using 1D code simulations by use of a Fourier series. We adopt a toroidal vacuum chamber segment of rectangular cross section of uniform curvature with infinite straight segments adjoining each end.

Chapter 4 – This chapter is a typographically edited version of our submission [5] to the Free Electron Laser Conference 2013 in New York. The document has also been modified for clarity and consistency with the rest of the thesis. Chronologically, the content of this chapter predates that of chapter 3; however, here we present the frequency-domain method without the Fourier series. Here, our goal is to accurately compute the electric fields to compare to [22, 23] using DG. We use the same vacuum chamber model as in chapter 3.

Chapter 5 – In this chapter, we examine methods for simulating wake fields in a flared vacuum chamber in the time-domain. Our goal is to accurately represent the physical vacuum chamber at the Canadian Light Source (CLS) and apply numerical techniques to solve for the fields given a prescribed electron bunch profile and initial field. The numerical parts of this work were published in [4].

Chapter 6 – In this chapter, we examine a different approximation to modeling the fields in a full 3D volume. We combine the Frenet-Serret and Galilean transformations to obtain new evolution PDEs for the 6 Maxwell field components. This approach is useful when the sources are localized, and the fields away from the source are not a quantity of interest. Portions of this work were presented at the International Conference on Spectral and High-Order Methods (ICOSAHOM) 2014 in Salt Lake City, UT.

Chapter 7 – In this chapter, we organize the results and conclusions of chapters 3–6. We discuss the viability of DG versus finite difference (FD) schemes in the various cases and their efficiency. Furthermore, we examine the difficulties encountered in programming the various codes used in our analyses. Lastly, we discuss possible directions for future work.

Chapter 2

Maxwell's Equations and Transformations

2.1 Maxwell's Equations

In this section we prove equivalence for the initial value problem of Maxwell's equations in two different forms as well as in the wave equation form. The goal of the equivalence is to allow for algorithms which use one form of the equations to automatically provide the solution to the other forms. Also, we consider only the full space domain $\mathbb{R}^3$ and defer details on boundary conditions to section 2.2.2.

2.1.1 Statement of Maxwell's Equations

We begin with a prescribed charge and current density $(\rho(\mathbf{R},t), \mathbf{j}(\mathbf{R},t))$, in Cartesian coordinates $\mathbf{R} := (Z, X, Y)^\top$, which satisfy the continuity equation:

$$\frac{\partial \rho}{\partial t} = -\boldsymbol{\nabla} \cdot \mathbf{j}(\mathbf{R}, t), \tag{2.1}$$

With the given charge and current densities satisfying (2.1), we pose the initial value problem for Maxwell's equations, for the electric and magnetic fields $\mathbf{E}(\mathbf{R},t)$ and $\mathbf{B}(\mathbf{R},t)$, as:

$$\frac{\partial \mathbf{E}}{\partial t} = c^2 \boldsymbol{\nabla} \times \mathbf{B} - cZ_0 \mathbf{j}(\mathbf{R},t), \tag{2.2}$$

$$\frac{\partial \mathbf{B}}{\partial t} = -\boldsymbol{\nabla} \times \mathbf{E}, \tag{2.3}$$

$$0 = \boldsymbol{\nabla} \cdot \mathbf{E} - cZ_0 \rho(\mathbf{R},t), \tag{2.4}$$

$$0 = \boldsymbol{\nabla} \cdot \mathbf{B}, \tag{2.5}$$

$$\mathbf{E}(\mathbf{R},0) = \mathbf{E}_0(\mathbf{R}), \tag{2.6}$$

$$\mathbf{B}(\mathbf{R},0) = \mathbf{B}_0(\mathbf{R}), \tag{2.7}$$

where $c = 1/\sqrt{\mu_0 \epsilon_0}$ is the speed of light and $Z_0 = \sqrt{\mu_0/\epsilon_0}$ is the free space impedance in SI units. We call (2.2)–(2.3) Maxwell's evolution equations, (2.4)–(2.5) Maxwell's constitutive equations, and (2.6)–(2.7) the initial conditions. The continuity equation (2.1) is derived by taking the divergence of (2.2) and applying (2.4).

Note, the constraints given in (2.4)–(2.5) need only be imposed on the initial data (2.6)–(2.7). This is proved in the following theorem:

Theorem 1 *If the fields,* $(\mathbf{E}(\mathbf{R},t), \mathbf{B}(\mathbf{R},t))$*, satisfy* (2.2)–(2.3) *and at* $t = 0$ *satisfy* (2.4)–(2.5)*, then* $(\mathbf{E}(\mathbf{R},t), \mathbf{B}(\mathbf{R},t))$ *automatically satisfy* (2.4)–(2.5) *for all time.*

Proof: Taking the divergence of (2.2) and applying the continuity equation (2.1) we obtain:

$$\begin{aligned}
\boldsymbol{\nabla} \cdot \frac{\partial \mathbf{E}}{\partial t} &= c^2 \boldsymbol{\nabla} \cdot \boldsymbol{\nabla} \times \mathbf{B} - cZ_0 \boldsymbol{\nabla} \cdot \mathbf{j} \\
\boldsymbol{\nabla} \cdot \frac{\partial \mathbf{E}}{\partial t} &= -cZ_0 \boldsymbol{\nabla} \cdot \mathbf{j} \\
\boldsymbol{\nabla} \cdot \frac{\partial \mathbf{E}}{\partial t} &= cZ_0 \frac{\partial \rho}{\partial t} \\
\frac{\partial}{\partial t}\left(\boldsymbol{\nabla} \cdot \mathbf{E} - cZ_0 \rho\right) &= 0. \qquad (2.8)
\end{aligned}$$

Thus $\boldsymbol{\nabla} \cdot \mathbf{E} - cZ_0\rho$ is constant in time. However, since $\mathbf{E}_0(\mathbf{R})$ satisfies (2.4), this constant is zero at $t = 0$. Therefore $\boldsymbol{\nabla} \cdot \mathbf{E} - cZ_0\rho = 0$ for all time, establishing (2.4). Similarly, taking the divergence of (2.3) we obtain:

$$\begin{aligned} \boldsymbol{\nabla} \cdot \frac{\partial \mathbf{B}}{\partial t} &= -\boldsymbol{\nabla} \cdot \boldsymbol{\nabla} \times \mathbf{E} \\ \boldsymbol{\nabla} \cdot \frac{\partial \mathbf{B}}{\partial t} &= 0 \\ \frac{\partial}{\partial t}\big(\boldsymbol{\nabla} \cdot \mathbf{B}\big) &= 0. \end{aligned} \tag{2.9}$$

Thus $\boldsymbol{\nabla} \cdot \mathbf{B}$ is constant in time. However, since $\mathbf{B}_0(\mathbf{R})$ satisfies (2.5), this constant is zero at $t = 0$. Therefore $\boldsymbol{\nabla} \cdot \mathbf{B} = 0$ for all time, establishing (2.5). □

For the proofs in the next section, we will assume the fields $(\mathbf{E}(\mathbf{R}, t), \mathbf{B}(\mathbf{R}, t))$ and sources $(\rho(\mathbf{R}, t), \mathbf{j}(\mathbf{R}, t))$ are smooth. However, this restriction to smooth fields and sources can be lifted by employing distribution theory not covered in the scope of this thesis. Furthermore, we do not address boundary conditions until section 2.2.2 and thus consider only the full space problem.

2.1.2 Wave Equation Forms

In this section we derive the wave equations for $(\mathbf{E}(\mathbf{R}, t), \mathbf{B}(\mathbf{R}, t))$ and state the standard existence and uniqueness theorem. We use this result to prove existence and uniqueness for the Maxwell equations in the form (2.2)–(2.3).

To derive the wave equation for $\mathbf{E}$, we begin by taking the curl of (2.3), applying vector identities, and substituting (2.2) and (2.4):

$$\begin{aligned}
\boldsymbol{\nabla} \times (\boldsymbol{\nabla} \times \mathbf{E}) &= -\boldsymbol{\nabla} \times \frac{\partial \mathbf{B}}{\partial t} \\
\boldsymbol{\nabla}(\boldsymbol{\nabla} \cdot \mathbf{E}) - \boldsymbol{\nabla}^2 \mathbf{E} &= -\frac{\partial}{\partial t}(\boldsymbol{\nabla} \times \mathbf{B}) \\
\boldsymbol{\nabla}(\boldsymbol{\nabla} \cdot \mathbf{E}) - \boldsymbol{\nabla}^2 \mathbf{E} &= -\frac{\partial}{\partial t}\left(\frac{1}{c^2}\frac{\partial \mathbf{E}}{\partial t} + \frac{Z_0}{c}\mathbf{j}\right) \\
\boldsymbol{\nabla}(\boldsymbol{\nabla} \cdot \mathbf{E}) - \boldsymbol{\nabla}^2 \mathbf{E} &= -\frac{1}{c^2}\frac{\partial^2 \mathbf{E}}{\partial t^2} - \frac{Z_0}{c}\frac{\partial \mathbf{j}}{\partial t} \\
cZ_0 \boldsymbol{\nabla}\rho - \boldsymbol{\nabla}^2 \mathbf{E} &= -\frac{1}{c^2}\frac{\partial^2 \mathbf{E}}{\partial t^2} - \frac{Z_0}{c}\frac{\partial \mathbf{j}}{\partial t} \\
\frac{1}{c^2}\frac{\partial^2 \mathbf{E}}{\partial t^2} - \boldsymbol{\nabla}^2 \mathbf{E} &= -cZ_0 \boldsymbol{\nabla}\rho - \frac{Z_0}{c}\frac{\partial \mathbf{j}}{\partial t}. \qquad (2.10)
\end{aligned}$$

Similarly, to derive the wave equation for $\mathbf{B}$, we take the curl of (2.3), apply appropriate vector identities, and substitute (2.2) and (2.5):

$$\begin{aligned}
\boldsymbol{\nabla} \times (\boldsymbol{\nabla} \times \mathbf{B}) &= \frac{1}{c^2}\boldsymbol{\nabla} \times \frac{\partial \mathbf{E}}{\partial t} + \frac{Z_0}{c}\boldsymbol{\nabla} \times \mathbf{j} \\
\boldsymbol{\nabla}(\boldsymbol{\nabla} \cdot \mathbf{B}) - \boldsymbol{\nabla}^2 \mathbf{B} &= \frac{1}{c^2}\frac{\partial}{\partial t}(\boldsymbol{\nabla} \times \mathbf{E}) + \frac{Z_0}{c}\boldsymbol{\nabla} \times \mathbf{j} \\
\boldsymbol{\nabla}(\boldsymbol{\nabla} \cdot \mathbf{B}) - \boldsymbol{\nabla}^2 \mathbf{B} &= -\frac{1}{c^2}\frac{\partial^2 \mathbf{B}}{\partial t^2} + \frac{Z_0}{c}\boldsymbol{\nabla} \times \mathbf{j} \\
\frac{1}{c^2}\frac{\partial^2 \mathbf{B}}{\partial t^2} - \boldsymbol{\nabla}^2 \mathbf{B} &= \frac{Z_0}{c}\boldsymbol{\nabla} \times \mathbf{j}. \qquad (2.11)
\end{aligned}$$

We now define the initial value problems for $\mathbf{E}(\mathbf{R}, t)$ and $\mathbf{B}(\mathbf{R}, t)$ separately for arbitrary but smooth $(\rho(\mathbf{R}, t), \mathbf{j}(\mathbf{R}, t))$ and initial conditions:

$$\begin{aligned}
\frac{1}{c^2}\frac{\partial^2 \mathbf{E}}{\partial t^2} - \boldsymbol{\nabla}^2 \mathbf{E} &= -\, cZ_0 \boldsymbol{\nabla}\rho - \frac{Z_0}{c}\frac{\partial \mathbf{j}}{\partial t}, & \frac{1}{c^2}\frac{\partial^2 \mathbf{B}}{\partial t^2} - \boldsymbol{\nabla}^2 \mathbf{B} &= \frac{Z_0}{c}\boldsymbol{\nabla} \times \mathbf{j}, \qquad (2.12) \\
\mathbf{E}(\mathbf{R}, 0) &= \mathbf{E}_0(\mathbf{R}), & \mathbf{B}(\mathbf{R}, 0) &= \mathbf{B}_0(\mathbf{R}), \qquad (2.13) \\
\frac{\partial}{\partial t}\mathbf{E}(\mathbf{R}, t)\Big|_{t=0} &= \mathbf{E}_1(\mathbf{R}), & \frac{\partial}{\partial t}\mathbf{B}(\mathbf{R}, t)\Big|_{t=0} &= \mathbf{B}_1(\mathbf{R}). \qquad (2.14)
\end{aligned}$$

Together, these initial value problems can be paired as:

$$\frac{1}{c^2}\frac{\partial^2 \mathbf{U}}{\partial t^2} - \boldsymbol{\nabla}^2\mathbf{U} = \mathbf{S}(\mathbf{R}, t), \tag{2.15}$$

$$\mathbf{U}(\mathbf{R}, 0) = \mathbf{U}_0(\mathbf{R}), \tag{2.16}$$

$$\frac{\partial}{\partial t}\mathbf{U}(\mathbf{R}, t)\Big|_{t=0} = \mathbf{U}_1(\mathbf{R}). \tag{2.17}$$

with $\mathbf{U}$ representing either $\mathbf{E}$ or $\mathbf{B}$, and $\mathbf{S}$ as the appropriate source term from the right-hand-side of (2.10) or (2.11).

Theorem 2 *The initial value problem of* (2.15)–(2.17) *has a unique solution* $\mathbf{U}(\mathbf{R}, t)$ *given smooth initial displacement* $\mathbf{U}_0(\mathbf{R})$*, initial velocity* $\mathbf{U}_1(\mathbf{R})$*, and source* $\mathbf{S}(\mathbf{R}, t)$*.*

Proof: Applying Kirchhoff's formula and Theorems 2, 4, and 5 in [9] (p72-82), the unique (see [16] for uniqueness) solution to (2.15)–(2.17) in Cartesian coordinates is:

$$\begin{aligned}\mathbf{U}(\mathbf{R}, t) = &\frac{1}{4\pi}\int_{|\mathbf{R}'|<ct} \frac{\mathbf{S}(\mathbf{y}, t - \frac{|\mathbf{R}'-\mathbf{R}|}{c})}{|\mathbf{R}' - \mathbf{R}|} d^3\mathbf{R}' \\ &+ \frac{1}{4\pi c^2 t^2}\int_{|\mathbf{R}'|=ct} \big[(\mathbf{R}' - \mathbf{R}) \cdot \boldsymbol{\nabla}\big]\mathbf{U}_0(\mathbf{R}') + \mathbf{U}_0(\mathbf{R}') + t\mathbf{U}_1(\mathbf{R}') dS(\mathbf{R}'),\end{aligned} \tag{2.18}$$

with the notation: $\big[\mathbf{b} \cdot \boldsymbol{\nabla}\big]\mathbf{A}(\mathbf{R}') := (\mathbf{b} \cdot \boldsymbol{\nabla} A_Z(\mathbf{R}'), \mathbf{b} \cdot \boldsymbol{\nabla} A_X(\mathbf{R}'), \mathbf{b} \cdot \boldsymbol{\nabla} A_Y(\mathbf{R}'))^\top$. We emphasize this solution is defined using Cartesian coordinates. $\square$

Next, to consider the existence and uniqueness of solutions to Maxwell's Equations, we must impose extra restrictions on the initial conditions and source terms:

$$\boldsymbol{\nabla} \cdot \mathbf{E}_0(\mathbf{R}) = cZ_0\rho(\mathbf{R}, 0), \tag{2.19}$$

$$\boldsymbol{\nabla} \cdot \mathbf{B}_0(\mathbf{R}) = 0, \tag{2.20}$$

$$\mathbf{E}_1(\mathbf{R}) = c^2\boldsymbol{\nabla} \times \mathbf{B}_0(\mathbf{R}) - cZ_0\mathbf{j}(\mathbf{R}, 0), \tag{2.21}$$

$$\mathbf{B}_1(\mathbf{R}) = -\boldsymbol{\nabla} \times \mathbf{E}_0(\mathbf{R}), \tag{2.22}$$

The additional requirements (2.19)–(2.22), which couple $\mathbf{E}(\mathbf{R}, t)$ and $\mathbf{B}(\mathbf{R}, t)$, are necessary to satisfy the Maxwell's equations in the form of (2.2)–(2.3).

Theorem 3 *The initial value problems of* (2.12)–(2.14) *with the restrictions* (2.19)–(2.22) *have a unique solution* $(\mathbf{E}(\mathbf{R}, t), \mathbf{B}(\mathbf{R}, t))$ *which is also the unique solution to Maxwell's equations* (2.2)–(2.3).

Proof: Part 1 (Existence)

We begin by applying Theorem 2 to construct the solutions $(\mathbf{E}(\mathbf{R}, t), \mathbf{B}(\mathbf{R}, t))$ from the initial value problem (2.12)–(2.14) subject to the restrictions on the source and initial data (2.19)–(2.22). Since (2.4)–(2.5) follow directly from the restrictions posed in (2.19)–(2.22) we must only show (2.2)–(2.3) are satisfied by $(\mathbf{E}, \mathbf{B})$.

To show (2.3) is satisfied, we take the curl of the wave equation for $\mathbf{E}$ and substitute the wave equation for $\mathbf{B}$ for the right-hand-side in (2.12):

$$\begin{aligned}
\frac{1}{c^2}\boldsymbol{\nabla}\times\frac{\partial^2\mathbf{E}}{\partial t^2} - \boldsymbol{\nabla}\times\boldsymbol{\nabla}^2\mathbf{E} &= -cZ_0\boldsymbol{\nabla}\times\boldsymbol{\nabla}\rho - \frac{Z_0}{c}\boldsymbol{\nabla}\times\frac{\partial\mathbf{j}}{\partial t}\\
\frac{1}{c^2}\frac{\partial^2}{\partial t^2}\left(\boldsymbol{\nabla}\times\mathbf{E}\right) - \boldsymbol{\nabla}^2\left(\boldsymbol{\nabla}\times\mathbf{E}\right) &= -\frac{\partial}{\partial t}\left(\frac{Z_0}{c}\boldsymbol{\nabla}\times\mathbf{j}\right)\\
\frac{1}{c^2}\frac{\partial^2}{\partial t^2}\left(\boldsymbol{\nabla}\times\mathbf{E}\right) - \boldsymbol{\nabla}^2\left(\boldsymbol{\nabla}\times\mathbf{E}\right) &= -\frac{\partial}{\partial t}\left(\frac{1}{c^2}\frac{\partial^2\mathbf{B}}{\partial t^2} - \boldsymbol{\nabla}^2\mathbf{B}\right)\\
\frac{1}{c^2}\frac{\partial^2}{\partial t^2}\left(\boldsymbol{\nabla}\times\mathbf{E} + \frac{\partial\mathbf{B}}{\partial t}\right) &= \boldsymbol{\nabla}^2\left(\boldsymbol{\nabla}\times\mathbf{E} + \frac{\partial\mathbf{B}}{\partial t}\right). \qquad (2.23)
\end{aligned}$$

We now have a wave equation for $\mathbf{U} \equiv \boldsymbol{\nabla}\times\mathbf{E} + \partial\mathbf{B}/\partial t$ with $\mathbf{S} = \mathbf{0}$. We also have $\mathbf{U}_0 \equiv \boldsymbol{\nabla}\times\mathbf{E}_0 + \mathbf{B}_1 = \mathbf{0}$ by (2.22). We must only show $\mathbf{U}_1 = \mathbf{0}$ to assert (2.3) is satisfied by invoking Theorem 2 again. Examining $\mathbf{U}_1$ and using (2.21) we note:

$$\begin{aligned}\mathbf{U}_1 &\equiv \boldsymbol{\nabla}\times\mathbf{E}_1 + \left.\frac{\partial^2\mathbf{B}}{\partial t^2}\right|_{t=0}\\ &= \boldsymbol{\nabla}\times\big(c^2\boldsymbol{\nabla}\times\mathbf{B}_0 - cZ_0\mathbf{j}(\mathbf{R},0)\big) + \left.\frac{\partial^2\mathbf{B}}{\partial t^2}\right|_{t=0}\\ &= c^2\boldsymbol{\nabla}\times(\boldsymbol{\nabla}\times\mathbf{B}_0) - cZ_0\boldsymbol{\nabla}\times\mathbf{j}(\mathbf{R},0) + \left.\frac{\partial^2\mathbf{B}}{\partial t^2}\right|_{t=0}\\ &= c^2\boldsymbol{\nabla}(\boldsymbol{\nabla}\cdot\mathbf{B}_0) - c^2\boldsymbol{\nabla}^2\mathbf{B}_0 - cZ_0\boldsymbol{\nabla}\times\mathbf{j}(\mathbf{R},0) + \left.\frac{\partial^2\mathbf{B}}{\partial t^2}\right|_{t=0}\\ &= -c^2\boldsymbol{\nabla}^2\mathbf{B}_0 - cZ_0\boldsymbol{\nabla}\times\mathbf{j}(\mathbf{R},0) + \left.\frac{\partial^2\mathbf{B}}{\partial t^2}\right|_{t=0}.\end{aligned}$$

Thus $\mathbf{U}_1 = \mathbf{0}$ since $\mathbf{B}(\mathbf{R},t)$ satisfies the wave equation (2.12) by construction. Therefore, $\mathbf{U} \equiv \boldsymbol{\nabla}\times\mathbf{E} + \partial\mathbf{B}/\partial t = \mathbf{0}$ proving (2.3) is satisfied.

Similarly, to show (2.2) is satisfied, we begin by taking the curl of the wave equation for $c^2\mathbf{B}$ and using the wave equation for $\mathbf{E}$ for the right-hand-side in (2.12):

$$\begin{aligned}\boldsymbol{\nabla}\times\frac{\partial^2\mathbf{B}}{\partial t^2} - c^2\boldsymbol{\nabla}\times(\boldsymbol{\nabla}^2\mathbf{B}) &= cZ_0\boldsymbol{\nabla}\times(\boldsymbol{\nabla}\times\mathbf{j})\\ \boldsymbol{\nabla}\times\frac{\partial^2\mathbf{B}}{\partial t^2} - c^2\boldsymbol{\nabla}^2(\boldsymbol{\nabla}\times\mathbf{B}) &= cZ_0\boldsymbol{\nabla}(\boldsymbol{\nabla}\cdot\mathbf{j}) - cZ_0\boldsymbol{\nabla}^2\mathbf{j}\\ \boldsymbol{\nabla}\times\frac{\partial^2\mathbf{B}}{\partial t^2} - c^2\boldsymbol{\nabla}\times\boldsymbol{\nabla}^2\mathbf{B} &= \frac{\partial}{\partial t}(-cZ_0\boldsymbol{\nabla}\rho) - cZ_0\boldsymbol{\nabla}^2\mathbf{j}\\ \boldsymbol{\nabla}\times\frac{\partial^2\mathbf{B}}{\partial t^2} - c^2\boldsymbol{\nabla}\times\boldsymbol{\nabla}^2\mathbf{B} &= \frac{\partial}{\partial t}\left(\frac{1}{c^2}\frac{\partial^2\mathbf{E}}{\partial t^2} - \boldsymbol{\nabla}^2\mathbf{E} + \frac{Z_0}{c}\frac{\partial\mathbf{j}}{\partial t}\right) - cZ_0\boldsymbol{\nabla}^2\mathbf{j}\\ \frac{1}{c^2}\frac{\partial^2}{\partial t^2}\left(c^2\boldsymbol{\nabla}\times\mathbf{B} - \frac{\partial\mathbf{E}}{\partial t} - cZ_0\mathbf{j}\right) &= \boldsymbol{\nabla}^2\left(c^2\boldsymbol{\nabla}\times\mathbf{B} - \frac{\partial\mathbf{E}}{\partial t} - cZ_0\mathbf{j}\right) \qquad (2.24)\end{aligned}$$

We now have a wave equation for $\mathbf{U} \equiv c^2\boldsymbol{\nabla}\times\mathbf{B} - \partial\mathbf{E}/\partial t - cZ_0\mathbf{j}$ with $\mathbf{S} = \mathbf{0}$. We also have $\mathbf{U}_0 \equiv c^2\boldsymbol{\nabla}\times\mathbf{B}_0 - \mathbf{E}_1 - cZ_0\mathbf{j}(\mathbf{R},0) = \mathbf{0}$ by (2.21). As before, must only show $\mathbf{U}_1 = \mathbf{0}$ to assert (2.2) is satisfied by invoking Theorem 2 once more. Examining $\mathbf{U}_1$

and using (2.22) we note:

$$\begin{aligned}
\mathbf{U}_1 &\equiv c^2\boldsymbol{\nabla}\times\mathbf{B}_1 - \frac{\partial^2\mathbf{E}}{\partial t^2}\Big|_{t=0} - cZ_0\frac{\partial\mathbf{j}}{\partial t}\Big|_{t=0}\\
&= -c^2\boldsymbol{\nabla}\times\big(\boldsymbol{\nabla}\times\mathbf{E}_0\big) - \frac{\partial^2\mathbf{E}}{\partial t^2}\Big|_{t=0} - cZ_0\frac{\partial\mathbf{j}}{\partial t}\Big|_{t=0}\\
&= -c^2\boldsymbol{\nabla}(\boldsymbol{\nabla}\cdot\mathbf{E}_0) + c^2\boldsymbol{\nabla}^2\mathbf{E}_0 - \frac{\partial^2\mathbf{E}}{\partial t^2}\Big|_{t=0} - cZ_0\frac{\partial\mathbf{j}}{\partial t}\Big|_{t=0}\\
&= -c^3Z_0\boldsymbol{\nabla}\rho(\mathbf{R},0) + c^2\boldsymbol{\nabla}^2\mathbf{E}_0 - \frac{\partial^2\mathbf{E}}{\partial t^2}\Big|_{t=0} - cZ_0\frac{\partial\mathbf{j}}{\partial t}\Big|_{t=0}.
\end{aligned}$$

Thus $\mathbf{U}_1 = \mathbf{0}$ since $\mathbf{E}(\mathbf{R},t)$ satisfies the wave equation (2.12) by construction. Therefore, $\mathbf{U} \equiv c^2\boldsymbol{\nabla}\times\mathbf{B} - \partial\mathbf{E}/\partial t - cZ_0\mathbf{j} = \mathbf{0}$ proving (2.2) is also satisfied.

Proof: Part 2 (Uniqueness)

The next task is to prove the solutions $(\mathbf{E}(\mathbf{R},t), \mathbf{B}(\mathbf{R},t))$, constructed in part 1, are unique. Assume two solutions $(\mathbf{E}_a, \mathbf{B}_a)$ and $(\mathbf{E}_b, \mathbf{B}_b)$ both satisfy (2.2)–(2.3) with the same initial conditions and sources. Now define the differences:

$$\mathbf{E}_D(\mathbf{R},t) := \mathbf{E}_a(\mathbf{R},t) - \mathbf{E}_b(\mathbf{R},t), \qquad \mathbf{B}_D(\mathbf{R},t) := \mathbf{B}_a(\mathbf{R},t) - \mathbf{B}_b(\mathbf{R},t),$$

$$\mathbf{E}_D(\mathbf{R},0) = \mathbf{E}_a(\mathbf{R},0) - \mathbf{E}_b(\mathbf{R},0) = \mathbf{0}, \qquad \mathbf{B}_D(\mathbf{R},0) = \mathbf{B}_a(\mathbf{R},0) - \mathbf{B}_b(\mathbf{R},0) = \mathbf{0}.$$

Furthermore, since the sources are the same for $(\mathbf{E}_a, \mathbf{B}_a)$ and $(\mathbf{E}_b, \mathbf{B}_b)$ and the divergence and curl operators are linear, applying (2.2)–(2.3) yields:

$$\boldsymbol{\nabla}\times\mathbf{E}_D = -\frac{\partial\mathbf{B}_D}{\partial t}, \qquad \boldsymbol{\nabla}\times\mathbf{B}_D = \frac{1}{c^2}\frac{\partial\mathbf{E}_D}{\partial t}.$$

Therefore, since $\mathbf{E}_D(\mathbf{R},0) = \mathbf{0}$ and $\mathbf{B}_D(\mathbf{R},0) = \mathbf{0}$ we have:

$$\frac{\partial\mathbf{E}_D}{\partial t}\Big|_{t=0} = \mathbf{0}, \qquad \frac{\partial\mathbf{B}_D}{\partial t}\Big|_{t=0} = \mathbf{0}.$$

Next, following the derivation of the wave equations as in (2.10)–(2.11):

$$\frac{1}{c^2}\frac{\partial^2\mathbf{E}_D}{\partial t^2} - \boldsymbol{\nabla}^2\mathbf{E}_D = \mathbf{0}, \qquad \frac{1}{c^2}\frac{\partial^2\mathbf{B}_D}{\partial t^2} - \boldsymbol{\nabla}^2\mathbf{B}_D = \mathbf{0}.$$

Lastly, we note that unique solution to the initial value problem for the homogeneous wave equation with zero initial data is zero. Thus:

$$\mathbf{E}_D(\mathbf{R},t) = \mathbf{0}, \qquad\qquad \mathbf{B}_D(\mathbf{R},t) = \mathbf{0}.$$

Therefore, solutions $(\mathbf{E}(\mathbf{R},t), \mathbf{B}(\mathbf{R},t))$ constructed by (2.12)–(2.14) with the restrictions (2.19)–(2.22) satisfy (2.2)–(2.3) and are unique. $\square$

2.2 Transformations and Transforms

Accelerator systems are designed so that particle bunches move roughly along a reference orbit. It is useful to consider Maxwell's equations in Frenet-Serret coordinates using the reference orbit of the particle motion. This is done for numerical convenience in modeling the charge and current densities.

2.2.1 Frenet-Serret Coordinates

In the cases studied in this thesis, we consider a planar reference orbit in the $Y = 0$ plane, defined in Cartesian coordinates $\mathbf{R} = (Z, X, Y)^\top$ by:

$$\mathbf{R}_r(s) = \begin{pmatrix} Z_r(s) \\ X_r(s) \\ 0 \end{pmatrix}, \tag{2.25}$$

where s is the arclength along the reference orbit. The unit tangent and normal vectors $\mathbf{t}(s)$, $\mathbf{n}(s)$, and $\mathbf{e}_Y$ are given by:

$$\mathbf{t}(s) = \mathbf{R}_r'(s) = \begin{pmatrix} Z_r'(s) \\ X_r'(s) \\ 0 \end{pmatrix}, \quad \mathbf{n}(s) = \begin{pmatrix} -X_r'(s) \\ Z_r'(s) \\ 0 \end{pmatrix}, \quad \mathbf{e}_Y = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \tag{2.26}$$

so that the Frenet-Serret transformation is defined as:

$$\mathbf{R}(\mathbf{r}) := \mathbf{R}_r(s) + x\mathbf{n}(s) + y\mathbf{e}_Y, \tag{2.27}$$

which is, in component form:

$$\begin{aligned} Z(s,x,y) &= Z_r(s) - xX_r'(s), \\ X(s,x,y) &= X_r(s) + xZ_r'(s), \\ Y(s,x,y) &= y. \end{aligned}$$

where $\mathbf{r} := (s,x,y)^\top$, the Frenet-Serret coordinates. We assume (2.27) is a diffeomorphism on a neighborhood of the reference orbit (2.25). The y coordinate is unchanged from Y. For a general smooth trajectory, the curvature, up to a sign, is defined as:

$$|\kappa(s)| := |\mathbf{R}_r''(s)| = |Z_r''(s)X_r'(s) - Z_r'(s)X_r''(s)|. \tag{2.28}$$

Choosing the counterclockwise orientation: $\kappa(s) = Z_r''(s)X_r'(s) - Z_r'(s)X_r''(s)$, we obtain the relations:

$$\mathbf{t}'(s) = -\kappa(s)\mathbf{n}(s), \qquad \mathbf{n}'(s) = \kappa(s)\mathbf{t}(s). \tag{2.29}$$

It is also useful to define the radius of curvature R and curvilinear scale factor η by:

$$R(s) := 1/|\kappa(s)|, \qquad \eta(s,x) := 1 + \kappa(s)x. \tag{2.30}$$

As an example, consider a circular reference orbit with center $\mathbf{R}_c = (0,R,0)$ starting at the origin: $\mathbf{R}_r(0) = \mathbf{0}$ and undergoing uniform curvature $\kappa = -1/R$. The components of $\mathbf{R}(\mathbf{r})$ are:

$$\begin{aligned} Z(s,x,y) &= R\sin(s/R) - x\sin(s/R) \\ X(s,x,y) &= R - R\cos(s/R) + x\cos(s/R) \\ Y(s,x,y) &= y \end{aligned} \tag{2.31}$$

with the transformation uniquely defined and invertible for $0 \leq s < 2\pi R$ and $x < R$.

Next, to transform the spatial derivatives of Maxwell's equations (2.2)–(2.5), into Frenet-Serret coordinates, we consider a vector field $\mathbf{F}(\mathbf{R})$ and its components in $\mathbf{r}$:

$$\begin{aligned} \mathbf{F}(\mathbf{R}) &:= F_Z(\mathbf{R})\mathbf{e}_Z + F_X(\mathbf{R})\mathbf{e}_X + F_Y(\mathbf{R})\mathbf{e}_Y \\ &\equiv F_s(\mathbf{r})\mathbf{t}(s) + F_x(\mathbf{r})\mathbf{n}(s) + F_y(\mathbf{r})\mathbf{e}_Y, \end{aligned} \tag{2.32}$$

where $\mathbf{R}$ and $\mathbf{r}$ are related by (2.27). We use (2.32) on the fields $\mathbf{E}(\mathbf{R}, t)$ and $\mathbf{B}(\mathbf{R}, t)$ to define $(E_s(\mathbf{r}, t), E_x(\mathbf{r}, t), E_y(\mathbf{r}, t))$ and $(B_s(\mathbf{r}, t), B_x(\mathbf{r}, t), B_y(\mathbf{r}, t))$, the $(\mathbf{t}, \mathbf{n}, \mathbf{e}_Y)$ components of $(\mathbf{E}, \mathbf{B})$. The Cartesian components are related to the Frenet-Serret components by:

$$F_Z(\mathbf{R}(\mathbf{r})) = F_s(\mathbf{r})Z_r'(s) - F_x(\mathbf{r})X_r'(s) \tag{2.33a}$$

$$F_X(\mathbf{R}(\mathbf{r})) = F_s(\mathbf{r})X_r'(s) + F_x(\mathbf{r})Z_r'(s) \tag{2.33b}$$

$$F_Y(\mathbf{R}(\mathbf{r})) = F_y(\mathbf{r}) \tag{2.33c}$$

To transform the derivatives we require the inverse $\mathbf{r}(\mathbf{R})$ of (2.27) which satisfies $\mathbf{R} = \mathbf{R}_r(s(\mathbf{R})) + x(\mathbf{R})\mathbf{n}(s(\mathbf{R})) + y(\mathbf{R})\mathbf{e}_Y$. Now, using (2.27), the partial derivatives of (s, x, y) in (Z, X, Y) satisfy:

$$\begin{aligned} 1 &= (Z_r' - X_r''x)\frac{\partial s}{\partial Z} - X_r'\frac{\partial x}{\partial Z}, & 0 &= (X_r' + Z_r''x)\frac{\partial s}{\partial Z} + Z_r'\frac{\partial x}{\partial Z}, & 0 &= \frac{\partial y}{\partial Z}, \\ 0 &= (Z_r' - X_r''x)\frac{\partial s}{\partial X} - X_r'\frac{\partial x}{\partial X}, & 1 &= (X_r' + Z_r''x)\frac{\partial s}{\partial X} + Z_r'\frac{\partial x}{\partial X}, & 0 &= \frac{\partial y}{\partial X}, \\ 0 &= (Z_r' - X_r''x)\frac{\partial s}{\partial Y} - X_r'\frac{\partial x}{\partial Y}, & 0 &= (X_r' + Z_r''x)\frac{\partial s}{\partial Y} + Z_r'\frac{\partial x}{\partial Y}, & 1 &= \frac{\partial y}{\partial Y}. \end{aligned}$$

Inverting this system with the definition of η in (2.28)–(2.30), results in:

$$\begin{aligned} \frac{\partial s}{\partial Z} &= \frac{1}{\eta}Z_r', & \frac{\partial s}{\partial X} &= \frac{1}{\eta}X_r', & \frac{\partial s}{\partial Y} &= 0, \\ \frac{\partial x}{\partial Z} &= -X_r', & \frac{\partial x}{\partial X} &= Z_r', & \frac{\partial x}{\partial Y} &= 0, \\ \frac{\partial y}{\partial Z} &= 0, & \frac{\partial y}{\partial X} &= 0, & \frac{\partial y}{\partial Y} &= 1. \end{aligned} \tag{2.34}$$

Lastly, differentiating with respect to $\mathbf{R}$ in $\mathbf{R} = \mathbf{R}(\mathbf{r}(\mathbf{R}))$ with the above relations, we obtain the following:

$$\frac{\partial F_Z}{\partial Z} = \frac{1}{\eta} Z_r'^2 \frac{\partial F_s}{\partial s} - Z_r' X_r' \frac{\partial F_s}{\partial x} - \frac{1}{\eta} Z_r' X_r' \frac{\partial F_x}{\partial s} + X_r'^2 \frac{\partial F_x}{\partial x} + \frac{\kappa}{\eta} Z_r' X_r' F_s + \frac{\kappa}{\eta} Z_r'^2 F_x, \tag{2.35a}$$

$$\frac{\partial F_X}{\partial Z} = \frac{1}{\eta} Z_r' X_r' \frac{\partial F_s}{\partial s} - X_r'^2 \frac{\partial F_s}{\partial x} + \frac{1}{\eta} Z_r'^2 \frac{\partial F_x}{\partial s} - X_r' Z_r' \frac{\partial F_x}{\partial x} - \frac{\kappa}{\eta} Z_r'^2 F_s + \frac{\kappa}{\eta} Z_r' X_r' F_x, \tag{2.35b}$$

$$\frac{\partial F_Y}{\partial Z} = \frac{1}{\eta} Z_r' \frac{\partial F_y}{\partial s} - X_r' \frac{\partial F_y}{\partial x}, \tag{2.35c}$$

$$\frac{\partial F_Z}{\partial X} = \frac{1}{\eta} Z_r' X_r' \frac{\partial F_s}{\partial s} + Z_r'^2 \frac{\partial F_s}{\partial x} - \frac{1}{\eta} X_r'^2 \frac{\partial F_x}{\partial s} - Z_r' X_r' \frac{\partial F_x}{\partial x} + \frac{\kappa}{\eta} X_r'^2 F_s + \frac{\kappa}{\eta} Z_r' X_r' F_x, \tag{2.35d}$$

$$\frac{\partial F_X}{\partial X} = \frac{1}{\eta} X_r'^2 \frac{\partial F_s}{\partial s} + Z_r' X_r' \frac{\partial F_s}{\partial x} + \frac{1}{\eta} Z_r' X_r' \frac{\partial F_x}{\partial s} + Z_r'^2 \frac{\partial F_x}{\partial x} - \frac{\kappa}{\eta} Z_r' X_r' F_s + \frac{\kappa}{\eta} X_r'^2 F_x, \tag{2.35e}$$

$$\frac{\partial F_Y}{\partial X} = \frac{1}{\eta} X_r' \frac{\partial F_y}{\partial s} + Z_r' \frac{\partial F_y}{\partial x}, \tag{2.35f}$$

$$\frac{\partial F_Z}{\partial Y} = Z_r' \frac{\partial F_s}{\partial y} - X_r' \frac{\partial F_x}{\partial y}, \tag{2.35g}$$

$$\frac{\partial F_X}{\partial Y} = X_r' \frac{\partial F_s}{\partial y} + Z_r' \frac{\partial F_x}{\partial y}, \tag{2.35h}$$

$$\frac{\partial F_Y}{\partial Y} = \frac{\partial F_y}{\partial y}, \tag{2.35i}$$

where we emphasize that the left-hand-sides of (2.35) are evaluated at $\mathbf{R}(\mathbf{r})$. We can now construct the gradient, divergence, and curl operators in Frenet-Serret coordinates by transforming the Cartesian operators with the partial derivatives (2.34). Given $f(\mathbf{R})$ we define $\phi(\mathbf{r}) := f(\mathbf{R}(\mathbf{r}))$ so that $\phi(\mathbf{r}(\mathbf{R})) \equiv f(\mathbf{R})$ and we have:

$$\begin{aligned}\boldsymbol{\nabla} f :=& \frac{\partial f}{\partial Z}\mathbf{e}_Z + \frac{\partial f}{\partial X}\mathbf{e}_X + \frac{\partial f}{\partial Y}\mathbf{e}_Y \\ =& \left(\frac{\partial \phi}{\partial s}\frac{\partial s}{\partial Z} + \frac{\partial \phi}{\partial x}\frac{\partial x}{\partial Z}\right)\mathbf{e}_Z + \left(\frac{\partial \phi}{\partial s}\frac{\partial s}{\partial X} + \frac{\partial \phi}{\partial x}\frac{\partial x}{\partial X}\right)\mathbf{e}_X + \frac{\partial \phi}{\partial y}\frac{\partial y}{\partial Y}\mathbf{e}_Y \\ =& \frac{1}{\eta}\frac{\partial \phi}{\partial s}\mathbf{t} + \frac{\partial \phi}{\partial x}\mathbf{n} + \frac{\partial \phi}{\partial y}\mathbf{e}_Y.\end{aligned} \tag{2.36}$$

Similarly, using $\mathbf{F}(\mathbf{R}) = F_s(\mathbf{r}(\mathbf{R}))\mathbf{t}(s(\mathbf{R})) + F_x(\mathbf{r}(\mathbf{R}))\mathbf{n}(s(\mathbf{R})) + F_y(\mathbf{r}(\mathbf{R}))\mathbf{e}_Y$ with the partial derivatives in (2.35), we obtain:

$$\begin{aligned}\boldsymbol{\nabla}\cdot\mathbf{F} :=& \frac{\partial F_Z}{\partial Z} + \frac{\partial F_X}{\partial X} + \frac{\partial F_Y}{\partial Y}\\ =& \frac{1}{\eta}\frac{\partial F_s}{\partial s} + \frac{1}{\eta}\frac{\partial(\eta F_x)}{\partial x} + \frac{\partial F_y}{\partial y},\end{aligned} \tag{2.37}$$

$$\begin{aligned}\boldsymbol{\nabla}\times\mathbf{F} :=& \left(\frac{\partial F_Y}{\partial X} - \frac{\partial F_X}{\partial Y}\right)\mathbf{e}_Z + \left(\frac{\partial F_Z}{\partial Y} - \frac{\partial F_Y}{\partial Z}\right)\mathbf{e}_X + \left(\frac{\partial F_X}{\partial Z} - \frac{\partial F_Z}{\partial X}\right)\mathbf{e}_Y\\ =& \left(\frac{\partial F_y}{\partial x} - \frac{\partial F_x}{\partial y}\right)\mathbf{t} + \left(\frac{\partial F_s}{\partial y} - \frac{1}{\eta}\frac{\partial F_y}{\partial s}\right)\mathbf{n} + \left(\frac{1}{\eta}\frac{\partial F_x}{\partial s} - \frac{1}{\eta}\frac{\partial(\eta F_s)}{\partial x}\right)\mathbf{e}_Y.\end{aligned} \tag{2.38}$$

Next, the scalar and vector Laplacian operators are built from (2.36)–(2.38):

$$\begin{aligned}\boldsymbol{\nabla}^2 f :=& \boldsymbol{\nabla}\cdot(\boldsymbol{\nabla} f)\\ =& \frac{1}{\eta}\frac{\partial}{\partial s}\left(\frac{1}{\eta}\frac{\partial\phi}{\partial s}\right) + \frac{1}{\eta}\frac{\partial}{\partial x}\left(\eta\frac{\partial\phi}{\partial x}\right) + \frac{\partial^2\phi}{\partial y^2},\end{aligned} \tag{2.39}$$

$$\begin{aligned}\boldsymbol{\nabla}^2\mathbf{F} :=& \boldsymbol{\nabla}(\boldsymbol{\nabla}\cdot\mathbf{F}) - \boldsymbol{\nabla}\times(\boldsymbol{\nabla}\times\mathbf{F})\\ =& \left[\frac{1}{\eta}\frac{\partial}{\partial s}\left(\frac{1}{\eta}\frac{\partial F_s}{\partial s}\right) + \frac{1}{\eta}\frac{\partial}{\partial x}\left(\eta\frac{\partial F_s}{\partial x}\right) + \frac{\partial^2 F_s}{\partial y^2} + \frac{2\kappa}{\eta^2}\frac{\partial F_x}{\partial s} + \frac{\kappa'}{\eta^3}F_x\right]\mathbf{t}\\ &+ \left[\frac{1}{\eta}\frac{\partial}{\partial s}\left(\frac{1}{\eta}\frac{\partial F_x}{\partial s}\right) + \frac{1}{\eta}\frac{\partial}{\partial x}\left(\eta\frac{\partial F_x}{\partial x}\right) + \frac{\partial^2 F_x}{\partial y^2} - \frac{2\kappa}{\eta^2}\frac{\partial F_s}{\partial s} - \frac{\kappa'}{\eta^3}F_s\right]\mathbf{n}\\ &+ \left[\frac{1}{\eta}\frac{\partial}{\partial s}\left(\frac{1}{\eta}\frac{\partial F_y}{\partial s}\right) + \frac{1}{\eta}\frac{\partial}{\partial x}\left(\eta\frac{\partial F_y}{\partial x}\right) + \frac{\partial^2 F_y}{\partial y^2}\right]\mathbf{e}_Y.\end{aligned} \tag{2.40}$$

With these relations, Maxwell's equations (2.2)–(2.5) can be transformed into Frenet-

Serret coordinates. Specifically applying the operators in (2.37)–(2.38) we note:

$$\frac{\partial E_s}{\partial t} = c^2\left(\frac{\partial B_y}{\partial x} - \frac{\partial B_x}{\partial y}\right) - cZ_0 j_s(\mathbf{r},t), \tag{2.41a}$$

$$\frac{\partial E_x}{\partial t} = c^2\left(\frac{\partial B_s}{\partial y} - \frac{1}{\eta}\frac{\partial B_y}{\partial s}\right) - cZ_0 j_x(\mathbf{r},t), \tag{2.41b}$$

$$\frac{\partial E_y}{\partial t} = c^2\left(\frac{1}{\eta}\frac{\partial B_x}{\partial s} - \frac{1}{\eta}\frac{\partial(\eta B_s)}{\partial x}\right) - cZ_0 j_y(\mathbf{r},t), \tag{2.41c}$$

$$\frac{\partial B_s}{\partial t} = -\left(\frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y}\right), \tag{2.41d}$$

$$\frac{\partial B_x}{\partial t} = -\left(\frac{\partial E_s}{\partial y} - \frac{1}{\eta}\frac{\partial E_y}{\partial s}\right), \tag{2.41e}$$

$$\frac{\partial B_y}{\partial t} = -\left(\frac{1}{\eta}\frac{\partial E_x}{\partial s} - \frac{1}{\eta}\frac{\partial(\eta E_s)}{\partial x}\right), \tag{2.41f}$$

$$0 = \frac{1}{\eta}\frac{\partial E_s}{\partial s} + \frac{1}{\eta}\frac{\partial(\eta E_x)}{\partial x} + \frac{\partial E_y}{\partial y} - cZ_0\rho(\mathbf{r},t), \tag{2.41g}$$

$$0 = \frac{1}{\eta}\frac{\partial B_s}{\partial s} + \frac{1}{\eta}\frac{\partial(\eta B_x)}{\partial x} + \frac{\partial B_y}{\partial y}. \tag{2.41h}$$

As a consequence of the transformation, it is important to note the Cartesian volume element is $d^3\mathbf{R} = dZdXdY$ and thus densities, when written in the Frenet-Serret coordinate system, must be integrated with respect to the curved volume element $d^3\mathbf{r} = \eta(s,x)dsdxdy$. Thus in (2.41), $\rho(\mathbf{r},t) \equiv \rho_L(\mathbf{R}(\mathbf{r}),t)$ where $\rho_L(\mathbf{R},t)$ is the charge density in Cartesian coordinates mentioned throughout section 2.1.

Another particular result of the curved volume element is that rigid charge densities of the form: $\rho(\mathbf{r},t) = \lambda(s-\beta ct)w(x)h(y)$ are not physically meaningful since the spread in x given by the distribution function $w(x)$ creates a velocity dispersion over the source. That is, charge along $x = x_1$ moves at a different speed than charge at $x \neq x_1$. To illustrate this in an example, consider the uniform circular trajectory given by (2.31), where $\kappa = -1/R$, with the density:

$$\rho(\mathbf{r}, t) = \begin{cases} 1/(1+\kappa x) & : \quad 0 \le s - \beta ct \le \Delta s, \quad 0 \le x \le \Delta x, \quad 0 \le y \le \Delta y \\ 0 & : \quad \text{otherwise} \end{cases} \tag{2.42}$$

This ρ represents a curved slab of uniform density of total charge $Q = \Delta s \Delta x \Delta y$ in Cartesian coordinates. The slab rigidly rotates about the axis $(Z, X) = (0, R)$ with constant angular speed $\beta c/R$. The portion of the source along $x = 0$ moves at speed βc however the portion of the source along $x = \Delta x$ moves at speed $\beta c(1 + \kappa \Delta x)$. This dispersion is unphysical for the bunches of electrons we consider in accelerator structures.

Therefore, we consider only thin sources where $w(x) = \delta(x)$ so that charge density can maintain a rigid profile without velocity dispersion. Another consequence of the thin source is that the scale factor $\eta(s, x)$ in the volume element $\eta ds dx dy$ simplifies to $ds dx dy$ for integrals involving the charge or current density.

2.2.2 Source Terms and Boundary Conditions

Following the derivation of the Frenet-Serret coordinates, we discuss the source terms for the charge and current densities ρ and $\mathbf{j}$ used throughout this thesis. Additionally, we address the boundary conditions which are used for Maxwell's equations inside the vacuum chamber.

To model a line charge beam, used in Chapters 3 and 4, we adopt the charge and current densities of a bunch moving in the $+s$ direction with the form:

$$\rho(\mathbf{r}, t) = q\lambda(s - \beta ct)\delta(x)\delta(y), \tag{2.43}$$

$$\mathbf{j}(\mathbf{r}, t) = q\beta c\lambda(s - \beta ct)\delta(x)\delta(y)\mathbf{t}, \tag{2.44}$$

where q is the total charge and $\lambda(z)$ is the longitudinal distribution. Clearly, these source terms satisfy (2.1). We consider a Gaussian distribution for $\lambda(z)$ given by:

$$\lambda(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{-z^2/2\sigma^2}, \tag{2.45}$$

where the standard deviation σ is often called the bunch length. Since the source travels very close to the speed of light, we additionally take $\beta = 1$ which is a good approximation for our work in this thesis. A particular advantage in setting $\beta = 1$ is that we can construct a steady-state initial condition for the time-domain problems where the initial conditions depend on s only through λ. This construction will be discussed in detail in Chapters 3–6.

Other sources we consider are those of a ribbon or round bunch where we allow the beam to have some transverse width in x and y. For Chapters 3 and 5 we adopt the ribbon beam source:

$$\rho(\mathbf{r}, t) = q\lambda(s - ct)\delta(x)G(y), \tag{2.46}$$

$$\mathbf{j}(\mathbf{r}, t) = qc\lambda(s - ct)\delta(x)G(y)\mathbf{t}, \tag{2.47}$$

and for Chapter 6 we use the round bunch:

$$\rho(\mathbf{r}, t) = q\lambda(s - ct)\lambda(x)\lambda(y), \tag{2.48}$$

$$\mathbf{j}(\mathbf{r}, t) = qc\lambda(s - ct)\lambda(x)\lambda(y)\mathbf{t}. \tag{2.49}$$

The distribution $G(y)$ in (2.46)–(2.47) is also assumed to be a Gaussian but of possibly a different transverse standard deviation σ_y. Note, the round bunch model (2.49) for Chapter 6 is not physical and is only used for algorithm testing purposes.

The boundary for the vacuum chamber is assumed to be perfectly conducting. While in Chapters 3 and 4 we only require the perfectly electrically conducting (PEC) boundary conditions, in Chapters 5 and 6 we require a different approach to handle the open boundaries in the longitudinal s direction. We will defer from these special considerations until discussions in Chapters 5 and 6.

The PEC boundary condition for a simply-connected volume $\Omega \subset \mathbb{R}^3$ enclosed by a piece-wise smooth boundary $\partial\Omega$ is:

$$\mathbf{n} \times \mathbf{E}\Big|_{\partial\Omega} = \mathbf{0}, \tag{2.50a}$$

$$\mathbf{n} \cdot \mathbf{B}\Big|_{\partial\Omega} = 0. \tag{2.50b}$$

where $\mathbf{n}$ is an outward normal vector along the boundary $\partial\Omega$. Considering sources whose support is interior to Ω, we conjecture that while the tangential $\mathbf{E}$ and normal $\mathbf{B}$ fields are imposed with the Dirichlet-type boundary conditions (2.50), the normal $\mathbf{E}$ and tangential $\mathbf{B}$ fields satisfy Neumann-type boundary conditions.

This conjecture is motivated by (2.41) and applying (2.50) with source terms which vanish on the boundary $\partial\Omega$. For example, for a planar PEC boundary at $x = a$ with the interior of Ω in the region $x < a$ we have:

$$\frac{\partial E_x}{\partial x}\bigg|_{x\to a^-} = 0, \qquad \frac{\partial B_s}{\partial x}\bigg|_{x\to a^-} = 0, \qquad \frac{\partial B_y}{\partial x}\bigg|_{x\to a^-} = 0, \tag{2.51}$$

and likewise for a planar PEC boundary at $y = b$ with the interior of Ω in the region $y < b$ we have:

$$\frac{\partial E_y}{\partial y}\bigg|_{y\to b^-} = 0, \qquad \frac{\partial B_s}{\partial y}\bigg|_{y\to b^-} = 0, \qquad \frac{\partial B_x}{\partial y}\bigg|_{y\to b^-} = 0. \tag{2.52}$$

These extra boundary conditions in (2.51) or (2.52) are redundant and are satisfied automatically with (2.50) for Maxwell's equations; however, they are necessary for the Maxwell wave equations. These Neumann conditions are derived from examining waveguide solutions to Maxwell's equations, see pages 358–359 in [12].

Another interesting fact is that (2.50b) must only be imposed for $\mathbf{B}$ initially. This is proven by considering the signed distance function of Ω:

$$f(\mathbf{r}) = \begin{cases} -d(\mathbf{r}, \Omega^c) & : \quad \mathbf{r} \in \Omega \\ d(\mathbf{r}, \Omega) & : \quad \mathbf{r} \in \Omega^c \end{cases} \tag{2.53}$$

with the distance function $d(\mathbf{r}, \Omega) := \inf_{\mathbf{r}' \in \Omega} d(\mathbf{r}, \mathbf{r}')$. If the boundary $\partial\Omega$ is smooth, then we can set $\mathbf{n} = \boldsymbol{\nabla} f$ and thus taking the time derivative of (2.50b) we obtain:

$$\begin{aligned}
\frac{\partial}{\partial t}(\mathbf{n} \cdot \mathbf{B})\Big|_{\partial\Omega} &= -\mathbf{n} \cdot (\boldsymbol{\nabla} \times \mathbf{E})\Big|_{\partial\Omega} \\
&= \boldsymbol{\nabla} \cdot (\mathbf{n} \times \mathbf{E})\Big|_{\partial\Omega} - \mathbf{E} \cdot (\boldsymbol{\nabla} \times \mathbf{n})\Big|_{\partial\Omega} \\
&= 0
\end{aligned}$$

since $\mathbf{E}$ satisfies (2.50a) and $\boldsymbol{\nabla} \times (\boldsymbol{\nabla} f) = \mathbf{0}$. Thus, if $\mathbf{B}$ satisfies (2.50b) initially, then $\mathbf{B}$ satisfies (2.50b) for all time. Therefore, this condition on $\mathbf{B}$ is only required in constructing initial conditions of Maxwell's equations. Corners along the boundary $\partial\Omega$ do not pose a problem since the normal $\mathbf{n}$ is well defined around the corner. Numerically, this is handled by placing an element vertex or grid point at the corner.

2.2.3 Fourier Series in y

In this subsection we consider Maxwell's equations (2.41) in the infinite slab of thickness h: $\Omega = \{\mathbf{r} | -h/2 < y < h/2\}$. Here we adopt the Frenet-Serret coordinates $\mathbf{r} = (s, x, y)^\top$ presented in section 2.2.1 as our starting point; however, we can apply the same transformations in Cartesian coordinates with no difference since $y = Y$. Applying the PEC boundary conditions at $y = \pm h/2$ we note:

$$\mathbf{n} \times \mathbf{E}\Big|_{y=\pm h/2} = \mathbf{0}, \qquad \mathbf{n} \cdot \mathbf{B}\Big|_{y=\pm h/2} = 0, \tag{2.54}$$

where $\mathbf{n} = \pm\mathbf{e}_Y$ denoting the normal of the boundary $\partial\Omega$. Applying (2.54) with Maxwell's equations results in $E_s = E_x = B_y \equiv 0$ and $\partial E_y/\partial y = \partial B_s/\partial y = \partial B_x/\partial y \equiv 0$ on $y = \pm h/2$. A useful transformation when the y-boundaries are independent of (s, x) and are of either Dirichlet-type or Neumann-type is to decompose fields modally with Fourier Series in y. That is, we decompose $f(s, x, y, t)$ in either a sine or cosine series:

$$\begin{aligned} f(s,x,y,t) &= \sum_{p=1}^{\infty} \hat{b}_p(s,x,t) \sin\left(\alpha_p(y+h/2)\right), \\ \hat{b}_p(s,x,t) &= \frac{2}{h}\int_{-h/2}^{h/2} f(s,x,y,t)\sin\left(\alpha_p(y+h/2)\right) dy, \end{aligned} \tag{2.55}$$

$$\begin{aligned} f(s,x,y,t) &= \frac{\hat{a}_0(s,x,t)}{2} + \sum_{p=1}^{\infty} \hat{a}_p(s,x,t) \cos\left(\alpha_p(y+h/2)\right), \\ \hat{a}_p(s,x,t) &= \frac{2}{h}\int_{-h/2}^{h/2} f(s,x,y,t)\cos\left(\alpha_p(y+h/2)\right) dy, \end{aligned} \tag{2.56}$$

with $\alpha_p = \pi p/h$. Now applying (2.55) to each of the 3 Dirichlet-type boundary fields yields motivated by (2.54), we obtain:

$$\begin{pmatrix} E_s(s,x,y,t) \\ E_x(s,x,y,t) \\ B_y(s,x,y,t) \end{pmatrix} = \sum_{p=1}^{\infty} \begin{pmatrix} \hat{E}_{sp}(s,x,t) \\ \hat{E}_{xp}(s,x,t) \\ \hat{B}_{yp}(s,x,t) \end{pmatrix} \sin\big(\alpha_p(y+h/2)\big). \tag{2.57}$$

And similarly, applying (2.56) to the remaining fields we obtain:

$$\begin{pmatrix} E_y(s,x,y,t) \\ B_s(s,x,y,t) \\ B_x(s,x,y,t) \end{pmatrix} = \sum_{p=0}^{\infty} \begin{pmatrix} \hat{E}_{yp}(s,x,t) \\ \hat{B}_{sp}(s,x,t) \\ \hat{B}_{xp}(s,x,t) \end{pmatrix} \cos\big(\alpha_p(y+h/2)\big). \tag{2.58}$$

It is important to note that Maxwell's equations are satisfied term-by-term in the Fourier series and we choose the appropriate expansion (2.55) or (2.56) depending on the types of boundary conditions. Additionally, the source components ρ, j_s, j_x are expanded with (2.55) while j_y is expanded with (2.56). In the particular case where the sources $(\rho, \mathbf{j})$ are either symmetric in y, all even p modes for both the fields and sources reduce to zero. At this point, each of the field equations in the volume (s,x,y) are reduced to equations in the plane (s,x) with the parameter p denoting the mode in the y direction.

2.2.4 Paraxial Approximation

In this subsection we consider a Fourier transform in $s-ct$ after the Frenet-Serret transformation in section 2.2.1 with or without the additional Fourier series in y given in section 2.2.3. In the derivation of the approximation, consider the fields without the Fourier series in y, namely the result of section 2.2.1 and define the Fourier transform and inverse as:

$$\begin{aligned}\hat{f}(s,x,y,k) &= \frac{c}{2\pi}\int_{-\infty}^{\infty} f(s,x,y,t)e^{-ik(s-ct)}dt,\\ f(s,x,y,t) &= \int_{-\infty}^{\infty}\hat{f}(s,x,y,k)e^{ik(s-ct)}dk.\end{aligned} \tag{2.59}$$

This pair follows easily from any of the standard 1D Fourier transform conventions. The motivation behind this transformation is to study the behavior of the fields when the sources and fields move in the same direction. In this case, we expect the fields $(\hat{\mathbf{E}},\hat{\mathbf{B}})$ to be slowly varying in s. By considering the ribbon source which obeys (2.46)–(2.47), we begin with the wave equations for $(\mathbf{E}(\mathbf{r},t),\mathbf{B}(\mathbf{r},t))$ in Frenet-Serret coordinates. By applying the vector identity (2.40), we note the components of $(\mathbf{E},\mathbf{B})$ from Maxwell's wave equations (2.12), satisfy equations of the form:

$$\begin{aligned}-\frac{1}{c^2}\frac{\partial^2 U_s}{\partial t^2} + \frac{1}{\eta^2}\frac{\partial^2 U_s}{\partial s^2} - \frac{\kappa' x}{\eta^3}\frac{\partial U_s}{\partial s} + \frac{\kappa^2}{\eta^2}U_s&\\ + \frac{\partial^2 U_s}{\partial x^2} + \frac{\kappa}{\eta}\frac{\partial U_s}{\partial x} + \frac{\partial^2 U_s}{\partial y^2} + \frac{2\kappa}{\eta^2}\frac{\partial U_x}{\partial s} + \frac{\kappa'}{\eta^3}U_x &= S_{s,U}(\mathbf{r},t),\end{aligned} \tag{2.60a}$$

$$\begin{aligned}-\frac{1}{c^2}\frac{\partial^2 U_x}{\partial t^2} + \frac{1}{\eta^2}\frac{\partial^2 U_x}{\partial s^2} - \frac{\kappa' x}{\eta^3}\frac{\partial U_x}{\partial s} - \frac{\kappa^2}{\eta^2}U_x&\\ + \frac{\partial^2 U_x}{\partial x^2} + \frac{\kappa}{\eta}\frac{\partial U_x}{\partial x} + \frac{\partial^2 U_x}{\partial y^2} - \frac{2\kappa}{\eta^2}\frac{\partial U_s}{\partial s} - \frac{\kappa'}{\eta^3}U_s &= S_{x,U}(\mathbf{r},t),\end{aligned} \tag{2.60b}$$

$$-\frac{1}{c^2}\frac{\partial^2 U_y}{\partial t^2} + \frac{1}{\eta^2}\frac{\partial^2 U_y}{\partial s^2} - \frac{\kappa' x}{\eta^3}\frac{\partial U_y}{\partial s} + \frac{\partial^2 U_y}{\partial x^2} + \frac{\kappa}{\eta}\frac{\partial U_y}{\partial x} + \frac{\partial^2 U_y}{\partial y^2} = S_{y,U}(\mathbf{r},t), \tag{2.60c}$$

where $\kappa(s)$ is the curvature and $\eta(s,x) = 1+\kappa(s)x$. Note that the s and x components of the fields are now coupled in (2.60) but the y component remains uncoupled. Due to the $\delta(x)$ term in the source terms, we take $\eta = 1$ in derivatives involving $(\rho,\mathbf{j})$

so the specific right-hand-sides of (2.60) in Frenet-Serret coordinate components $(\mathbf{t}, \mathbf{n}, \mathbf{e}_Y)$ are:

$$\begin{pmatrix} S_{s,E}(\mathbf{r},t) \\ S_{x,E}(\mathbf{r},t) \\ S_{y,E}(\mathbf{r},t) \end{pmatrix} = \begin{pmatrix} 0 \\ qcZ_0\lambda(s-ct)\delta'(x)G(y) \\ qcZ_0\lambda(s-ct)\delta(x)G'(y) \end{pmatrix}, \tag{2.61a}$$

$$\begin{pmatrix} S_{s,B}(\mathbf{r},t) \\ S_{x,B}(\mathbf{r},t) \\ S_{y,B}(\mathbf{r},t) \end{pmatrix} = \begin{pmatrix} 0 \\ -qZ_0\lambda(s-ct)\delta'(x)G(y) \\ qZ_0\lambda(s-ct)\delta(x)G'(y) \end{pmatrix}. \tag{2.61b}$$

Now we apply (2.59) to (2.60) to obtain the frequency-domain equations:

$$\begin{aligned} k^2\hat{U}_s + \frac{1}{\eta^2}\left(\frac{\partial^2\hat{U}_s}{\partial s^2} + 2ik\frac{\partial\hat{U}_s}{\partial s} - k^2\hat{U}_s\right) - \frac{\kappa' x}{\eta^3}\left(\frac{\partial\hat{U}_s}{\partial s} + ik\hat{U}_s\right) + \frac{\kappa^2}{\eta^2}\hat{U}_s \\ + \frac{\partial^2\hat{U}_s}{\partial x^2} + \frac{\kappa}{\eta}\frac{\partial\hat{U}_s}{\partial x} + \frac{\partial^2\hat{U}_s}{\partial y^2} + \frac{2\kappa}{\eta^2}\left(\frac{\partial\hat{U}_x}{\partial s} + ik\hat{U}_x\right) + \frac{\kappa'}{\eta^3}\hat{U}_x = \hat{S}_{s,U}(x,y,k), \end{aligned} \tag{2.62a}$$

$$\begin{aligned} k^2\hat{U}_x + \frac{1}{\eta^2}\left(\frac{\partial^2\hat{U}_x}{\partial s^2} + 2ik\frac{\partial\hat{U}_x}{\partial s} - k^2\hat{U}_x\right) - \frac{\kappa' x}{\eta^3}\left(\frac{\partial\hat{U}_x}{\partial s} + ik\hat{U}_x\right) - \frac{\kappa^2}{\eta^2}\hat{U}_x \\ + \frac{\partial^2\hat{U}_x}{\partial x^2} + \frac{\kappa}{\eta}\frac{\partial\hat{U}_x}{\partial x} + \frac{\partial^2\hat{U}_x}{\partial y^2} - \frac{2\kappa}{\eta^2}\left(\frac{\partial\hat{U}_s}{\partial s} + ik\hat{U}_s\right) - \frac{\kappa'}{\eta^3}\hat{U}_s = \hat{S}_{x,U}(x,y,k), \end{aligned} \tag{2.62b}$$

$$\begin{aligned} k^2\hat{U}_y + \frac{1}{\eta^2}\left(\frac{\partial^2\hat{U}_y}{\partial s^2} + 2ik\frac{\partial\hat{U}_y}{\partial s} - k^2\hat{U}_y\right) - \frac{\kappa' x}{\eta^3}\left(\frac{\partial\hat{U}_y}{\partial s} + ik\hat{U}_y\right) \\ + \frac{\partial^2\hat{U}_y}{\partial x^2} + \frac{\kappa}{\eta}\frac{\partial\hat{U}_y}{\partial x} + \frac{\partial^2\hat{U}_y}{\partial y^2} = \hat{S}_{y,U}(x,y,k). \end{aligned} \tag{2.62c}$$

with the right-hand-sides corresponding to the Fourier transform of (2.61):

$$\begin{pmatrix} \hat{S}_{s,E}(x,y,k) \\ \hat{S}_{x,E}(x,y,k) \\ \hat{S}_{y,E}(x,y,k) \end{pmatrix} = \begin{pmatrix} 0 \\ qcZ_0\hat{\lambda}(k)\delta'(x)G(y) \\ qcZ_0\hat{\lambda}(k)\delta(x)G'(y) \end{pmatrix}, \tag{2.63a}$$

$$\begin{pmatrix} \hat{S}_{s,B}(x,y,k) \\ \hat{S}_{x,B}(x,y,k) \\ \hat{S}_{y,B}(x,y,k) \end{pmatrix} = \begin{pmatrix} 0 \\ -qZ_0\hat{\lambda}(k)\delta'(x)G(y) \\ qZ_0\hat{\lambda}(k)\delta(x)G'(y) \end{pmatrix}. \tag{2.63b}$$

It is important to note that the frequency-domain equations (2.62) are elliptic and thus ill-posed as initial value problems with s as the evolution variable. If we restrict our approach to consider only slowly varying solutions in s, we can take $\partial^2/\partial s^2$ terms as "small" and thus ignore them. Also, if we consider only portions of trajectories of uniform or slowly varying curvature, where $\kappa'(s)x/\eta \ll k$, we can omit those terms as well. With these changes, the altered equations are almost Schrödinger-like and are of the form:

$$\begin{aligned} &k^2\hat{U}_s + \frac{1}{\eta^2}\Big(2ik\frac{\partial\hat{U}_s}{\partial s} - k^2\hat{U}_s\Big) + \frac{\kappa^2}{\eta^2}\hat{U}_s \\ &\qquad + \frac{\partial^2\hat{U}_s}{\partial x^2} + \frac{\kappa}{\eta}\frac{\partial\hat{U}_s}{\partial x} + \frac{\partial^2\hat{U}_s}{\partial y^2} + \frac{2\kappa}{\eta^2}\Big(\frac{\partial\hat{U}_x}{\partial s} + ik\hat{U}_x\Big) = \hat{S}_s(x,y,k), \end{aligned} \tag{2.64a}$$

$$\begin{aligned} &k^2\hat{U}_x + \frac{1}{\eta^2}\Big(2ik\frac{\partial\hat{U}_x}{\partial s} - k^2\hat{U}_x\Big) - \frac{\kappa^2}{\eta^2}\hat{U}_x \\ &\qquad + \frac{\partial^2\hat{U}_x}{\partial x^2} + \frac{\kappa}{\eta}\frac{\partial\hat{U}_x}{\partial x} + \frac{\partial^2\hat{U}_x}{\partial y^2} - \frac{2\kappa}{\eta^2}\Big(\frac{\partial\hat{U}_s}{\partial s} + ik\hat{U}_s\Big) = \hat{S}_x(x,y,k), \end{aligned} \tag{2.64b}$$

$$k^2\hat{U}_y + \frac{1}{\eta^2}\Big(2ik\frac{\partial\hat{U}_y}{\partial s} - k^2\hat{U}_y\Big) + \frac{\partial^2\hat{U}_y}{\partial x^2} + \frac{\kappa}{\eta}\frac{\partial\hat{U}_y}{\partial x} + \frac{\partial^2\hat{U}_y}{\partial y^2} = \hat{S}_y(x,y,k). \tag{2.64c}$$

Note, (2.64a) and (2.64b) still remain coupled from the Frenet-Serret transformation. This omission of the $\partial^2/\partial s^2$ terms is commonly referred to as the paraxial approximation and is valid in systems where the fields and sources move together in the same direction. More precisely, this approximation is an ad hoc method to ignore contributions to fields which travel away from the source. Thus, while the new equations (2.64) can be solved instead of (2.62), the solutions are very different if the $\partial^2/\partial s^2$ terms are not "small". One example of where the paraxial approximation is invalid, where the fields propagate in a direction different than that of the source, is when reflections occur due to a reflecting boundary such as a mirror.

Two important issues arise from the use of the paraxial equations. {1} How is the solution of (2.60a)–(2.60c) used to construct the initial fields of (2.64a)–(2.64c) and {2} when are the fields governed by (2.64a)–(2.64c), when transformed back to the time domain with (2.59), a good approximation to the fields which satisfy (2.60a)–(2.60c).

To address {1}, we consider the IBVP of (2.60a)–(2.60c) in an infinite pipe Ω with cross-section $\mathcal{A}$; that is, $\Omega = \{(s,x,y)|(x,y) \in \mathcal{A}\}$. We have $\mathbf{E}_0(s,x,y) = \mathbf{E}(s,x,y,0)$ and $\mathbf{B}_0(s,x,y) = \mathbf{B}(s,x,y,0)$ given as initial data in Ω. We replace s with $-ct$ in $\mathbf{E}_0$ and $\mathbf{B}_0$ and construct $\hat{\mathbf{E}}_0(x,y,k)$ and $\hat{\mathbf{B}}_0(x,y,k)$ by (2.59):

$$\hat{\mathbf{E}}_0(x,y,k) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \mathbf{E}_0(-ct,x,y)e^{ikct}dt \tag{2.65}$$

$$\hat{\mathbf{B}}_0(x,y,k) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \mathbf{B}_0(-ct,x,y)e^{ikct}dt \tag{2.66}$$

Now we evolve (2.64a)–(2.64c) in s using $\hat{\mathbf{E}}_0(x,y,k)$ and $\hat{\mathbf{B}}_0(x,y,k)$ as initial conditions. The transverse (x,y) boundary conditions of (2.60a)–(2.60c) on $\partial\Omega$ are converted naturally to boundary conditions for (2.64a)–(2.64c) provided they are not time nor s-dependent.

One key fact implicit in (2.65)–(2.66) is that we assume the solutions of (2.60a)–(2.60c) for $\mathbf{E}$ and $\mathbf{B}$ are plane-wave solutions of the form $\mathbf{F}(s-ct,x,y)$. This assumption constructs the solution at $s=0$ for all time for use in the Fourier transform in (2.65)–(2.66). Of course, the general solution of (2.60a)–(2.60c) admits solutions other than the form $\mathbf{F}(s-ct,x,y)$; in which case extracting the solution at $s=0$ would require knowledge of the fields for all time a priori.

To address {2}, we begin by examining the validity of taking $\partial^2\hat{U}/\partial s^2 = 0$. Solving for $\hat{U}$ we obtain:

$$\hat{U}(s,x,y,k) = \hat{F}(x,y,k) + s\hat{G}(x,y,k), \tag{2.67}$$

where $\hat{F}$ and $\hat{G}$ are arbitrary functions of (x,y,k). Next, taking the inverse Fourier transform with respect to k yields:

$$U(s,x,y,t) = F(s-ct,x,y) + sG(s-ct,x,y) \tag{2.68}$$

In particular, (2.68) admits plane-wave solutions of a rigid field profile moving in the $+s$ direction at speed c. Thus any plane-wave solutions of the form $F(s-ct,x,y)$ in (2.60a)–(2.60c) will exactly satisfy the paraxial equations (2.64a)–(2.64c) in the frequency-domain provided the trajectory is straight by $\kappa(s) \equiv 0$. Equivalently, if a solution of the form (2.68) is a good approximation to the solution of (2.60a)–(2.60c), then the paraxial approximation is valid.

Chapter 3

1D Frequency Domain Study – Proceedings of FEL2014

3.1 Statement of the Physical Problem

We start with the wave equation for the E_y and $H_y := (c/Z_0)B_y$ fields given by (2.12) in the Frenet-Serret coordinates $(\mathbf{r}, t)$:

$$\boldsymbol{\nabla}^2 E_y - \frac{1}{c^2}\frac{\partial^2 E_y}{\partial t^2} = Z_0\left(\frac{1}{c}\frac{\partial j_y}{\partial t}(\mathbf{r}, t) + c\frac{\partial \rho}{\partial y}(\mathbf{r}, t)\right), \tag{3.1a}$$

$$\boldsymbol{\nabla}^2 H_y - \frac{1}{c^2}\frac{\partial^2 H_y}{\partial t^2} = -\frac{1}{\eta}\frac{\partial j_x}{\partial s}(\mathbf{r}, t) + \frac{\partial j_s}{\partial x}(\mathbf{r}, t) + \frac{\kappa}{\eta} j_s(\mathbf{r}, t), \tag{3.1b}$$

where $\kappa(s)$ is the local curvature as in (2.28), $\eta(s, x) := 1 + \kappa(s)x$ and $\boldsymbol{\nabla}^2$ is defined from (2.39). We aim to solve for the fields in a rectangular cross-section vacuum chamber with perfectly conducting walls at $x = x_{\text{in}}, x_{\text{out}}$ and $y = \pm h/2$. The reference orbit is centered in the chamber and has bending radius R. We next apply the Fourier series in y from section 2.2.3, and the Fourier transform in $s - ct$ from section

2.2.4. With these transformations, the fields and sources are expressed in the form:

$$\begin{aligned} F(s,x,y,t) &= \int_{-\infty}^{\infty} dk\, e^{ik(s-ct)} \sum_{p=1}^{\infty} \phi_p(y)\hat{F}_p(s,x,k), \\ \phi_p(y) &= \begin{bmatrix} \cos \\ \sin \end{bmatrix} \Big(\alpha_p(y+h/2)\Big), \quad \alpha_p = \frac{\pi p}{h}. \end{aligned} \tag{3.2}$$

Note, recalling from section 2.2.3, if the vertical distribution of charge is an even function of y, which we assume, then only odd integers p are involved. Motivation by the appropriate boundary conditions for each field, the factor ϕ_p contains only cosine terms for $E_y,\ H_x,\ H_s,\ j_y$, whereas it contains only sine terms for $H_y,\ E_x,\ E_s,\ \rho,\ j_x,\ j_s$. After applying the transformations in (3.2), and using equations (2.62c), (3.1) with $\hat{F}_p(s,x,k)$ representing either $\hat{E}_{yp}$ or $\hat{H}_{yp}$ become:

$$\begin{aligned} &\frac{2ik}{\eta^2}\frac{\partial^2 \hat{F}_p}{\partial s^2} + \left(\frac{1}{\eta^2} - \frac{\kappa' x}{\eta^3}\right)\frac{\partial \hat{F}_p}{\partial s} + \frac{\partial^2 \hat{F}_p}{\partial x^2} + \frac{\kappa}{\eta}\frac{\partial \hat{F}_p}{\partial x} \\ &\qquad\qquad + \left(\gamma_p^2 - \frac{k^2\eta + ik\kappa' x}{\eta^3}\right)\hat{F}_p = \hat{S}_p(x,k), \end{aligned} \tag{3.3}$$

where $\gamma_p^2 = k^2 - \alpha_p^2$ and $S_p(x,k)$ is the transform of the specific right-hand-sides of (3.1) given explicitly in (3.5). In the approximation of slowly varying amplitude (paraxial approximation), where $\partial^2/\partial s^2$ terms are neglected, and in a constant curvature bend where $\kappa'(s) = 0$, the transformed equations (3.3) for $\hat{F}_p(s,x,k)$ become:

$$\frac{2ik}{\eta^2}\frac{\partial \hat{F}_p}{\partial s} = -\frac{\partial^2 \hat{F}_p}{\partial x^2} - \frac{\kappa}{\eta}\frac{\partial \hat{F}_p}{\partial x} - \left(\gamma_p^2 - \frac{k^2}{\eta^2}\right)\hat{F}_p + \hat{S}_p(x,k). \tag{3.4}$$

For the ribbon charge and current densities (2.46)–(2.47), the source terms $\hat{S}_p(x,k)$ for $\hat{E}_{yp},\ \hat{H}_{yp}$ in (3.4) are:

$$\hat{S}_{\hat{E}yp}(x,k) = \sigma_p(k)\delta(x), \qquad \hat{S}_{\hat{H}yp}(x,k) = \tau_p(k)\left(\delta'(x) + \delta(x)\kappa\right), \tag{3.5a}$$

$$\sigma_p(k) = qZ_0\alpha_p c\hat{\lambda}(k)G_p, \qquad \tau_p(k) = qc\hat{\lambda}(k)G_p, \tag{3.5b}$$

where $\hat{\lambda}$ and G_p are Fourier transforms of λ and G respectively. The factors $\sigma_p(k)$ and $\tau_p(k)$ are introduced here for convenience. For a Gaussian $G(y)$ with width $\sigma_y \ll h$ we have $G_p = (-1)^{(p-1)/2}(2/h)\exp(-(\alpha_p\sigma_y)^2/2)$.

Next, the perfectly conducting boundary conditions from section 2.2.2 are imposed by (2.50) and (2.51) and satisfy the following:

$$\hat{E}_{yp}\Big|_{x=x_{\text{in}},x_{\text{out}}} = 0, \qquad\qquad \frac{\partial \hat{H}_{yp}}{\partial x}\Big|_{x=x_{\text{in}},x_{\text{out}}} = 0. \tag{3.6}$$

As an aside, we must impose the Neumann boundary conditions on $\hat{H}_{yp}$ since we are evolving the wave equations which require a boundary condition for each field on each boundary. To construct initial conditions for $\hat{E}_{yp}, \hat{H}_{yp}$, we assume an infinite straight prior to the entrance of the bend and use the steady-state solutions $\hat{F}_{p0} = \hat{E}_{yp0},\ \hat{H}_{yp0}$ which satisfy $\partial \hat{F}_{p0}/\partial s = 0$ with $\kappa = 0$, thus from (3.4) we have:

$$\frac{d^2 \hat{F}_{0p}}{dx^2} - \alpha_p^2 \hat{F}_{0p} = \hat{S}_p(x,k). \tag{3.7}$$

With the solutions to (3.7) as initial conditions for $(\hat{E}_{yp}, \hat{H}_{yp})$ we pose the initial boundary value problem given by (3.4)–(3.7) for each k and p. We don't require any initial conditions for the remaining fields: $(\hat{E}_{sp}, \hat{E}_{xp}, \hat{H}_{sp}, \hat{H}_{xp})$ since they can be obtained from $(\hat{E}_{yp}, \hat{H}_{yp})$ through additional relations discussed in (3.42).

3.2 Statement of the Computational Problem

To solve (3.4) numerically, we first introduce a transformation, credited to Robert Warnock, to remove the singularities of (3.5a):

$$V_p(s,x,k) = \hat{E}_{yp}(s,x,k) - \sigma_p(k) x \Theta(x), \tag{3.8a}$$

$$W_p(s,x,k) = \hat{H}_{yp}(s,x,k) - \tau_p(k) \Theta(x), \tag{3.8b}$$

with $\Theta(x)$ being the Heaviside step function. Under these transformations, (V_p, W_p) still satisfy (3.4); except, the sources now become:

$$S_{Vp}(x,k) = -\sigma_p(k)\Theta(x)\left[\frac{\kappa}{\eta} + x\left(\gamma_p^2 - \frac{k^2}{\eta^2}\right)\right], \tag{3.9a}$$

$$S_{Wp}(x,k) = -\tau_p(k)\Theta(x)\left(\gamma_p^2 - \frac{k^2}{\eta}\right). \tag{3.9b}$$

We have eliminated δ and δ' from the source, allowing a standard numerical implementation. A jump in the sources at $x = 0$ remains, and while that too can be removed by a second transformation, it is not necessary here since we only require that the source terms be bounded and resolvable numerically.

The boundary conditions for V_p and W_p are:

$$\begin{aligned} V_p|_{x=x_{in}} &= 0, & V_p|_{x=x_{out}} &= -\sigma_p x_{\text{out}}, \\ \left.\frac{\partial W_p}{\partial x}\right|_{x=x_{\text{in}}} &= 0, & \left.\frac{\partial W_p}{\partial x}\right|_{x=x_{\text{out}}} &= 0. \end{aligned} \tag{3.10}$$

Next, we rewrite (3.4) as:

$$i\frac{\partial u}{\partial s} = a(x)\frac{\partial^2 u}{\partial x^2} + b(x)\frac{\partial u}{\partial x} + c(x)u + \tilde{S}_u(x) \tag{3.11}$$

with u representing either V_p or W_p, and $\tilde{S}_u = \eta^2 S_{Vp,Wp}/2k$. We have omitted the k and p dependence for convenience. It is important to note that our Schrödinger-type equation (3.11) is only parabolic in the sense of infinite propagation speed. It is hyperbolic-like in the sense that initial conditions are not smoothed by the PDE evolution. The solutions of (3.7) to provide initial conditions are found readily by variation of parameters:

$$\begin{aligned} \hat{E}_{yp0}(x,k) = &-\frac{\sigma_p(k)}{\alpha_p}\frac{\sinh(\alpha_p x_{\text{out}})\sinh(\alpha_p(x-x_{\text{in}}))}{\sinh(\alpha_p(x_{\text{out}}-x_{\text{in}}))} \\ &+\frac{\sigma_p(k)}{\alpha_p}\sinh(\alpha_p x)\Theta(x), \end{aligned} \tag{3.12a}$$

$$\begin{aligned} \hat{H}_{yp0}(x,k) = &-\tau_p(k)\frac{\sinh(\alpha_p x_{\text{out}})\cosh(\alpha_p(x-x_{\text{in}}))}{\sinh(\alpha_p(x_{\text{out}}-x_{\text{in}}))} \\ &+\tau_p(k)\cosh(\alpha_p x)\Theta(x). \end{aligned} \tag{3.12b}$$

3.3 Numerical Implementation

In this section we describe our numerical algorithms for integrating (3.11) by finite difference (FD) and nodal discontinuous Galerkin (DG) numerical schemes. We use explicit time-stepping in s for both methods.

3.3.1 Finite Difference Scheme

We begin by discretization of $[x_{\text{in}}, x_{\text{out}}]$ into $N_{\text{res}}+1$ equidistant nodes spaced by Δx. The nodal coordinates are defined by $x_i = x_{\text{in}} + (i-1)\Delta x$ and $u_i = u(x_i)$. We define the 4th order differentiation operators by:

$$\begin{aligned} \left.\frac{du}{dx}\right|_{x_i} &\approx \frac{u_{i-2} - 8u_{i-1} + 8u_{i+1} - u_{i+2}}{12\Delta x}, \\ \left.\frac{d^2u}{dx^2}\right|_{x_i} &\approx \frac{-u_{i-2} + 16u_{i-1} - 30u_i + 16u_{i+1} - u_{i+2}}{12\Delta x^2}. \end{aligned} \tag{3.13}$$

For points near the boundaries, i.e. $i = 1, 2, N_{\text{res}}, N_{\text{res}} + 1$, we use lopsided 5 point stencils. For explicit time stepping, we employ a leap-frog scheme. It is important to note: the leap-frog scheme is unstable for the heat equation [19] and does have a weak instability for our Schrödinger-type equation in (3.11). Von Neumann analysis asserts that this weak instability, due to the $b(x)\partial u/\partial x$ term in (3.11), is not significant in the parameter studies we consider with $\eta \sim 1$. The leap-frog scheme is:

$$u^{n+1} = u^{n-1} + 2\Delta s \Phi(u^n), \tag{3.14}$$

with $\Phi(u^n)$ denoting the right-hand-side of (3.11) with the discretization (3.13) at $s = n\Delta s$. For stability, we take:

$$\Delta s = C_{CFL} k \Delta x^2. \tag{3.15}$$

In our tests, we determine experimentally that $C_{CFL} < 0.3$ results in a stable scheme for $p \lesssim 40$. Due to the sharp peaks in the solution for large p, a higher resolution must be used if larger p-modes are desired. This will be discussed in section 3.4.2.

The boundary conditions on V_p are imposed by setting $V_{p;1} = 0$ and $V_{p;N_{\text{res}}+1} = -\sigma_p x_{\text{out}}$ while the boundary conditions for W_p are imposed with a one-sided derivative stencil and solving for $W_{p;1}$ and $W_{p;N_{\text{res}}+1}$ as functions of the nearby grid values

$W_{p;2,\ldots,5}$ and $W_{p;N_{\text{res}}-3,\ldots,N_{\text{res}}}$ respectively. The initial conditions on (V_p, W_p) are computed analytically from (3.12).

3.3.2 Discontinuous Galerkin Scheme

We derive our DG scheme in some detail since it is not as well known as FD. DG methods have features taken from finite element (FE) and finite volume methods (FV). Solutions are represented by polynomials local to each element as in FE; however, the PDE can be represented in an explicit semi-discrete form. The PDE is satisfied using fluxes between elements similar to FV. This results in a scheme which can maintain high-order accuracy (*hp*-adaptivity) and stability for wave-dominated problems [11].

We rewrite the PDE in (3.11) for both V_p or W_p each as a first order system:

$$i\frac{\partial u}{\partial s} = a(x)\frac{\partial q}{\partial x} + b(x)q + c(x)u + f(x), \qquad q = \frac{\partial u}{\partial x}. \tag{3.16}$$

To find an approximate solution, we begin by partitioning the domain $[x_{\text{in}}, x_{\text{out}}]$ into K elements. In each element, we represent the solution as a polynomial of degree N. Thus total number of nodes is given by $(N+1)K$. We next focus on a particular element $D^k = [x_1^k, x_{N+1}^k]$ (note, this k is *not* related to Fourier transform frequency variable in (3.2)). We approximate the solution on this element in the Lagrange polynomial basis:

$$u^k(x,s) = \sum_{j=1}^{N+1} u_j^k(s)\ell_j^k(x), \tag{3.17}$$

$$q^k(x,s) = \sum_{j=1}^{N+1} q_j^k(s)\ell_j^k(x), \tag{3.18}$$

with $\ell_j^k(x_i^k) = \delta_{ij}$ for nodal coordinates x_i^k. The derivatives of u in (3.16) are given by differentiating (3.17). The terms bq, cu and f are replaced by their natural interpolating polynomials

$$\begin{aligned}(bq)^k &= \sum_{j=1}^{N+1} b_j^k q_j^k(s)\ell_j^k(x),\\ (cu)^k &= \sum_{j=1}^{N+1} c_j^k u_j^k(s)\ell_j^k(x),\\ f^k &= \sum_{j=1}^{N+1} f_j^k(s)\ell_j^k(x),\end{aligned} \tag{3.19}$$

and $\partial q/\partial x$ is replaced by

$$\left(\frac{\partial q}{\partial x}\right)^k = \sum_{j=1}^{N+1} q_j^k(s)[\ell_j^k(x)]'. \tag{3.20}$$

Note that each of these is a polynomial of degree N or $N-1$. Inserting (3.17), (3.19), (3.20) into (3.16) we obtain the following residuals $\mathcal{R}_{1,2}^k(x,s)$ defined by

$$\mathcal{R}_1^k(x,s) := i\frac{\partial u^k}{\partial s} - a^k\frac{\partial q^k}{\partial x} - (bq)^k - (cu)^k - f^k, \tag{3.21a}$$

$$\mathcal{R}_2^k(x,s) := q^k - \frac{\partial u^k}{\partial x}. \tag{3.21b}$$

If we require that the residuals $\mathcal{R}_{1,2}^k(x,s)$ be orthogonal to the ℓ_i^k, this yields $N+1$ equations for each (u_j^k, q_j^k) of the form

$$\int_{D^k} \mathcal{R}_1^k(x,s)\ell_i^k(x)dx = 0, \qquad \int_{D^k} \mathcal{R}_2^k(x,s)\ell_i^k(x)dx = 0, \tag{3.22}$$

for $i = 1, ..., N+1$. Clearly, requiring these to be zero will not yield a viable algorithm, as there would be no coupling between elements. The heart of DG is to couple adjacent elements using the numerical flux. This is obtained by integrating the $[\ell_i^k(x)]'$ terms by parts, inserting the flux condition, and then reversing the integration by parts. We illustrate this on $\mathcal{R}_2^k(x,s)$:

$$\begin{aligned}\int_{D^k} -\frac{\partial u^k}{\partial x}\ell_i^k dx = \int_{D^k} u^k \ell_i^k(x)' dx - \left[u^k \ell_i^k\right]_{x_1^k}^{x_{N+1}^k} \approx \\ \int_{D^k} u^k \ell_i^k(x)' dx - \left[u^* \ell_i^k\right]_{x_1^k}^{x_{N+1}^k} = \\ \int_{D^k} -\frac{\partial u^k}{\partial x}\ell_i^k dx + \left[(u^k - u^*)\ell_i^k\right]_{x_1^k}^{x_{N+1}^k}\end{aligned} \tag{3.23}$$

and the calculation is identical for the x derivative of q in (3.22). The approximation step introduces u^* and q^* which give rise to the numerical fluxes which combine boundary information from the two elements in contact at the interface at x_1^k and x_{N+1}^k. While many choices of numerical fluxes exist, we opt to use a local DG flux for better convergence properties [11]. These numerical fluxes have the form:

$$\begin{aligned} u^*(x_1^k) = u_1^k, \quad u^*(x_{N+1}^k) = u_1^{k+1}, \\ q^*(x_1^k) = q_{N+1}^{k-1}, \quad q^*(x_{N+1}^k) = q_{N+1}^k. \end{aligned} \tag{3.24}$$

Remark: at this point we will present the motivation for the choice of fluxes in (3.24) which lead to a stable scheme. A requirement for stability is that the total energy $|u|^2$ be non-increasing in s; we enforce this by ensuring the energy for a single element is non-increasing. For this derivation, we consider the simpler problem with $a(x) = -1$, $b(x) = 0$, $c(x) = 0$, and $f(x) = 0$:

$$\frac{\partial u}{\partial s} = i\frac{\partial q}{\partial x}, \qquad q = \frac{\partial u}{\partial x}. \tag{3.25}$$

Now, constructing the residuals as in (3.21) leads to:

$$\int_{D^k} \mathcal{R}_1^k(x,s) v^k(x) dx = 0, \qquad \int_{D^k} \mathcal{R}_2^k(x,s) w^k(x) dx = 0, \tag{3.26}$$

for any test functions $v^k(x)$ and $w^k(x)$ in the same space of polynomials spanned by $\ell_i^k(x)$ for $i = 1, ..., N+1$. To examine the local energy $|u^k|^2$ it is convenient to choose $v^k = \bar{u}^k$ and $w^k = \bar{q}^k$, where the bar $^-$ denotes complex conjugation. Integrating the residual expressions by parts, with our choices for v^k and w^k, we obtain:

$$\int_{D^k} \frac{\partial u^k}{\partial s}\bar{u}^k + iq^k\frac{\partial \bar{u}^k}{\partial x}dx - i\left[q^*\bar{u}^k\right]_{x_1^k}^{x_{N+1}^k} = 0, \tag{3.27}$$

$$\int_{D^k} q^k\bar{q}^k + u^k\frac{\partial \bar{q}^k}{\partial x}dx - \left[u^*\bar{q}^k\right]_{x_1^k}^{x_{N+1}^k} = 0, \tag{3.28}$$

where we have introduced the numerical fluxes u^* and q^* as before but have not yet specified them. Now, integrating (3.28) by parts once more we obtain:

$$\int_{D^k} q^k\bar{q}^k - \bar{q}^k\frac{\partial u^k}{\partial x}dx - \left[(u^* - u^k)\bar{q}^k\right]_{x_1^k}^{x_{N+1}^k} = 0. \tag{3.29}$$

Next, we complex conjugate (3.27) and (3.29) respectively to obtain:

$$\int_{D^k} \frac{\partial \bar{u}^k}{\partial s}u^k - i\bar{q}^k\frac{\partial u^k}{\partial x}dx + i\left[\bar{q}^*u^k\right]_{x_1^k}^{x_{N+1}^k} = 0, \tag{3.30}$$

$$\int_{D^k} \bar{q}^k q^k - q^k\frac{\partial \bar{u}^k}{\partial x}dx - \left[(\bar{u}^* - \bar{u}^k)q^k\right]_{x_1^k}^{x_{N+1}^k} = 0. \tag{3.31}$$

Now we add equations (3.27) and (3.30) to arrive at:

$$\int_{D^k} \frac{\partial u^k}{\partial s}\bar{u}^k + \frac{\partial \bar{u}^k}{\partial s}u^k + iq^k\frac{\partial \bar{u}^k}{\partial x} - i\bar{q}^k\frac{\partial u^k}{\partial x}dx - i\left[q^*\bar{u}^k - \bar{q}^*u^k\right]_{x_1^k}^{x_{N+1}^k} = 0. \tag{3.32}$$

Noting that $(\partial u^k/\partial s)\bar{u}^k + (\partial \bar{u}^k/\partial s)u^k = \partial |u^k|^2/\partial s$ is the s derivative of the energy, and substituting (3.29) and (3.31) into (3.32), we obtain:

$$\frac{\partial}{\partial s}\int_{D^k} |u^k|^2 dx = -i\left[\bar{q}^*u^k - q^*\bar{u}^k - \bar{u}^*q^k + \bar{u}^k q^k + u^*\bar{q}^k - u^k\bar{q}^k\right]_{x_1^k}^{x_{N+1}^k}. \tag{3.33}$$

For stability, we require that the energy be non-increasing in s on each element D^k. Thus at each interface between elements, we must choose a u^* and q^* which ensures the net contribution of the flux from each elements adjoining at the interface of the right-hand-side of (3.33) be less than or equal to zero. We now show that (3.24) results in a stable method. Considering the interface between elements D^k and D^{k+1} we note the energy flux at the interface is:

$$\begin{aligned}\text{Flux} = -i[&\bar{q}^*u^{k+1} - q^*\bar{u}^{k+1} - \bar{u}^*q^{k+1} + \bar{u}^{k+1}q^{k+1} + u^*\bar{q}^{k+1} - u^{k+1}\bar{q}^{k+1} \\ &-\bar{q}^*u^k + q^*\bar{u}^k + \bar{u}^*q^k - \bar{u}^k q^k - u^*\bar{q}^k + u^k\bar{q}^k].\end{aligned} \tag{3.34}$$

Inserting (3.24), i.e. $u^* = u^{k+1}$ and $q^* = q^k$ into (3.34) we obtain Flux $= 0$. Equivalently, we could have chosen $u^* = u^k$ and $q^* = q^{k+1}$ for a stable scheme. The important aspect in the choice of numerical flux is that u and q are taken from opposite directions at the interface. This concludes our remark on the motivation for the choice of numerical flux. For more details on deriving these numerical flux conditions, see [21].

Returning to the original equations (3.16) and continuing onward from (3.24), we define the vectors $\boldsymbol{\ell}^k(x) = (\ell_1^k(x), ..., \ell_{N+1}^k(x))^\top$ and similarly for $\mathbf{u}^k(s)$, $\mathbf{q}^k(s)$, and $\mathbf{f}^k$. We also introduce the mass and stiffness matrices:

$$\mathcal{M}_{ij}^k = \int_{D^k} \ell_i^k \ell_j^k dx, \qquad \mathcal{S}_{ij}^k = \int_{D^k} \frac{d\ell_i^k}{dx} \ell_j^k dx. \tag{3.35}$$

Thus (3.22) and (3.23) yield:

$$\mathbf{q}^k = (\mathcal{M}^k)^{-1}\mathcal{S}^k\mathbf{u}^k - (\mathcal{M}^k)^{-1}\left[(u-u^*)\boldsymbol{\ell}^k\right]_{x_1^k}^{x_{N+1}^k}, \tag{3.36a}$$

$$\begin{aligned} i\frac{d\mathbf{u}^k}{ds} =& \mathcal{A}^k(\mathcal{M}^k)^{-1}\mathcal{S}^k\mathbf{q}^k - \mathcal{A}^k(\mathcal{M}^k)^{-1}\left[(q-q^*)\boldsymbol{\ell}^k\right]_{x_1^k}^{x_{N+1}^k} \\ &+ \mathcal{B}^k\mathbf{q}^k + \mathcal{C}^k\mathbf{u}^k + \mathbf{f}^k. \end{aligned} \tag{3.36b}$$

where $\mathcal{A}^k = \operatorname{diag}(a_1^k, ..., a_{N+1}^k)$ and similarly for $\mathcal{B}^k$ and $\mathcal{C}^k$. We introduce a slight error in (3.36b) from commuting $(\mathcal{M}^k)^{-1}$ and $\mathcal{A}^k$. This is acceptable since $\mathcal{A}^k$ does not vary much within each element and this error scales as $\max_{i,j} |\partial a/\partial x||x_i - x_j|N^{-1}$ over the element k which is described in detail in [11] on p254.

In our code, we combine all K elements, thus we can arrange the solution $\mathbf{u}$ and $\mathbf{q}$ as $(N+1) \times K$ arrays with mass and stiffness matrices $\mathcal{M}$ and $\mathcal{S}$ common to all elements. This approach enables the right-hand-side operations to be done using dense matrix-matrix multiplication. Boundary conditions are handled by adjustment of the fluxes (3.24). Dirichlet conditions for V_p are imposed by:

$$\begin{aligned} u_V^*(x_1^1) &= 0, \qquad & u_V^*(x_{N+1}^K) &= -\sigma_p x_{\text{out}}, \\ q_V^*(x_1^1) &= q_1^1, \qquad & q_V^*(x_{N+1}^K) &= q_{N+1}^K, \end{aligned} \tag{3.37}$$

and Neumann conditions for W_p are imposed on q^* instead of u^*:

$$\begin{aligned} u^*_W(x^1_1) &= u^1_1, \qquad u^*_W(x^K_{N+1}) = u^K_{N+1}, \\ q^*_W(x^1_1) &= 0, \qquad q^*_W(x^K_{N+1}) = 0, \end{aligned} \tag{3.38}$$

The $(\mathbf{u}, \mathbf{q})$ systems for V_p and W_p are evolved in s separately using a 4th order low-storage Runge-Kutta scheme. The restriction for Δs scales as in (3.15) with Δx as the minimal distance between two nodes on an element. For the x^k_i nodal locations in (3.17)–(3.18), we use Legendre-Gauss-Lobatto quadrature nodes (see p.43-51 in [11]) due to advantages in matrix conditioning; however, this choice of nodes imposes severe restrictions on Δs at high orders N. A balance of K and N is needed for optimal efficiency.

3.4 Numerical Results

In this section we examine several aspects for a line charge model (i.e. $\sigma_y \to 0$ and thus $G(y) \to \delta(y)$). First we compare results for V_p in the FD and DG schemes of the previous section, emphasizing both computational accuracy and efficiency. Next, we examine the convergence of the Fourier series sum over p of $\hat{E}_{yp}$. Lastly, we compute the longitudinal impedance from a relation for $\hat{E}_{sp}$ using V_p and W_p to compare to results presented in [22, 23].

3.4.1 Finite Difference versus Discontinuous Galerkin

For our numerical tests, we use the following parameters (as in [5, 22, 23]):

x_{in}	$= -0.030\,\text{m}$	x_{out}	$= 0.030\,\text{m}$
h	$= 0.020\,\text{m}$	κ	$= 1.000\,\text{m}^{-1}$
q	$= 10^{-12}\,\text{C}$	β	$= 1$
k	$= 8 \cdot 10^3\,\text{m}^{-1}$	p	$= 1$

A common grid of points in x is used for the error comparisons of both FD and DG methods. A reference solution for V_p at $s = 0.200\,\text{m}$ is computed by the FD scheme with $N_{\text{res}} = 5760$. The FD scheme errors are displayed in Table 3.1.

Table 3.1: FD $\hat{E}_{yp}$ Error and Computation Time

N_{res}	240	480	960	1920
L^∞ Error	1.46e-4	1.06e-5	1.31e-6	4.58e-7
L^2 Error	1.24e-4	9.08e-6	1.40e-6	4.61e-7
Time	0.4 s	2.4 s	6.9 s	15.0 s

While our FD stencil is fourth-order accurate, the method exhibits lesser-order self-convergence (convergence with respect to a higher resolution solution). We attribute this reduction in order of convergence to the jump discontinuities at $x = 0$ of the source terms in (3.11). The use of a higher-order stencil was partly unnecessary in this context and a simpler second-order stencil would have performed at the same level of convergence albeit with higher error constants.

The fourth-order stencil would exhibit its optimal convergence rate if additional transformations were made to V_p and W_p to smooth the source terms further. Specifically, this would require transforming V_p and W_p so that the $\Theta(x)$ terms in (3.9) have the form $x^2\Theta(x)$, ensuring enough smoothness on the initial conditions to enable the fourth-order convergence. More details on this issue for FD can be found on page 214 [19].

Next, the DG scheme with the same parameters is compared for varying order N and elements K. A reference solution for V_p is computed using $(N, K) = (12, 80)$

which is compared on the set of nodes for $(N, K) = (2, 20)$, common to all solutions. The common set of nodes x_i^c lie at the locations $x_i^c = x_{\text{in}} + (i-1)(x_{\text{out}} - x_{\text{in}})/40$ for $i = 1, ..., 41$. Note, the DG solutions are multivalued along the interior interfaces between elements located on the common set of nodes at x_{2m+1}^c for integers m. Thus while there are a total 60 of nodes in the $(N, K) = (2, 20)$ solution, there are only 41 unique coordinates. We take the average of the solutions on these multivalued DG nodes for our comparison.

Furthermore, the DG reference solution with $(N, K) = (12, 80)$, while different from the FD reference solution with $N_{\text{res}} = 5760$, exhibits a maximum point-wise relative difference of 1.62e-7 over the 41 common coordinates x_i^c.

Table 3.2: DG $\hat{E}_{yp}$ Error and Computation Time

$K \backslash N$	2	4	6	8
	2.91e-2	9.01e-3	5.67e-4	1.97e-5
20	3.69e-2	5.22e-3	2.67e-4	8.49e-6
	0.04 s	0.22 s	1.0 s	2.8 s
	8.89e-3	1.92e-4	2.18e-6	3.36e-8
40	9.08e-3	2.03e-4	2.13e-6	3.10e-8
	0.10 s	1.0 s	4.4 s	14 s
	1.88e-3	1.22e-5	3.64e-8	1.82e-9
80	1.26e-3	7.98e-6	2.23e-8	1.96e-9
	0.50 s	5.0 s	24 s	77 s

In Table 3.2, we list the L^∞ (top), L^2 (middle), and CPU times (bottom) for varying values of K and N. We observe spectral convergence, where the errors scale with the element sizes h_0 and polynomial order N as h_0^{N+1} with $h_0 \propto 1/K$. The DG method overcomes the issue of reduction in the order of convergence due to discontinuous source terms if an element interface is placed at the discontinuity. For this example, we ensure an element vertex is placed at $x = 0$. Both FD and DG methods display comparable efficiency by comparing the CPU times in the tables for similar errors for the parameter ranges we studied.

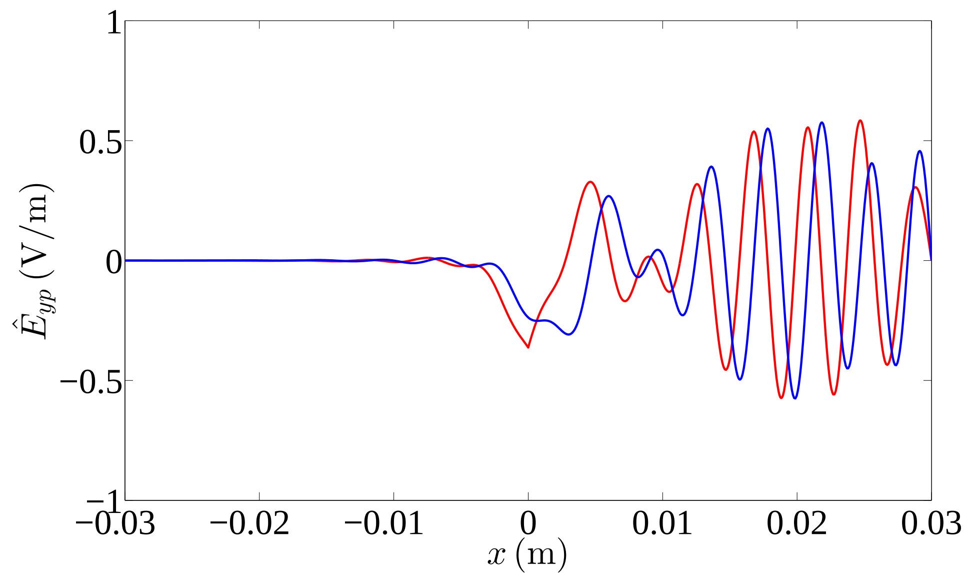


Figure 3.1: Real (red) and imaginary (blue) parts of DG reference solution for $\hat{E}_{yp}$ using $(N, K) = (12, 80)$, for $p = 1$, $k = 8 \cdot 10^3\,\mathrm{m}^{-1}$, at $s = 0.200\,\mathrm{m}$. The corner at $x = 0$ is due to the $\delta(x)$ term in the sources ρ and $\mathbf{j}$.

3.4.2 Fourier Series Convergence

To construct the solutions in y, we use the Fourier series as defined in (3.2). We concentrate on $\hat{E}_y$ defined as:

$$\hat{E}_y(s, x, y, k) = \sum_{p=1}^{\infty} \hat{E}_{yp}(s, x, k) \cos\left(\alpha_p(y + h/2)\right). \tag{3.39}$$

It is important to note that for the line source $G(y) = \delta(y)$, while each $\hat{E}_{yp}$ is bounded, the infinite sum introduces a singularity at $(x, y) = (0, 0)$ for a line charge model. One method to study the L^2 self-convergence of the partial sums in (3.39), is to subtract the singularity by adopting the approach used in [1, 2, 20, 22, 23]. The singular term in $\hat{E}_y$, by the expression in [2], is:

$$\hat{E}_y^b = \frac{qZ_0c\hat{\lambda}(k)}{2\pi}\frac{y}{x^2 + y^2}. \tag{3.40}$$

Defining $\hat{E}_y^r := \hat{E}_y - \hat{E}_y^b$, we obtain

$$\hat{E}_y^r = \sum_{p=1}^{\infty} \left[\hat{E}_{yp} - \hat{E}_{yp}^b\right] \cos\left(\alpha_p(y + h/2)\right), \tag{3.41}$$

with $\hat{E}_{yp}^b$ as the Fourier series components of (3.40). Thus, we can examine the L^2 self-convergence of $\hat{E}_y^r$. We take the $p_{\max} = 39$ partial sum ($p = 1, 3, ..., p_{\max}$) as a reference solution for (3.41) for $k = 8 \cdot 10^3\,\mathrm{m}^{-1}$. This solution is compared to the partial sums for varying $p_{\max}$.

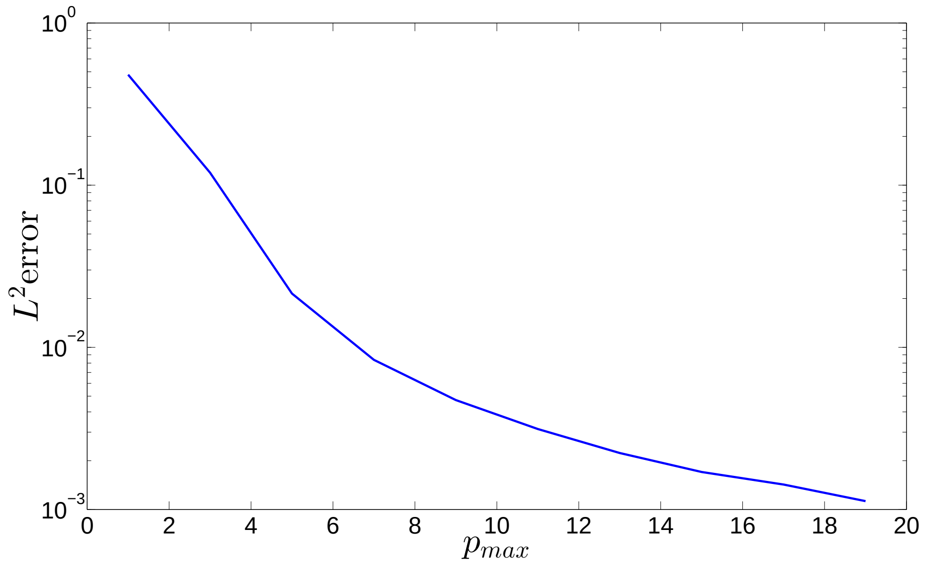


Figure 3.2: L^2 error $\hat{E}_y^r$ for varying $p_{\max}$ using $(N, K) = (12, 80)$ evaluated on a 1441×481 (x, y) grid, for $k = 8 \cdot 10^3\,\mathrm{m}^{-1}$, at $s = 0.200\,\mathrm{m}$ using the line charge model $G(y) = \delta(y)$.

The self-convergence plot in Figure 3.2 shows only a few p-modes are necessary to obtain a solution for $\hat{E}_y^r$ accurate to 10^{-2} for our choice of parameters. The solutions were computed in ~100 seconds for $p_{\max} = 19$ at this resolution. In contrast, in [5], this took ~1000 seconds for the same parameters using a 2D FD method. Thus, this 1D approach provides a large computational speed-up over the 2D FD method in [5] using the same computer hardware. The reference solution of $\hat{E}_y^r$ as a surface plot is displayed in Figure 3.4.

Alternatively, we can examine the convergence of $\hat{E}_y$ directly if we use the Gaussian source with $\sigma_y \neq 0$ where $G_p = (-1)^{(p-1)/2}(2/h)\exp(-(\alpha_p\sigma_y)^2/2)$. For this convergence we will examine the L^∞ norm of the p-modes: $\hat{E}_{yp}$ with increasing p. Since $G_p \sim \exp(p^{-2})$ and scales each field, then the Fourier series $\hat{E}_y$ converges uniformly for all x as $p_{\max} \to \infty$. If σ_y is very small however, a larger $p_{\max}$ must be used to resolve the Gaussian $G(y)$. This effect is shown in Figure 3.3 where the L^∞ norm of the p-modes decays rapidly in p once the Gaussian $G(y)$ is resolved (i.e. when G_p is small). For the line source model $G(y) = \delta(y)$, where $|G_p| = 2/h$, the maximum norm of the p-modes do not decay in p at $x = 0$.

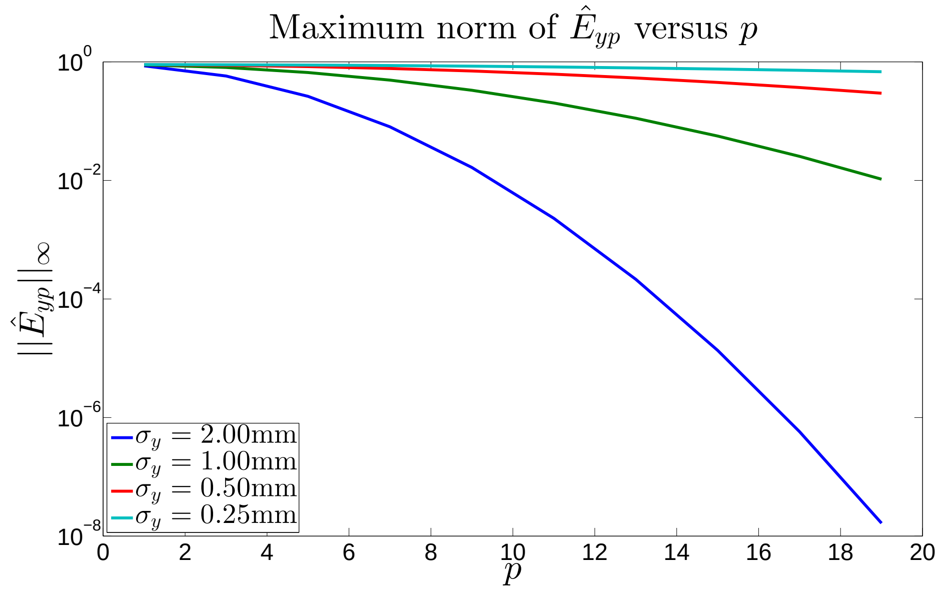


Figure 3.3: L^∞ norm $\hat{E}_{yp}$ for varying p using a Gaussian source $G(y)$ with varying σ_y. The solution is computed at $s = 0.200$ m using the DG method with $(N, K) = (8, 30)$.

3.4.3 CSR Impedance

In this section we compute the longitudinal impedance by examining the dependence of $\hat{E}_s$ on k/κ. The $\hat{E}_s$ field Fourier series components are given in terms of $V_p(s,x,k)$ and $W_p(s,x,k)$ by:

$$\hat{E}_{sp}(s,x,k) = \frac{-1}{\gamma_p^2}\left[\frac{\alpha_p}{\eta}\left(ik\left(V_p+\sigma_p x\Theta\right)+\frac{\partial V_p}{\partial s}\right)-ikZ_0\frac{\partial W_p}{\partial x}\right]. \tag{3.42}$$

We mention here that the transforms in $s-ct$ and y also yield formulas for $\hat{E}_{xp}$, $\hat{H}_{xp}$, $\hat{H}_{sp}$ which depend only on V_p and W_p and their s and x derivatives. The $\partial/\partial s$ terms in (3.42) are computed by the right-hand-sides of (3.11).

In the case of the smooth Gaussian $G(y)$, the high p-mode terms are attenuated by $G_p \sim \exp(p^{-2})$ and thus while the sharp peaks in $\hat{E}_{yp}$ and $\hat{H}_{yp}$ become badly resolved, the G_p factor scaling the fields dominates the bound on these sharp peaks. Thus a maximum resolution can be used in x for all p-modes since the sharp peak in the higher p-mode solutions are greatly attenuated.

However, in the case of the line source $G(y)=\delta(y)$, a resolution issue occurs as p increases. This issue is seen in the calculations of $\hat{E}_{yp}$ and $\hat{H}_{yp}$ which become very sharply peaked at $x=0$ with increasing p and thus finer spatial resolution is required as the desired accuracy is increased. For large p-modes using the line source model: $G_p=(-1)^{(p+1)/2}(2/h)$, the $\hat{E}_{yp}$ and $\hat{H}_{yp}$ fields become arbitrarily sharp at $x=0$. This issue implies that if the higher p-modes are desired, finer grids in x are required. Thus, for $\sigma_y=0$, while $\hat{E}_s$ can be computed from:

$$\hat{E}_s(s,x,y,k)=\sum_{p=1}^{\infty}\hat{E}_{sp}(s,x,k)\cos\left(\alpha_p(y+h/2)\right), \tag{3.43}$$

the sharply peaked solution: $(V_p+\sigma_p x\Theta)$ and $\partial W_p/\partial x$ in (3.42) for large p introduces a large error in $\hat{E}_{sp}(s,x,k)$ at $x=0$. The plot for $\hat{E}_{yp}(s,x,k)$ for $p=19$ with $\sigma_y=0$ is shown in Figure 3.5. As p increases, the peak in Figure 3.5 becomes narrower but its height stays constant.

As a workaround, the solution $\hat{E}_s(s, x, y, k)$ computed with equation (3.43) can be evaluated at $(x, y) = (0, 0)$ by using interpolation. We interpolate each p-mode $\hat{E}_{sp}$ at $x = 0$ with a spline formed by nodes near $x = 0$. We then sum the interpolated values at $x = 0$ to obtain $\hat{E}_s(s, 0, 0, k)$.

Figure 3.6 shows the value of $\hat{E}_s$ at the end of the bend: $s_{\max} = 0.200\,\mathrm{m}$ sampled at the source location $(x, y) = (0, 0)$. The spline interpolation was not necessary for this plot as the source distribution $G(y)$ was taken to be a Gaussian with $\sigma_y = 2\,\mathrm{mm}$.

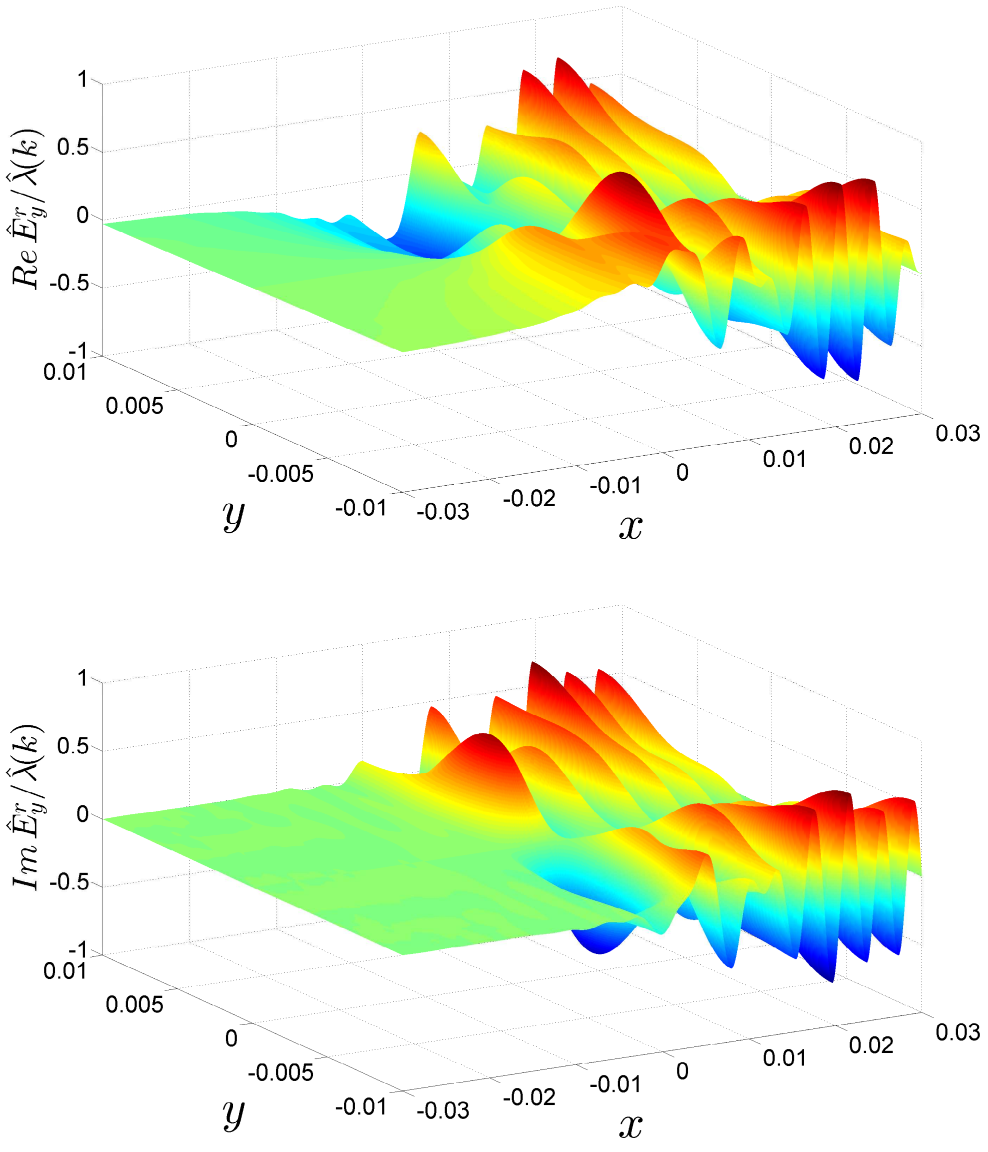


Figure 3.4: DG reference solution for $Re(\hat{E}_y^r)/\hat{\lambda}(k)$ (top), $Im(\hat{E}_y^r)/\hat{\lambda}(k)$ (bottom) with $p_{\max} = 39$ and $(N, K) = (12, 80)$ evaluated on a 1441×481 (x, y) grid, for $k = 8 \cdot 10^3\,\mathrm{m}^{-1}$, at $s = 0.200\,\mathrm{m}$.

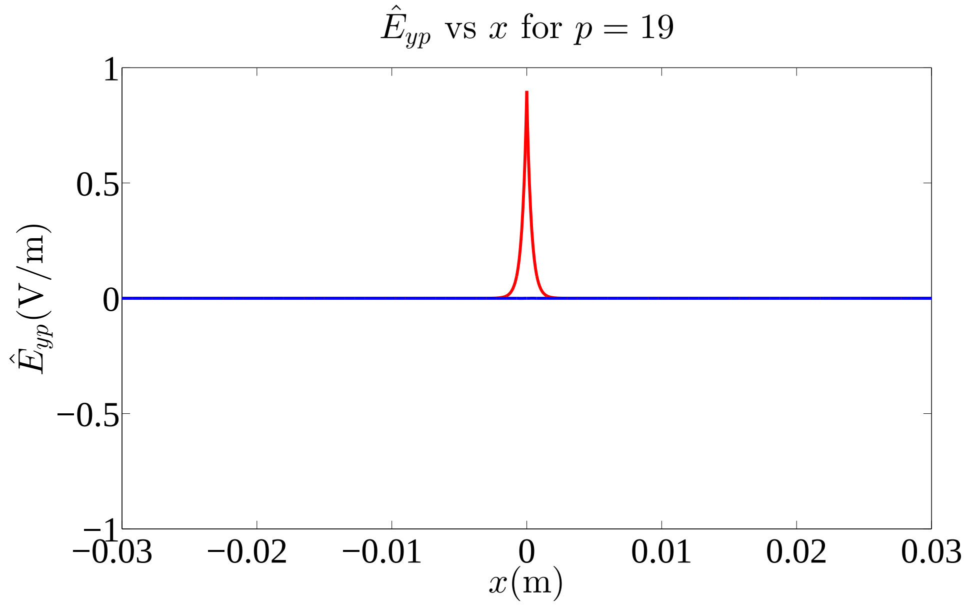


Figure 3.5: Real (red) and imaginary (blue) parts of $\hat{E}_{yp}$ for $p = 19$ with using $(N, K) = (12, 80)$, for $k = 8 \cdot 10^3\,\text{m}^{-1}$, at $s = 0.200\,\text{m}$.

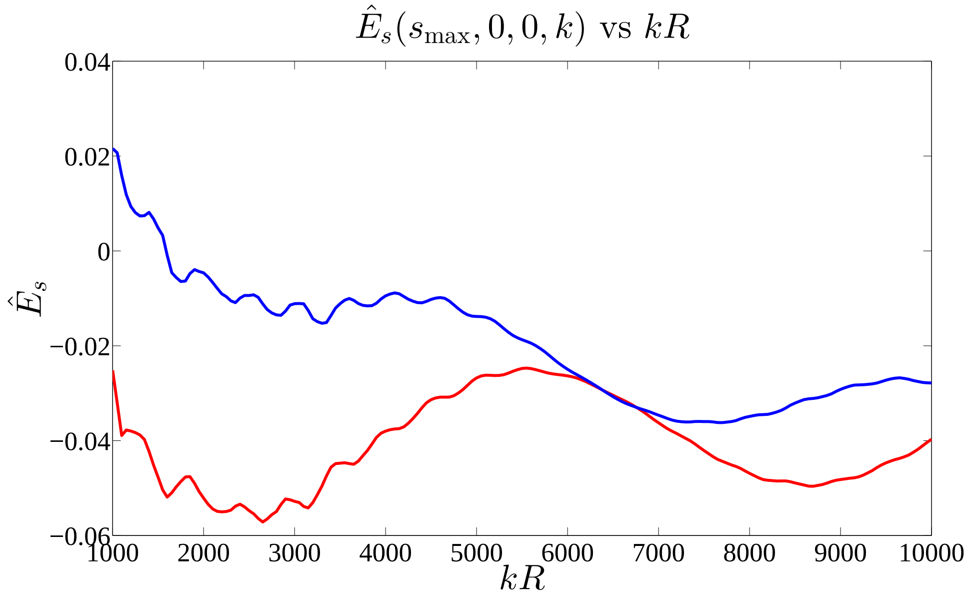


Figure 3.6: Real (red) and imaginary (blue) parts of $\hat{E}_s(s_{\text{max}}, 0, 0, k)$ for a source with $\sigma_y = 0.002\,\text{m}$ using $p_{\text{max}} = 9$ and $(N, K) = (8, 30)$.

Chapter 4

2D Frequency Domain Study – Proceedings of FEL2013

4.1 Statement of the Physical Problem

Derivations of the paraxial approximation can be found in section 2.2.4 as well as in the theses by T. Agoh [1] and D. Zhou [22]. The starting point here are Maxwell's equations with the source given by a line charge moving at near the speed of light, where for our study, taking $\beta = 1$ is an appropriate approximation. We consider the source orbit to be on circular arc of radius R and length L, and confine the domain by a perfectly conducting rectangular cross-section vacuum chamber as in [1, 2, 22, 23]. The electric field components E_x and E_y written in Frenet-Serret coordinates $\mathbf{r} = (s, x, y)$ for a bend of constant curvature $\kappa = 1/R$, are obtained from (2.60b)–(2.60c) and satisfy:

$$-\frac{1}{c^2}\frac{\partial^2 E_x}{\partial t^2}+\frac{1}{\eta^2}\frac{\partial^2 E_x}{\partial s^2}-\frac{\kappa^2}{\eta^2}E_x+\frac{\partial^2 E_x}{\partial x^2}+\frac{\kappa}{\eta}\frac{\partial E_x}{\partial x}+\frac{\partial^2 E_x}{\partial y^2}-\frac{2\kappa}{\eta^2}\frac{\partial E_s}{\partial s}=S_x(\mathbf{r},t), \tag{4.1a}$$

$$-\frac{1}{c^2}\frac{\partial^2 E_y}{\partial t^2}+\frac{1}{\eta^2}\frac{\partial^2 E_y}{\partial s^2}+\frac{\partial^2 E_y}{\partial x^2}+\frac{\kappa}{\eta}\frac{\partial E_y}{\partial x}+\frac{\partial^2 E_y}{\partial y^2}=S_y(\mathbf{r},t). \tag{4.1b}$$

where $S_x(\mathbf{r},t)$ and $S_y(\mathbf{r},t)$ are the source term components for the electric field wave equations. Specifically, the source terms follow from (2.61a) with $G(y)=\delta(y)$ for the line charge:

$$\begin{pmatrix} S_x(\mathbf{r},t) \\ S_y(\mathbf{r},t) \end{pmatrix}=\begin{pmatrix} qcZ_0\lambda(s-ct)\delta'(x)\delta(y) \\ qcZ_0\lambda(s-ct)\delta(x)\delta'(y) \end{pmatrix}. \tag{4.2}$$

Next, applying the Fourier transform in $s-ct$ from section 2.2.4 to (4.1) results in:

$$\begin{aligned} k^2\hat{E}_x+\frac{1}{\eta^2}\left(\frac{\partial^2\hat{E}_x}{\partial s^2}+2ik\frac{\partial\hat{E}_x}{\partial s}-k^2\hat{E}_x\right)-\frac{\kappa^2}{\eta^2}\hat{E}_x \\ +\frac{\partial^2\hat{E}_x}{\partial x^2}+\frac{\kappa}{\eta}\frac{\partial\hat{E}_x}{\partial x}+\frac{\partial^2\hat{E}_x}{\partial y^2}-\frac{2\kappa}{\eta^2}\left(\frac{\partial\hat{E}_s}{\partial s}+ik\hat{E}_s\right)=\hat{S}_x(x,y,k), \end{aligned} \tag{4.3a}$$

$$\begin{aligned} k^2\hat{E}_y+\frac{1}{\eta^2}\left(\frac{\partial^2\hat{E}_y}{\partial s^2}+2ik\frac{\partial\hat{E}_y}{\partial s}-k^2\hat{E}_y\right) \\ +\frac{\partial^2\hat{E}_y}{\partial x^2}+\frac{\kappa}{\eta}\frac{\partial\hat{E}_y}{\partial x}+\frac{\partial^2\hat{E}_y}{\partial y^2}=\hat{S}_y(x,y,k), \end{aligned} \tag{4.3b}$$

where the Fourier transform of the source, which is independent of s, is:

$$\begin{pmatrix} \hat{S}_x(x,y,k) \\ \hat{S}_y(x,y,k) \end{pmatrix}=\begin{pmatrix} qcZ_0\hat{\lambda}(k)\delta'(x)\delta(y) \\ qcZ_0\hat{\lambda}(k)\delta(x)\delta'(y) \end{pmatrix}. \tag{4.4}$$

To compare with results in [22, 23], we rearrange (4.3) to collect all the derivatives:

$$\begin{aligned}\frac{1}{\eta^2}\frac{\partial^2 \hat{E}_x}{\partial s^2} + \frac{2ik}{\eta^2}\frac{\partial \hat{E}_x}{\partial s} + \frac{\partial^2 \hat{E}_x}{\partial x^2} + \frac{\kappa}{\eta}\frac{\partial \hat{E}_x}{\partial x} + \frac{\partial^2 \hat{E}_x}{\partial y^2}& \\ + k^2\left(1 - \frac{1}{\eta^2}\right)\hat{E}_x - \frac{2\kappa}{\eta^2}\left(\frac{\partial \hat{E}_s}{\partial s} + ik\hat{E}_s\right) &= \hat{S}_x(x,y,k),\end{aligned} \tag{4.5a}$$

$$\begin{aligned}\frac{1}{\eta^2}\frac{\partial^2 \hat{E}_y}{\partial s^2} + \frac{2ik}{\eta^2}\frac{\partial \hat{E}_y}{\partial s} + \frac{\partial^2 \hat{E}_y}{\partial x^2} + \frac{\kappa}{\eta}\frac{\partial \hat{E}_y}{\partial x} + \frac{\partial^2 \hat{E}_y}{\partial y^2}& \\ + k^2\left(1 - \frac{1}{\eta^2}\right)\hat{E}_y &= \hat{S}_y(x,y,k),\end{aligned} \tag{4.5b}$$

Applying the paraxial approximation from section 2.2.4 we omit the $\partial^2/\partial s^2$ terms to arrive at:

$$\begin{aligned}\frac{2ik}{\eta^2}\frac{\partial \hat{E}_x}{\partial s} + \frac{\partial^2 \hat{E}_x}{\partial x^2} + \frac{\kappa}{\eta}\frac{\partial \hat{E}_x}{\partial x} + \frac{\partial^2 \hat{E}_x}{\partial y^2}& \\ + k^2\left(1 - \frac{1}{\eta^2}\right)\hat{E}_x - \frac{2\kappa}{\eta^2}\left(\frac{\partial \hat{E}_s}{\partial s} + ik\hat{E}_s\right) &= \hat{S}_x(x,y,k),\end{aligned} \tag{4.6a}$$

$$\begin{aligned}\frac{2ik}{\eta^2}\frac{\partial \hat{E}_y}{\partial s} + \frac{\partial^2 \hat{E}_y}{\partial x^2} + \frac{\kappa}{\eta}\frac{\partial \hat{E}_y}{\partial x} + \frac{\partial^2 \hat{E}_y}{\partial y^2}& \\ + k^2\left(1 - \frac{1}{\eta^2}\right)\hat{E}_y &= \hat{S}_y(x,y,k),\end{aligned} \tag{4.6b}$$

Now, we apply the low-curvature expansion $(1 - \eta^{-2}) \approx 2\kappa x$ where $\eta \approx 1$ and assume $\kappa \ll k$, as in [22, 23], to arrive at the following equations:

$$\frac{\partial \hat{E}_x}{\partial s} = \frac{i}{2k}\left(\frac{\partial^2 \hat{E}_x}{\partial x^2} + \frac{\partial^2 \hat{E}_x}{\partial y^2}\right) + ik\kappa x\hat{E}_x - \frac{cZ_0 i}{2k}\hat{S}_x(x,y,k), \tag{4.7a}$$

$$\frac{\partial \hat{E}_y}{\partial s} = \frac{i}{2k}\left(\frac{\partial^2 \hat{E}_y}{\partial x^2} + \frac{\partial^2 \hat{E}_y}{\partial y^2}\right) + ik\kappa x\hat{E}_y - \frac{cZ_0 i}{2k}\hat{S}_y(x,y,k), \tag{4.7b}$$

with the specific forms of the source terms for the electric field wave equation. We now combine the equations in (4.7) by defining $\hat{\mathbf{E}} := (\hat{E}_x, \hat{E}_y)^\top$ and $\hat{\mathbf{S}} := (\hat{S}_x, \hat{S}_y)^\top$. Due to the singularity in the sources of (4.7), we introduce the decomposition given in [2]: $\hat{\mathbf{E}} \equiv \hat{\mathbf{E}}^r + \hat{\mathbf{E}}^b$ so that the singular source terms in $\hat{\mathbf{S}}$ are absorbed by the $\hat{\mathbf{E}}^b$

term. This decomposition is not unique. For a line charge, one choice for $\hat{\mathbf{E}}^b$, which satisfies Poisson's equation $\boldsymbol{\nabla}_\perp^2 \hat{\mathbf{E}}^b = cZ_0\hat{\mathbf{S}}$, where $\boldsymbol{\nabla}_\perp^2 := (\partial^2/\partial x^2) + (\partial^2/\partial y^2)$, is:

$$E_x^b(x,y,k) = C(k)\frac{x}{x^2+y^2}, \qquad E_y^b(x,y,k) = C(k)\frac{y}{x^2+y^2}, \tag{4.8}$$

where $C(k) = qcZ_0\hat{\lambda}(k)/2\pi$. Now we rewrite (4.7) as:

$$\frac{\partial \hat{\mathbf{E}}^r}{\partial s} + \frac{\partial \hat{\mathbf{E}}^b}{\partial s} = \frac{i}{2k}\boldsymbol{\nabla}_\perp^2 \hat{\mathbf{E}}^r + \frac{i}{2k}\boldsymbol{\nabla}_\perp^2 \hat{\mathbf{E}}^b + ik\kappa x\hat{\mathbf{E}}^r + ik\kappa x\hat{\mathbf{E}}^b - \frac{cZ_0 i}{2k}\hat{\mathbf{S}}. \tag{4.9}$$

Our choice in $\hat{\mathbf{E}}^b$ allows the simplification to:

$$\frac{\partial \hat{\mathbf{E}}^r}{\partial s} = \frac{i}{2k}\boldsymbol{\nabla}_\perp^2 \hat{\mathbf{E}}^r + ik\kappa x\hat{\mathbf{E}}^r + ik\kappa x\hat{\mathbf{E}}^b, \tag{4.10}$$

For a rectangular vacuum chamber, we consider a cross-section of size $2a \times 2b$ with perfectly conducting boundary conditions as defined in equations (2.50)–(2.52). Specifically, the boundary conditions for this geometry are:

$$\hat{E}_x\Big|_{y=\pm b} \equiv \hat{E}_x^r + \hat{E}_x^b\Big|_{y=\pm b} = 0, \quad \frac{\partial \hat{E}_x}{\partial x}\Big|_{x=\pm a} \equiv \frac{\partial \hat{E}_x^r}{\partial x} + \frac{\partial \hat{E}_x^b}{\partial x}\Big|_{x=\pm a} = 0, \tag{4.11a}$$

$$\hat{E}_y\Big|_{x=\pm a} \equiv \hat{E}_y^r + \hat{E}_y^b\Big|_{x=\pm a} = 0, \quad \frac{\partial \hat{E}_y}{\partial y}\Big|_{y=\pm b} \equiv \frac{\partial \hat{E}_y^r}{\partial y} + \frac{\partial \hat{E}_y^b}{\partial y}\Big|_{y=\pm b} = 0. \tag{4.11b}$$

Lastly, the initial conditions are given by assuming the solution $\hat{\mathbf{E}}$ has achieved a steady state in an infinite straight preceding the bend. The derivation of the initial conditions follows in next the section.

4.2 Statement of the Mathematical Problem

Here we study the initial boundary value problem for the two Schrödinger-type equations given in (4.10). Our goal is to solve for the unknown radiation field $\hat{\mathbf{E}}^r$ which yields the electric field $\hat{\mathbf{E}}$ when combined with the known beam field $\hat{\mathbf{E}}^b$ of (4.8) as described in [2]. The decoupled initial boundary value problems for $\hat{\mathbf{E}}^r(x, y, k)$ are:

$$\frac{\partial \hat{E}_x^r}{\partial s} = \frac{i}{2k}\boldsymbol{\nabla}_\perp^2 \hat{E}_x^r + ik\kappa x\hat{E}_x^r + ik\kappa x\hat{E}_x^b(x, y, k), \tag{4.12a}$$

$$\frac{\partial \hat{E}_y^r}{\partial s} = \frac{i}{2k}\boldsymbol{\nabla}_\perp^2 \hat{E}_y^r + ik\kappa x\hat{E}_y^r + ik\kappa x\hat{E}_y^b(x, y, k), \tag{4.12b}$$

on the domain $0 \leq s \leq L$, $-a \leq x \leq a$, $-b \leq y \leq b$, with the boundary conditions in (4.11). These conditions for $\hat{E}_x^r$ and $\hat{E}_y^r$ are given in terms of $\hat{E}_x^b$ and $\hat{E}_y^b$:

$$\hat{E}_x^r = -\hat{E}_x^b \qquad \text{for} \qquad y = \pm b, \tag{4.13a}$$

$$\frac{\partial \hat{E}_x^r}{\partial x} = -\frac{\partial \hat{E}_x^b}{\partial x} \qquad \text{for} \qquad x = \pm a, \tag{4.13b}$$

$$\hat{E}_y^r = -\hat{E}_y^b \qquad \text{for} \qquad x = \pm a, \tag{4.13c}$$

$$\frac{\partial \hat{E}_y^r}{\partial y} = -\frac{\partial \hat{E}_y^b}{\partial y} \qquad \text{for} \qquad y = \pm b. \tag{4.13d}$$

By the same approach in Chapter 3, we build the initial conditions by assuming the fields have reached a steady state $\partial\hat{\mathbf{E}}/\partial s = \mathbf{0}$ in an infinite straight prior to the bend where $\kappa = 0$. Using (4.12)–(4.13), at $s = 0$ we have:

$$\boldsymbol{\nabla}_\perp^2 E_x^r = 0, \qquad \boldsymbol{\nabla}_\perp^2 E_y^r = 0. \tag{4.14}$$

The initial fields are given by the solutions to Laplace's equation in (4.14) with the boundary conditions in (4.13). The solution to these can be found analytically using Fourier series; however, we use a Poisson solver to construct the solution numerically. We note that the initial boundary value problems for $\hat{E}_x^r$ and $\hat{E}_y^r$ are uncoupled.

The field $\hat{E}_s^r$ is needed on $0 \leq s \leq L$ in order to compare with the impedance calculation in [22]. The field $\hat{E}_s^r$ is derived from Maxwell's equation (2.41g) and taking $\eta \to 1$ in the low curvature limit, that is:

$$\frac{\partial E_s}{\partial s} + \frac{\partial E_x}{\partial x} + \frac{\partial E_y}{\partial y} = qcZ_0\lambda(s - ct)\delta(x)\delta(y). \tag{4.15}$$

Next, applying the Fourier transform in $s - ct$ to the above yields:

$$ik\hat{E}_s = -\frac{\partial \hat{E}_x}{\partial x} - \frac{\partial \hat{E}_y}{\partial y} + qcZ_0\hat{\lambda}(k)\delta(x)\delta(y). \tag{4.16}$$

Lastly, by splitting the field into $\hat{E}_s \equiv \hat{E}_s^r + \hat{E}_s^b$, the beam field contribution $\hat{E}_s^b$, generated from $\boldsymbol{\nabla}_\perp^2 \hat{\mathbf{E}}^b = cZ_0\hat{\mathbf{S}}$, reduces to zero since:

$$\frac{\partial \hat{E}_x^b}{\partial x} + \frac{\partial \hat{E}_y^b}{\partial y} = qcZ_0\hat{\lambda}(k)\delta(x)\delta(y). \tag{4.17}$$

Therefore, only $\hat{E}_s^r \neq 0$ and we obtain:

$$\hat{E}_s^r = \frac{i}{k}\left(\frac{\partial \hat{E}_x^r}{\partial x} + \frac{\partial \hat{E}_y^r}{\partial y}\right). \tag{4.18}$$

The impedance at $(x, y) = (0, 0)$ in our notation is given by:

$$Z(k) = -\frac{1}{q\hat{\lambda}(k)}\int_0^\infty \hat{E}_s^r(s, 0, 0, k)ds. \tag{4.19}$$

The calculation of $E_s(s, 0, 0, k)$ for $s \geq L$ is discussed in section 4.4.4.

4.3 Overview of Our DG Approach

To make the discussion of the DG approach simpler, we introduce dimensionless variables through the rescaling:

$$s \to 2ka^2\tilde{s}, \quad x \to a\tilde{x}, \quad y \to a\tilde{y}, \quad \hat{E}_x^r \to C(k)U/a, \quad \hat{E}_y^r \to C(k)V/a$$

but in this subsection we will write (s, x, y) for the dimensionless variables $(\tilde{s}, \tilde{x}, \tilde{y})$. In terms of these rescaled variables, (4.12a) becomes:

$$-i\frac{\partial U}{\partial s} = \boldsymbol{\nabla}_\perp^2 U + F_1(U, x, y, k), \tag{4.20}$$

where $F_1(U,x,y,k) = 2k^2\kappa a^3 x(U + x/(x^2+y^2))$. The boundary conditions (4.13a)–(4.13b) become:

$$\frac{\partial U}{\partial x}\Big|_{x=\pm 1} = \frac{1-y^2}{(1+y^2)^2}, \qquad U\Big|_{y=\pm b/a} = -\frac{x}{x^2+(b/a)^2}. \tag{4.21}$$

Similarly, with the rescaling of E_y^r, (4.12b) becomes:

$$-i\frac{\partial V}{\partial s} = \boldsymbol{\nabla}_\perp^2 V + F_2(V,x,y,k), \tag{4.22}$$

where $F_2(V,x,y,k) = 2k^2\kappa a^3 x(V + y/(x^2+y^2))$. The boundary conditions (4.13c)–(4.13d) similarly become:

$$V\Big|_{x=\pm 1} = -\frac{y}{1+y^2}, \qquad \frac{\partial V}{\partial y}\Big|_{y=\pm b/a} = \frac{(b/a)^2-x^2}{(x^2+(b/a)^2)^2}. \tag{4.23}$$

The initial conditions are also rescaled by $C(k)/a$, and we continue to write (s,x,y) in place of $(\tilde{s},\tilde{x},\tilde{y})$. We note that U and V only depend parametrically on $2k^2a^3\kappa$ and b/a, and that the integration domain $0 \le s \le L$ becomes $0 \le s \le L/2ka^2$. The factor $C(k)$ only enters into the magnitude of the fields $\hat{E}_x^r$ and $\hat{E}_y^r$.

Both (4.20) and (4.22) can be written in the form,

$$-i\frac{\partial u}{\partial s} = \frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial y} + F(u,x,y), \tag{4.24a}$$

$$q_x = \frac{\partial u}{\partial x}, \tag{4.24b}$$

$$q_y = \frac{\partial u}{\partial y}. \tag{4.24c}$$

Now, we partition $\Omega \equiv [-1,1]\times[-b/a,b/a]$ into K triangular elements, focus on a single element $D^k \subset \Omega$, and assume that on D^k the local solution $u^k \in \mathcal{P}^N(D^k)$ is polynomial of degree N. Multiplication of each equation in (4.24) by a test function:

$v \in \mathcal{P}^N(D^k)$ and the integration over D^k yields

$$-i\int_{D^k} v\frac{\partial u^k}{\partial s}dA = \int_{D^k}\left[v\frac{\partial q_x^k}{\partial x} + v\frac{\partial q_y^k}{\partial y} + vF^k\right]dA \tag{4.25a}$$

$$\int_{D^k} vq_x^k dA = \int_{D^k} v\frac{\partial u^k}{\partial x}dA, \tag{4.25b}$$

$$\int_{D^k} vq_y^k dA = \int_{D^k} v\frac{\partial u^k}{\partial y}dA. \tag{4.25c}$$

Equations (4.25) are exact, but involve only the local polynomials u^k, q_x^k, q_y^k, and v on D^k. To couple adjacent elements, we now adjust the above equations by introducing flux terms along the boundary ∂D^k which couple D^k to neighboring elements. In the strong form, the so-called residual equations (4.25) become:

$$\begin{aligned}-i\int_{D^k} v\frac{\partial u^k}{\partial s}dA = &\int_{D^k}\left[v\frac{\partial q_x^k}{\partial x} + v\frac{\partial q_y^k}{\partial y} + vF^k(u,x,y)\right]dA \\ &- \int_{\partial D^k} v\left[n_x(q_x^k - q_x^*) + n_y(q_y^k - q_y^*)\right]dL,\end{aligned} \tag{4.26a}$$

$$\int_{D^k} vq_x^k dA = \int_{D^k} v\frac{\partial u^k}{\partial x}dA - \int_{\partial D^k} n_x v(u^k - u^*)dL, \tag{4.26b}$$

$$\int_{D^k} vq_y^k dA = \int_{D^k} v\frac{\partial u^k}{\partial y}dA - \int_{\partial D^k} n_y v(u^k - u^*)dL, \tag{4.26c}$$

In these formulas dL and (n_x, n_y) respectively specify the arc-length measure and outward normal vector for ∂D^k. The terms u^*, q_x^*, q_y^* are numerical fluxes which depend not only on the local solutions u^k, q_x^k, q_y^k on D^k but also on the solutions belonging to adjacent elements. We specify our choice of fluxes in (4.31). One way to "derive" (4.26) from (4.25) is to first integrate by parts, shifting all derivatives onto each test polynomial v. This process generates integrals along the boundary ∂D^k. Next, in these boundary integrals one makes the intermediate replacements $u^k, q_x^k, q_y^k \to u^*, q_x^*, q_y^*$, and then invokes a second round of integration by parts to arrive at (4.26). Of course, without the intermediate replacements, this process would arrive back at (4.25), as is easily seen by taking $(u^*, q_x^*, q_y^*) = (u^k, q_x^k, q_y^k)$ in (4.26).

To obtain matrix formulas from (4.26), we first express the local solution on element D^k and each test polynomial as follows:

$$\begin{aligned} u^k(x,y) &= \sum_{j=1}^{N_p} u_j^k \ell_j^k(x,y), & v(x,y) &= \ell_i^k(x,y), \\ q_x^k(x,y) &= \sum_{j=1}^{N_p} q_{x,j}^k \ell_j^k(x,y), & q_y^k(x,y) &= \sum_{j=1}^{N_p} q_{y,j}^k \ell_j^k(x,y). \end{aligned} \tag{4.27}$$

Here, N_p denotes the number of nodes on the element D^k, which is related to the polynomial order N through $N_p = (N+1)(N+2)/2$. Moreover, $\ell_i^k(x,y)$ is the Lagrange basis polynomial which is 1 at the ith node $(x_i^k, y_i^k) \in D^k$, but zero at all other nodes (x_j^k, y_j^k), $j \neq i$. Since $\ell_i \in \mathcal{P}^N(D^k)$, by taking $i \in \{1, \dots, N_p\}$ as arbitrary, we sample the whole test space since every polynomial of degree N is uniquely decomposed by the N_p basis polynomials. We note that the index variable i here is distinct from the imaginary unit $\sqrt{-1}$ coefficient in the PDEs (4.20) and (4.22). With expansion coefficients as in (4.27), we define the corresponding vectors:

$$\begin{aligned} \mathbf{u}^k &= (u_1^k, u_2^k, \dots, u_{N_p}^k)^\top, \\ \mathbf{q}_x^k &= (q_{x,1}^k, q_{x,2}^k, \dots, q_{x,N_p}^k)^\top, \\ \mathbf{q}_y^k &= (q_{y,1}^k, q_{y,2}^k, \dots, q_{y,N_p}^k)^\top, \\ \boldsymbol{\ell}^k &= (\ell_1^k, \ell_2^k, ..., \ell_{N_p}^k)^\top. \end{aligned} \tag{4.28}$$

Substitution of (4.27) and the vectors into (4.26) yields:

$$\begin{aligned} -i\frac{d\mathbf{u}^k}{ds} =& (\mathcal{M}^k)^{-1}\mathcal{S}_x^k \mathbf{q}_x^k + (\mathcal{M}^k)^{-1}\mathcal{S}_y^k \mathbf{q}_y^k + \mathbf{F}^k \\ & - (\mathcal{M}^k)^{-1} \int_{\partial D^k} \left[n_x(q_x^k - q_x^*) + n_y(q_y^k - q_y^*) \right] \boldsymbol{\ell}^k dL \end{aligned} \tag{4.29a}$$

$$\mathbf{q}_x^k = (\mathcal{M}^k)^{-1}\mathcal{S}_x^k \mathbf{u}^k - (\mathcal{M}^k)^{-1} \int_{\partial D^k} n_x(u^k - u^*) \boldsymbol{\ell}^k dL \tag{4.29b}$$

$$\mathbf{q}_y^k = (\mathcal{M}^k)^{-1}\mathcal{S}_y^k \mathbf{u}^k - (\mathcal{M}^k)^{-1} \int_{\partial D^k} n_y(u^k - u^*) \boldsymbol{\ell}^k dL, \tag{4.29c}$$

where $\mathbf{F}^k := F(\mathbf{u}^k, \mathbf{x}^k, \mathbf{y}^k)$ is evaluated at the nodal coordinates. In these expressions the mass $\mathcal{M}^k$ and stiffness $\mathcal{S}_x^k$, $\mathcal{S}_y^k$ matrices are defined as:

$$\begin{aligned} [\mathcal{M}^k]_{ij} &= \int_{D^k} \ell_i^k(x,y)\ell_j^k(x,y)dA, \\ [\mathcal{S}_x^k]_{ij} &= \int_{D^k} \frac{\partial \ell_i^k}{\partial x}(x,y)\ell_j^k(x,y)dA, \\ [\mathcal{S}_y^k]_{ij} &= \int_{D^k} \frac{\partial \ell_i^k}{\partial y}(x,y)\ell_j^k(x,y)dA, \end{aligned} \tag{4.30}$$

To reach the given form (4.29) of the local semi-discrete equations, we invert the local mass matrix $\mathcal{M}^k$, and put these equations into their final form by defining u^*, q_x^*, q_y^* as follows:

$$u^* = \{\{u\}\}, \qquad q_x^* = \{\{q_x\}\} - \tau[\![u]\!]_x, \qquad q_y^* = \{\{q_y\}\} - \tau[\![u]\!]_y. \tag{4.31}$$

The $\{\{\cdot\}\}$ and $[\![\cdot]\!]$ operations respectively denote the average value and jump in a value across a boundary. For example, if two elements D^{k_1} and D^{k_2} share a common boundary segment $\partial D^{k_{1,2}}$, then along the segment $q_x^* = \frac{1}{2}(q_x^+ + q_x^-) - \tau(n_x^- u^- + n_x^+ u^+)$. In the case $\tau = 0$ we recover the central flux scheme which is stable; however, for better conditioned solution, we set the penalty parameter τ as described on p263 of [11]. Every element follows the same construction yielding a total $N_p \times K$ nodes for Ω, where K is the total number of elements.

4.4 Numerical Implementation

This section presents the steps used to obtain the electric fields and longitudinal impedances for the curved and straight portions of the vacuum chamber. This process is repeated for every wave number k we consider. We now return to the notation in the section on the statement of the problem, since our actual code employs the physical variables.

4.4.1 Construction of the Elements and Matrices

The first step begins by dividing the rectangular cross-section Ω of the vacuum chamber into rectangles and then subdividing each rectangle diagonally into two triangles. The total number of elements $K = 2N_x^{\text{res}}N_y^{\text{res}}$, with N_x^{res} and N_y^{res} denoting the resolution of elements in the x and y directions. Figure 4.1 shows an example configuration of nodes and elements with $N_x^{\text{res}} = 6$, $N_y^{\text{res}} = 2$, of polynomial order $N = 4$. We employ Lagrange-Gauss-Lobatto nodal spacing for better conditioning of the mass and stiffness matrices. More discussion on this choice of spacing is found on p47 of [11]. Next, the mass and stiffness matrices are computed via (4.30). These

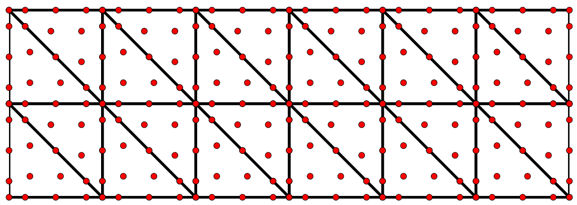


Figure 4.1: Example of a 24 element domain using 4th order elements (15 nodes per element). The nodes (red dots) are not equally spaced among each element.

matrices are then used to obtain the derivative matrix products $\mathcal{D}_x = \mathcal{M}^{-1}\mathcal{S}_x$ and $\mathcal{D}_y = \mathcal{M}^{-1}\mathcal{S}_y$, appearing in (4.29).

4.4.2 Construction of the Initial Data

The next step is to construct the initial conditions at $s = 0$ using (4.14) subject to the boundary conditions given by (4.13). The transverse fields $\hat{E}_x^r$ and $\hat{E}_y^r$ are constructed numerically with the DG Poisson solver described on p275-280 of [11]. Next, $\hat{E}_s^r$ is computed with the derivative matrices from step 1:

$$\hat{E}_s^r = \frac{i}{k}(\mathcal{D}_x\hat{E}_x^r + \mathcal{D}_y\hat{E}_y^r). \tag{4.32}$$

4.4.3 Evolution of the Fields

The third, and most computationally intensive, step is to evolve the PDEs in (4.12) for the transverse fields $\hat{E}_x^r$ and $\hat{E}_y^r$. These are evolved using the classical 4th order explicit Runge-Kutta scheme. The time step for the evolution is first determined by

$$\Delta s = C_{\text{CFL}} \cdot k \cdot r_{\min}^2, \tag{4.33}$$

where $r_{\min}$ is minimal distance between the nodes. Note that $r_{\min}$ decreases quadratically with the order N; therefore, for large N this time step restriction is severe. We have taken the CFL constant C_{CFL} as 0.25 for our computations. This constant was determined experimentally.

The evaluations of $d\mathbf{u}/ds$ required for timestepping are implemented as follows: $\mathbf{q}_x$ and $\mathbf{q}_y$ are obtained from (4.29b)–(4.29c), with the results then substituted into (4.29a). This process is done for both $\hat{E}_x^r$ and $\hat{E}_y^r$. Finally, $\hat{E}_s^r$ is computed from $\hat{E}_x^r$ and $\hat{E}_y^r$ at each timestep with (4.32).

4.4.4 Computation of the Impedance

We separate the impedance integral in (4.19) into two parts, $Z = Z_b + Z_s$, where:

$$Z_b(k) = -\frac{1}{q\hat{\lambda}(k)} \int_0^L \hat{E}_s^r(s,0,0,k)ds, \tag{4.34a}$$

$$Z_s(k) = -\frac{1}{q\hat{\lambda}(k)} \int_L^\infty \hat{E}_s^r(s,0,0,k)ds. \tag{4.34b}$$

Throughout the evolution process in step 3, we record $\hat{E}_s^r$ at the origin: $\hat{E}_{s0}^r(s,k) := \hat{E}_s^r(s,0,0,k)$. Z_b is then approximated by the trapezoidal rule:

$$Z_b(k) \approx -\frac{\Delta s}{q\hat{\lambda}(k)} \left[\frac{\hat{E}_{s0}^r(0,k) + \hat{E}_{s0}^r(L,k)}{2} + \sum_{n=1}^{N_{\text{steps}}-1} \hat{E}_{s0}^r(n\Delta s, k) \right]. \tag{4.35}$$

Following [22], we evaluate the remaining integral Z_s as a mode expansion:

$$Z_s(k) \approx \frac{1}{q\hat{\lambda}(k)} \sum_{m=1}^{M} \sum_{p=1}^{P} D_{mp}(k) \sin\left(\frac{m\pi}{2}\right) \sin\left(\frac{p\pi}{2}\right), \tag{4.36}$$

where the coefficients $D_{mp}(k)$ are determined by the transverse fields $\hat{E}_x^r$ and $\hat{E}_y^r$ at the end of the bend, $s = L$, through the following expressions:

$$D_{mp}(k) = \frac{8(A_{mp}(k)k_x + B_{mp}(k)k_y)}{k_x^2 + k_y^2}, \tag{4.37a}$$

$$A_{mp}(k) = \frac{1}{ab} \iint\limits_{\Omega} \hat{E}_x^r(L, x, y, k) \cos(k_x x) \sin(k_y y)\, dA, \tag{4.37b}$$

$$B_{mp}(k) = \frac{1}{ab} \iint\limits_{\Omega} \hat{E}_y^r(L, x, y, k) \sin(k_x x) \cos(k_y y)\, dA, \tag{4.37c}$$

$$k_x = \frac{m\pi}{2a}, \qquad k_y = \frac{p\pi}{2b}. \tag{4.37d}$$

The values of M and P should ideally be infinite; however, in our experiments, $M \gtrsim (N+1)N_x^{\text{res}}$ and $P \gtrsim (N+1)N_y^{\text{res}}$ are sufficient for a good approximation.

4.5 Numerical Results

In this section, we consider a narrow source length where we take $\hat{\lambda}(k)$ to be constant for the k values of interest. We further note that the initial conditions and solutions scale in direct proportion with $\hat{\lambda}(k)$.

4.5.1 DG Results

Although comprehensive strategies [14, 15] exist for optimized DG simulations on GPUs, we have adopted a simple approach based on MATLAB's `gpuArray`. Our simulations have been performed on an Nvidia GeForce GTX Titan with the following parameters: $a = 60$ mm, $b = 20$ mm, $L = 200$ mm, $R = 1$ m, and $k = 8$ mm^{-1}. We have used K elements with $N_p = (N+1)(N+2)/2$ nodes per element, and the internal penalty flux parameter τ mentioned earlier. We consider results for both (i) impedance calculations and (ii) performance and accuracy.

The initial condition is shown in Figure 4.2 and is dependent on k only through $\hat{\lambda}(k)$. Figure 4.3 shows an example of the real and imaginary parts of $\hat{E}_x^r$, for $k = 8$ mm^{-1}, at $s = 200$ mm, i.e. at the end of the bend by integrating (4.12) with $(N, K) = (8, 2400)$.

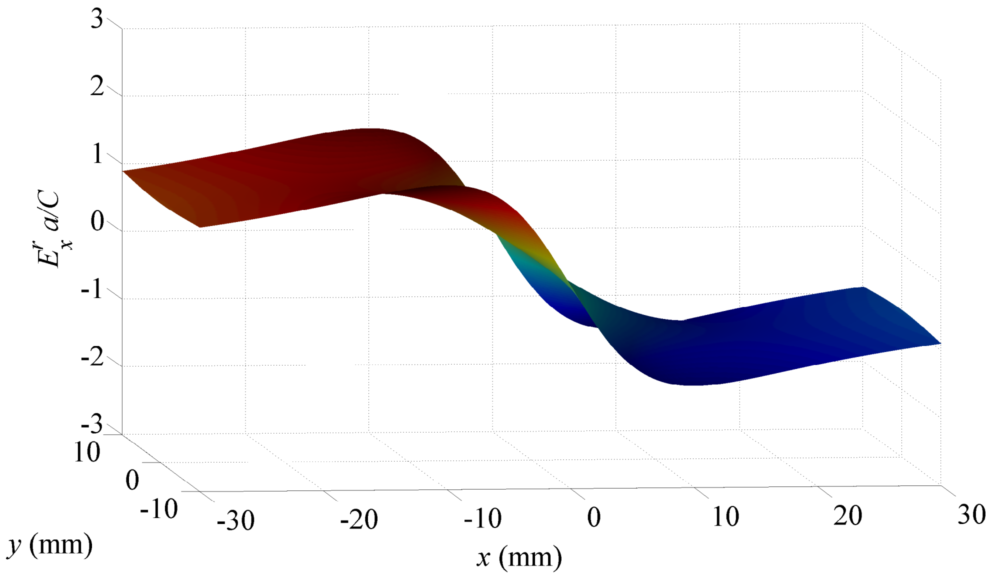


Figure 4.2: Initial condition for $\hat{E}_x^r$ with $C(k) = qcZ_0\hat{\lambda}(k)/2\pi$. This initial condition is the solution to (4.14) with the boundary conditions (4.13).

We have performed a high resolution simulation with $(N, K) = (8, 2400)$, and taken the resulting numerical solution as the "exact" solution. This simulation took about one hour. Errors for lower-resolution solutions have been computed against this reference solution, with the results for $\hat{E}_x^r$ at $s = 200$ mm and $k = 8$ mm^{-1} shown in Table 4.1.

Table 4.1: DG E_x^r Error and Computation Time

N\K	**150**	**600**	**1350**	**2400**
2	8.107e-1	3.915e-1	7.471e-2	2.453e-2
	8.032e-1	2.528e-1	4.291e-2	1.484e-2
	9s	28s	49s	81s
	122	486	1093	1943
4	1.265e-1	4.897e-3	9.344e-4	3.590e-4
	8.017e-2	3.427e-3	9.462e-4	4.342e-4
	29s	88s	202s	392s
	539	2156	4850	8622
6	1.122e-2	5.177e-4	1.407e-4	5.483e-5
	6.974e-3	6.187e-4	1.932e-4	8.045e-5
	83s	283s	677s	1319s
	1691	6764	15218	27054
8	1.569e-3	1.672e-4	5.612e-5	N\A*
	1.493e-3	2.098e-4	7.473e-5	N\A*
	208s	723s	1867s	3630s
	4174	16693	37559	66771

*:Used for comparison to other tests.

The table shows relative L^∞ (top) and relative L^2 (upper middle) errors corresponding to $\hat{E}_x^r$ evaluated on a 31×11 grid. This evaluation grid was the largest set of nodes common to all DG grids. Computation times (lower middle) and time-step counts (bottom) are also listed.

The large errors for $(N, K) = (2, 150)$, $(2, 600)$, and $(4, 150)$ are due to the fields not being sufficiently resolved. The errors decrease and the time-step counts increase with increasing K and N. For stability, we note the stepsize from (4.33) follows $\Delta s \propto 1/(KN^2)$ since the nodes are not equally spaced throughout each triangular element. The minimal nodal spacing within an element scales with the polynomial order as N^{-2}.

Our MATLAB DG code was first written for a CPU; the GPU version required little additional work. Our GPU calculations become more efficient for larger matrix systems, and when less communication between the CPU and GPU is required. For the lower-order tests, the GPU functioned at around 20% of its maximum capacity. However, for the higher-order tests, the GPU efficiency increased to over 60%. In our GPU simulations we have observed speed-ups of up to ~ 10 over our CPU simulations.

4.5.2 FD Results and Impedance Comparison

We have also written a MATLAB finite difference (FD) code modeled after the FD method discussed in [1, 2, 22, 23]. This FD method uses leap-frog as the time-stepper. Our DG code allows for any spatial order N, whereas the FD code employs a second-order stencil, commonly known as Yee's method.

The FD grids were of size $(N_x^{\text{res}} + 1) \times (N_y^{\text{res}} + 1)$. As a test of the two codes, we calculated the impedance using (4.19). For both the FD and DG approaches, Figure 4.4 depicts the resulting real and imaginary parts.

To the eye, these impedances are in agreement with Figure 3 in [23]. The discrepancy between our DG and FD results stems from how the spatial derivatives in (4.18) are computed. The FD method uses lower order difference stencils instead of the derivative matrices $\mathcal{D}_x$ and $\mathcal{D}_y$.

We have run the FD code for $(N_x^{\text{res}}, N_y^{\text{res}}) = (60, 20)$, $(120, 40)$, $(180, 60)$, and $(240, 80)$, and the results are shown in Table 4.2. The errors were calculated with respect to the high resolution DG calculation, using a 61×21 grid which is common to all the cases. The organization of the table is as in Table 4.1, with listings for the relative L^∞ and L^2 error, run time, and time-step count. Clearly, our DG GPU code outperforms our FD CPU code. However, this is in part due to the GPU implementation of the DG code. The FD code could not be easily extended to GPU computing since sparse matrices are not natively supported in MATLAB's CUDA library. However, using the CPU only, our DG code still outperforms our FD code.

Table 4.2: FD E_x^r Error

Grid	**61 × 21**	**121 × 41**	**181 × 61**	**241 × 81**
	3.845e-1	1.126e-1	4.016e-2	3.440e-2
	4.539e-1	1.223e-1	4.551e-2	2.840e-2
	8s	125s	642s	2032s
	200	800	1800	3200

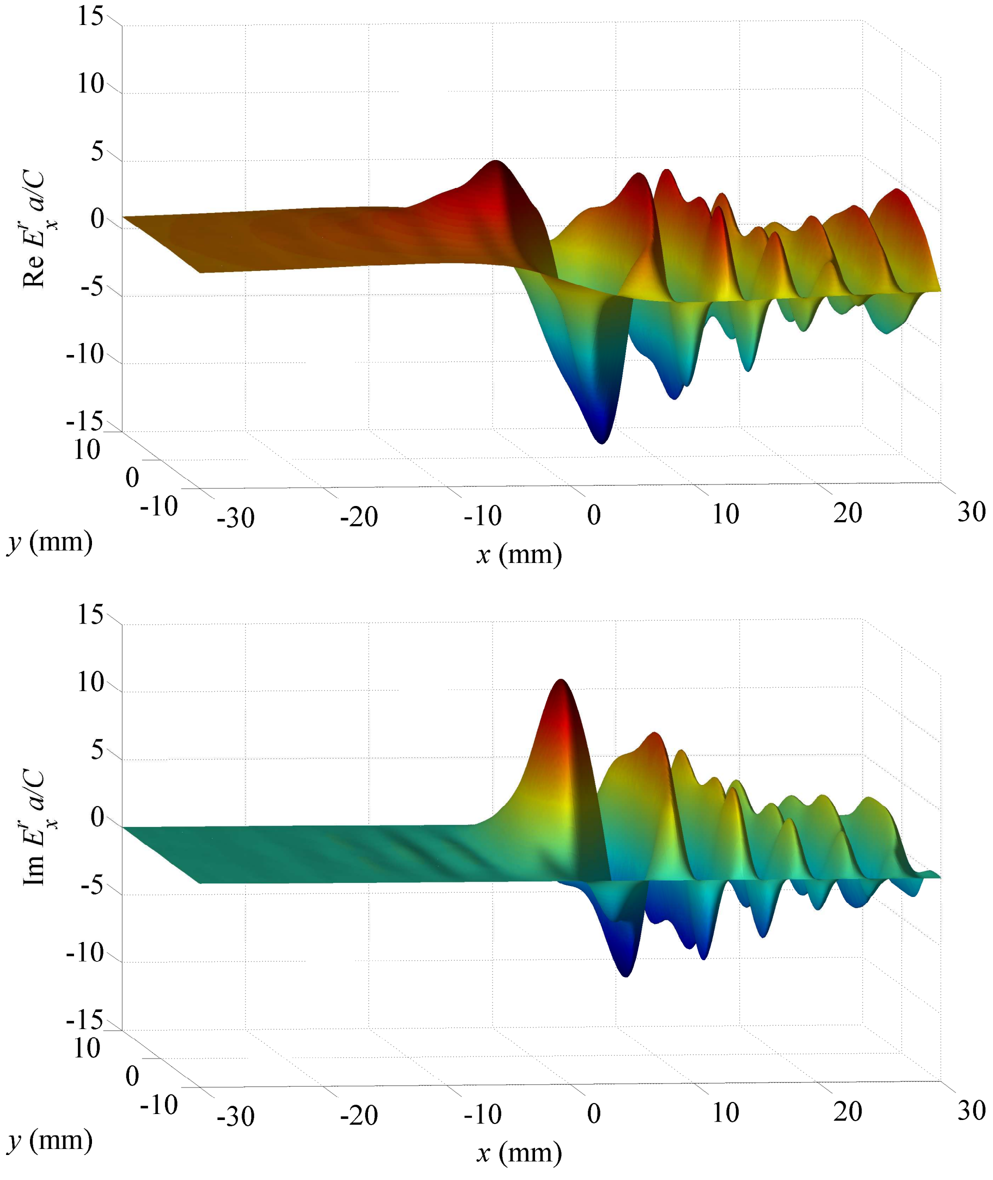


Figure 4.3: Real (top) and imaginary (bottom) parts of $\hat{E}_x^r$ for $R = 1$ m, $L = 200$ mm, $a = 30$ mm, $b = 10$ mm,$k = 8$ mm^{-1}.

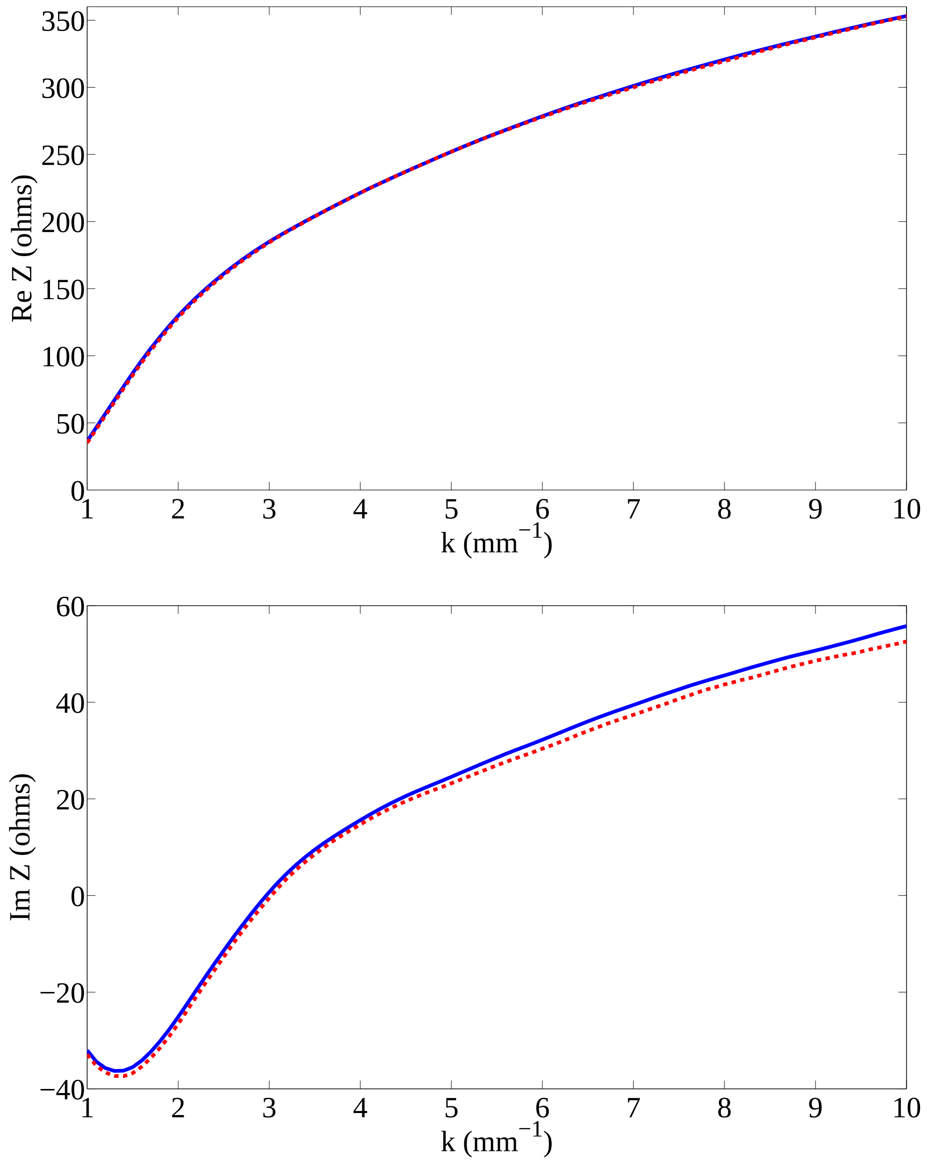


Figure 4.4: Real (top) and imaginary (bottom) parts of the impedance for $\rho = 1$ m, $L = 200$ mm, $a = 30$ mm, $b = 10$ mm. DG (blue solid), FD (red dashed)

Chapter 5

2D Time Domain Study – Work for the Canadian Light Source

5.1 Statement of the Problem Domain

In this chapter, we solve Maxwell's equations in a vacuum chamber in a bend at the CLS using the Frenet-Serret coordinates and a Fourier series in y as seen in sections 2.2.1 and 2.2.3. We adopt the trajectory of the electron bunch as the reference orbit of the Frenet-Serret transformation. The reference orbit is a fixed distance from the inner wall of the bend as shown in Figure 5.1.

For solutions in the time-domain, we adopt the scaled time coordinate $\tau := ct$. Next, our domain of interest is a vacuum chamber of fixed height h with perfectly conducting walls at $y = \pm h/2$. Along the transverse direction x, the boundary is allowed to vary in shape with s, and also has perfectly conducting walls. Furthermore, we assume the inner wall does not vary in s, thus we denote the x-boundaries by $x = x_{\text{in}}$ and $x = x_{\text{out}}(s)$. We close the computational domain by bounding the s-direction at $s = s_{\min}$ and $s = s_{\max}$ with an open boundary condition; this will be discussed in detail in section 5.2.3. We mark the transition from a straight section to the beginning of the bend by $s = s_0$.

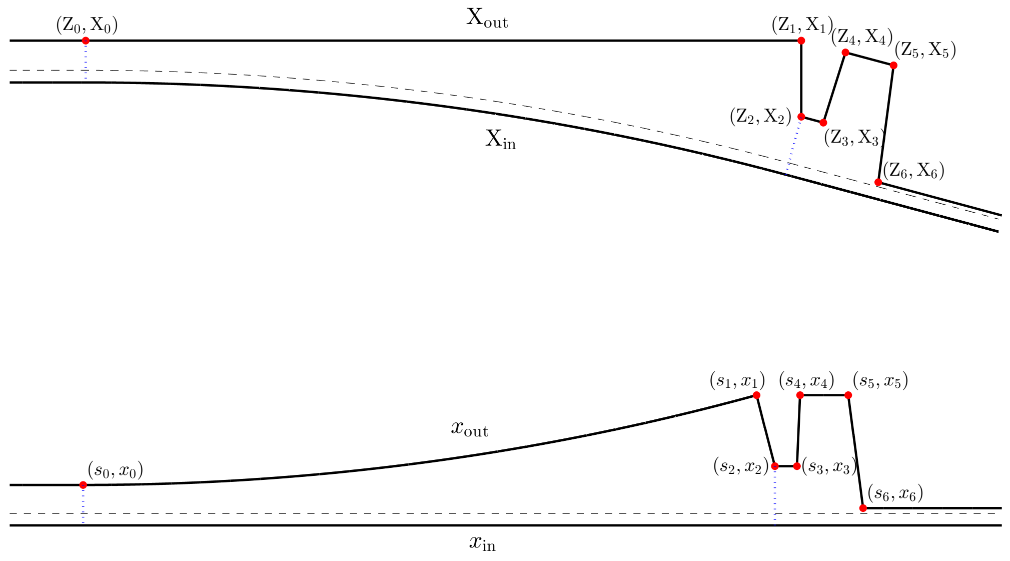


Figure 5.1: Physical laboratory frame (top) and Frenet-Serret transformed frame (bottom). The dashed line indicates the source's trajectory, the region between the dotted blue lines indicate where the curvature is non-zero, and the red dots indicate points where the boundary geometry transitions.

The cross-sectional dimensions and bending radius of the CLS chamber are:

$$h = 0.032\text{m}, \quad x_{\text{in}} = -0.032\text{m}, \quad x_{\text{out}}(s_0) = 0.078\text{m}, \quad R = 7.143\text{m} \tag{5.1}$$

The curvature of the reference orbit $\kappa(s)$ is piecewise constant:

$$\kappa(s) = \begin{cases} 1/R & : \quad s_0 \le s \le s_2 \\ 0 & : \quad \textit{otherwise} \end{cases} \tag{5.2}$$

The outer wall profile $x_{\text{out}}(s)$ in Frenet-Serret coordinates is piecewise continuous:

$$x_{\text{out}}(s) = \begin{cases} x_0 & : \quad s \le s_0 \\ (R + x_0)\sec(s/R) - R & : \quad s_0 < s \le s_1 \\ (R + x_0)\tan(s_1/R)\csc(s/R) - R & : \quad s_1 < s \le s_2 \\ x_2 & : \quad s_2 < s \le s_3 \\ x_2 + (s - s_3)\cot(2.5^\circ) & : \quad s_3 < s \le s_4 \\ x_4 & : \quad s_4 < s \le s_5 \\ x_5 - (s - s_5)\cot(7.5^\circ) & : \quad s_5 < s \le s_6 \\ x_6 & : \quad s > s_6 \end{cases} \tag{5.3}$$

The points along the outer wall boundary (s_i, x_i) denote transition points between the different features of the boundary. The chamber begins as a semi-infinite straight of fixed width $w = x_{\text{out}}(s_0) - x_{\text{in}}$ for $s < s_0$. The inner wall in this region follows along this curved trajectory in Cartesian coordinates maintaining a fixed distance from the reference orbit to the wall x_{in}. The outer wall, a straight in the laboratory space, is curved in the Frenet-Serret coordinates and flares outward to a maximum of $x_{\text{out}}(s_1)$. At s_1, the location of a photon absorber in the chamber, the wall forms a right angle and bends inwards towards the beam.

At $s = s_2$, the end of the photon absorber, the bunch again travels along a straight path and the outer wall remains at a distance x_2 parallel to the source trajectory. As $s = s_3$, the wall moves outward at 2.5° from the normal to form a cavity behind the photon absorber. The corners at $s = s_4, s_5$ form the bounds of the cavity which then angles towards the beam at 7.5° from the normal. The bottom portion of this wall is the M1 mirror. Lastly, at $s = s_6$, the end of the M1 mirror, the outer wall is assumed to continue onward and parallel to the source trajectory at distance x_6 from the source. The corners and transition points along the boundary profile $x_{\text{out}}(s)$ are approximately located at:

$$
\begin{aligned}
s_0 &= 0.0000\text{m}, & x_0 &:= x_{\text{out}}(s_0) = 0.07800\text{m},\\
s_1 &= 1.8326\text{m}, & x_1 &:= x_{\text{out}}(s_1) = 0.32236\text{m},\\
s_2 &= 1.8825\text{m}, & x_2 &:= x_{\text{out}}(s_2) = 0.12908\text{m},\\
s_3 &= 1.9425\text{m}, & x_3 &:= x_{\text{out}}(s_3) = 0.12908\text{m},\\
s_4 &= 1.9509\text{m}, & x_4 &:= x_{\text{out}}(s_4) = 0.32236\text{m},\\
s_5 &= 2.0820\text{m}, & x_5 &:= x_{\text{out}}(s_5) = 0.32236\text{m},\\
s_6 &= 2.1225\text{m}, & x_6 &:= x_{\text{out}}(s_6) = 0.01500\text{m}.
\end{aligned}
\tag{5.4}
$$

These locations are the transformed coordinates of (Z_i, X_i), the Cartesian coordinate laboratory frame locations where the boundary geometry transitions.

Lastly, to model the electron bunch at the CLS we consider the ribbon charge and current densities (2.46)–(2.47). However, we offset the argument of λ to $(s - ct - s_{\text{off}})$ where $s_{\text{off}} < s_0$ is an offset distance so that the bunch starts at $s = s_{\text{off}}$ at $t = 0$. In Figure 5.1, the starting position of the source at s_{off} is located in the initial straight segment to the left of s_0. The value of s_{off} is chosen to ensure the source is supported in the region $s < s_0$.

5.2 Problem Setup and Formulation

5.2.1 Maxwell's Equations and Transforms

Using the geometry and source for the time-domain problem defined in the previous section, we pose Maxwell's equations transformed into Frenet-Serret coordinates from section 2.3.1. Maxwell's evolution equations for $(\mathbf{E}, \mathbf{H})$, where $\mathbf{H} := c\mathbf{B}/Z_0$, with the source prescribed by (2.46)–(2.47) in the scaled time coordinate $\tau = ct$ are:

$$\frac{\partial \mathbf{E}}{\partial \tau} = Z_0 \boldsymbol{\nabla} \times \mathbf{H} - Z_0 \mathbf{j}, \tag{5.5}$$

$$\frac{\partial \mathbf{H}}{\partial \tau} = -\frac{1}{Z_0} \boldsymbol{\nabla} \times \mathbf{E}. \tag{5.6}$$

Next, by recalling the components of Maxwell's equations using the Frenet-Serret vector formulas in (2.41) we have:

$$\frac{1}{Z_0}\frac{\partial E_s}{\partial \tau} = \frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} - qcG(y)\lambda(s-\tau-s_{\text{off}})\delta(x) \tag{5.7a}$$

$$\frac{1}{Z_0}\frac{\partial E_x}{\partial \tau} = \frac{\partial H_s}{\partial y} - \frac{1}{\eta}\frac{\partial H_y}{\partial s} \tag{5.7b}$$

$$\frac{1}{Z_0}\frac{\partial E_y}{\partial \tau} = \frac{1}{\eta}\frac{\partial H_x}{\partial s} - \frac{\partial H_s}{\partial x} - \frac{\kappa}{\eta}H_s \tag{5.7c}$$

$$Z_0\frac{\partial H_s}{\partial \tau} = \frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x} \tag{5.7d}$$

$$Z_0\frac{\partial H_x}{\partial \tau} = \frac{1}{\eta}\frac{\partial E_y}{\partial s} - \frac{\partial E_s}{\partial y} \tag{5.7e}$$

$$Z_0\frac{\partial H_y}{\partial \tau} = \frac{\partial E_s}{\partial x} + \frac{\kappa}{\eta}E_s - \frac{1}{\eta}\frac{\partial E_x}{\partial s} \tag{5.7f}$$

noting the curvilinear scale factor $\eta(s,x) = 1 + \kappa(s)x$. The curvature $\kappa(s)$ in (5.2) is discontinuous which assumes the source particles encounter hard-edged magnets to generate the prescribed reference orbit. While hard-edged magnets result in $\kappa'(s)$ becoming unbounded at the interface, this is only an issue in the wave equation forms of Maxwell's equations since the $\kappa'(s)$ terms do not appear in (5.7). Lastly, the parameter s_{off} is a fixed offset distance to the initial source position which centers the distribution $\lambda(s)$ to $s = s_{\text{off}}$ at $t = 0$. More details on this offset distance are discussed in section 5.2.2.

Now, applying Fourier Series in y from section 2.2.3 to (5.7) with $\alpha_p = \pi p/h$ corresponding to wavenumber p:

$$F_j(s,x,y,\tau) = \sum_{p=1}^{\infty} \phi_j(\alpha_p(y+h/2))F_{jp}(s,x,\tau), \tag{5.8}$$

with $\phi_j(\cdot) = \cos(\cdot)$ for E_y, H_s, H_x, and $\phi_j(\cdot) = \sin(\cdot)$ for E_s, E_x, H_y and $G(y)$. The six field equations are expressed modally as:

$$\frac{1}{Z_0}\frac{\partial E_{sp}}{\partial \tau} = \frac{\partial H_{yp}}{\partial x} + \alpha_p H_{xp} - qcG_p\lambda(s-\tau-s_{\text{off}})\delta(x), \tag{5.9a}$$
$$\frac{1}{Z_0}\frac{\partial E_{xp}}{\partial \tau} = -\alpha_p H_{sp} - \frac{1}{\eta}\frac{\partial H_{yp}}{\partial s}, \tag{5.9b}$$
$$\frac{1}{Z_0}\frac{\partial E_{yp}}{\partial \tau} = \frac{1}{\eta}\frac{\partial H_{xp}}{\partial s} - \frac{\partial H_{sp}}{\partial x} - \frac{\kappa}{\eta}H_{sp}, \tag{5.9c}$$
$$Z_0\frac{\partial H_{sp}}{\partial \tau} = \alpha_p E_{xp} - \frac{\partial E_{yp}}{\partial x}, \tag{5.9d}$$
$$Z_0\frac{\partial H_{xp}}{\partial \tau} = \frac{1}{\eta}\frac{\partial E_{yp}}{\partial s} - \alpha_p E_{sp}, \tag{5.9e}$$
$$Z_0\frac{\partial H_{yp}}{\partial \tau} = \frac{\partial E_{sp}}{\partial x} + \frac{\kappa}{\eta}E_{sp} - \frac{1}{\eta}\frac{\partial E_{xp}}{\partial s}. \tag{5.9f}$$

5.2.2 Initial Conditions

In this section we choose s_{off} so that the initial longitudinal distribution of the source $\lambda(s - s_{\text{off}})$ is assumed to be compactly supported in $s < s_0 \equiv 0$. While our Gaussian source distribution: $\lambda(z) = (2\pi\sigma^2)^{-1/2}\exp(-z^2/2\sigma^2)$ is not compactly supported, we choose $s_{\text{off}} = -10\sigma$ so that $\lambda(s_{\text{off}})/\lambda(0) < \epsilon_{\text{mach}} \approx 10^{-16}$. The initial fields are constructed by assuming the source and fields are steadily transported as they enter the domain from an infinitely long straight chamber. Steady-state refers to functions depending on s and τ only through the argument $(s-\tau)$. Using $\kappa = 0$ for the straight region, we seek a solution which satisfies (5.9a)–(5.9f) with $F_p(s,x,\tau) = F_p(s-\tau,x)$ for each field; this last restriction is thus a plane wave solution.

To construct these fields directly, we use the wave equation forms of Maxwell's equations in Cartesian coordinates since $(Z,X,Y) = (s,x,y)$ and $\mathbf{t} = \mathbf{e}_Z$ in the straight where $s < s_0$ and $\kappa = 0$:

$$\begin{aligned}\frac{\partial^2 \mathbf{E}}{\partial s^2} + \frac{\partial^2 \mathbf{E}}{\partial x^2} + \frac{\partial^2 \mathbf{E}}{\partial y^2} - \frac{\partial^2 \mathbf{E}}{\partial \tau^2} &= Z_0 \left(\frac{\partial \mathbf{j}}{\partial \tau} + c\boldsymbol{\nabla}\rho \right), \\ \frac{\partial^2 \mathbf{H}}{\partial s^2} + \frac{\partial^2 \mathbf{H}}{\partial x^2} + \frac{\partial^2 \mathbf{H}}{\partial y^2} - \frac{\partial^2 \mathbf{H}}{\partial \tau^2} &= -\boldsymbol{\nabla} \times \mathbf{j}.\end{aligned} \tag{5.10}$$

Next, using ρ and $\mathbf{j}$ from (2.46)–(2.47), and applying the plane wave condition to $\mathbf{E}$ and $\mathbf{H}$, the fields satisfy:

$$\frac{\partial^2 E_s}{\partial x^2} + \frac{\partial^2 E_s}{\partial y^2} = 0, \tag{5.11a}$$

$$\frac{\partial^2 E_x}{\partial x^2} + \frac{\partial^2 E_x}{\partial y^2} = qZ_0cG(y)\lambda(s-\tau-s_{\text{off}})\delta'(x), \tag{5.11b}$$

$$\frac{\partial^2 E_y}{\partial x^2} + \frac{\partial^2 E_y}{\partial y^2} = qZ_0cG'(y)\lambda(s-\tau-s_{\text{off}})\delta(x), \tag{5.11c}$$

$$\frac{\partial^2 H_s}{\partial x^2} + \frac{\partial^2 H_s}{\partial y^2} = 0, \tag{5.11d}$$

$$\frac{\partial^2 H_x}{\partial x^2} + \frac{\partial^2 H_x}{\partial y^2} = -qcG'(y)\lambda(s-\tau-s_{\text{off}})\delta(x), \tag{5.11e}$$

$$\frac{\partial^2 H_y}{\partial x^2} + \frac{\partial^2 H_y}{\partial y^2} = qcG(y)\lambda(s-\tau-s_{\text{off}})\delta'(x). \tag{5.11f}$$

Lastly, applying the Fourier Series in y, the fields satisfy the following ordinary differential equations (taking $s - \tau$ as a parameter):

$$\frac{d^2 E_{sp}}{dx^2} - \alpha_p^2 E_{sp} = 0, \tag{5.12a}$$

$$\frac{d^2 E_{xp}}{dx^2} - \alpha_p^2 E_{xp} = qZ_0cG_p\lambda(s-\tau-s_{\text{off}})\delta'(x), \tag{5.12b}$$

$$\frac{d^2 E_{yp}}{dx^2} - \alpha_p^2 E_{yp} = qZ_0c\alpha_pG_p\lambda(s-\tau-s_{\text{off}})\delta(x), \tag{5.12c}$$

$$\frac{d^2 H_{sp}}{dx^2} - \alpha_p^2 H_{sp} = 0, \tag{5.12d}$$

$$\frac{d^2 H_{xp}}{dx^2} - \alpha_p^2 H_{xp} = -qc\alpha_pG_p\lambda(s-\tau-s_{\text{off}})\delta(x), \tag{5.12e}$$

$$\frac{d^2 H_{yp}}{dx^2} - \alpha_p^2 H_{yp} = qcG_p\lambda(s-\tau-s_{\text{off}})\delta'(x). \tag{5.12f}$$

The boundary conditions for each field follow from the assumption of perfectly conducting walls as in (2.50)–(2.51), thus we have:

$$\left.\frac{dE_{xp}}{dx}\right|_{x_{\rm in},x_0} = 0, \qquad \left.E_{yp}\right|_{x_{\rm in},x_0} = 0, \qquad \left.E_{sp}\right|_{x_{\rm in},x_0} = 0,$$
$$\left.H_{xp}\right|_{x_{\rm in},x_0} = 0, \qquad \left.\frac{dH_{yp}}{dx}\right|_{x_{\rm in},x_0} = 0, \qquad \left.\frac{dH_{sp}}{dx}\right|_{x_{\rm in},x_0} = 0.$$

Due to the homogeneous boundary conditions with zero source term for E_{sp} and H_{sp}, the unique solutions are $E_{sp} = 0$ and $H_{sp} = 0$. The remaining $\mathbf{E}$ fields have the form:

$$E_{xp}(x; s-\tau) = -qZ_0cG_p\lambda(s-\tau-s_{\rm off})\Phi_p(x),$$
$$E_{yp}(x; s-\tau) = -qZ_0cG_p\lambda(s-\tau-s_{\rm off})\Psi_p(x),$$

where the functions $\Phi_p(x)$ and $\Psi_p(x)$ are:

$$\Phi_p(x) = \sinh(\alpha_p x_0)\frac{\cosh(\alpha_p(x-x_{\rm in}))}{\sinh(\alpha_p(x_0-x_{\rm in}))} - \cosh(\alpha_p x)\Theta(x),$$
$$\Psi_p(x) = \sinh(\alpha_p x_0)\frac{\sinh(\alpha_p(x-x_{\rm in}))}{\sinh(\alpha_p(x_0-x_{\rm in}))} - \sinh(\alpha_p x)\Theta(x),$$

with $\Theta(x)$ as the Heaviside step function. With these formulas and noting the relations: $E_{xp} = Z_0H_{yp}$ and $E_{yp} = -Z_0H_{xp}$ in the straight, all six initial fields are constructed by setting $\tau = 0$:

$$E_{sp}(s,x,0) = 0, \tag{5.13a}$$
$$E_{xp}(s,x,0) = -qZ_0cG_p\lambda(s-s_{\rm off})\Phi_p(x), \tag{5.13b}$$
$$E_{yp}(s,x,0) = -qZ_0cG_p\lambda(s-s_{\rm off})\Psi_p(x), \tag{5.13c}$$
$$H_{sp}(s,x,0) = 0, \tag{5.13d}$$
$$H_{xp}(s,x,0) = qcG_p\lambda(s-s_{\rm off})\Psi_p(x), \tag{5.13e}$$
$$H_{yp}(s,x,0) = -qcG_p\lambda(s-s_{\rm off})\Phi_p(x). \tag{5.13f}$$

5.2.3 Boundary Conditions

We have stated the necessary boundary conditions along the transverse perfectly conducting walls in section 2.2.2. However, the remaining boundaries at $s = s_{\rm min}$

and $s = s_{\text{max}}$, the entrance and exit to the vacuum chamber, need to be open boundaries allowing for radiation to pass through without reflection. Here, s_{min} is some distance at which $\lambda(s - s_{\text{off}})$ is compactly supported in the region $s_{\text{min}} < s < s_0$ at $\tau = 0$; thus $s_{\text{min}} \leq 2s_{\text{off}} = -20\sigma$. Several options exist to emulate this type of boundary condition; however, we opt to simply increase the length of the domain in each direction. The reason for this choice is that we are only interested in what occurs inside the region $s_0 \leq s \leq s_6$ up to moderate times no longer than a few times the length of the domain: $\tau < \tau_{\text{max}} \lesssim 3(s_6 - s_0)$. Since the chamber is narrow in these extensions, the number of elements does not increase substantially.

By adjusting s_{min} and s_{max} we can apply the global boundary condition: $\mathbf{n} \times \mathbf{E} = \mathbf{0}$, i.e. perfectly conducting walls, for all boundaries. We denote the surfaces $s = s_{\text{min}}$ and $s = s_{\text{max}}$ to be the s boundaries for the problem. With our given geometry, source, and initial conditions, $s_{\text{min}} < \min((2s_1 - \tau_{\text{max}})/2, 2s_{\text{off}})$ and $s_{\text{max}} > \max((\tau_{\text{max}} + s_6)/2, s_6)$ will ensure that the non-physical reflections produced in the buffer regions: $s_{\text{min}} < s < s_0$ and $s_6 < s < s_{\text{max}}$ do not enter the domain of interest: $0 \leq s \leq s_6$ for $0 \leq \tau \leq \tau_{\text{max}}$.

5.2.4 Evolution Procedure

Before we derive a semi-discrete form of (5.9) for numerical evaluation, we must treat the singularity on the right-hand-side of (5.9a). The following transformation eliminates the $\delta(x)$ term in the source in favor of a Heaviside function:

$$\widetilde{H}_{yp} = H_{yp} - qcG_p\lambda(s - \tau - s_{\text{off}})\Theta(x). \tag{5.14}$$

With this transformation, (5.9) becomes:

$$\frac{1}{Z_0}\frac{\partial E_{sp}}{\partial \tau} = \frac{\partial \widetilde{H}_{yp}}{\partial x} + \alpha_p H_{xp}, \tag{5.15a}$$

$$\frac{1}{Z_0}\frac{\partial E_{xp}}{\partial \tau} = -\alpha_p H_{sp} - \frac{1}{\eta}\frac{\partial \widetilde{H}_{yp}}{\partial s} - \frac{1}{\eta} qcG_p\lambda'(s-\tau-s_{\text{off}})\Theta(x), \tag{5.15b}$$

$$\frac{1}{Z_0}\frac{\partial E_{yp}}{\partial \tau} = \frac{1}{\eta}\frac{\partial H_{xp}}{\partial s} - \frac{\partial H_{sp}}{\partial x} - \frac{\kappa}{\eta} H_{sp}, \tag{5.15c}$$

$$Z_0\frac{\partial H_{sp}}{\partial \tau} = \alpha_p E_{xp} - \frac{\partial E_{yp}}{\partial x}, \tag{5.15d}$$

$$Z_0\frac{\partial H_{xp}}{\partial \tau} = \frac{1}{\eta}\frac{\partial E_{yp}}{\partial s} - \alpha_p E_{sp}, \tag{5.15e}$$

$$Z_0\frac{\partial \widetilde{H}_{yp}}{\partial \tau} = \frac{\partial E_{sp}}{\partial x} + \frac{\kappa}{\eta} E_{sp} - \frac{1}{\eta}\frac{\partial E_{xp}}{\partial s} + qZ_0cG_p\lambda'(s-\tau-s_{\text{off}})\Theta(x). \tag{5.15f}$$

This system of equations (5.15) can be discretized in (s, x) and evolved in τ with a suitable time-stepping method. A useful memory-conservative method is the 4th order Low-Storage Explicit Runge-Kutta (LSERK) method. At the cost of an extra function evaluation compared to the classical 4th order Runge-Kutta method, LSERK requires only 1 residual storage array instead of 4 per field. This memory-saving feature is invaluable for our simulations by allowing larger numbers of elements and/or higher order DG polynomials for our computations on a single GPU. The method for an ordinary differential equation: $du/d\tau = \mathcal{L}(u, \tau)$, is outlined below:

$$k_1 = \Delta\tau \mathcal{L}(u(\tau), \tau), \qquad u_1 = u(\tau) + b_1 \cdot k_1, \tag{5.16a}$$

$$k_2 = a_2 \cdot k_1 + \Delta\tau \mathcal{L}(u_1, \tau + c_2 \cdot \Delta\tau), \qquad u_2 = u_1 + b_2 \cdot k_2, \tag{5.16b}$$

$$k_3 = a_3 \cdot k_2 + \Delta\tau \mathcal{L}(u_2, \tau + c_3 \cdot \Delta\tau), \qquad u_3 = u_2 + b_3 \cdot k_3, \tag{5.16c}$$

$$k_4 = a_4 \cdot k_3 + \Delta\tau \mathcal{L}(u_3, \tau + c_4 \cdot \Delta\tau), \qquad u_4 = u_3 + b_4 \cdot k_4, \tag{5.16d}$$

$$k_5 = a_5 \cdot k_4 + \Delta\tau \mathcal{L}(u_4, \tau + c_5 \cdot \Delta\tau), \qquad u(\tau + \Delta\tau) = u_4 + b_5 \cdot k_5. \tag{5.16e}$$

This algorithm requires 5 function evaluations of the right-hand-side $\mathcal{L}$ and one storage variable denoted by k. The subscripts denote the stages but can be overwritten in memory since k_{i-1} and u_{i-1} are no longer needed after k_i and u_i are computed respectively. The constants a_i, b_i, and c_i are listed in the following table:

	$b_1 = 0.149659021999229$	
$a_2 = -0.417890474499852$	$b_2 = 0.379210312999627$	$c_2 = 0.149659021999229$
$a_3 = -1.192151694642677$	$b_3 = 0.822955029386982$	$c_3 = 0.370400957364205$
$a_4 = -1.697784692471528$	$b_4 = 0.699450455949122$	$c_4 = 0.622255763134443$
$a_5 = -1.514183444257156$	$b_5 = 0.153057247968152$	$c_5 = 0.958282130674690$

5.3 Numerical Methods

In this section we describe the procedures for evaluating the PDEs outlined in the previous section using a Discontinuous Galerkin method on an unstructured mesh. We also compare the results to experimental data collected at the CLS and examine convergence with respect to a high-resolution simulation.

5.3.1 Discontinuous Galerkin Formulation

We begin by truncating the domain in s by placing a numerical boundary at $s = s_{\min}$ and $s = s_{\max}$ as described in section 5.2.3. Next, we partition the domain $s_{\min} \leq s \leq s_{\max}$, $x_{\text{in}} \leq x \leq x_{\text{out}}(s)$ into K triangular elements. The elements with an edge shared on the outer wall $x_{out}(s)$ on $s_0 < s < s_2$ are not conformal since the boundary is curved. This boundary error will degrade the overall convergence of the scheme to 1st order; however, since the curvature we consider is very small relative to the element size, this choice does not limit the accuracy unless we use a coarser mesh with very high order elements. We have examined the use of curvilinear Gordon–Hall blended elements; however this significantly increased computational work with no improvement in accuracy for the parameters tested. More explanation on this fact can be found in section 5.3.2.

Each element consists of $N_p = (N+1)(N+2)/2$ nodes where N is the order of the polynomial used to locally represent the solution. The nodal locations on a particular element k are denoted by (s_j^k, x_j^k) for $j = 1, ..., N_p$.

Along interior edges: edges which are shared by two elements, the field is multivalued and discontinuous since the elements may have differing values along the shared edge. For each field component on a particular element D^k, we denote the approximate solution u^k by:

$$u^k(s, x, \tau) = \sum_{i=1}^{N_p} u_i^k(\tau)\ell_i^k(s, x), \tag{5.17}$$

where ℓ_i^k is the Lagrange interpolating polynomial of degree N with the property: $\ell_i^k(s_j^k, x_j^k) = \delta_{ij}$. As an aside, for each PDE, only 3 fields are coupled; for example, in (5.15f), only E_{xp}, E_{sp} and $\widetilde{H}_{yp}$ appear. The residual of each of our linear first-order PDEs has the form:

$$\mathcal{R}(s, x, \tau) := \frac{\partial u^k}{\partial \tau} - a(s, x)\frac{\partial v^k}{\partial s} - b\frac{\partial w^k}{\partial x} - c(s, x)w^k - f(s, x, \tau). \tag{5.18}$$

Here, for the example of (5.15f), u^k represents $\widetilde{H}_{yp}$, v^k represents E_{xp}, and w^k represents E_{sp}. The function forms for the coefficients are: $a(s, x) = \pm 1/\eta$, $b = \pm 1$ or 0, and $c(s, x) = \pm\alpha_p$ or $\pm\kappa/\eta$ depending on the PDE. The residual DG formulation can be formed by enforcing:

$$\int_{D^k} \mathcal{R}(s, x, \tau)\ell_j^k(s, x) ds dx = 0, \tag{5.19}$$

for each $j = 1, ..., N_p$ on each element k. Thus we have N_p equations of the form (5.19) for the N_p nodal values in element k. While this system is complete, it decouples all the elements producing a nonsense algorithm which will not yield a meaningful solution. To couple the elements to their neighbors along their shared edges, we integrate (5.19) by parts:

$$\int_{D^k}\left[\frac{\partial u_h^k}{\partial \tau}\ell_j^k + av_h^k\frac{\partial \ell_j^k}{\partial s} + \frac{\partial a}{\partial s}v_h^k\ell_j^k + bw_h^k\frac{\partial \ell_j^k}{\partial x} - cw_h^k\ell_j^k - f\ell_j^k\right]dsdx$$
$$= \int_{\partial D^k}\mathbf{n}\cdot(av_h^k, bw_h^k)^\top\ell_j^k dl. \qquad (5.20)$$

where $\mathbf{n}$ denotes the outward normal (n_s, n_x) along the boundary: ∂D^k. Next, we adjust the terms on the right-hand-side to incorporate information from elements neighboring D^k along the shared boundary ∂D^k. We denote these adjustments by $(av)^*$ and $(bw)^*$, commonly known as numerical fluxes. Then, after integrating by parts again to the form of (5.19), we obtain:

$$\int_{D^k}\mathcal{R}_h(s,x,\tau)\ell_j^k(s,x)dsdx = -\int_{\partial D^k}\mathbf{n}\cdot(av_h^k-(av)^*, bw_h^k-(bw)^*)^\top\ell_j^k(s,x)dl. \qquad (5.21)$$

Since u_h^k, v_h^k, and w_h^k, are sums of the same Lagrange polynomials as used in the residual, it becomes useful to define the mass matrix $\mathcal{M}^k$ and stiffness matrices $\mathcal{S}_s$, $\mathcal{S}_x$ of size $N_p \times N_p$:

$$[\mathcal{M}^k]_{ij} = \int_{D^k}\ell_i^k(s,x)\ell_j^k(s,x)dsdx, \qquad (5.22)$$
$$[\mathcal{S}_s^k]_{ij} = \int_{D^k}\ell_i^k(s,x)\frac{\partial \ell_j^k(s,x)}{\partial s}dsdx, \qquad (5.23)$$
$$[\mathcal{S}_x^k]_{ij} = \int_{D^k}\ell_i^k(s,x)\frac{\partial \ell_j^k(s,x)}{\partial x}dsdx. \qquad (5.24)$$

Next, since each triangle is different, we use a Jacobian matrix J^k to map each element to a reference element. This conveniently allows for the use of global differentiation matrices $\mathcal{D}_s = \mathcal{M}^{-1}\mathcal{S}_s$ and $\mathcal{D}_x = \mathcal{M}^{-1}\mathcal{S}_x$ which are the same for all elements k. The caveat for this advantage is that we must commute $\mathcal{M}^{-1}$ with $a(s,x)$; however, if we align the element interfaces so that $\kappa(s)$ is constant in each triangle then this issue is avoided. This is done by enforcing the mesh generating routine to align elements to have interfaces along $s = s_0$ and $s = s_2$, thus $\kappa = 0$ or $1/R$ depending on the element. Combining all these terms, the 6 system of Maxwell's equations written in DG semi-discrete form is:

$$\begin{aligned}\frac{dE_{sp}}{d\tau} &= Z_0\mathcal{D}_xH_{yp} + Z_0\alpha_pH_{xp}\\ &\quad+\frac{1}{2}(J\mathcal{M})^{-1}\Big(-Z_0\mathbf{n}_x[\![H_{yp}]\!] - [\![E_{sp}]\!] + \mathbf{n}_s\left(\mathbf{n}_s[\![E_{sp}]\!] + \mathbf{n}_x[\![E_{xp}]\!]\right)\Big),\end{aligned} \tag{5.25a}$$

$$\begin{aligned}\frac{dE_{xp}}{d\tau} &= -Z_0\alpha_pH_{sp} - \frac{Z_0}{\eta}\mathcal{D}_sH_{yp} - \frac{Z_0}{\eta}qcG_p\lambda'(s-\tau)\Theta(x)\\ &\quad+\frac{1}{2}(J\mathcal{M})^{-1}\left(\frac{Z_0}{\eta}\mathbf{n}_s[\![H_{yp}]\!] - [\![E_{xp}]\!] + \mathbf{n}_x\left(\mathbf{n}_s[\![E_{sp}]\!] + \mathbf{n}_x[\![E_{xp}]\!]\right)\right),\end{aligned} \tag{5.25b}$$

$$\begin{aligned}\frac{dE_{yp}}{d\tau} &= \frac{Z_0}{\eta}\mathcal{D}_sH_{xp} - Z_0\mathcal{D}_xH_{sp} - \frac{Z_0\kappa}{\eta}H_{sp}\\ &\quad+\frac{1}{2}(J\mathcal{M})^{-1}\left(-\frac{Z_0}{\eta}\mathbf{n}_s[\![H_{xp}]\!] + Z_0\mathbf{n}_x[\![H_{sp}]\!] - [\![E_{yp}]\!]\right),\end{aligned} \tag{5.25c}$$

$$\begin{aligned}\frac{dH_{sp}}{d\tau} &= -\frac{1}{Z_0}\mathcal{D}_xE_{yp} + \frac{\alpha_p}{Z_0}E_{xp}\\ &\quad+\frac{1}{2}(J\mathcal{M})^{-1}\left(\frac{1}{Z_0}\mathbf{n}_x[\![E_{yp}]\!] - [\![H_{sp}]\!] + \mathbf{n}_s\left(\mathbf{n}_s[\![H_{sp}]\!] + \mathbf{n}_x[\![H_{xp}]\!]\right)\right),\end{aligned} \tag{5.25d}$$

$$\begin{aligned}\frac{dH_{xp}}{d\tau} &= -\frac{\alpha_p}{Z_0}E_{sp} + \frac{1}{Z_0\eta}\mathcal{D}_sE_{yp}\\ &\quad+\frac{1}{2}(J\mathcal{M})^{-1}\left(\frac{-1}{Z_0\eta}\mathbf{n}_s[\![E_{yp}]\!] - [\![H_{xp}]\!] + \mathbf{n}_x\left(\mathbf{n}_s[\![H_{sp}]\!] + \mathbf{n}_x[\![H_{xp}]\!]\right)\right),\end{aligned} \tag{5.25e}$$

$$\begin{aligned}\frac{dH_{yp}}{d\tau} &= \frac{1}{Z_0}\mathcal{D}_xE_{sp} - \frac{1}{Z_0\eta}\mathcal{D}_sE_{xp} + \frac{\kappa}{Z_0\eta}E_{sp} + qcG_p\lambda'(s-\tau)\Theta(x)\\ &\quad+\frac{1}{2}(J\mathcal{M})^{-1}\left(\frac{-1}{Z_0}\mathbf{n}_x[\![E_{sp}]\!] + \frac{1}{Z_0\eta}\mathbf{n}_s[\![E_{xp}]\!] - [\![H_{yp}]\!]\right).\end{aligned} \tag{5.25f}$$

The $[\![u]\!] = u^- - u^+$ operator denotes the jump between interior and exterior values of the field along an interface between triangles. The J operator denotes the mapping Jacobian to a reference element which has a special structure for the curvilinear elements. The $\mathbf{n}_s$ and $\mathbf{n}_x$ arrays store the normal components to the interfaces of the triangles and the products including $\mathbf{n}_s$ and $\mathbf{n}_x$ denote Hadamard matrix multiplication (i.e. element-wise multiplication).

The boundary condition $\mathbf{n} \times \mathbf{E} = \mathbf{0}$ is imposed on all exterior edges, edges which are not shared with another element, through the numerical flux. We adopt the mirror principle for the boundary condition which has the form:

$$\mathbf{n} \times [\![\mathbf{E}]\!] = 2\mathbf{n} \times \mathbf{E}^- \tag{5.26}$$

No condition must be imposed on $\mathbf{H}$ for a well-posed system since (2.50b) is automatically satisfied for all time if the initial condition on $\mathbf{H}$ satisfies (2.50b). The jump in fields, $[\![\mathbf{H}]\!]$, on the global edges is simply set to zero. A full MATLAB implementation of these equations based on J. Hesthaven and T. Warburton's *Nodal Discontinuous Galerkin Methods* [11] is found in Appendix A.

5.3.2 Convergence Test Results

In this section we present convergence tests and solution results for (5.15a)–(5.15f) with the discontinuous Galerkin formulation of section 5.3.1. We begin with the convergence tests for the following set of parameters and constants in addition to those found in (5.1) and (5.4):

$$\begin{aligned} s_{\min} &= -400\text{mm}, & s_{\max} &= 2500\text{mm}, & s_{\text{off}} &= -200\text{mm}, & \tau_{\max} &= 200\text{mm}, \\ \sigma &= 20\text{mm}, & q &= 10^{-12}\text{C}, & c &= 2.998 \cdot 10^8\text{m/s}, & Z_0 &= 376.7\Omega \end{aligned}$$

We examine the solution for E_{yp} along the curve $s = 0$ at $\tau = 200$mm sampled at 500 equally spaced points in $x \in [-5\text{mm}, 50\text{mm}]$. By using meshes with edge-length $h_0 = \{25.0, 14.3, 10.0, 7.14, 5.56, 4.55, 3.85, 3.33\}$mm, for DG polynomial orders $N =$ 2, 4, 6, 8, we calculate the L_2 error with respect to a reference solution generated with $h_0 = 3.33$m and $N = 12$, see Figure 5.2. For $N = 2, 4, 6$ the convergence rate displayed spectral convergence: Error $\propto h_0^{N+1}$. However, for $N = 8$, the errors saturated around $1\text{e} - 13$ likely due to double-precision limitations.

We repeated the convergence tests using curvilinear elements but did not observe a noticeable difference in errors as the accuracy degradation due to triangular elements over curvilinear elements was not a limiting factor for our choice of parameters.

The triangular elements were a very good approximation to the conformal curved elements, generated by Gordon-Hall blending, due to the curvature of the outer wall compared to the element size and the modest DG orders we considered: $h_0 \ll R$, $N \leq 12$.

5.3.3 Physical Results

In this section we discuss the comparisons to physical experimental data taken at the CLS and presented in [4]. In particular, we examine the structure of the wake-fields produced by the source (2.46)–(2.47) as it travels through the vacuum chamber. For our tests, we focus on the electric field at a location near the outer wall just inside the entrance of the bend known as the "backward port".

The entrance of the backward port leading to the diode detector in the CLS is located at approximately $(s_{\text{bp}}, x_{\text{bp}}) = (330\text{mm}, 80\text{mm})$, see Figure 5.3. At this coordinate, we extract the E_{xp} field for the $p = 1$ mode only. This sampling is done using Nth-order polynomial interpolation at $(s_{\text{bp}}, x_{\text{bp}})$ at each timestep using the N_p nodal values of the element containing the point.

Next, we take the Fourier transform in τ of E_{xp} at $(s_{\text{bp}}, x_{\text{bp}})$ to apply a low-pass frequency filter to simulate the response of the diode detector at the backward port. Thus, the filtered field E^F_{xp} is:

$$\hat{E}_{xp}(s_{\text{bp}}, x_{\text{bp}}, k) = \int_0^{\tau_{\max}} e^{ik\tau} E_{xp}(s_{\text{bp}}, x_{\text{bp}}, \tau)d\tau, \tag{5.27}$$

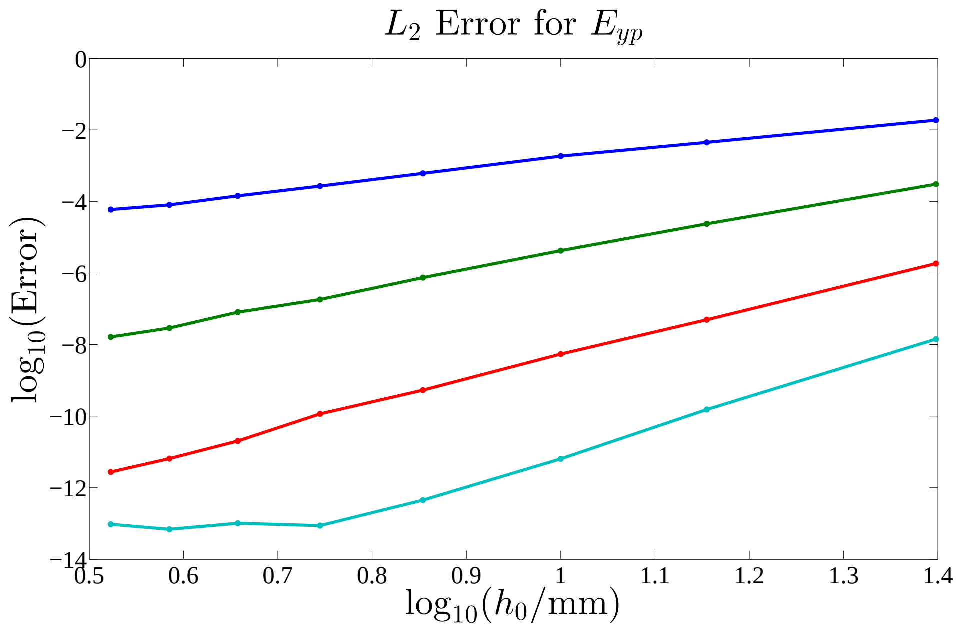
$$E^F_{xp}(s_{\text{bp}}, x_{\text{bp}}, \tau) = \frac{1}{2\pi} \int_{-k_{\max}}^{k_{\max}} e^{-ik\tau} \frac{1}{1 + ik/k_0} \hat{E}_{xp}(s_{\text{bp}}, x_{\text{bp}}, k)dk, \tag{5.28}$$

where $\tau_{\max}$ is the simulation time which is taken to be large enough so that $E_{xp} \sim 0$ for $\tau > \tau_{\max}$. In our experiments, we use $\tau_{\max} = 5000\text{mm}$ which is long enough to capture most reflections from the photon absorber and M1 mirror. We apply the filter cutoff frequency $k_0 = 0.0033\text{mm}^{-1}$ corresponding to a temporal frequency of 1GHz and we consider a maximal frequency $k_{\max} = 1\text{mm}^{-1}$.

Now we compare the plot of $|E^F_{xp}(s_{\text{bp}}, x_{\text{bp}}, \tau)|^2$ to the experimental data output from [4] shown in Figure 5.4. In both plots, the peaks A–G represent the times when certain features of CSR, generated in the bend, reflect off the conducting walls and reach the backward port.

The peaks A and C are attributed to the primary reflections from the photon absorber and M1 mirror respectively. This is evident by the time delay between peaks A and C corresponding to twice the difference in light travel time between the photon absorber and M1 mirror. Next, the peaks B and D–G correspond to the wake pulses from A and C respectively. The peak intensities and precise positions vary likely due to the simplified geometry of the model.

Lastly, Figure 5.5 displays contour plots of E_{xp}. These contours further highlight the complex wake-field structures generated by the geometry. In the top part of Figure 5.5 we show the E_{xp} field just after the source as exited the domain of interest ($s_0 \leq s \leq s_6$) on the right. The reflections of the photon absorber ($s \approx 1.4$m) and M1 mirror ($s \approx 1.8$m) are clearly visible and traveling to the left towards the backward port. In the bottom part of Figure 5.5, we highlight the complex structure of the wake-field pulse generated from the M1 mirror near the backward port. The wake-field pulse from the photon absorber is not visible as it has already exited the domain of interest on the left.



h_0	$N = 2$	$N = 4$	$N = 6$	$N = 8$
25.0mm	1.8676e − 02	3.0409e − 04	1.8365e − 06	1.4201e − 08
14.3mm	4.4789e − 03	2.3760e − 05	4.9445e − 08	1.5273e − 10
10.0mm	1.8475e − 03	4.2277e − 06	5.4275e − 09	6.4027e − 12
7.14mm	6.1020e − 04	7.4178e − 07	5.3219e − 10	4.4872e − 13
5.56mm	2.6820e − 04	1.8159e − 07	1.1521e − 10	8.7046e − 14
4.55mm	1.4322e − 04	8.0395e − 08	2.0178e − 11	1.0133e − 13
3.85mm	8.0400e − 05	2.9002e − 08	6.4940e − 12	6.8786e − 14
3.33mm	5.9269e − 05	1.6313e − 08	2.7470e − 12	9.4810e − 14
Rate	**2.93**	**4.95**	**6.70**	**6.21**

Figure 5.2: L_2 error for E_{yp} along $s = 0\text{mm}$ at $\tau = 200\text{mm}$ for order $N = 2$ (blue), $N = 4$ (green), $N = 6$ (red), and $N = 8$ (cyan) elements of approximate size h_0. The optimal rate of $N + 1$ is achieved for lower orders; however for $N = 8$, at the higher resolutions, we conjecture that the saturation in error is caused by double precision limitations.

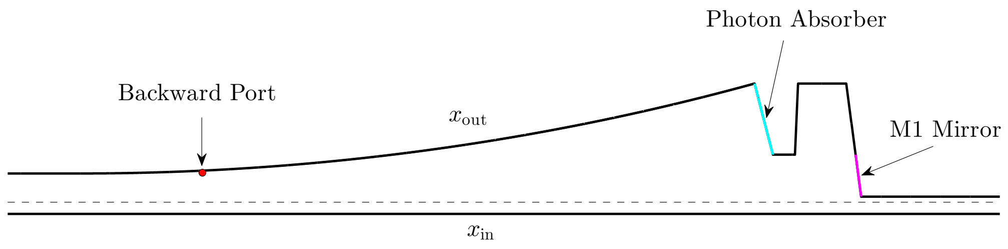


Figure 5.3: Backward port (red dot) where E_{xp} is sampled at each timestep. The source at $x = 0$ (dashed line) travels from left to right. The photon absorber (cyan) and M1 mirror assembly (magenta) are also shown.

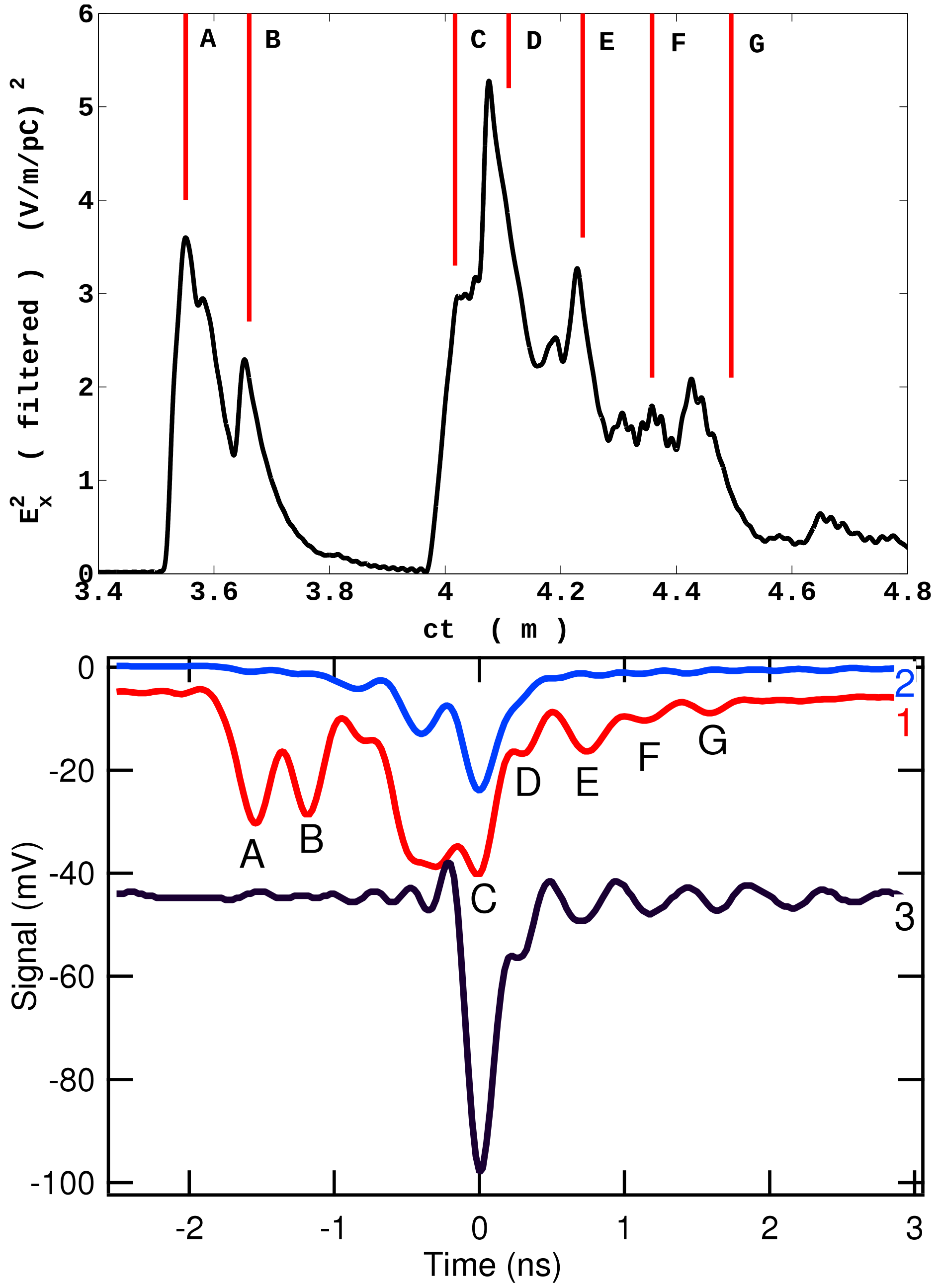
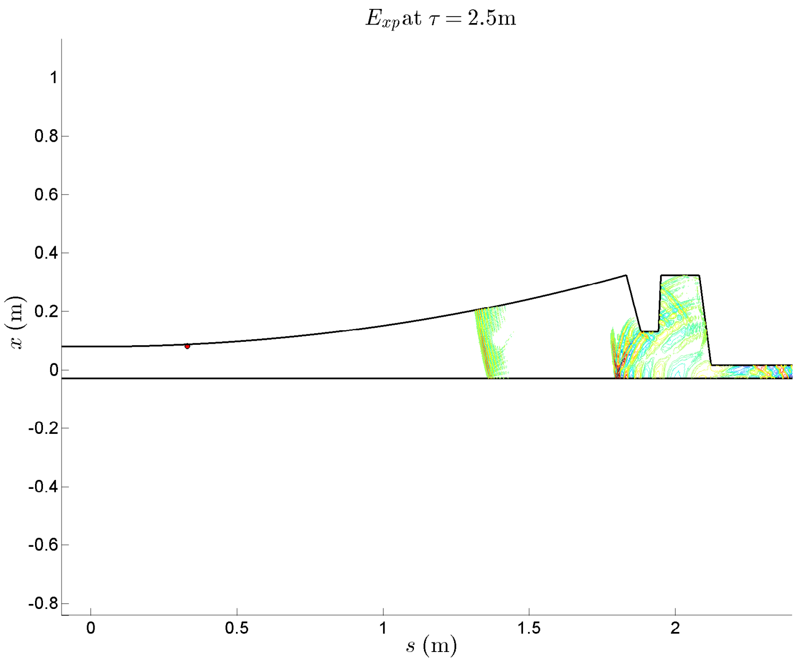


Figure 5.4: (Top) Low-pass filtered E_{xp}^2 field for $\sigma = 2$mm bunch length source at the backward port. (Bottom) Experimental RF diode measurements (oscilloscope traces) for various diode configuration and polarization: (1) [red] backward/horizontal, (2) [blue] backward/vertical, (3) [black] forward/horizontal. For clarity, the curves have been separated vertically.



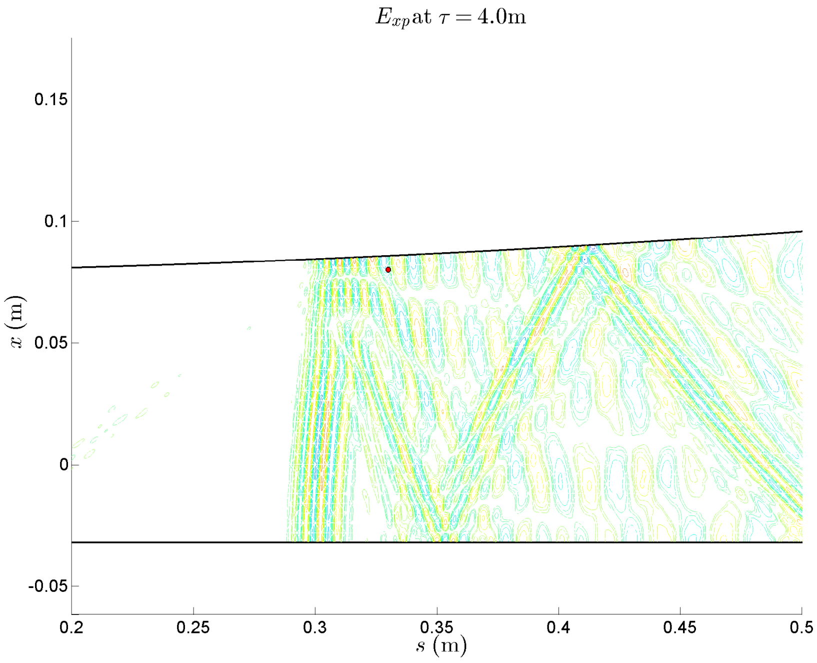


Figure 5.5: (Top) E_{xp} field for $\sigma = 2$mm at $\tau = 2.5$m on the entire domain. (Bottom) Zoom-in view of E_{xp} field at $\tau = 4.0$m at the backward port (red dot).

Chapter 6

Full 3D Time Domain Study – Work presented at ICOSAHOM14

6.1 Introduction

In this chapter, we will examine a new approach to solving the full 3D Maxwell's equations in the time-domain in a vacuum chamber with a constant cross-section. Particularly, we do not apply the Fourier series in y as in Chapters 3 and 5. A full 3D tetrahedral mesh discretization of an entire bend would not be computationally feasible due to memory limitations with our computing platform. If the electric and magnetic fields of interest are localized near the source, an attractive option is to use an adaptive mesh which follows the source.

One option for this adaptive mesh is to generate new meshes and interpolate as the source travels through the bend. However, this option introduces interpolation steps and new elements and associated operators which must be reconstructed throughout the time evolution. We instead adopt a Galilean transformation to model a localized source in a moving reference frame to restrict the computational domain to a region containing the source. This transformation enables us to use a common fixed mesh which travels with the source. The study and implementation of this method here is intended as a proof-of-concept and an application of the Galilean transformation.

6.2 Galilean Transformation

The central idea for the Galilean transformation we use is to transform Maxwell's equations to a coordinate system moving at speed v_g in the $+s$ direction. We define the Galilean transformation from the Frenet-Serret coordinates $\mathbf{r} = (s, x, y)$ by:

$$\begin{pmatrix} s \\ x \\ y \\ t \end{pmatrix} = \begin{pmatrix} \tilde{s} + v_g\tilde{t} \\ \tilde{x} \\ \tilde{y} \\ \tilde{t} \end{pmatrix} \tag{6.1}$$

with $v_g > 0$ but not necessarily bounded by c and for the moment we consider no curvature: $\kappa(s) = 0$. The reasoning behind this idea is to maintain a well-defined transformation at $v_g = c$, unlike the Lorentz transformation in which the scale factor $\gamma := (1 - v_g^2/c^2)^{-1/2}$ diverges as $v_g \to c$. The main advantage of a Galilean transformation with velocity $v_g \geq c$ is that boundaries at $\tilde{s} =$ constant become simplified to inflow/outflow boundaries. Considering a field $u(s, x, y, t)$, we have the transformed field related by:

$$\tilde{u}(\tilde{s}, \tilde{x}, \tilde{y}, \tilde{t}) := u(\tilde{s} + v_g\tilde{t}, \tilde{x}, \tilde{y}, \tilde{t}) = u(s, x, y, t) \tag{6.2}$$

With (6.1)–(6.2), the time derivatives of $u(s, x, y, t)$ in the new coordinates are:

$$\frac{\partial u}{\partial t} = \frac{\partial \tilde{u}}{\partial \tilde{t}} - v_g \frac{\partial \tilde{u}}{\partial \tilde{s}} \tag{6.3}$$

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 \tilde{u}}{\partial \tilde{t}^2} - 2v_g \frac{\partial^2 \tilde{u}}{\partial \tilde{t}\partial \tilde{s}} + v_g^2 \frac{\partial^2 \tilde{u}}{\partial \tilde{s}^2} \tag{6.4}$$

while the Frenet-Serret spatial derivatives: $\partial/\partial\tilde{s}$, $\partial/\partial\tilde{x}$, and $\partial/\partial\tilde{y}$ remain unchanged from (2.36)–(2.40) but are written with the new coordinates $(\tilde{s},\tilde{x},\tilde{y})$. Thus, a wave equation in Frenet-Serret coordinates for a vector field $\mathbf{U}(s,x,y,t)$, in the new coordinates $\tilde{\mathbf{U}}(\tilde{s},\tilde{x},\tilde{y},\tilde{t}) \equiv \mathbf{U}(s,x,y,t)$, has the form:

$$-\frac{1}{c^2}\Big(\frac{\partial^2\tilde{\mathbf{U}}}{\partial\tilde{t}^2} - 2v_g\frac{\partial^2\tilde{\mathbf{U}}}{\partial\tilde{t}\partial\tilde{s}} + v_g^2\frac{\partial^2\tilde{\mathbf{U}}}{\partial\tilde{s}^2}\Big) + \boldsymbol{\nabla}^2\tilde{\mathbf{U}} = \tilde{\mathbf{S}}(\tilde{s},\tilde{x},\tilde{y},\tilde{t}) \equiv \mathbf{S}(s,x,y,t) \tag{6.5}$$

where we use the $\boldsymbol{\nabla}^2$ operator from (2.40). Of particular note, in the case of $\kappa(s)=0$ and $v_g > c$, the $\partial^2\tilde{\mathbf{U}}/\partial\tilde{s}^2$ term changes sign. This alters the dynamics in (6.5) to no waves propagating the in the $+\tilde{s}$ direction. While we do not directly work with the Maxwell wave equations, we use the wave equation example here to demonstrate the $\partial^2\tilde{\mathbf{U}}/\partial\tilde{s}^2$ term sign changing property for $v_g > c$.

Lastly, if we include nonzero curvature $\kappa(s) \neq 0$, we must adjust the minimum v_g to ensure the waves do not propagate in the $+\tilde{s}$ direction throughout the domain. We will discuss this issue further in the next section.

6.3 Statement of the Problem

In this section we derive the initial value problem for Maxwell's equations with the Frenet-Serret and Galilean transformations. Recalling Chapter 2, equations (2.41), we begin with Maxwell's evolution equations for $(\mathbf{E},\mathbf{B})$ in Frenet-Serret coordinates:

$$\frac{\partial E_s}{\partial t} = c^2\left(\frac{\partial B_y}{\partial x} - \frac{\partial B_x}{\partial y}\right) - cZ_0 j_s(\mathbf{r},t), \tag{6.6a}$$

$$\frac{\partial E_x}{\partial t} = c^2\left(\frac{\partial B_s}{\partial y} - \frac{1}{\eta}\frac{\partial B_y}{\partial s}\right) - cZ_0 j_x(\mathbf{r},t), \tag{6.6b}$$

$$\frac{\partial E_y}{\partial t} = c^2\left(\frac{1}{\eta}\frac{\partial B_x}{\partial s} - \frac{1}{\eta}\frac{\partial(\eta B_s)}{\partial x}\right) - cZ_0 j_y(\mathbf{r},t), \tag{6.6c}$$

$$\frac{\partial B_s}{\partial t} = -\left(\frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y}\right), \tag{6.6d}$$

$$\frac{\partial B_x}{\partial t} = -\left(\frac{\partial E_s}{\partial y} - \frac{1}{\eta}\frac{\partial E_y}{\partial s}\right), \tag{6.6e}$$

$$\frac{\partial B_y}{\partial t} = -\left(\frac{1}{\eta}\frac{\partial E_x}{\partial s} - \frac{1}{\eta}\frac{\partial(\eta E_s)}{\partial x}\right). \tag{6.6f}$$

Next, we apply the Galilean transformation from (6.1) with v_g arbitrary for the moment. For the remainder of this chapter, to avoid confusion, we omit the tilde symbols $\tilde{}$ on the x, y, and t variables since $(x,y,t) = (\tilde{x},\tilde{y},\tilde{t})$. Thus, in the new variables $(\tilde{s},x,y,t) \equiv (s - v_g t, x, y, t)$, the system (6.6) becomes:

$$\frac{\partial E_{\tilde{s}}}{\partial t} = c^2\left(\frac{\partial B_y}{\partial x} - \frac{\partial B_x}{\partial y}\right) - cZ_0 \tilde{j}_{\tilde{s}} + v_g\frac{\partial E_{\tilde{s}}}{\partial \tilde{s}}, \tag{6.7a}$$

$$\frac{\partial E_x}{\partial t} = c^2\left(\frac{\partial B_{\tilde{s}}}{\partial y} - \frac{1}{\eta}\frac{\partial B_y}{\partial \tilde{s}}\right) - cZ_0 \tilde{j}_x + v_g\frac{\partial E_x}{\partial \tilde{s}}, \tag{6.7b}$$

$$\frac{\partial E_y}{\partial t} = c^2\left(\frac{1}{\eta}\frac{\partial B_x}{\partial \tilde{s}} - \frac{1}{\eta}\frac{\partial(\eta B_{\tilde{s}})}{\partial x}\right) - cZ_0 \tilde{j}_y + v_g\frac{\partial E_y}{\partial \tilde{s}}, \tag{6.7c}$$

$$\frac{\partial B_{\tilde{s}}}{\partial t} = -\left(\frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y}\right) + v_g\frac{\partial B_{\tilde{s}}}{\partial \tilde{s}}, \tag{6.7d}$$

$$\frac{\partial B_x}{\partial t} = -\left(\frac{\partial E_{\tilde{s}}}{\partial y} - \frac{1}{\eta}\frac{\partial E_y}{\partial \tilde{s}}\right) + v_g\frac{\partial B_x}{\partial \tilde{s}}, \tag{6.7e}$$

$$\frac{\partial B_y}{\partial t} = -\left(\frac{1}{\eta}\frac{\partial E_x}{\partial \tilde{s}} - \frac{1}{\eta}\frac{\partial(\eta E_{\tilde{s}})}{\partial x}\right) + v_g\frac{\partial B_y}{\partial \tilde{s}}. \tag{6.7f}$$

We denote the transformation in (6.7), as the Frenet-Serret–Galilean (FSG) system of Maxwell's equations.

For our domain, we consider a box in FSG coordinates $\Omega = [\tilde{s}_{\min}, \tilde{s}_{\max}] \times [-a,a] \times [-b,b]$ with a constant curvature $\kappa = 1/R$. In the original Cartesian coordinates, this volume Ω corresponds to a curved slab with rectangular cross sectional area $2a \times 2b$ of arc length $(\tilde{s}_{\max} - \tilde{s}_{\min})$ along the center of the cross section. This volume rotates about the axis $(Z, X) = (0, -R)$ with angular speed v_g/R.

Throughout this section, we consider the equations in (6.7) with $v_g \geq v_c$ where v_c is dependent on the spatial extent in x and the curvature κ. Specifically, we require: $v_c = \max(c/\eta(s,x))$ over the boundary to ensure that along the segments $\tilde{s} = \tilde{s}_{\max}$ and $\tilde{s} = \tilde{s}_{\min}$ the conditions are purely inflow and outflow respectively. Thus for our domain Ω, we have $v_c = c/(1 - a/R)$. Lastly, we only consider cases where $a \ll R$ and thus $v_g \geq v_c \gtrsim c$.

For the boundaries of Ω, we adopt the perfectly conducting boundary conditions as in (2.50) for the transverse walls of the domain along $x = \pm a$ and $y = \pm b$. We additionally set the inflow boundary at $\tilde{s} = \tilde{s}_{\max}$ to be zero for all fields.

Next, we must construct a set of initial conditions on $(\mathbf{E}, \mathbf{B})$ for the system in (6.7). We begin by assuming the source has a rigid profile in Frenet-Serret coordinates and moves in the $+s$ direction at speed c. We take the profile as compactly (or essentially) supported in Ω for some finite time $0 \leq t \leq T$. Since $v_g > c$, then any localized physical source, with speed bounded by c, cannot remain in Ω for all time. We consider a source charge and current density in FSG coordinates with the form given in (2.48)–(2.49):

$$\begin{aligned} \tilde{\rho}(\tilde{s}, x, y, t) &= q\lambda(\tilde{s} + (v_g - c)t)\lambda(x)\lambda(y), \\ \tilde{j}_s(\tilde{s}, x, y, t) &= qc\lambda(\tilde{s} + (v_g - c)t)\lambda(x)\lambda(y), \end{aligned} \tag{6.8}$$

with the distribution λ, as Gaussian with zero mean and variance σ. For this round distribution, if σ is sufficiently small, then the source is effectively contained in Ω until $t \simeq \tilde{s}_{\min}/(v - c)$. Note, for this source, we take $\tilde{j}_x = \tilde{j}_y = 0$.

Using the method in Chapter 5 for constructing the initial conditions by assuming a plane-wave steady-state solution with $\kappa = 0$. This plane-wave condition assumes solutions of $(\mathbf{E},\mathbf{B})$ initially are of the form $\mathbf{F}(\tilde{s}+(v_g-c)t,x,y)$ which cancel all $\tilde{s}$ and t derivatives in (6.7). By setting the parameter $\tilde{s}+(v_g-c)t\to\tilde{s}$ to compute the initial condition, we solve the following Poisson equations, derived from the wave equations of (6.7), subject to the transverse perfectly conducting boundary conditions:

$$\frac{\partial^2 E_{\tilde{s}}}{\partial x^2}+\frac{\partial^2 E_{\tilde{s}}}{\partial y^2}=0, \tag{6.9a}$$

$$\frac{\partial^2 E_x}{\partial x^2}+\frac{\partial^2 E_x}{\partial y^2}=qZ_0c\lambda(\tilde{s})\lambda'(x)\lambda(y), \tag{6.9b}$$

$$\frac{\partial^2 E_y}{\partial x^2}+\frac{\partial^2 E_y}{\partial y^2}=qZ_0c\lambda(\tilde{s})\lambda(x)\lambda'(y), \tag{6.9c}$$

$$\frac{\partial^2 B_{\tilde{s}}}{\partial x^2}+\frac{\partial^2 B_{\tilde{s}}}{\partial y^2}=0, \tag{6.9d}$$

$$\frac{\partial^2 B_x}{\partial x^2}+\frac{\partial^2 B_x}{\partial y^2}=-qZ_0\lambda(\tilde{s})\lambda(x)\lambda'(y), \tag{6.9e}$$

$$\frac{\partial^2 B_y}{\partial x^2}+\frac{\partial^2 B_y}{\partial y^2}=qZ_0\lambda(\tilde{s})\lambda'(x)\lambda(y). \tag{6.9f}$$

The transverse boundary conditions follow from (2.50)–(2.51). Along $x=\pm a$ and $y=\pm b$, these conditions are:

$$E_{\tilde{s}}\Big|_{x=\pm a}=0,\qquad \frac{\partial E_x}{\partial x}\Big|_{x=\pm a}=0,\qquad E_y\Big|_{x=\pm a}=0, \tag{6.10a}$$

$$\frac{\partial B_{\tilde{s}}}{\partial x}\Big|_{x=\pm a}=0,\qquad B_x\Big|_{x=\pm a}=0,\qquad \frac{\partial B_y}{\partial x}\Big|_{x=\pm a}=0, \tag{6.10b}$$

$$E_{\tilde{s}}\Big|_{y=\pm b}=0,\qquad E_x\Big|_{y=\pm b}=0,\qquad \frac{\partial E_y}{\partial y}\Big|_{y=\pm b}=0, \tag{6.10c}$$

$$\frac{\partial B_{\tilde{s}}}{\partial y}\Big|_{y=\pm b}=0,\qquad \frac{\partial B_x}{\partial y}\Big|_{y=\pm b}=0,\qquad B_y\Big|_{y=\pm b}=0. \tag{6.10d}$$

The solutions to (6.9) are constructed via Fourier series and have the form:

$$E_x(\tilde{s},x,y,0) = \sum_{m=0}^{\infty}\sum_{p=1}^{\infty} E^x_{mp}(\tilde{s})\cos(\alpha_m(x-a))\sin(\beta_p(y-b)), \tag{6.11a}$$

$$E_y(\tilde{s},x,y,0) = \sum_{m=1}^{\infty}\sum_{p=0}^{\infty} E^y_{mp}(\tilde{s})\sin(\alpha_m(x-a))\cos(\beta_p(y-b)), \tag{6.11b}$$

$$B_x(\tilde{s},x,y,0) = \sum_{m=1}^{\infty}\sum_{p=0}^{\infty} B^x_{mp}(\tilde{s})\sin(\alpha_m(x-a))\cos(\beta_p(y-b)), \tag{6.11c}$$

$$B_y(\tilde{s},x,y,0) = \sum_{m=0}^{\infty}\sum_{p=1}^{\infty} B^y_{mp}(\tilde{s})\cos(\alpha_m(x-a))\sin(\beta_p(y-b)), \tag{6.11d}$$

while the fields $E_{\tilde{s}}$ and $B_{\tilde{s}}$, due to homogeneous sources, are zero (we assign the constant in the solution for $B_{\tilde{s}}$ to be zero). The Fourier coefficients in (6.11) are given by the following integrals over the transverse domain $\Omega_{2D} := [-a,a]\times[-b,b]$:

$$E^x_{mp}(\tilde{s}) = \frac{-qZ_0c\lambda(\tilde{s})}{ab\phi_{mp}} \iint_{\Omega_{2D}} \lambda'(x)\lambda(y)\cos(\alpha_m(x-a))\sin(\beta_p(y-b))dydx, \tag{6.12a}$$

$$E^y_{mp}(\tilde{s}) = \frac{-qZ_0c\lambda(\tilde{s})}{ab\phi_{mp}} \iint_{\Omega_{2D}} \lambda(x)\lambda'(y)\sin(\alpha_m(x-a))\cos(\beta_p(y-b))dydx, \tag{6.12b}$$

$$B^x_{mp}(\tilde{s}) = \frac{qZ_0\lambda(\tilde{s})}{ab\phi_{mp}} \iint_{\Omega_{2D}} \lambda(x)\lambda'(y)\sin(\alpha_m(x-a))\cos(\beta_p(y-b))dydx, \tag{6.12c}$$

$$B^y_{mp}(\tilde{s}) = \frac{-qZ_0\lambda(\tilde{s})}{ab\phi_{mp}} \iint_{\Omega_{2D}} \lambda'(x)\lambda(y)\cos(\alpha_m(x-a))\sin(\beta_p(y-b))dydx, \tag{6.12d}$$

where $\alpha_m = \pi m/2a$, $\beta_p = \pi p/2b$, and $\phi_{mp} = \alpha_m^2 + \beta_p^2$ for $m, p \neq 0$. For the modes with m or $p = 0$, then we replace ϕ_{mp} with $2\phi_{mp}$ in (6.12). With (6.11)–(6.12) and $E_{\tilde{s}}(\tilde{s},x,y,0) = B_{\tilde{s}}(\tilde{s},x,y,0) \equiv 0$, we have constructed the initial data for (6.7).

In practice, we do not compute the Fourier series given by (6.11)–(6.12) and instead use a Poisson solver on a 2D DG scheme with triangles and interpolate to the 3D DG nodes based on their (x, y) coordinates. We then scale the fields by $\lambda(\tilde{s})$, based on their $\tilde{s}$ coordinate. This process will be described in section 6.5.2.

6.4 DG Formulation

In this section we pose the DG implementation to solve (6.7) with the initial and boundary conditions described in section 6.3. Much of this implementation follows from the procedures found in Chapter 10 of *Nodal Discontinuous Galerkin Methods* by Hesthaven and Warburton [11].

For the construction of an Nth order solution on $\Omega = [\tilde{s}_{\min}, \tilde{s}_{\max}] \times [-a, a] \times [-b, b]$, we begin with considering a single tetrahedral element, denoted by D^k, from the total number of elements K which partition Ω. For Nth order elements, we require $N_p = (N+1)(N+2)(N+3)/6$ nodes per element. Using the N_p nodes, we define a local polynomial solution for each field on the element D^k by:

$$u^k(\tilde{s}, x, y, t) = \sum_{i=1}^{N_p} u_i^k(t) \ell_i^k(\tilde{s}, x, y), \tag{6.13}$$

where ℓ_i^k is the 3D Lagrange interpolating polynomial of degree N with the property $\ell_i^k(\tilde{s}_j^k, x_j^k, y_j^k) = \delta_{ij}$ for each node $(\tilde{s}_j^k, x_j^k, y_j^k)$ in element k.

Now we construct the residuals for (6.7). We will only illustrate the process in detail for (6.7a); however, the construction for the remaining 5 fields follows in a similar manner and the results will be presented in the next section. By applying the expansion in (6.13) to $E_{\tilde{s}}$, B_x, B_y, and $j_{\tilde{s}}$, we obtain the residual:

$$\mathcal{R}(\tilde{s}, x, y, t) := \frac{\partial E_{\tilde{s}}^k}{\partial t} - c^2 \frac{\partial B_y^k}{\partial x} + c^2 \frac{\partial B_x^k}{\partial y} + cZ_0 j_{\tilde{s}}^k - v_g \frac{\partial E_{\tilde{s}}^k}{\partial \tilde{s}}. \tag{6.14}$$

Next, we desire orthogonality of this residual to the same Lagrange polynomials

$\ell_j^k(\tilde{s},x,y)$ for $j = 1, ..., N_p$:

$$\int_{D^k} \mathcal{R}(\tilde{s},x,y,t)\ell_j^k(\tilde{s},x,y)d\tilde{s}dxdy = 0. \tag{6.15}$$

The above residual condition (6.15) yields N_p equations for the N_p unknown nodal values $\partial E_{\tilde{s}i}^k/\partial t$. These equations are still uncoupled from the global domain; to couple element k to its neighbors we integrate (6.15) by parts:

$$\begin{aligned}\int_{D^k} \frac{\partial E_{\tilde{s}}^k}{\partial t}\ell_j^k - c^2 B_y^k \frac{\partial \ell_j^k}{\partial x} + c^2 B_x^k \frac{\partial \ell_j^k}{\partial y} + cZ_0 j_{\tilde{s}}^k \ell_j^k - v_g E_{\tilde{s}}^k \frac{\partial \ell_j^k}{\partial \tilde{s}} d\tilde{s}dydx \\ = \int_{\partial D^k} \mathbf{n}\cdot[-v_g E_{\tilde{s}}^k, -c^2 B_y^k, c^2 B_x^k]\ell_j^k dA,\end{aligned} \tag{6.16}$$

where $\mathbf{n} := [n_{\tilde{s}}, n_x, n_y]$ denotes the outward normal of D^k along the boundary. Next, we adjust the right-hand-side of (6.16) with numerical flux terms which incorporate information from the neighboring elements of D^k:

$$\begin{aligned}\int_{D^k} \frac{\partial E_{\tilde{s}}^k}{\partial t}\ell_j^k - c^2 B_y^k \frac{\partial \ell_j^k}{\partial x} + c^2 B_x^k \frac{\partial \ell_j^k}{\partial y} + cZ_0 j_{\tilde{s}}^k \ell_j^k + v_g E_{\tilde{s}}^k \frac{\partial \ell_j^k}{\partial \tilde{s}} d\tilde{s}dydx \\ = \int_{\partial D^k} \mathbf{n}\cdot[v_g E_{\tilde{s}}^*, -c^2 B_y^*, c^2 B_x^*]\ell_j^k dA.\end{aligned} \tag{6.17}$$

Lastly, integrating by parts again, we return to form of (6.15) but with the adjusted right-hand-side incorporating the neighboring elements. This is commonly referred as the strong form:

$$\begin{aligned}&\int_{D^k} \mathcal{R}(\tilde{s},x,y,t)\ell_j^k(\tilde{s},x,y)d\tilde{s}dxdy = \\ &\quad -\int_{\partial D^k} \mathbf{n}\cdot[v_g(E_{\tilde{s}}^k - E_{\tilde{s}}^*), -c^2(B_y^k - B_y^*), c^2(B_x^k - B_x^*)]\ell_j^k(\tilde{s},x,y)dA.\end{aligned} \tag{6.18}$$

At this point, we introduce the mass and stiffness matrices $\mathcal{M}^k$, $\mathcal{S}_{\tilde{s}}^k$, $\mathcal{S}_x^k$, and $\mathcal{S}_y^k$

which are central to representing (6.18) in semi-discrete form:

$$[\mathcal{M}^k]_{ij} = \int_{D^k} \ell_i^k(\tilde{s},x,y)\ell_j^k(\tilde{s},x,y)d\tilde{s}dxdy, \tag{6.19a}$$

$$[\mathcal{S}_{\tilde{s}}^k]_{ij} = \int_{D^k} \ell_i^k(\tilde{s},x,y)\frac{\partial \ell_j^k(\tilde{s},x,y)}{\partial \tilde{s}}d\tilde{s}dxdy, \tag{6.19b}$$

$$[\mathcal{S}_{x}^k]_{ij} = \int_{D^k} \ell_i^k(\tilde{s},x,y)\frac{\partial \ell_j^k(\tilde{s},x,y)}{\partial x}d\tilde{s}dxdy, \tag{6.19c}$$

$$[\mathcal{S}_{y}^k]_{ij} = \int_{D^k} \ell_i^k(\tilde{s},x,y)\frac{\partial \ell_j^k(\tilde{s},x,y)}{\partial y}d\tilde{s}dxdy. \tag{6.19d}$$

We also define the differentiation matrices $\mathcal{D}_{\tilde{s}}^k := (\mathcal{M}^k)^{-1}\mathcal{S}_{\tilde{s}}^k$, $\mathcal{D}_{x}^k := (\mathcal{M}^k)^{-1}\mathcal{S}_{x}^k$, and $\mathcal{D}_{y}^k := (\mathcal{M}^k)^{-1}\mathcal{S}_{y}^k$. We now construct the semi-discrete form of (6.7a) as:

$$\frac{dE_{\tilde{s}}^k}{dt} = c^2\mathcal{D}_x^k B_y^k - c^2\mathcal{D}_y^k B_x^k - cZ_0 j_{\tilde{s}}^k + v_g \mathcal{D}_{\tilde{s}}^k E_{\tilde{s}}^k + \frac{1}{2}(\mathcal{M}^k)^{-1}\int_{\partial D^k} E_{\tilde{s}}^{\mathrm{F}} \boldsymbol{\ell}^k dA, \tag{6.20a}$$

$$\begin{aligned} E_{\tilde{s}}^{\mathrm{F}} =& c^2 n_x [\![B_y]\!] - c^2 n_y [\![B_x]\!] \\ &+ \left(1 + \frac{v_g}{c}n_{\tilde{s}}\right)[\![E_{\tilde{s}}]\!] - n_{\tilde{s}}(n_{\tilde{s}}[\![E_{\tilde{s}}]\!] + n_x[\![E_x]\!] + n_y[\![E_y]\!]). \end{aligned} \tag{6.20b}$$

The term $E_{\tilde{s}}^F$ denotes the upwind numerical flux term which incorporates jumps in values of the fields from neighboring elements with $[\![u]\!] = u^+ - u^-$ where u^- is the value of the field from within the element k and u^+ is the value of the field from the adjacent element along an interface. The outward normal components: $n_{\tilde{s}}$, n_x, and n_y, are defined along the 4 faces of the element k. Lastly, the operator $(\mathcal{M}^k)^{-1}\int_{\partial D^k} \boldsymbol{\ell}^k dA$ is the surface-to-volume "LIFT" operator as mentioned in [11].

Imposing the boundary conditions are a bit more involved as the transverse boundaries $x = \pm a$ and $y = \pm b$ are perfectly conducting while the boundaries along $s = s_{\max}, s_{\min}$ are inflow/outflow. Thus 3 different boundary conditions are required along the 6 planar boundaries of $\partial\Omega$: the PEC condition, the inflow condition, and the outflow condition.

For the PEC transverse boundaries we only impose (2.50a). Since (2.50b) is satisfied by the initial condition, then it is satisfied for all time and need not be imposed. The condition in (2.50a) is imposed via the mirror principle so that $\mathbf{n} \times [\![\mathbf{E}]\!] = -2\mathbf{n} \times \mathbf{E}^-$. For the inflow boundary, we take all 6 fields as zero upwind. Specifically, the inflow condition relates the jumps in fields as $[\![\mathbf{E}]\!] = -\mathbf{E}^-$ and $[\![\mathbf{B}]\!] = -\mathbf{B}^-$, while the outflow boundary simply requires no condition. More details on the implementation of these conditions are discussed in section 6.5.3.

6.5 Numerical Implementation

In this section we describe step-by-step the entire process for solving the initial value problem (6.7) for $v_g \geq v_c$ on Ω with initial data satisfying (6.11) and the perfectly conducting boundary conditions for $x = \pm a$ and $y = \pm b$, and inflow/outflow conditions on $s = s_{\max}, s_{\min}$.

6.5.1 Preprocessing Steps

We begin by partitioning the domain $\Omega = [\tilde{s}_{\min}, \tilde{s}_{\max}] \times [-a, a] \times [-b, b]$ into K tetrahedral elements. While many mesh generating algorithms exist for this application, we simply partition Ω into $N_s^{\text{res}} N_x^{\text{res}} N_y^{\text{res}}$ rectangular boxes (typically cubes) and then subpartition each of the boxes into 6 tetrahedrons generating $K = 6N_s^{\text{res}} N_x^{\text{res}} N_y^{\text{res}}$ elements. An example of a box and tetrahedron is shown in Figure 6.1.

We proceed by defining the nodal coordinates for the N_p nodes within each element. These nodes are first generated on a standard reference element: the tetrahedron defined by the vertices $(-1, -1, -1)$, $(1, -1, -1)$, $(-1, 1, -1)$, $(-1, -1, 1)$ and later linearly mapped to each of the physical elements. We employ Lagrange–Gauss–Lobatto nodal spacing for better matrix conditioning; these nodes are described in detail on p407–418 of [11].

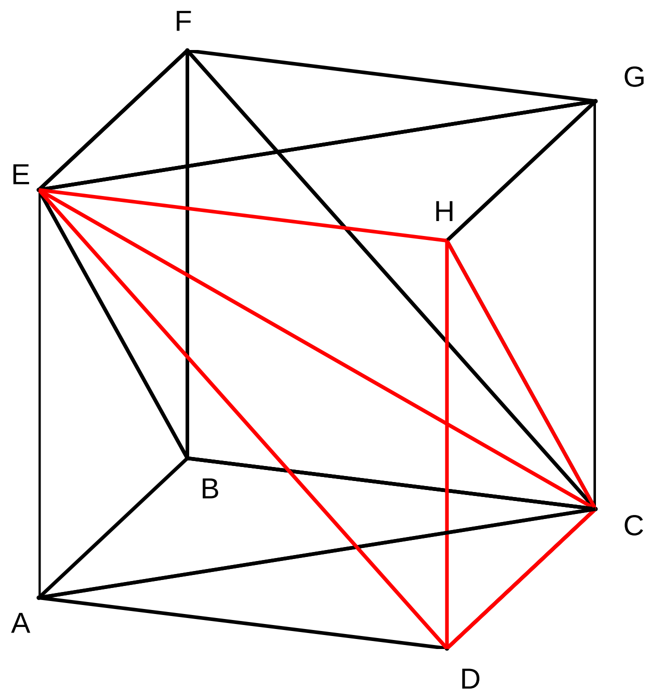


Figure 6.1: An example cube containing 6 congruent tetrahedral elements denoted by vertices: $ACBE$, $BEFC$, $FCGE$, $DEAC$, $HCDE$, and $GEHC$. One tetrahedron, $HCDE$, is highlighted in red.

Using the nodal coordinates on a standard reference element, we build the differentiation matrices $\mathcal{D}_{\tilde{s}}$, $\mathcal{D}_x$, $\mathcal{D}_y$, and the surface-to-volume integral operator $(\mathcal{M})^{-1}\mathcal{E}$ used in the numerical flux for the reference element. As a note, the operator we denote $(\mathcal{M})^{-1}\mathcal{E}$ is a matrix of size $N_p \times 4N_{fp}$ where $N_{fp} = (N+1)(N+2)/2$ are the number of nodes along each triangular face of the elements. The purpose of $(\mathcal{M})^{-1}\mathcal{E}$ is to evaluate the surface integral over the element faces as seen on the right-hand-side of (6.16). A detailed description of these steps are found on p418–425 of [11].

Next, we generate the geometric factors which transform the coordinates between the reference element and the physical elements. These factors are the terms in the Jacobian array J of size $N_p \times K$ which transform the physical element to the reference coordinates. Additionally, we generate the face normal arrays $n_{\tilde{s}}$, n_x, n_y each of size $4N_{fp} \times K$ which contain the outward normal components for each node along the element faces. These steps are covered on p425–427 of [11].

With the K tetrahedral elements and their vertices from mesh generation, we must also build the connectivity maps. These arrays incorporate the information about which elements are connected to a given element as well as which faces of the elements are shared. These maps are used to generate the node lists for which nodes are in contact along an interface. This information is used in constructing the numerical flux based on the interior and exterior values of an element. It is important to note that nodes along element edges share faces with more than one other element and as such appear multiple times in the connectivity maps.

In Figure 6.2, the red nodes lie in the interior of the element, the green nodes lie on the triangular faces of the element, and the blue nodes lie on the edges and corners of the element. The numerical flux jumps $[\![\cdot]\!]$ are generated with the boundary node values (green and blue) based on which face, not edge, they correspond to. Thus, while the edge nodes (blue) may be shared with many elements, only the corresponding triangular faces form numerical fluxes. For the example of element ACBE and the edge nodes along segment EC, only the elements BEFC and DEAC have faces shared with ACBE. The nodal values of elements FCGE, HCDE, and GEHC do not contribute to the numerical flux for element ACBE. With the nodes and connectivity maps complete, we can set up the routine to construct for the initial conditions for each field.

It is worth clarifying that the edge and corner nodes, where the faces of an individual element meet, are listed 2 and 3 times respectively in the face node lists. Recalling a given element has $N_p = (N+1)(N+2)(N+3)/6$ nodes in total, precisely $N_{p,vol} = (N-3)(N-2)(N-1)/6$ nodes lie in the interior, $N_{p,faces} = 2(N-2)(N-1)$ nodes lie strictly on faces but not edges, $N_{p,edges} = 6(N-1)$ nodes lie strictly on edges

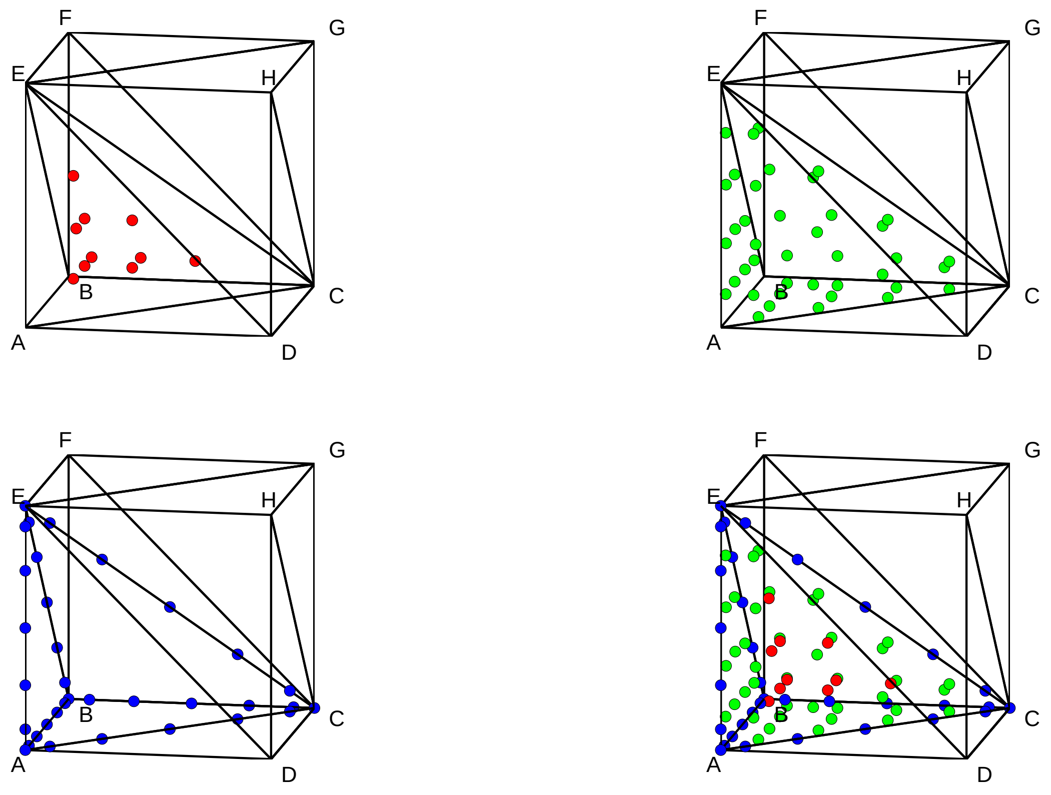


Figure 6.2: Tetrahedral 6th-order nodes for element ACBE. (Top left) interior volume nodes in red. (Top right) face nodes only shared with 1 other element in green. (Bottom left) Edge and corner nodes shared with several other elements in blue. (Bottom right) all $N_p = 84$ nodes displayed together.

but not corners, and $N_{p,verts} = 4$ nodes lie on the corner vertices. We therefore have the decomposition: $N_p = N_{p,vol} + N_{p,faces} + N_{p,edges} + N_{p,verts}$. In Figure 6.2 we display $N_{p,vol}$ nodes in red, $N_{p,faces}$ nodes in green, and $N_{p,edges} + N_{p,verts}$ nodes in blue. Thus, the face node list for a single element is of length $4N_{fp} = N_{p,faces} + 2N_{p,edges} + 3N_{p,verts}$.

6.5.2 Initial Condition Setup

To generate the initial conditions which satisfy (6.9)–(6.10), we use a sparse DG Poisson solver instead of the Fourier series solutions from (6.11)–(6.12). To build a

2D solution from (6.9), we first divide out the $\lambda(\tilde{s})$ dependence as a parameter to solve for the quantities $\mathbf{E}_{2D}(x,y) := \mathbf{E}(\tilde{s},x,y,0)/\lambda(\tilde{s})$ and $\mathbf{B}_{2D}(x,y) := \mathbf{B}(\tilde{s},x,y,0)/\lambda(\tilde{s})$.

Next, we partition the 2D domain $\Omega_{2D} := [-a,a] \times [-b,b]$ into K_{2D} triangular, order N_{2D} elements; the order N_{2D} should be at least the order N used in the 3D problem. We apply the 2D sparse DG Poisson solver detailed on p275–280 of [11] to solve for $(\mathbf{E}_{2D}, \mathbf{B}_{2D})$ on the set of nodes $(x_{2D}, y_{2D}) \in \Omega_{2D}$ where the total number of nodes is $N_{p2D} \times K_{2D}$ where $N_{p2D} = (N_{2D}+1)(N_{2D}+2)/2$.

We next must interpolate the solution of $(\mathbf{E}_{2D}, \mathbf{B}_{2D})$, defined on $(x_{2D}, y_{2D}) \in \Omega_{2D}$, to the 3D nodal $(\tilde{s},x,y)$ coordinates. For every 3D node $(\tilde{s},x,y)$, we project the location onto Ω_{2D} by its (x,y) coordinate. We then locate which triangular 2D element in Ω_{2D} the point (x,y) is located in. Next, we interpolate the solution of $(\mathbf{E}_{2D}, \mathbf{B}_{2D})$ at (x,y) using its N_{2D} order polynomial solution on that element. If the point (x,y) lies on an edge or corner in Ω_{2D}, then we take the average of the interpolated value of $(\mathbf{E}_{2D}, \mathbf{B}_{2D})$ at (x,y) among all elements containing (x,y).

Once the solution $(\mathbf{E}_{2D}, \mathbf{B}_{2D})$ has been computed for each 3D node's (x,y) coordinate, we multiply this solution by $\lambda(\tilde{s})$ to construct the initial condition for the 3D problem which satisfies (6.9)–(6.10). To summarize, we solve for (6.9)–(6.10) independent of $\lambda(\tilde{s})$ with a 2D solution in (x,y) and then extend this 2D solution to 3D by multiplication of $\lambda(\tilde{s})$.

6.5.3 Time Evolution

With the preprocessing steps for constructing the elements, nodes, connectivity maps, derivative matrices, Jacobian and outward normal arrays, and initial fields, we can now evolve (6.7) in time. We adopt the same low-storage 4th order Runge-Kutta scheme as given in (5.16). Following the DG approach used to derive (6.20), we now list the steps performed at each Runge-Kutta time stage.

1 - Compute Field Differences

We first compute the jumps between elements of $[\![E_{\tilde{s}}]\!]$, $[\![E_x]\!]$, $[\![E_y]\!]$, $[\![B_{\tilde{s}}]\!]$, $[\![B_x]\!]$, and $[\![B_y]\!]$. This entails using the connectivity map to determine which elements are connected to a given element along each of its 4 faces. For each of the N_{fp} nodes along a face, the difference $[\![u]\!] = u^+ - u^-$ where the $^-$ denotes the value of the field from within the element and $^+$ denotes the value of the field at its corresponding node in the neighboring element along the face. When complete we have generated $[\![E_{\tilde{s}}]\!]$, $[\![E_x]\!]$, $[\![E_y]\!]$, $[\![B_{\tilde{s}}]\!]$, $[\![B_x]\!]$, and $[\![B_y]\!]$ which are each arrays of size $4N_{fp} \times K$; that is, the jump in value for each field along each face, at the N_{fp} nodes on each face, for all K elements.

Lastly, for the face nodes with no corresponding element along the boundary $\partial\Omega$, we set the jump to zero for the moment. That is we set $u^+ \equiv u^-$ if no adjacent element exists. We will impose the boundary conditions next.

2 - Impose Boundary Conditions

In this step we must impose the 3 different boundary conditions: PEC, inflow, and outflow along $\partial\Omega$. We begin by imposing the PEC condition. As we note in section 2.2.2, only (2.50a) is required for the perfectly conducting boundary; (2.50b) is automatically imposed with (2.50a) provided the initial condition satisfies (2.50b); which it does by construction. Thus for face nodes along $x = \pm a$ we set: $[\![E_{\tilde{s}}]\!] = -2E_{\tilde{s}}^-$ and $[\![E_y]\!] = -2E_y^-$, effectively setting the average value for the tangential electric fields to zero. Similarly, for $y = \pm b$ we set: $[\![E_{\tilde{s}}]\!] = -2E_{\tilde{s}}^-$ and $[\![E_x]\!] = -2E_x^-$.

Next, the inflow condition is handled by setting each field's jump by assuming the upwind value is zero since no preexisting fields exist ahead of the source. Specifically, we set $[\![u]\!] = -u^-$ for each of the 6 fields along the inflow wall at $\tilde{s} = \tilde{s}_{\max}$.

Lastly, the outflow condition does not need to be imposed as in the case of an advective transport equation. We simply leave the jumps in the fields along the outflow boundary, at $\tilde{s} = \tilde{s}_{\min}$, as $[\![u]\!] = 0$ which was done automatically in the previous step of computing the field differences. Thus, unless otherwise stated in the PEC or inflow boundary condition, $[\![u]\!] = 0$ for each field not mentioned along $\partial\Omega$.

3 - Set Up Numerical Fluxes

In this step we generate the flux terms from the field differences computed in step 1 of the evolution procedure. To begin, we compute the outward normal field components:

$$\mathbf{n} \cdot [\![\mathbf{E}]\!] = n_{\tilde{s}}[\![E_{\tilde{s}}]\!] + n_x[\![E_x]\!] + n_y[\![E_y]\!], \tag{6.21a}$$

$$\mathbf{n} \cdot [\![\mathbf{B}]\!] = n_{\tilde{s}}[\![B_{\tilde{s}}]\!] + n_x[\![B_x]\!] + n_y[\![B_y]\!]. \tag{6.21b}$$

where $n_{\tilde{s}}$, n_x, and n_y are the outward normal components along the faces computed in preprocessing. Note, the products on the right-hand-sides of (6.21) denote Hadamard matrix multiplication (component-wise products) resulting in output arrays of size $4N_{fp} \times K$; the total number of face nodes over all elements.

Now we set up the individual flux terms for each of the fields. Following the procedure in section 6.4 for $E_{\tilde{s}}$ which led to (6.20), we now list all 6 flux terms:

$$E_{\tilde{s}}^{\mathrm{F}} = c^2 n_x[\![B_y]\!] - c^2 n_y[\![B_x]\!] + \left(1 + \frac{v_g}{c} n_{\tilde{s}}\right)[\![E_{\tilde{s}}]\!] - n_{\tilde{s}}(\mathbf{n} \cdot [\![\mathbf{E}]\!]), \tag{6.22a}$$

$$E_x^{\mathrm{F}} = c^2 n_y[\![B_{\tilde{s}}]\!] - \frac{c^2}{\eta} n_{\tilde{s}}[\![B_y]\!] + \left(1 + \frac{v_g}{c} n_{\tilde{s}}\right)[\![E_x]\!] - n_x(\mathbf{n} \cdot [\![\mathbf{E}]\!]), \tag{6.22b}$$

$$E_y^{\mathrm{F}} = \frac{c^2}{\eta} n_{\tilde{s}}[\![B_x]\!] - c^2 n_x[\![B_{\tilde{s}}]\!] + \left(1 + \frac{v_g}{c} n_{\tilde{s}}\right)[\![E_y]\!] - n_y(\mathbf{n} \cdot [\![\mathbf{E}]\!]), \tag{6.22c}$$

$$B^{\mathrm{F}}_{\tilde{s}} = -n_x[\![E_y]\!] + n_y[\![E_x]\!] + \left(1 + \frac{v_g}{c}n_{\tilde{s}}\right)[\![B_{\tilde{s}}]\!] - n_{\tilde{s}}(\mathbf{n}\cdot[\![\mathbf{B}]\!]), \tag{6.22d}$$
$$B^{\mathrm{F}}_{x} = -n_y[\![E_{\tilde{s}}]\!] + \frac{1}{\eta}n_{\tilde{s}}[\![E_y]\!] + \left(1 + \frac{v_g}{c}n_{\tilde{s}}\right)[\![B_x]\!] - n_x(\mathbf{n}\cdot[\![\mathbf{B}]\!]), \tag{6.22e}$$
$$B^{\mathrm{F}}_{y} = \frac{-1}{\eta}n_{\tilde{s}}[\![E_x]\!] + n_x[\![E_{\tilde{s}}]\!] + \left(1 + \frac{v_g}{c}n_{\tilde{s}}\right)[\![B_y]\!] - n_y(\mathbf{n}\cdot[\![\mathbf{B}]\!]). \tag{6.22f}$$

Note, the terms $(1+(v_g/c)n_{\tilde{s}})$ denote the incrementation of $(v_g/c)n_{\tilde{s}}$ by 1 to each of the $4N_{fp}\times K$ components in the array. Also, the factors of η appearing in (6.22) are also arrays of size $4N_{fp}\times K$ since they contain the value $\eta(s,x)$ at each face node for each element; all multiplication in (6.22) is Hadamard matrix multiplication.

4 - Compute Right-Hand-Side Terms

This step constructs the right-hand-sides of (6.7) incorporating the flux terms as in (6.20). The numerical flux terms in (6.22) are integrated over the boundary of each element which is handled by the surface-to-volume LIFT matrix on page 422 of [11]. The discrete right-hand-sides of (6.7) are:

$$\frac{dE_{\tilde{s}}}{dt} = c^2\mathcal{D}_xB_y - c^2\mathcal{D}_yB_x - cZ_0j_{\tilde{s}} + v_g\mathcal{D}_{\tilde{s}}E_{\tilde{s}} + \mathrm{LIFT}\cdot(E^{\mathrm{F}}_{\tilde{s}}/(2J)), \tag{6.23a}$$
$$\frac{dE_x}{dt} = c^2\mathcal{D}_yB_{\tilde{s}} - \frac{c^2}{\eta}\mathcal{D}_{\tilde{s}}B_y - cZ_0j_x + v_g\mathcal{D}_{\tilde{s}}E_x + \mathrm{LIFT}\cdot(E^{\mathrm{F}}_{x}/(2J)), \tag{6.23b}$$
$$\frac{dE_y}{dt} = \frac{c^2}{\eta}\mathcal{D}_{\tilde{s}}B_x - c^2\mathcal{D}_xB_{\tilde{s}} + \frac{c^2\kappa}{\eta}B_{\tilde{s}} - cZ_0j_y + v_g\mathcal{D}_{\tilde{s}}E_y + \mathrm{LIFT}\cdot(E^{\mathrm{F}}_{y}/(2J)), \tag{6.23c}$$
$$\frac{dB_{\tilde{s}}}{dt} = -\mathcal{D}_xE_y + \mathcal{D}_yE_x + v_g\mathcal{D}_{\tilde{s}}B_{\tilde{s}} + \mathrm{LIFT}\cdot(B^{\mathrm{F}}_{\tilde{s}}/(2J)), \tag{6.23d}$$
$$\frac{dB_x}{dt} = -\mathcal{D}_yE_{\tilde{s}} + \frac{1}{\eta}\mathcal{D}_{\tilde{s}}E_y + v_g\mathcal{D}_{\tilde{s}}B_x + \mathrm{LIFT}\cdot(B^{\mathrm{F}}_{x}/(2J)), \tag{6.23e}$$
$$\frac{dB_y}{dt} = \frac{-1}{\eta}\mathcal{D}_{\tilde{s}}E_x + \mathcal{D}_xE_{\tilde{s}} - \frac{\kappa}{\eta}E_{\tilde{s}} + v_g\mathcal{D}_{\tilde{s}}B_y + \mathrm{LIFT}\cdot(B^{\mathrm{F}}_{y}/(2J)). \tag{6.23f}$$

where the products of the derivative matrices with the fields, such as $\mathcal{D}_xE_y$, are matrix products per element. One could compute $\mathcal{D}_xE_y$ by building a very large block diagonal matrix of size $N_pK\times N_pK$ with $\mathcal{D}_x = diag(\mathcal{D}^k_x)$, and then subsequently multiplying this matrix by E_y reshaped as a $N_pK \times 1$ vector where the entries $1, ..., N_p$ correspond to $k = 1$, and $N_p + 1, ..., 2N_p$ correspond to $k = 2$ and so forth. However, this is an inefficient method; instead we use the reference element maps in the preprocessing stage to use a matrix $\mathcal{D}_x$ for the reference element and compute the quantity element-by-element with array operations which is detailed on pages 423–425 of [11].

Another note is that the $\eta(s, x)$ arrays in (6.23) are Hadamard matrix multiplied and are computed at the 3D nodal values and are of size $N_p \times K$. This is in contrast to the $\eta(s, x)$ arrays in (6.22), which are of size $4N_{fp} \times K$.

5 - Evaluate RK Step

Once the right-hand-sides in (6.23) have been computed, we apply the low-storage Runge-Kutta algorithm from (5.16). The timestep for the scheme is determined by the scale length of the elements and the DG order N. The scale length for an element is the diameter of the inscribed sphere within a tetrahedral element. For a good approximation to this, we use the minimal distance between nodes as the scale size:

$$dt \leq \frac{1}{2c} \cdot \min_k \left(\min_{i,j,\, i \neq j} \sqrt{(\tilde{s}_i^k - \tilde{s}_j^k)^2 + (x_i^k - x_j^k)^2 + (y_i^k - y_j^k)^2} \right). \qquad (6.24)$$

The factor of $1/2c$ is a reasonable CFL constant we have determined experimentally.

With this timestep bound, we compute the necessary number of steps required to reach a desired final time. We repeat steps 1–4 in each of the 5 stages in the RK loop given by (5.16) until the final time.

6.6 Convergence Results

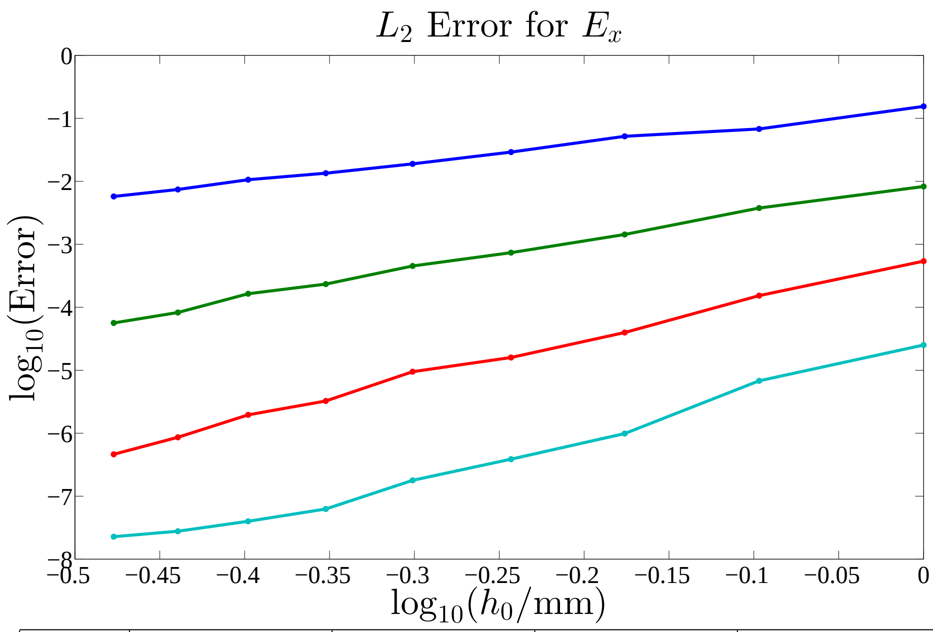
In this last section, we test the convergence of the DG algorithm. We evolve the fields in a bend in a cube: $\Omega = [-2\text{m}, 2\text{m}]^3$ with $v_g = 1.05c$. Specifically, we use the following parameters for our tests:

$$\begin{array}{cccc} s_{\min} = -2\text{m}, & s_{\max} = 2\text{m}, & a = 2\text{m} & b = 2\text{m}, \\ \kappa = 0.02\text{m}^{-1}, & \sigma = 0.50\text{m}, & t_{\max} = 1.0\text{m}/c, & q = 1\text{pC}. \end{array}$$

We partition the domain Ω evenly with $N^{\text{res}} \equiv N_s^{\text{res}} = N_x^{\text{res}} = N_y^{\text{res}}$ cubes leading to a total number of $K = 6(N^{\text{res}})^3$ elements for $N^{\text{res}} = \{5, 7, 9, 11\}$ and consider DG polynomial orders $N = \{2, 4, 6, 8\}$. We compare solutions on a sample grid of 400 points: $(\tilde{s}_i, x_j, 0)$ where $\tilde{s}_i = -1/2 + (i-1)/19$ and $x_j = -1/2 + (j-1)/19$ for $i, j = 1, ..., 20$. This sample grid comprises a planar slice of 20×20 equidistant points in a $1\text{m} \times 1\text{m}$ square in the $\tilde{s}$-x plane. We interpolate all solutions to this sample grid using their Nth order polynomial solution. For solutions occurring on a boundary between elements, we take the average of all adjoining element polynomial solutions. Our reference solution for the tests used $K = 10368$ elements of order $N = 12$.

For our initial condition, we solve Poisson's equation in section 6.5.2 using $K_{2D} = 2304$ triangles of $N_{2D} = 12$ order. We then interpolate this 2D solution on (x, y) to the 3D grid for each (N, K) by multiplying by the factor $\lambda(\tilde{s})$.

Figure 6.3 shows the $N+1$ order spectral convergence of the DG scheme. The highest order examined $N = 8$ appears to have a diminished convergence rate at higher resolutions; we conjecture this is caused by error introduced by the time-stepping method. Future work will check this. The L_2 errors are particularly large by comparison to our work in other chapters; however, the small 3D source used here was not well resolved with fewer elements of lower order. We chose a physically unrealistic source and domain to test our model as a physically accurate and meaningful 3D solution would require a larger computing system. All simulation tests here were computed on our Nvidia GTX Titan using MATLAB scripts partially developed by [11] in under 4 hours total.



K	$N = 2$	$N = 4$	$N = 6$	$N = 8$
384	1.5460e − 01	8.2595e − 03	5.3970e − 04	2.5137e − 05
750	6.7766e − 02	3.7587e − 03	1.5272e − 04	6.7853e − 06
1296	5.1801e − 02	1.4286e − 03	3.9652e − 05	9.8637e − 07
2058	2.9106e − 02	7.3478e − 04	1.5911e − 05	3.8661e − 07
3072	1.8907e − 02	4.5200e − 04	9.4717e − 06	1.7859e − 07
4374	1.3448e − 02	2.3298e − 04	3.2508e − 06	6.2646e − 08
6000	1.0602e − 02	1.6372e − 04	1.9549e − 06	3.9972e − 08
7986	7.4090e − 03	8.2594e − 05	8.6091e − 07	2.7784e − 08
10368	5.7390e − 03	5.6247e − 05	4.6203e − 07	2.2747e − 08
Rate	**2.95**	**4.56**	**6.37**	**6.68**

Figure 6.3: L_2 error for E_x for the 3D common grid at $t = 1.0\text{m}/c$ for order $N = 2$ (blue), $N = 4$ (green), $N = 6$ (red), and $N = 8$ (cyan) and varying number of elements with a corresponding edge length of $h_0 = 4(K/6)^{-1/3}$m.

Chapter 7

Conclusions and Future Work

7.1 Overview of Completed Work

In this final chapter, we discuss the various approaches used to analyze CSR in both the frequency and time domain. Here we summarize the results of the 4 methods investigated from Chapters 3–6 with the benefits and drawbacks we encountered.

7.1.1 Chapter 3 – Summary

Here we studied CSR using a 1D frequency-domain simulation with the Fourier series in y, the Fourier transform in $s - ct$, and the paraxial approximation. We have modeled CSR in a rectangular toroidal bend and achieved results similar to [5] in a fraction of the computation time. This method, using the Fourier series in y, greatly reduces the computational complexity of the system and results in a reasonable approximation to the CSR fields using only a few p-modes.

In our tests, both our FD and DG codes performed comparably in efficiency, i.e. similar errors for similar computation times. The comparison between FD and DG methods enables us to explore newer numerical techniques and make decisions on future codes based on the different methods.

7.1.2 Chapter 4 – Summary

Next we analyzed CSR using a 2D frequency-domain simulation with the paraxial approximation but without the Fourier series in y used in Chapter 3. Our primary intent here was to study the advantages of 2D DG methods over 2D FD methods. While both our DG and FD results are based on MATLAB codes which are not fully optimized, we have observed a significant speed-up using our DG method developed by [11]. This advantage of DG over FD was considerably more pronounced after the implementation of GPU computing into our MATLAB code. In MATLAB, the GPU favors the large dense matrix-matrix product computations of our DG scheme but does not support sparse matrix products used for our FD scheme.

We did not examine higher order finite difference schemes. On rectangular domains, a high order FD method may possibly outperform our DG method. However, our DG method does handle complex geometry better [11] which would be difficult to incorporate into FD methods without degrading the accuracy of the method. One such example is considering a curved domain, where the stair-step boundary contour approximation for the FD method is particularly limiting.

7.1.3 Chapter 5 – Summary

In this chapter we studied the solution of Maxwell's equations in the time-domain to allow for more complex geometries where the paraxial approximation would not be valid. To reduce the computational work involved in a 3D simulation we adopted the use of the Fourier series in y as in Chapter 3. One particular advantage examined in the time domain simulations, is the ability to examine the effects of reflected and scattered waves, unlike in the frequency-domain methods of Chapters 3 and 4.

Importantly, we were able model the larger scale wake field pulses in agreement with experimental data collected at the CLS [4] using our simplified geometry with only the $p = 1$ mode in the Fourier series in y. However, the finer frequency structures observed in the CLS experimental data were not fully captured by our simplified model.

7.1.4 Chapter 6 – Summary

Lastly, we analyzed a different approach for solving Maxwell's equations in the time domain using a 3D DG method. Here we adopted the Galilean transformation to essentially form a traveling grid of nodes which follow the source. The idea behind this approach was to limit the computational domain to a neighborhood of the source to mimic the concept of the paraxial approximation where the fields and source travel together in the same direction.

We successfully demonstrated the stability and spectral convergence of this approach; however, we have not yet examined parameter ranges of interest for accelerator and beam physics. Our goal was to establish a proof of concept for the Galilean transformation in the Frenet-Serret coordinates using DG.

7.2 Conclusions

In conclusion, we have demonstrated the robustness and efficiency of the nodal DG methods outlined in [11] when applied to both the frequency and time domain Maxwell equations in several cases. While we have examined nodal DG methods in comparison to FD, we have not considered other techniques such as continuous finite element methods or modal DG schemes.

Additionally, we computed CSR fields and impedances consistent with [1, 2, 22, 23]. We achieved our goal of utilizing DG to accurately model CSR in an effort to promote awareness of these newer numerical techniques to the accelerator and beam physics community.

7.3 Future Work

In this final section, we outline possible avenues of exploration not addressed in this thesis. In our work on the various topics covered here, we also list some difficulties which also could be improved upon in future work.

For our analysis in Chapter 3, we seek to adjust our algorithms to allow for perturbations in the wall positions $x_{\text{in}}(s)$ and $x_{\text{out}}(s)$ to more accurately describe complex vacuum chambers. This adjustment must be made carefully as to not violate the paraxial approximation. Additionally, we are investigating the use of gauge transformations on the Schrödinger-type equations for better computational results. We also could study the construction of the time domain fields using the inverse Fourier transform in $s - ct$.

Regarding our studies in Chapter 4, we have only considered rectangular domains and have not tested the efficiency of higher order finite difference grids. We also have not implemented GPU compatibility for the finite difference methods since sparse arrays are not natively supported by MATLAB's CUDA library from the Parallel Computing Toolbox at this time. As in the case of Chapter 3, we also have not explored the time domain wake fields from the inverse Fourier transform.

In Chapter 5, we aim to improve upon our simplified geometry of the problem by including a tunnel for the backward port. Furthermore, we have not studied the effects of including more p-modes; however, preliminary tests suggest the higher p-modes do not contribute significantly to the solution away from the reference orbit. Another topic of study involves transcribing the time domain code used here into a lower level computing platform such as Fortran or C++ with parallel computing.

Lastly, for Chapter 6 we would like to consider applying the Fourier series in y as in Chapters 3 and 5 to reduce the 3D problem into 2D while maintaining the Galilean transformation. While the goal here was to establish a proof-of-concept for Maxwell's equations under the Frenet-Serret and Galilean transformations, a full 3D problem with realistic parameters was not feasible on our single GPU-enabled system due to memory constraints. In addition to extending our CLS code in Chapter 5 into other programming languages, we would also like to transcribe our 3D MATLAB code used in Chapter 6 into a lower level programming language, such as C++ or Fortran, to allow for the simulation of larger-scale problems.

Appendix A

DG Code for CLS

A.1 CLS Algorithm Overview

In this section, we will outline every step in a complete simulation for the PDE system defined in the CLS chapter. To begin, we must complete a number of preprocessing routines to generate matrices and operators. Next, the PDEs are evolved in a time-stepping routine to a desired end point. Lastly, the results can be processed to obtain specific information. This approach is broken down into the following steps (code line numbers are in brackets):

1. **Preprocessing routines**
   (a) Define simulation parameters and constants
   (b) Mesh generation
      i. Generate vertices with mesh generation
      ii. Create triangular elements from vertices
   (c) Node and operator generation

   i. Construct Nth order nodal coordinates for reference triangle
   ii. Generate Vandermonde matrix from reference nodal coordinates
   iii. Create derivative matrices from Vandermonde matrix
   iv. Build physical nodes from elements and reference triangle nodes
   v. Construct lift matrix for surface integral terms
   vi. Generate reference element to physical element maps and normals
   vii. Create element-to-element and element-to-face connectivity maps
   viii. Locate physical nodes along edges between elements
   ix. Build node lists for interior and exterior nodes by element

(d) Initialization of fields and sources

   i. Compute stable time-step
   ii. Combine matrix factors in PDE operator for faster computation
   iii. Construct residual storage arrays for fields
   iv. Generate source terms used in PDE
   v. Initialize fields using PDE initial conditions

(e) Optional preprocessing

   i. Set up curvilinear element operators
   ii. Create interpolation matrix to map data to equally spaced nodes
   iii. Locate probe elements for time-series data storage
   iv. Move all necessary matrices and arrays to GPU

2. Timestepping routines

(a) Main Timestep Loop (LSERK)

   i. Calculate current time from step number and RK stage
   ii. Compute jump terms along edge nodes

   iii. Impose boundary conditions on edge nodes along global boundary
   iv. Generate $\mathbf{n} \cdot \mathbf{E}$ and $\mathbf{n} \cdot \mathbf{H}$ terms along edges for fluxes
   v. Construct numerical flux terms for PDE using upwinding
   vi. Compute necessary partial derivatives of fields
   vii. Evaluate right-hand-sides for PDE including sources
   viii. Update residual storage arrays used for LSERK
   ix. Compute fields at next timestep using residual storage arrays

(b) Optional timestepping routines

   i. Process curvilinear elements
   ii. Extract value of fields at desired probe locations for time-series data
   iii. Plot fields every few timesteps for debugging

3. Postprocessing routines

(a) Clean up and save data to workspace file

(b) Optional postprocessing

   i. Interpolate fields to equally spaced nodes
   ii. Slice fields along contours
   iii. Examine frequency content of time-series data
   iv. Other plotting methods

Now we will detail each of the itemized routines. We also will explain the MATLAB implementation for each routine some of which are modeled after the work in *Nodal Discontinuous Galerkin Methods* by J. Hesthaven and T. Warburton [11]. We have implemented several improvements from those MATLAB codes including: implementation of GPU array objects, removal of global variable dependencies, decomposition of structure array objects, and preallocation of memory for faster preprocessing.

A.2 Preprocessing Routines

A.2.1 Parameter Definitions

First, we must define or input the constants and parameters to be used in the simulation. This code block includes all the user specified data and some quick computations. The mesh geometry details are not included here but are located in a separate routine. The units of each quantity are listed in the comment blocks in parenthesis.

```
% Primary input parameters
x_in = -0.032;      % (m) Initial distance of source to inner wall
x_out = 0.078;      % (m) Initial distance of source to outer wall
x_max = 0.330;      % (m) Maximal distance of source to outer wall (est)
s_min = -0.10;      % (m) Starting region before bend
s_max = 3.500;      % (m) Length of computational domain
h = 0.032;          % (m) Height of chamber
R = 7.143;          % (m) Radius of curvature
q = 1e-12;          % (C) Charge of source (assumed negative)
c = 299792458;      % (m/s) Speed of light
Z0 = 376.730313;    % (ohms) Impedance of free space
p = 1;              % ( ) Mode number for Fourier series in y
alpha = pi*p/h;     % (m^-1) Wave number for Fourier series in y
tau_max = 3.500;    % (s) Final time of simulation
```

Here, it is important to note that `t_max` should be no larger than `s_max` to avoid issues near the boundary. If a longer time is desired, or the geometry produces reflections sooner, an increased `s_min` and `s_max` may be required. Also, the code is currently set up to handle a single y Fourier-mode p but if a sum over p is desired, a loop can be made over p. The caveat however is that a new grid may need to be constructed to resolve higher order p-modes requiring the entire code to be run again.

Next, the source is defined in terms of its longitudinal and vertical distributions in s and y. We opt to use the simplified formula for G_p if the vertical source distribution $G(y)$ is very narrow (i.e. $\sigma_y = 0$); this is a good approximation for small p modes; for this case, G_p is constant. For $\sigma_y \neq 0$ use $G_p = (-1)^{(p-1)/2}(2/h)\exp(-(\alpha_p\sigma_y)^2/2)$.

```
% Logitudinal source distribution in s
sig_s = 0.002;      %(m) Standard deviation of source width in s
t_delay = 0.05;     %(s) Temporal delay for source (scalar shift)
f1 = 1/(sig_s*sqrt(2*pi));
f2 = 2*sig_s^2;
f3 = f1/(-sig_s^2);
lambda = @(smt) f1*exp(-(smt+t_delay).^2/f2);
lambdap = @(smt) f3*(smt+t_delay).*exp(-(smt+t_delay).^2/f2);

% Vertical source distribution in y
Gp = 2/h*(-1)^((p-1)/2);   %(m^-1) Fourier coefficient for G(y) mode p
```

The functions `lambda` and `lambdap` refer to the longitudinal distributions $\lambda(s)$ and $\lambda'(s)$ respectively (assuming a Gaussian bunch). The temporal delay `t_delay` is used to offset the source such that it is effectively compactly supported in the region $s_{\min} < s < 0$; a positive offset must be used and is recommended to be at least twice the bunch length `sig_s`. The intermediate function stages `f1`, `f2`, and `f3` are used to eliminate redundant computations.

Lastly, we include some additional parameters used for the numerical algorithm. Here we define the order N to be used on all elements and also specify several option flags such as enabling GPU computing and plotting routines.

```
% Additional code components
h0 = 0.005;           % (m) Scaled edge length for resolution
N = 6;                % ( ) Order of approximation (for DG method)
Nout = 12;            % ( ) Order of output nodes for plotting
Np = (N+1)*(N+2)/2;   % ( ) Number of nodes per element
Nfp = N+1;            % ( ) Number of nodes per edge
Ntol = 1e-12;         % ( ) Tolerance in locating nodes along boundaries
alp = 1;              % ( ) Denotes flux type (1 = upwind, 0 = central)
```

The quantity `h0` is used to define a rough element size which should be small enough to resolve the source bunch length. A safe choice for this parameter for up to moderate orders $N \lesssim 20$ is $\texttt{h0} \leq N \cdot \texttt{sig\_s}/5$ which ensures the Gaussian source is resolved well enough. The parameter `Nout` is used only for plotting; this number can be greater than N to produce smoother upscaled resolution images.

A.2.2 Mesh Generation

Next in the preprocessing chain is the mesh generation routines. We opt to use the distmesh subroutines to generate the triangular mesh. These routines were developed by Per-Olof Persson (UC Berkeley) and Gilbert Strang (MIT) [17]. The desired output of these routines are: total number of elements K, total number of vertices `Nv`, s-coordinates of vertices `VS`, x-coordinates of vertices `VX`, and the connectivity matrix of elements `EToV`. The usage of the routines for our application is as follows:

```
% Define boundary corner locations
s1 = 1.832632366508338;    x1 = 0.322357941766471;
s2 = 1.882483505139345;    x2 = 0.129081490102798;
s3 = 1.942483505139345;    x3 = 0.129081490102798;
```

```
s4 = 1.950922137260992;    x4 = 0.322357941766471;
s5 = 2.082038177552993;    x5 = 0.322357941766471;
s6 = 2.122502618239870;    x6 = 0.015000000000000;

% Define geometry by signed distance function
fd = @(p) max([-((p(:,1) < 1e-12).*x_out + ...
(1e-12 <= p(:,1) & p(:,1) < s1).* ...
((R+x_out)./cos(p(:,1)/R)-R) + ...
(s1 <= p(:,1) & p(:,1) < s2).* ...
((R+x_out)*tan(s1/R)./sin(p(:,1)/R)-R) + ...
(s2 <= p(:,1) & p(:,1) < s3).*x2 + ...
(s3 <= p(:,1) & p(:,1) < s4).* ...
(x3 + (p(:,1)-s3)/tand(2.5)) + ...
(s4 <= p(:,1) & p(:,1) < s5).*x4 + ...
(s5 <= p(:,1) & p(:,1) < s6).* ...
(x5 - (p(:,1)-s5)/tand(7.5)) + ...
(s6 <= p(:,1)).*x6) + ...
p(:,2),s_min-p(:,1),-s_max+p(:,1),x_in-p(:,2)],[],2);

fh = @huniform;    %Use uniform vertex spacing weights
bbox = [s_min,x_in;s_max,x_max];    %Define global domain box
```

Here, `fd` defines an unnormalized signed distance function (in the format required for distmesh) from the boundary of the domain which incorporates the complex outer boundary $x_{\text{out}}(s)$ and the buffer regions extending for $s < s_0 = 0$ and $s > s_6$. While evaluation speed of this function is not optimized for extensive use, in this preprocessing routine it performs adequately. Next, the function `fh` defines the element weighting function for which we choose uniform weights. And lastly, `bbox` defines a bounding box for the domain used by distmesh to restrict where to generate elements.

```
% Specify fixed vertices required for conformal boundary
hline1_s = linspace(s_min,0,ceil(-s_min/h0)+1)';
hline1_x = 0*ones(size(hline1_s));
hline2_s = linspace(0,s2,ceil(s2/h0)+1)';
hline2_x = 0*ones(size(hline2_s));
hline3_s = linspace(s2,s_max,ceil((s_max-s2)/h0)+1)';
hline3_x = 0*ones(size(hline3_s));

vline1m_x = linspace(x_in,0,ceil(-x_in/h0)+2)';
vline1m_s = 0*ones(size(vline1m_x));
vline1p_x = linspace(0,x_out,ceil(x_out/h0)+3)';
vline1p_s = 0*ones(size(vline1p_x));
vline2m_x = linspace(x_in,0,ceil(-x_in/h0)+2)';
vline2m_s = s2*ones(size(vline2m_x));
vline2p_x = linspace(0,x2,ceil(x2/h0)+3)';
vline2p_s = s2*ones(size(vline2p_x));

ffix = [s_min,x_in;s_min,x_out;s_max,x_in;s_max,x7;...
s1,x1;s2,x2;s3,x3;s4,x4;s5,x5;s7,x7];
ffix = [ffix ;...
hline1_s, hline1_x; hline2_s, hline2_x; hline3_s, hline3_x; ...
vline1m_s, vline1m_x;  vline1p_s, vline1p_x; ...
vline2m_s, vline2m_x; vline2p_s, vline2p_x];
```

This code block defines a list of vertices which must be included in the triangulation of the domain. The `hline` vectors define vertices along the source trajectory at $x = 0$ while maintaining a node exists at $s = 0$ and $s = s_2$, `vline1` vectors define vertices along the interface at $s = 0$, and the `vline2` vectors define vertices along the interface at $s = s_2$. The other vertices include all corners of the domain; the $(s_1 - h_0/2, x_1 - 2h_0/3)$ coordinate pair is an ad hoc method for ensuring that the no single element contains boundary edges along both $0 < s < s_1$ and $s_1 < s < s_2$. These are necessary to ensure that the fields in the constructed elements are continuous, i.e. $\kappa(s)$ and $\Theta(x)$ are constant in each element.

```
[P,EToV] = distmesh2d(fd,fh,h0,bbox,ffix);
Nv = size(P,1);      % Number of vertices
VS = P(:,1)';        % Locations of vertices in s
VX = P(:,2)';        % Locations of vertices in x
K = size(EToV,1);    % Number of elements
```

Lastly, we generate the mesh and extract the required parameters needed to proceed in the preprocessing chain. The vertices along the outer wall $x_{\text{out}}(s)$ may need to be adjusted slightly to exactly conform to the boundary; distmesh does not guarantee boundary vertices are located on the exact boundary. To fix this issue, the following code adjusts the locations:

```
wall = @(s) x_out.*(s<0) + ...
    ((R+x_out)./cos(s/R)-R).*((0<=s)&(s<s1)) + ...
    ((R+x_out)*tan(s1/R)./sin(s/R)-R).*((s1<=s)&(s<s2)) + ...
    x2.*(s>=s2);

% Locate vertices along curivilinear edges of domain and move VX locations
vflag = find((VS>=-h0/10 & VS<=(s2+h0/10) & abs(wall(VS)-VX)<h0/10));
VX(vflag) = wall(VS(vflag));
```

These steps may require up to a few minutes but the 5 quantities: `K`, `Nv`, `VS`, `VX`, and `EToV`, may be saved for future simulations if the mesh doesn't need to be modified. Additionally, we must treat the curvilinear elements located along the outer boundary $x_{\text{out}}(s)$ for $0 < s < s_2$ differently to ensure the entire method's order of accuracy is not limited by the boundary geometry. Numerical evidence suggests that this correction does not show an improvement in error unless high orders on coarse meshes are used or the domain contains more strongly curved boundaries. These extra steps will be listed in the optional preprocessing section.

A.2.3 DG Operator Setup

The next series of subroutines are inspired by the setup structure given in chapters 3 and 6 of *Nodal Discontinuous Galerkin Methods* by J. Hesthaven and T. Warburton. The codes developed in the book use global variables and a large number of function files. We condense some function files into larger subroutines for efficiency and defer from using global variables to avoid compatibility issues if our code is nested in another environment.

The first step is to compute the nodal locations on a reference triangle for the desired order of the method. This can be accomplished with the function files `Nodes2D.m`, `xytors.m`, `Warpfactor.m`, `JacobiP.m`, `JacobiGQ.m`, and `JacobiGL.m` from NUDG.org. We condense these into 4 subroutines since several routines are only referenced once.

```
function [r,t] = DG_Nodes(N,Np)
% This subroutine generates the reference element nodal corrdinates given
% the order of the elements to be used. This subroutine is based on
% Nodes2D.m and xytors.m from nudg.org.
%
%  Inputs:
%  N  = order of elements (1 x 1)
%  Np = number of nodes per element (1 x 1)
%
%  Outputs:
%  r = reference element coordinate 1 (Np x 1)
%  t = reference element coordinate 2 (Np x 1)

alpopt = [0.0000 0.0000 1.4152 0.1001 0.2751 0.9800 1.0999 1.2832 ...
          1.3648 1.4773 1.4959 1.5743 1.5770 1.6223 1.6258 1.6667];

% Set optimized parameter, alpha, depending on order N
a = alpopt(min(N,16));

% Create equidistributed nodes on an equilateral triangle
L1 = zeros(Np,1);    L2 = zeros(Np,1);    L3 = zeros(Np,1);
count = 0;
for i = 1:N+1
    for j = 1:N+2-i
        count = count + 1;
        L1(count) = (i-1)/N;
        L3(count) = (j-1)/N;
    end
end
L2 = 1-L1-L3;
r = -L2+L3;                 % Undeformed coordinates of x1
t = (-L2-L3+2*L1)/sqrt(3);  % Undeformed coordinates of x2

% Compute blending function for each node per edge
b1 = 4*L2.*L3;          b2 = 4*L1.*L3;          b3 = 4*L1.*L2;

% Compute warping scale factors
```

```
w1 = Ewarp(N,Np,L3-L2); w2 = Ewarp(N,Np,L1-L3); w3 = Ewarp(N,Np,L2-L1);

% Combine blend and warping
w1 = b1.*w1.*(1 + (a*L1).^2);
w2 = b2.*w2.*(1 + (a*L2).^2);
w3 = b3.*w3.*(1 + (a*L3).^2);

r = r + w1 - w2/2 - w3/2;
t = t + w2*sqrt(3)/2 - w3*sqrt(3)/2;

% Map equilateral triangle to reference triangle
L1 = (sqrt(3)*t+1)/3;
L2 = (-3*r - sqrt(3)*t + 2)/6;
L3 = (3*r - sqrt(3)*t + 2)/6;

r = -L2 + L3 - L1;
t = -L2 - L3 + L1;

end % End of DG_Nodes subroutine
```

Essentially, our DG_Nodes routine merges Nodes2D.m and xytors.m in one combination. The xytors.m part is the mapping the equilateral triangle to the reference triangle: the triangle generated from the vertices $(-1,1)$, $(-1,1)$, and $(1,-1)$. The Ewarp routine, referenced 3 times in DG_Nodes, is defined next.

```
function [W] = Ewarp(N,Np,r)
%This subroutine computes the 1D warping function for a given edge. This
%subroutine is based on Warpfactor.m from nudg.org.
%
```

```
%  Inputs:
%  N  = order of elements (1 x 1)
%  Np = number of nodes per element (1 x 1)
%  r  = reference element coodinate (Np x 1)
%
%  Outputs:
%  W = warping function for given blended nodes (Np x 1)

gX = JGL(N);                  % Function for Gauss-Lobatto quadrature
eqX = linspace(-1,1,N+1)';    % Equidistant nodes on [-1,1]
eqV = zeros(N+1);
P = zeros(N+1,Np);
for i = 1:N+1
    eqV(:,i) = JP(eqX(:),0,0,i-1);   % Evaluate Jacobi Polynomial
    P(i,:) = JP(r(:),0,0,i-1)';
end

L = eqV'\P;             % Lagrange polynomial evaluated at r
W = L'*(gX - eqX);      % Warp factor

% Scale the warp factor (remove zeros from scaling)
zf = (abs(r)<1-1e-12);  % Index of zeros
sf = 1 - (zf.*r).^2;    % Scaling for zeros
W = W./sf + W.*(zf-1);  % Rescale warpfactor

end % End of EWarp subroutine
```

The two routines referenced in `Ewarp` are `JGL` and `JP` which are closely modeled after `JacobiP.m`, `JacobiGQ.m`, and `JacobiGL.m`. These routines are used to compute

Jacobi polynomials $P_n^{(\alpha,\beta)}(x)$ and the Gauss-Lobatto node locations. The interior Gauss-Lobatto nodes are found from the $N-1$ roots r_i of $P_{N-2}^{(\alpha+1,\beta+1)}(r) = 0$; the full set of nodes includes the two endpoints $r_1 = -1$, $r_{N+1} = 1$. This is done in the `JGL` and `JP` routines.

```
function [x] = JGL(N)
% This subroutine computes the Nth order Legandre-Gauss-Lobatto quadrature
% points associated with the Jacobi polynomial with alpha = beta = 0. This
% subroutine is based on JacobiGL.m and JacobiGQ.m from nudg.org.
%
%  Inputs:
%  N = order of elements (1 x 1)
%
%  Outputs:
%  x = quadrature point locations in [-1,1] (N+1 x 1)

if N<=2
    x = linspace(-1,1,N+1)';    % Equidistant nodes for N <= 2
else
    h1 = linspace(2,2*N-4,N-2);
    h2 = linspace(1,N-2,N-2);
    J = diag(2./(h1+2).*sqrt(h2.*(h2+2).*(h2+1).*(h2+1)./ ...
        (h1+1)./(h1+3)),1);
    J = J + J';
    [~,D] = eig(J);

    x = [-1.0 diag(D)' 1.0]';   % Compute nodal locations for N > 2
end
```

```
end % End of JGL subroutine
```

```
function [P] = JP(x,a,b,N)
% This subroutine computes the Jacobi polynomial of order N at points x
% with the parameters a and b. The polynomials are orthonormal. This
% subroutine is based on JacobiP.m from nudg.org.
%
%  Inputs:
%  x = vector of points to evaluate
%  a = parameter alpha (1 x 1)
%  b = parameter beta (1 x 1)
%  N = order of polynomial (1 x 1)
%
%  Outputs:
%  P = polynomial evaluated at x (length(x) x 1)

if size(x,2) == 1   % Form x into row vector if not already
    x = x';
end

PL = zeros(N+1,length(x));

% First iteration of polynomial
g0 = 2^(a+b+1)/(a+b+1)*gamma(a+1)*gamma(b+1)/gamma(a+b+1);
PL(1,:) = 1/sqrt(g0);
if N == 0
    P = PL';
    return
end
```

```
% Second iteration of polynomial
g1 = (a+1)*(b+1)/(a+b+3)*g0;
PL(2,:) = ((a+b+2)*x/2 + (a-b)/2)/sqrt(g1);
if N == 1
    P = PL(2,:)';
    return
end

% Recursive iteration of polynomial
a0 = 2/(2+a+b)*sqrt((a+1)*(b+1)/(a+b+3));
for i = 1:N-1
    b0 = 2*i+a+b;
    a1 = 2/(b0+2)*sqrt((i+1)*(i+1+a+b)*(i+1+a)*(i+1+b)/((b0+1)*(b0+3)));
    b1 = -(a^2-b^2)/(b0*(b0+2));
    PL(i+2,:) = 1/a1*(-a0*PL(i,:)+(x-b1).*PL(i+1,:));
    a0 = a1;
end

P = PL(N+1,:)';    % Output polynomial

end % End of JP subroutine
```

Now that the `r` and `t` reference coordinates are constructed with `DG_Nodes` the next step is to generate the Vandermonde matrix. This matrix generates the nodal weights associated with modal representation of the Jacobi polynomials on the reference triangle. For the implementation here in the preprocessing, the matrix `V` is a square matrix of size $N_p \times N_p$ but this subroutine will be used for interpolation later which may generate non-square matrices. Our implementation combines the `Vandermonde2D.m`, `rstoab.m`, and `Simplex2DP.m` routines from NUDG.org.

```
function [V] = DG_Vandermonde(N,Np,r,t)
% This subroutine computes the Vandermonde matrix associated with the LGL
% optimized warped nodes. This subroutine is based on Vandermonde2D.m,
% rstoab.m, and Simplex2DP from nudg.org.
%
%  Inputs:
%  N  = order of elements (1 x 1)
%  Np = number of nodes per element (1 x 1)
%  r  = reference element coordinate 1 (Np x 1)
%  t  = reference element coordinate 2 (Np x 1)
%
%  Outputs:
%  V = Vandermonde matrix (Np x Np)

V = zeros(length(r),Np);

% Map reference triangle to standard triangle coordinates
a = zeros(length(r),1);
for i = 1:Np
        if (t(i) ~= 1)
            a(i) = 2*(1+r(i))/(1-t(i))-1;
    else
        a(i) = -1;
        end
end

% Fill-in columns of Vandermonde matrix
count = 0;
for i = 0:N
```

```
    for j = 0:N-i
        count = count + 1;
        h1 = JP(a,0,0,i);
        h2 = JP(t,2*i+1,0,j);
        V(:,count) = sqrt(2)*h1.*h2.*(1-t).^i;
    end
end


end % End of DG_Vandermonde subroutine
```

Next we construct the collocation derivative matrices using the reference triangle nodes and Vandermonde matrix. The matrices `Dr` and `Dt` denote the partial derivative operators along the two reference coordinates `r` and `t`. The procedure is modeled after `Dmatrices2D.m`, `rstoab.m`, `GradVendermonde2D`, and `GradSimplex2DP`

```
function [Dr,Dt] = DG_Collocation(N,Np,r,t,V)
% This subroutine computes the collocation derivative matrices from the
% reference element nodes and the Vandermonde matrix. This subroutine is
% based on Dmatrices.m, rstoab.m, GradVandermonde2D.m, and
% GradSimplex2DP.m from nudg.org.
%
%  Inputs:
%  N  = order of elements (1 x 1)
%  Np = number of nodes per element (1 x 1)
%  r  = reference element coordinate 1 (Np x 1)
%  t  = reference element coordinate 2 (Np x 1)
%  V  = Vandermonde matrix (Np x Np)
%
%  Outputs:
```

```
%  Dr = collocation derivative matrix in r (Np x Np)
%  Dt = collocation derivative matrix in s (Np x Np)

Dr = zeros(Np);     Dt = zeros(Np);

% Map reference triangle to standard triangle coordinates
a = zeros(length(r),1);
for i = 1:Np
        if (t(i) ~= 1)
            a(i) = 2*(1+r(i))/(1-t(i))-1;
    else
        a(i) = -1;
        end
end

count = 0;
for i = 0:N
    for j = 0:N-i
        count = count+1;
        h1 = JP(a,0,0,i);            dh1 = GJP(a,0,0,i);
        h2 = JP(t,2*i+1,0,j);        dh2 = GJP(t,2*i+1,0,j);

        % Compute modal derivative in r by column
        V2Dr = dh1.*h2;
        if i > 0;
            V2Dr = V2Dr.*((0.5*(1-t)).^(i-1));
        end

        % Compute modal derivative in t by column
        V2Dt = dh1.*(h2.*(0.5*(1+a)));
        temp = dh2.*((0.5*(1-t)).^i);
        if i > 0;
            V2Dt = V2Dt.*((0.5*(1-t)).^(i-1));
            temp = temp - 0.5*i*h2.*((0.5*(1-t)).^(i-1));
        end
        V2Dt = V2Dt + h1.*temp;

        % Normalize vectors and insert into matrices
        Dr(:,count) = 2^(i+0.5)*V2Dr;
        Dt(:,count) = 2^(i+0.5)*V2Dt;
    end
end

% Invert by Vandermonde matrix to obtain collocation derivative matrices
Dr = Dr/V;      Dt = Dt/V;

end % End of DG_Collocation subroutine
```

The next step is to flag the boundary nodes on each element. This is done by locating the nodes on each element which are along each edge of the triangle. Each of the 3 columns of `Fm` list the local node numbers along each edge. The routine is based on a small subsection of `StartUp2D.m` from NUDG.org.

```
function [Fm] = DG_Mask(r,t,Ntol)
% This subroutine locates edge nodes on each element. This subroutine is
% based on a section of StartUp2D.m from nudg.org.
%
%  Inputs:
%  r    = reference element coordinate 1 (Np x 1)
%  s    = reference element coordinate 2 (Np x 1)
%  Ntol = specified tolerance for node locating (1 x 1)
%
%  Outputs:
%  Fm = element array of edge indices (Nfp x 3)

fm1 = find(abs(t+1) < Ntol)';
fm2 = find(abs(r+t) < Ntol)';
fm3 = find(abs(r+1) < Ntol)';

% Concatenate face node ids
Fm = [fm1;fm2;fm3]';

end % End of DG_Mask subroutine
```

Another step is to generate the “lifting” matrix in the DG formulation. This matrix maps information of an entire element to the edge nodes. This will be used for the boundary integrals involving the numerical flux and jumps between the elements. Our routine is the same as the NUDG.org version: `Lift2D`.

```
function [Lift] = DG_Lift(N,Np,Nfp,r,t,Fm,V)
% This subroutine computes the area to edge "Lift" operator used in the
% DG formulation. This subroutine is based on Lift2D.m from nudg.org.
%
%  Inputs:
%  N   = order of elements (1 x 1)
%  Np  = number of nodes per element (1 x 1)
%  Nfp = number of nodes per edge (1 x 1)
%  r   = reference element coordinate 1 (Np x 1)
%  t   = reference element coordinate 2 (Np x 1)
%  Fm  = element array of face indices (Nfp x 3)
%  V   = Vandermonde matrix (Np x Np)
%
%  Outputs:
%  Lift = edge integral Lifting matrix (Np x 3*Nfp)

Emat = zeros(Np,3*Nfp);

% Lift face 1
fr = r(Fm(:,1));
V1D = zeros(length(fr),N+1);
for i = 1:N+1
    V1D(:,i) = JP(fr(:),0,0,i-1);
end
Emat(Fm(:,1),1:Nfp) = inv(V1D*V1D');

% Lift face 2
fr = r(Fm(:,2));
V1D = zeros(length(fr),N+1);
for i = 1:N+1
    V1D(:,i) = JP(fr(:),0,0,i-1);
end
Emat(Fm(:,2),Nfp+1:2*Nfp) = inv(V1D*V1D');

% Lift face 3
fr = t(Fm(:,3));
V1D = zeros(length(fr),N+1);
for i = 1:N+1
    V1D(:,i) = JP(fr(:),0,0,i-1);
end
```

```
Emat(Fm(:,3),2*Nfp+1:3*Nfp) = inv(V1D*V1D');


% Invert by the 2D mass matrix
Lift = V*(V'*Emat);


end % End of DG_Lift subroutine
```

Next, we construct the reference to physical element maps which are used to extend all the single element operations to the global triangulation. Also, we construct the normal vectors which point outward along each edge of the triangles. The construction combines the routines: `GeometricFactors2D.m`, `Normals2D.m`, and a portion of `StartUp2D.m` from NUDG.org.

```
function [rs,rx,ts,tx,J,ns,nx,Fs] = DG_GF2D(s,x,Dr,Dt,Nfp,K,Fm)
% This subroutine generates the metric elements for the local mappings of
% the elements, the outward pointing normals at element faces, and the
% inverse surface metric Fs. This subroutine is based on
% GeometricFactors2D.m, Normals2D.m, and a section of StartUp2D.m from
% nudg.org.
%
%  Inputs:
%  x   = matrix of physical x locations by element (Np x K)
%  y   = matrix of physical y locations by element (Np x K)
%  Dr  = collocation derivative matrix in r (Np x Np)
%  Dt  = collocation derivative matrix in t (Np x Np)
%  Nfp = number of nodes per edge (1 x 1)
%  K   = number of elements (1 x 1)
%  Fm  = element array of edge indices (Nfp x 3)
%
```

```
%  Outputs:
%  rs,rx = metric constants for r along x,y (Np x K)
%  ts,tx = metric constants for t along x,y (Np x K)
%  J     = Jacobian matrix for r,t,s,x (Np x K)
%  ns,nx = s,x outward normal components on edges (3*Nfp x K)
%  Fs    = ratio of edge to area Jacobian (3*Nfp x K)

% Generate collocation partial derivatives
sr = Dr*s;      st = Dt*s;
xr = Dr*x;      xt = Dt*x;

% Compute Jacobian matrix
J = sr.*xt - st.*xr;

% Invert coordinate mappings
rs =  xt./J;
rx = -st./J;
ts = -xr./J;
tx =  sr./J;

% Extract only edge node metrics
fsr = sr(Fm,:);         fss = st(Fm,:);
fxr = xr(Fm,:);         fxs = xt(Fm,:);

% Create reference ids by edges
ns = zeros(3*Nfp,K);    nx = zeros(3*Nfp,K);
fid1 = (1:Nfp)';
fid2 = (Nfp+1:2*Nfp)';
fid3 = (2*Nfp+1:3*Nfp)';
```

```
% Compute normals for edge 1
ns(fid1,:) = fxr(fid1,:);
nx(fid1,:) = -fsr(fid1,:);

% Compute normals for edge 2
ns(fid2,:) = fxs(fid2,:) - fxr(fid2,:);
nx(fid2,:) = fsr(fid2,:) - fss(fid2,:);

% Compute normals for edge 3
ns(fid3,:) = -fxs(fid3,:);
nx(fid3,:) = fss(fid3,:);

% Generate edge to area Jacobian map
sJ = sqrt(ns.*ns+nx.*nx);
ns = ns./sJ;    nx = nx./sJ;

Fs = sJ./(J(Fm,:));

end % End of DG_GF2D subroutine
```

To connect the elements together, we introduce the connectivity matrices `EToE` and `EToF`. The index arrays list the which elements are adjacent to which elements and which edges correspond to those elements. This routine is modeled after `tiConnect2D.m` from NUDG.org.

```
function [EToE,EToF] = DG_Connect(EToV,Nv,K)
% This subroutine builds the element-to-element and element-to-edge maps to
% be used for numerical fluxes between elements. This subroutine is based
% on tiConnect2D.m from nudg.org.
%
```

```
%  Inputs:
%  EToV = element-to-vertex mapping matrix (K x 3)
%  Nv   = number of vertices (1 x 1)
%  K    = number of elements (1 x 1)
%
%  Outputs:
%  EToE = element-to-element mapping matrix (K x 3)
%  EToF = element-to-edge mapping matrix (K x 3)

% Extract the edge vertices of each edge of each triangle
fn = [EToV(:,[1,2]);EToV(:,[2,3]);EToV(:,[3,1])];
fn = sort(fn,2)-1;

EToE = (1:K)'*ones(1,3);
EToF = ones(K,1)*(1:3);

id = fn(:,1)*Nv + fn(:,2) + 1;
ntn = [id, (1:3*K)', EToE(:), EToF(:)];
ntn = sortrows(ntn,1);

% Locate indices of common edges among elements
[ind,~] = find(ntn(1:(end-1),1) == ntn(2:end,1));

L = [ntn(ind,:); ntn(ind+1,:)];
R = [ntn(ind+1,:); ntn(ind,:)];

EToE(L(:,2)) = R(:,3);
EToF(L(:,2)) = R(:,4);
```

```
end % End of DG_Connect subroutine
```

The next step is to create node index arrays for each element. These arrays contain information on which nodes within an element correspond to which nodes on neighboring elements. The routine also generates a list of the nodes located along the global boundary. This routine is based on the routine `BuildMaps2D.m` from NUDG.org.

```
function [vM,vP,vB,mB] =DG_Maps(s,x,VS,VX,Fm,EToE,EToF,EToV,Np,Nfp,K,Ntol)
% This subroutine generates the maps identifying local and global boundary
% nodes. This subroutine is based on BuildMaps2D.m from nudg.org.
%
%  Inputs:
%  s    = matrix of physical s locations by element (Np x K)
%  x    = matrix of physical x locations by element (Np x K)
%  VS   = vector of s-coordinates of vertices (1 x Nv)
%  VX   = vector of x-coordinates of vertices (1 x Nv)
%  Fm   = element array of edge indices (Nfp x 3)
%  EToE = element to element mapping matrix (K x 3)
%  EToF = element to edge mapping matrix (K x 3)
%  EToV = element to vertex mapping matrix (K x 3)
%  Np   = number of nodes per element (1 x 1)
%  Nfp  = number of nodes per edge (1 x 1)
%  K    = number of elements (1 x 1)
%  Ntol = specified tolerance for node locating (1 x 1)
%
%  Outputs:
%  vM = vector of global nodal numbers on edges at interiors (3*K*Nfp x 1)
%  vP = vector of global nodal numbers on edges at exteriors (3*K*Nfp x 1)
%  vB = vector of global nodal numbers on boundary edges
%  mB = vector of global edge nodal numbers on boudary edges

id = reshape(1:Np*K,Np,K);
vM = zeros(Nfp,3,K);
vP = zeros(Nfp,3,K);
mM = (1:3*Nfp*K)';
mP = reshape(mM,Nfp,3,K);

for k1 = 1:K
    for n1 = 1:3
        vM(:,n1,k1) = id(Fm(:,n1),k1);
    end
end

t1 = ones(1,Nfp);
for k1 = 1:K
    for n1 = 1:3
        % Locate neighboring element
        k2 = EToE(k1,n1);     n2 = EToF(k1,n1);
        v1 = EToV(k1,n1);     v2 = EToV(k1,1+mod(n1,3));
        rd = sqrt((VS(v1)-VS(v2))^2 + (VX(v1)-VX(v2))^2);

        vidM = vM(:,n1,k1);   vidP = vM(:,n2,k2);
        x1 = s(vidM)*t1;      y1 = x(vidM)*t1;
        x2 = s(vidP)*t1;      y2 = x(vidP)*t1;

        % Define distance from interior and exterior nodes
        D = (x1-x2').^2 + (y1-y2').^2;
        % Locate nodal coordinates common to multiple elements
        [idM,idP] = find(sqrt(abs(D)) < Ntol*rd);
        vP(idM,n1,k1) = vidP(idP);
        mP(idM,n1,k1) = idP + (n2-1)*Nfp + (k2-1)*3*Nfp;
    end
end

% Reshape vM, vP, and mP to be vectors and locate boundary nodes
vM = vM(:);     vP = vP(:);     mP = mP(:);
mB = find(vP==vM);
vB = vM(mB);

end % End of DG_Maps subroutine
```

This concludes the section on DG operator construction and nodal generation from the triangular elements.

A.2.4 Initialization of fields and sources

The first subroutine here calculates a stable timestep and number of timesteps needed for the simulation by finding the minimal nodal distance among all elements and the area of the smallest element. The approach follows from `dtscale2D.m`, `JacobiGQ.m`, and a portion of `Maxwell2D.m` from NUDG.org.

```
function [Nsteps,dtau] = DG_Timestep(N,r,t,s,x,tau_max,Ntol)
% This subroutine computes the number of timesteps required to maintain a
% stable scheme. This subroutine is based on a section of JacobiGQ.m,
% Maxwell2D.m, and dtscale2D.m from nudg.org.
%
```

```
%  Inputs:
%  N       = order of elements (1 x 1)
%  r,t     = reference element coordinate 1,2 (Np x 1)
%  s,x     = matrix of physical s,x locations by element (Np x K)
%  tau_max = final simulation time (1 x 1)
%  Ntol    = specified tolerance for node locating (1 x 1)
%
%  Outputs:
%  Nsteps = number of timesteps to complete evolution (1 x 1)
%  dtau   = s-length per timestep (1 x 1)

% Construct standard nodal coordinates
if N == 0
    rGQ = 0;
else
    h1 = 2*(0:N-1);
    JGQ = diag(2*(1:N).^2./(h1+2)./sqrt((h1+1).*(h1+3)),1);
    JGQ(1,1) = 0;
    JGQ = JGQ + JGQ';
    [~,D] = eig(JGQ);
    rGQ = diag(D);
end
rmin = abs(rGQ(1)-rGQ(2));

% Determine element scale size
vm1 = find(abs(t+r+2)<Ntol);
vm2 = find(abs(r-1)<Ntol);
vm3 = find(abs(t-1)<Ntol);
vm = [vm1,vm2,vm3];
vs = s(vm(:),:);    vx = x(vm(:),:);
```

```
% Find area of triangular element
L1 = sqrt((vs(1,:)-vs(2,:)).^2+(vx(1,:)-vx(2,:)).^2);
L2 = sqrt((vs(2,:)-vs(3,:)).^2+(vx(2,:)-vx(3,:)).^2);
L3 = sqrt((vs(3,:)-vs(1,:)).^2+(vx(3,:)-vx(1,:)).^2);
SP = (L1+L2+L3)/2;
AR = sqrt(SP.*(SP-L1).*(SP-L2).*(SP-L3));

% Determine maximal stable timestep
dts = AR./SP;
dtau = (2/3)*(min(dts(dts ~= 0))*rmin);

% Find number of timesteps and adjust dt for equally spaced timesteps
Nsteps = ceil(tau_max/dtau);
dtau = tau_max/Nsteps;

end % End of DG_Initialize subroutine
```

Next, we preallocate memory to be used for the fields and their jumps. Each field only requires one residual storage array (e.g. `Esu`) since we opt to use LSERK for our timestepping algorithm. The variables: `dEs`, `dEx`, `dEy`, `dHs`, `dHx`, and `dHy` store the jumps in the field values along the edges of the elements.

```
% Preallocate field arrays, residual storage arrays, edge arrays
Es = zeros(Np,K);   Esu = zeros(Np,K);  dEs = zeros(3*Nfp,K);
Ex = zeros(Np,K);   Exu = zeros(Np,K);  dEx = zeros(3*Nfp,K);
Ey = zeros(Np,K);   Eyu = zeros(Np,K);  dEy = zeros(3*Nfp,K);
Hs = zeros(Np,K);   Hsu = zeros(Np,K);  dHs = zeros(3*Nfp,K);
Hx = zeros(Np,K);   Hxu = zeros(Np,K);  dHx = zeros(3*Nfp,K);
```

```
Hy = zeros(Np,K);   Hyu = zeros(Np,K);  dHy = zeros(3*Nfp,K);
```

Another step which reduces CPU work during the timestepping routine is to combine multiplicative factors and evaluate some expressions involving the Heaviside function. These Heaviside factors are ideal since we have defined our mesh to have elements where the Heaviside functions evaluate to a constant over an element. We can also define the low-storage explicit Runge-Kutta (LSERK) coefficients used in the timestepping loop. This RK method uses 5 stages for a 4th order method but only requires 1 residual storage array per field in contrast to the classical 4th order RK method which requires 4 residual storage arrays with 4 stages. Since the field arrays can be quite large, we opt to use the more memory efficient method.

```
% Locate elements along x=0 and split them for heaviside
theta_x = repmat(mean(x)>0,Np,1);

% Define function for s-dependant curvature
kap = @(s) (1/R)*((s>=0)&(s<=s2));
kap_s = repmat(kap(mean(s)),Np,1);

% Define additional multiplicative factors
RoxpR = 1./(1+kap_s.*x);
IoxpR = kap_s./(1+kap_s.*x);
RoxpRE = 1./(1+kap(reshape(s(vM),3*Nfp,K)).*reshape(x(vM),3*Nfp,K));
IoZ0 = 1/Z0;

% Generate butcher table of LSERK coefficients
rk4a = [            0 ...
        -567301805773/1357537059087   ...
        -2404267990393/2016746695238  ...
        -3550918686646/2091501179385  ...
        -1275806237668/842570457699];
rk4b = [ 1432997174477/9575080441755  ...
         5161836677717/13612068292357 ...
         1720146321549/2090206949498  ...
         3134564353537/4481467310338  ...
         2277821191437/14882151754819];
rk4c = [             0  ...
         1432997174477/9575080441755  ...
         2526269341429/6820363962896  ...
         2006345519317/3224310063776  ...
         2802321613138/2924317926251];
```

The multiplicative factors `RoxpR`, `IoxpR`, and `IoZ0` are matrix equivalents of $1/(1+\kappa x)$, $\kappa/(1+\kappa x)$, and $1/Z_0$ respectively; while the factor `RoxpRE` is $1/(1+\kappa x)$ evaluated along edges only. Next, the source and initial conditions of the fields are given by:

```
% Define source terms for RHS of Ex and Hy
Hy_S = q*Z0*c*Gp*theta_x;
Ex_S = q*c*Gp*theta_x;

% Define initial condition from analytic solution in straight pipe
Ex_0 = q*Z0*c*Gp*(-sinh(alpha*x_out)*cosh(alpha*(x-x_in))/...
    sinh(alpha*(x_out-x_in)) + cosh(alpha*x).*theta_x).*lambda(s);
Ey_0 = q*Z0*c*Gp*(-sinh(alpha*x_out)*sinh(alpha*(x-x_in))/...
    sinh(alpha*(x_out-x_in)) + sinh(alpha*x).*theta_x).*lambda(s);

% Set initial conditions for fields
Ex = Ex_0;
Ey = Ey_0;
```

```
Hy = Ex_0/Z0 - q*c*Gp*lambda(s).*theta_x;
Hx = -Ey_0/Z0;
```

At this point, we can complete a few optional preprocessing routines or begin the timestepping loop to evolve the 6 fields.

A.2.5 Optional Preprocessing

The first optional preprocessing focuses on setup of the curvilinear element operators for the elements along the curved outer wall at $x_{\text{out}}(s)$. Next, depending on the output desired, for example contour plots or time series data slices for Fourier transforms, we can construct some additional arrays to store this data. We also will want to generate interpolation matrices to extract data at points which do not lie exactly on a node.

The first routine will adjust the nodal locations of the elements located along the boundary by using Gordon-Hall blending. The output of this routine also includes the list of element numbers along the curved outer boundary.

```
function [s,x,cKs] = DG_CurveFix(r,t,s,x,VS,vflag,...
    Fm,EToV,N,K,R,x_out,s1,s2,x2)
% This subroutine adjusts the s and x coordinates of vertices located
% along the curved outer wall and flags the elements in the cf array.
% This subroutine is based on parts of MakeCylinder2D.m from nudg.org.
%
%  Inputs:
%  r,t   = reference element coordinate 1,2 (Np x 1)
%  s,x   = matrix of physical s,x locations by element (Np x K)
%  VS    = vector of s-coordinates of vertices (1 x Nv)
%  vflag = vector of vertex numbers along curvilinear edges (1 x varies)
%  Fm    = element array of edge indices (Nfp x 3)
%  EToV  = element to vertex mapping matrix (K x 3)
%  N     = order of elements (1 x 1)
%  K     = number of elements (1 x 1)
%  R     = radius of curvature (1 x 1)
%  x_out = initial distance of source to outer wall (1 x 1)
%  s1    = s-coordinate of first corner (1 x 1)
%  s2    = s-coordinate of second corner (1 x 1)
%  x2    = x-coordinate of second corner (1 x 1)
%
%  Outputs:
%  s_new = matrix of updated physical s locations by element (Np x K)
%  x_new = matrix of updated physical x locations by element (Np x K)
%  cKs   = vector of element numbers containing curvilinear edges (cK x 1)

% Define outer wall profile function and segmented arclength integrands
wall = @(s) x_out.*(s<0) + ((R+x_out)./cos(s/R)-R).*((0<=s)&(s<s1)) + ...
    ((R+x_out)*tan(s1/R)./sin(s/R)-R).*((s1<=s)&(s<s2)) + x2.*(s>=s2);
arc1 = @(s) sqrt(1+((R+x_out)/R*sin(s/R)./cos(s/R).^2).^2);
arc2 = @(s) sqrt(1+((R+x_out)/R*tan(s1/R)*cos(s/R)./sin(s/R).^2).^2);

% Locate elements containing curvilinear edges
cKs = [];    dKs = [];
for vert = 1:length(vflag)
    elems = find((vflag(vert) == EToV));    % Find test elements with vert
    for etest = 1:length(elems)
        elem = mod(elems(etest),K); % Select a test element of vert
        v1 = EToV(elem,1);          % Vertex 1 of test element
        v2 = EToV(elem,2);          % Vertex 2 of test element
        v3 = EToV(elem,3);          % Vertex 3 of test element
        if sum(v1 == vflag) && sum(v2 == vflag)
            cKs = [cKs;elem];       % Element is part of curved boundary
            dKs = [dKs;1];          % Edge 1 of element is curved
        elseif sum(v2 == vflag) && sum(v3 == vflag)
            cKs = [cKs;elem];       % Element is part of curved boundary
            dKs = [dKs;2];          % Edge 2 of element is curved
        elseif sum(v3 == vflag) && sum(v1 == vflag)
            cKs = [cKs;elem];       % Element is part of curved boundary
            dKs = [dKs;3];          % Edge 3 of element is curved
        end
    end
end

% Deform nodes in triangles along curved boundary
for e = 1:length(cKs)
    switch dKs(e)
        case 1  % Edge 1 is curved
            va = EToV(cKs(e),1);
            vb = EToV(cKs(e),2);
            vr = r;
            ids = find(abs(1-vr)>1e-7); % Node numbers of element interior
            bl = -(r(ids)+t(ids))./(1-vr(ids));
        case 2  % Edge 2 is curved
            va = EToV(cKs(e),2);
            vb = EToV(cKs(e),3);
            vr = t;
            ids = find(abs(1-vr)>1e-7); % Node numbers of element interior
            bl = (r(ids)+1)./(1-vr(ids));
        case 3  % Edge 3 is curved
```

```
            va = EToV(cKs(e),1);
            vb = EToV(cKs(e),3);
            vr = t;
            ids = find(abs(1-vr)>1e-7); % Node numbers of element interior
            bl = -(r(ids)+t(ids))./(1-vr(ids));
    end
    fnodes = Fm(:,dKs(e));   % Face node numbers
    fr = vr(fnodes);         % Reference triangle edge nodes
    s_new = s(:,cKs(e));     x_new = x(:,cKs(e));
    fds = zeros(N+1,1);      fdx = zeros(N+1,1);

    if min(VS(va),VS(vb)) < s1
        arc = arc1; % Triangle is along arc segment 1
    else
        arc = arc2; % Triangle is along arc segment 2
    end

    alen = integral(arc,VS(va),VS(vb));    % Arclength of edge
    for n = 1:N+1
        enode = Fm(n,dKs(e));    % Select node along curved edge
        arcint = @(s) integral(arc,VS(va),s)-(fr(n)+1)*alen/2;
        s_new(enode) = fzero(arcint,s(enode,cf(e)));
        x_new(enode) = wall(s_new(enode));
        fds(n) = s_new(enode)-s(enode,cf(e));
        fdx(n) = x_new(enode)-x(enode,cf(e));
    end

    % Blend deformation into all nodes in element
    Vf = zeros(length(fr),N+1);    Vv = zeros(length(vr),N+1);
    for n = 1:N+1
```

```
        Vf(:,n) = JP(fr(:),0,0,n-1);
        Vv(:,n) = JP(vr(:),0,0,n-1);
    end
    Vds = Vv*(Vf\fds);
    Vdx = Vv*(Vf\fdx);
          s_new(ids) = s_new(ids) + bl.*Vds(ids);
    x_new(ids) = x_new(ids) + bl.*Vdx(ids);

    % Update element with new node locations
    s(:,cKs(e)) = s_new;
    x(:,cKs(e)) = x_new;
end

end % End of DG_CurveFix subroutine
```

The `DG_CurveFix` routine should be run just after the nodes have been assigned in `DG_Coordinates` but can be run at any point until the curvilinear operators are needed or the initial conditions are set. The next step in constructing the curvilinear operators is as follows and uses parts of `BuildCurvedOPS2D.m` from NUDG.org.

```
function [cKse,cDs,cDx,gns,gnx,gs,gx,gVM,gVP,glift,eflag] = ...
    DG_CurvedOPS(N,Np,V,s,x,cKs,EToE,EToF)
% This subroutine completes the construction of all curvilinear element
% operators. The operators are computed using the cubature nodes. This
% subroutine is based on BuildCurvedOPS.m from nudg.org.
%
%  Inputs:
%  N      = order of elements (1 x 1)
%  Np     = number of nodes per element (1 x 1)
%  V      = Vandermonde matrix (Np x Np)
%  s,x    = matrix of physical s,x locations by element (Np x K)
%  cKs    = vector of element numbers with curvilinear edges (cK x 1)
%  EToE   = element to element mapping matrix (K x 3)
%  EToF   = element to edge mapping matrix (K x 3)
%
%  Outputs:
%  cKse  = list of neighboring element numbers of curved elements (cK x 3)
%  cDs   = sparse Ds operator of curved element (Np*cK x Np*cK)
%  cDx   = sparse Dx operator of curved element (Np*cK x Np*cK)
%  gns   = cell array of ns operator curved element {1x3}(Ni*cK x 1)
%  gnx   = cell array of nx operator curved element {1x3}(Ni*cK x 1)
%  gs    = cell array of s coords of curved edge nodes {1x3}(Ni*cK x 1)
%  gx    = cell array of x coords of curved edge nodes {1x3}(Ni*cK x 1)
%  gVM   = cell array of sparse curved vM traces {1x3}(Ni*cK x Ni*cK)
%  gVP   = cell array of sparse curved vP traces {1x3}(Ni*cK x Ni*cK)
%  glift = cell array of sparse curved lift matrices {1x3}(Ni*cK x Ni*cK)
%  eflag = cell array of face nodes for curved edges {1x3}(varies x 1)

Ninput = 3*N; % Can be adjusted but should be at least 3*N

% Generate cubature nodes and weights
[cr,ct,cw,Nc] = DG_Cubature(Ninput);

% Create interpolation matrix between cubature and LGL nodes
cV = DG_InterpMatrix(N,Np,V,Nc,cr,ct);

% Construct cubature derivative matrices
[cDr,cDt] = DG_Collocation(N,Np,cr,ct,V);
```

```
% Compute (r,t) Gauss nodes and weights used for 1D integration
J = zeros(Ninput+1);      h = 2*(0:Ninput);
J = diag(2./(h(1:Ninput)+2).*sqrt((1:Ninput).^4./ ...
        (h(1:Ninput)+1)./(h(1:Ninput)+3)),1);
J(1,1) = 0;
J = J + J';
[v1,d1] = eig(J);   gz = diag(d1);
gw = (v1(1,:)').^2*2;

% Generate (r,t) Gauss nodes on element
gr = [gz,-gz,-ones(size(gz))];
gt = [-ones(size(gz)),gz,-gz];

% Create Gauss quadrature interpolation and collocation matrices
gV = zeros(length(gz),Np,3);
gDr = zeros(length(gz),Np,3);
gDt = zeros(length(gz),Np,3);
for f1 = 1:3
    gV(:,:,f1) = DG_InterpMatrix(N,Np,V,Nc,gr(:,f1),gt(:,f1));
    [gDr(:,:,f1),gDt(:,:,f1)] = DG_Collocation(N,Np,gr(:,f1),gt(:,f1),V);
end

% Contruct the curved information sparse and normal arrays
cK = length(cKs);
Ni = Ninput+1;
cKse = zeros(cK,3);
cDs = spalloc(Np*cK,Np*cK,Np*Np*cK);
cDx = spalloc(Np*cK,Np*cK,Np*Np*cK);
gns{1} = zeros(Ni*cK,1);
gns{2} = zeros(Ni*cK,1);
```

```
gns{3} = zeros(Ni*cK,1);
gnx{1} = zeros(Ni*cK,1);
gnx{2} = zeros(Ni*cK,1);
gnx{3} = zeros(Ni*cK,1);
gs{1} = zeros(Ni*cK,1);
gs{2} = zeros(Ni*cK,1);
gs{3} = zeros(Ni*cK,1);
gx{1} = zeros(Ni*cK,1);
gx{2} = zeros(Ni*cK,1);
gx{3} = zeros(Ni*cK,1);
gVM{1} = spalloc(Ni*cK,Np*cK,Ni*Np*cK);
gVM{2} = spalloc(Ni*cK,Np*cK,Ni*Np*cK);
gVM{3} = spalloc(Ni*cK,Np*cK,Ni*Np*cK);
gVP{1} = spalloc(Ni*cK,Np*cK,Ni*Np*cK);
gVP{2} = spalloc(Ni*cK,Np*cK,Ni*Np*cK);
gVP{3} = spalloc(Ni*cK,Np*cK,Ni*Np*cK);
glift{1} = spalloc(Np*cK,Ni*cK,Np*Ni*cK);
glift{2} = spalloc(Np*cK,Ni*cK,Np*Ni*cK);
glift{3} = spalloc(Np*cK,Ni*cK,Np*Ni*cK);
eflag{1} = zeros(Ni*cK,1);
eflag{2} = zeros(Ni*cK,1);
eflag{3} = zeros(Ni*cK,1);

for i = 1:cK;
    % Extract element number and LGL nodes of element
    k1 = cKs(i);     s1 = s(:,k1);        x1 = x(:,k1);

    % Compute geometric factors for curved element
    [crs,crx,cts,ctx,cJ,~,~,~] = DG_GF2D(s1,x1,cDr,cDt);
    cMM = cV'*diag(cJ.*cw)*cV;        %Mass matrix of element
```

```
% Store Ds and Dx operators by element in sparse array
cDs(1+(i-1)*Np:Np+(i-1)*Np,1+(i-1)*Np:Np+(i-1)*Np) = ...
    cMM\(cV'*diag(cJ.*cw)*(diag(crs)*cDr+diag(cts)*cDt));
cDx(1+(i-1)*Np:Np+(i-1)*Np,1+(i-1)*Np:Np+(i-1)*Np) = ...
    cMM\(cV'*diag(cJ.*cw)*(diag(crx)*cDr+diag(ctx)*cDt));

for f1 = 1:3
    %Extract element and face numbers of adjacent elements
    cKse(i,f1) = EToE(k1,f1);    f2 = EToF(k1,f1);

    %Compute geometric factors of faces
    [grs,grx,gts,gtx,gJ,~,~,~] = DG_GF2D(s1,x1,...
        gDr(:,:,f1),gDt(:,:,f1));

    switch f1
        case 1
            gnsf = -gts;     gnxf = -gtx;
        case 2
            gnsf = grs+gts;  gnxf = grx+gtx;
        case 3
            gnsf = -grs;     gnxf = -grx;
    end

    %Surface Jacobian and normals for face f1
    gsJ = sqrt(gnsf.*gnsf + gnxf.*gnxf);
    gnsf = gnsf./gsJ;     gnxf = gnxf./gsJ;     gsJ = gsJ.*gJ;

    gns{f1}(1+(i-1)*Ni:Ni+(i-1)*Ni) = gnsf;
    gnx{f1}(1+(i-1)*Ni:Ni+(i-1)*Ni) = gnxf;
```

```
        %Store s and x coordinates for face f1 nodes
        gs{f1}(1+(i-1)*Ni:Ni+(i-1)*Ni) = gV(:,:,f1)*s1;
        gx{f1}(1+(i-1)*Ni:Ni+(i-1)*Ni) = gV(:,:,f1)*x1;

        %Store Vandermonde traces for face f1 and face f2 edges
        gVM{f1}(1+(i-1)*Ni:Ni+(i-1)*Ni,1+(i-1)*Np:Np+(i-1)*Np) = ...
            gV(:,:,f1);
        gVP{f1}(1+(i-1)*Ni:Ni+(i-1)*Ni,1+(i-1)*Np:Np+(i-1)*Np) = ...
            gV(end:-1:1,:,f2);

        %Store lift operator for curved element
        glift{f1}(1+(i-1)*Np:Np+(i-1)*Np,1+(i-1)*Ni:Ni+(i-1)*Ni) = ...
            cMM\(gV(:,:,f1)'*diag(gw.*gsJ));

        if cKs(i) == cKse(i,f1)
            eflag{f1}(1+(i-1)*Ni:Ni+(i-1)*Ni) = 1+(i-1)*Ni:Ni+(i-1)*Ni;
        end
    end
end

% Delete face numbers of non-boundary faces for indexing
for f1 = 1:3
    eflag{f1}(eflag{f1}==0) = [];
end

end % End of DG_CurvedOPS subroutine
```

The routine DG_CurvedOPS uses on two routines not yet defined. The first routine is

DG_Cubature which is based on Cubature2D.m from NUDG.org which in turn uses the cubature tables distributed in the *Encyclopaedia of Cubature Formulas* developed by R. Cools [8]. The list of tables is extensive thus we only include the table for $N_{\text{input}} = 6$ corresponding to $N = 2$.

```
function [cr,ct,cw,Nc] = DG_Cubature(Ninput)
% This subroutine generates the cubature operators used in the curvilinear
% elements. The order of the cubature input is taken as 3 times the
% element order: i.e. Ninput = 3*N. This subroutine is based on
% Cubature2D.m and CubatureData2D.m from nudg.org.
% The tables are obtained from:
%
% Cools, R. "Monomial Cubature Rules Since Stroud: A Compilation-Part 2."
% J. Comput. Appl. Math. 112, 21-27, 1999.
%
% Cools, R. "Encyclopaedia of Cubature Formulas."
% http://www.cs.kuleuven.ac.be/~nines/research/ecf/ecf.html
%
%  Inputs:
%  Ninput = order of cubature table (generally 3*N) (1 x 1)
%
%  Outputs:
%  cr = cubature node coordinate 1 (Nc x 1)
%  ct = cubature node coordinate 2 (Nc x 1)
%  cw = cubature node weights (Nc x 1)
%  Nc = number of cubature nodes per element (1 x 1)

if Ninput <= 28
    switch Ninput
    case 6
    cub2D = [[-0.501426509658179 -0.501426509658179 0.233572551452759];...
        [0.002853019316358 -0.501426509658179 0.233572551452759];...
        [-0.501426509658179 0.002853019316358 0.233572551452759];...
        [-0.873821971016996 -0.873821971016996 0.101689812740414];...
        [0.747643942033991 -0.873821971016996 0.101689812740414];...
        [-0.873821971016996 0.747643942033991 0.101689812740414];...
        [-0.379295097932431 -0.893709900310366 0.165702151236747];...
        [-0.893709900310366 -0.379295097932431 0.165702151236747];...
        [0.273004998242797 -0.893709900310366 0.165702151236747];...
        [-0.893709900310366 0.273004998242797 0.165702151236747];...
        [0.273004998242797 -0.379295097932431 0.165702151236747];...
        [-0.379295097932431 0.273004998242797 0.165702151236747]];
    end
    cr = cub2D(:,1);
    ct = cub2D(:,2);
    cw = cub2D(:,3);
else    % General formula for Ninput > 28
    cN = ceil((Ninput+1)/2)-1;
    J1 = zeros(cN);   h1 = 2*(0:cN);
    J1 = diag(2./(h1(1:cN)+2).*sqrt((1:cN).^4./ ...
        (h1(1:cN)+1)./(h1(1:cN)+3)),1);
    J1(1,1) = 0;
    J1 = J1 + J1';

    J2 = zeros(cN);   h2 = 2*(0:cN)+1;
    J2 = diag(-1/2./(h2+2)./h2) + ...
        diag(2./(h2(1:cN)+2).*sqrt((1:cN).^2.*((1:cN)+1).^2./ ...
        (h2(1:cN)+1)./(h2(1:cN)+3)),1);
    J2 = J2 + J2';
    [v1,d1] = eig(J1);
    ca = diag(d1);
    wa = (v1(1,:)').^2*2;
    [v2,d2] = eig(J2);
    cb = diag(d2);
    wb = (v2(1,:)').^2*2;

    ca = ones(cN+1,1)*ca';
    cb = cb*ones(1,cN+1);

    cr = 0.5*(1+ca).*(1-cb)-1;
    ct = cb;
    cw = 0.5*wb*(wa');

    cr = cr(:);     ct = ct(:);     cw = cw(:);
end

Nc = length(cw);

end % End of DG_Cubature subroutine
```

The other routine required for `DG_CurvedOPS` is `DG_InterpMatrix` which involves generating an interpolation matrix to map the solution from the LGL nodes to an equally spaced grid of nodes of an arbitrary order N_{out}, unless a reference nodal set r_{out} and t_{out} are given. Generally, it is desired to take $N_{\text{out}} > N$ to resolve the polynomial on each element for a contour or surface plot. The resultant interpolation matrix can be multiplied by the fields of size $N_p \times K$ to map them to an equally spaced set of nodes; which can later be plotted. Our routine is based off `InterpMatrix2D.m` from NUDG.org.

```
function [interp] = DG_InterpMatrix(N,Np,V,Nout,rout,tout)
% This subroutine generates the interpolation matrix between LGL nodes and
% equally spaced nodes. If other nodal spacing is desired, the subroutine
% accepts the extra inputs rout and tout. This subroutine is based on
% InterpMatrix2D.m from nudg.org.
%
%  Inputs:
%  N    = order of elements (1 x 1)
%  Np   = number of nodes per element (1 x 1)
%  V    = Vandermonde matrix (Np x Np)
%  Nout = order of output nodes (1 x 1)
%  rout = input nodal locations coordinate 1 [optional] (Nout x 1)
%  tout = input nodal locations coordinate 2 [optional] (Nout x 1)
%
%  Outputs:
%  interp = interpolation matrix (Npout x Np)

if nargin < 5   % Default to equidistant nodal spacing
    % Generate equally spaced reference triangle coordinates
    Npout = (Nout+1)*(Nout+2)/2;
    rout = zeros(Npout,1);        tout = zeros(Npout,1);
    sk = 1;
    for n=1:Nout+1
        for m=1:Nout+2-n
            rout(sk) = -1 + 2*(m-1)/Nout;
            tout(sk) = -1 + 2*(n-1)/Nout;
            sk = sk + 1;
        end
    end
```

```
else
    Npout = length(rout);
end

% Map reference triangle to standard triangle
V2D = zeros(Npout,Np);
a = zeros(Npout,1);
for  n = 1:Npout
    if (tout(n) ~= 1)
        a(n) = 2*(1+rout(n))/(1-tout(n))-1;
    else
        a(n) = -1;
    end
end
b = tout;

sk = 1;
for i = 0:N
    for j = 0:N-i
        h1 = JP(a,0,0,i);
        h2 = JP(b,2*i+1,0,j);
        V2D(:,sk) = sqrt(2)*h1.*h2.*(1-b).^i;
        sk = sk + 1;
    end
end

% Construct interpolation matrix
interp = V2D*inv(V);
```

```
end % End of DG_InterpMatrix subroutine
```

The next step for this interpolation is to create a new triangulation with the equally spaced nodes generated by multiplying the interpolation matrix to the coordinates. This next routine is based on parts of `PlotField2D.m` from NUDG.org.

```
function [tri] = DG_EqualTriangles(Nout,K)
% This subroutine generates an equally spaced triangulation of the domain.
% The triangulation will later be used for plotting using trisurf. This
% subroutine is based on part of PlotField2D.m from nudg.org.
%
%  Inputs:
%  Nout = order of output nodes (1 x 1)
%  K    = number of elements (1 x 1)
%
%  Outputs:
%  tri = triangulation based on order and number of elements (Nt x K)
%        (Nt is the number of triangles in one element)

% Generate triangle node numbering matrix
Npout = (Nout+1)*(Nout+2)/2;
tr = zeros(Nout+1);
count = 0;
for i = 1:Nout+1
    for j = 1:Nout+2-i
        count = count + 1;
        tr(i,j) = count;
    end
end
```

```
% Generate triangles for one element
tris = zeros(Nout^2,3);
count = 0;
for i = 1:Nout+1
    for j = 1:Nout+1-i
        v1 = tr(i,j);   v2 = tr(i,j+1);
        v3 = tr(i+1,j); v4 = tr(i+1,j+1);
        count = count + 1;
        tris(count,:) = [v1 v2 v3];
        if v4
            count = count + 1;
            tris(count,:) = [v2 v4 v3];
        end
    end
end

% Extend triangulation to all elements
tri = zeros(Nout^2*K,3);
for i = 1:K
    tri((1+(i-1)*Nout^2):(i*Nout^2),:) = tris + (i-1)*Npout;
end

end % End of DG_EqualTriangles subroutine
```

The size of this triangulation of sub-elements $K_{\text{equal}} = K \times N_{\text{out}}^2$. Each of these sub-triangles are used to plot the field using the 3 vertices of the sub-triangle to define a plane: polynomial of degree 1. The MATLAB command: `trisurf` can plot these sub-triangles and interpolate their height and color linearly using those 3 points per triangle.

The next preprocessing step is required only if timeseries data is desired at specific locations. If a large number of probe points is desired, such as to produce a contour slice, then it is recommended to pass certain variables through the function which are the same at all time-steps to save on computations. We outline the general algorithm here beginning with locating the element containing given points in the `probe` array of size $2 \times n_{\text{probe}}$.

```
function [probeE] = DG_ElementFind(probe,VS,VX,EToV)
% This subroutine locates the elements containing the probe points. The
% method involves locating the nearest vertex to the probes and checking
% adjacent elements for if they contain the probe points.
%
%  Inputs:
%  probe = array of (s,x) coordinates for probe points (nprobe x 2)
%  VS    = vector of s-coordinates of vertices (1 x Nv)
%  VX    = vector of x-coordinates of vertices (1 x Nv)
%  EToV  = element to vertex mapping matrix (K x 3)
%
%  Outputs:
%  probeE = vector of element numbers containing probes (nprobe x 1)

nprobe = size(probe,1);
probeE = zeros(nprobe,1);
for p = 1:nprobe

    % Location of probe p
    ps = probe(p,1);    px = probe(p,2);
    % Locate the vertex nearest probe p
    probeD = sqrt((VS-ps).^2+(VX-px).^2);
    probeVs = find(probeD == min(probeD));
    probeV = probeVs(1); % Select only one vertex if more than one

    % Locate the elements containing the vertex nearest probe p
    probeEs = find(sum(EToV == probeV,2));

    for q = 1:length(probeEs)

        % Contruct barycentric coordinates to check if probe is in element
        s1 = VS(EToV(probeEs(q),1));   x1 = VX(EToV(probeEs(q),1));
        s2 = VS(EToV(probeEs(q),2));   x2 = VX(EToV(probeEs(q),2));
        s3 = VS(EToV(probeEs(q),3));   x3 = VX(EToV(probeEs(q),3));
        detE = det([s1 - s3, s2 - s3; x1 - x3, x2 - x3]);

        lam1 = ((x2-x3)*(ps-s3)+(s3-s2)*(px-x3))/detE;
        lam2 = ((x3-x1)*(ps-s3)+(s1-s3)*(px-x3))/detE;
        lam3 = 1 - lam1 - lam2;

        if (lam1>=0) && (lam2>=0) && (lam3>=0)
            probeE(p) = probeEs(q);
            break;  % Stop once one element is found to contain probe p
        end
    end
end

% Output warning message if some probe points outside of domain
if sum(probeE==0)
    disp('Warning: one or more probe locations not in domain');
    probeE = probeE(probeE~=0); % Omit probes outside domain
end


end % End of DG_ElementFind subroutine
```

To use these probes in the timestepping routine, we also must include a method of interpolating the element containing the probes at those points. We include a modified version of the `DG_InterpMatrix` routine to allow for a more general interpolation of an element at any specified point.

```
% This subroutine interpolates the field f(s,x) at (sp,xp) and returns the
% value fp = f(sp,xp) interpolated using the Nth order polynomial defined
% by the f(s,x) points.
%
%  Inputs:
%  F   = field values at (s,x) (Np x 1)
%  sp  = sample point s-coordinate (1 x 1)
%  xp  = sample point x-coordinate (1 x 1)
%  N   = order of elements (1 x 1)
%  Np  = number of nodes per element (1 x 1)
%  V   = Vandermonde matrix (Np x Np)
%  Vsx = array storing s,x coordinates of vertices (3 x 2)
%
%  Outputs:
%  fp = Lagrange interpolated value of f(sp,xp) (1 x 1)


% Compute transformation to reference triangle
R = [-Vsx(1,1)+Vsx(2,1), -Vsx(1,1)+Vsx(3,1); ...
     -Vsx(1,2)+Vsx(2,2), -Vsx(1,2)+Vsx(3,2)];
R = R\[2*sp-Vsx(2,1)-Vsx(3,1);2*xp-Vsx(2,2)-Vsx(3,2)];
```

```

% Transform to standard triangle
if R(2) ~= 1
    R(1) = 2*(1+R(1))/(1-R(2))-1;
else
    R(1) = -1;
end

% Contruct interpolation weights
V2D = zeros(1,Np);
sk = 1;
for i = 0:N
    for j = 0:N-i
        h1 = JP(R(1),0,0,i);
        h2 = JP(R(2),2*i+1,0,j);
        V2D(sk) = sqrt(2)*h1.*h2.*(1-R(2)).^i;
        sk = sk + 1;
    end
end

% Interpolate using Nth order Lagrange polynomial
fp = V2D*(V\f);

end % End of DG_Interp subroutine
```

The two routines: `DG_ElementFind` and `DG_Interp` can be used at every timestep if the probe points are not stationary in time but instead move along an arbitrary path. If a large number of stationary probes are necessary, it is recommended to compute the quantity `V2D*inv(V)` in the last line of `DG_Interp` outside of the timestepping loop and multiply the fields directly by that quantity in the timestepping loop.

Lastly, if using an Nvidia CUDA-enabled GPU with double-precision processing power greater than the CPU (e.g. Nvidia GTX Titan), the following lines can be added before the timestepping loop to move all necessary workspace variables to the GPU memory. This instructs MATLAB to use the GPUArray versions of the same commands such as `*` and `.*` which may drastically improve performance if N and K are large enough for good parallelization. A quick toggle for this option can be specified in the simulation parameters section by setting the flag `GPU_flag` to be `1` or `0` for "on" or "off". The short `pause` command is necessary to force the CPU to wait until the GPU is finished loading all the variables.

```
% Move necessary variables to GPU for GPU processing if selected
if GPUflag
    Es = gpuArray(Es);    Esu = gpuArray(Esu);  dEs = gpuArray(dEs);
    Ex = gpuArray(Ex);    Exu = gpuArray(Exu);  dEx = gpuArray(dEx);
    Ey = gpuArray(Ey);    Eyu = gpuArray(Eyu);  dEy = gpuArray(dEy);
    Hs = gpuArray(Hs);    Hsu = gpuArray(Hsu);  dHs = gpuArray(dHs);
    Hx = gpuArray(Hx);    Hxu = gpuArray(Hxu);  dHx = gpuArray(dHx);
    Hy = gpuArray(Hy);    Hyu = gpuArray(Hyu);  dHy = gpuArray(dHy);
    Hy_S = gpuArray(Hy_S);          Ex_S = gpuArray(Ex_S);
    RoxpR = gpuArray(RoxpR);        RoxpRE = gpuArray(RoxpRE);
    IoxpR = gpuArray(IoxpR);        tau = gpuArray(time);
    alpha = gpuArray(alpha);        theta_x = gpuArray(theta_x);
    vM = gpuArray(vM);      vP = gpuArray(vP);      alp = gpuArray(alp);
    mB = gpuArray(mB);      vB = gpuArray(vB);      dtau = gpuArray(dtau);
    ns = gpuArray(ns);      nx = gpuArray(nx);      rs = gpuArray(rs);
    rx = gpuArray(rx);      ts = gpuArray(ts);      tx = gpuArray(tx);
    Dr = gpuArray(Dr);      Dt = gpuArray(Dt);      IoZ0 = gpuArray(IoZ0);
    Z0 = gpuArray(Z0);      Fsc = gpuArray(Fsc);    Lift = gpuArray(Lift);
    rk4a = gpuArray(rk4a);  rk4b = gpuArray(rk4b);  rk4c = gpuArray(rk4c);
    pause(0.01)
    disp('   GPU variable transfer complete.')
end
```

This concludes the optional preprocessing routines and these may be inserted after the necessary routines but before the timestepping routines. **NOTE**: The curvilinear subroutines do **NOT** work in conjunction with GPU processing as it is implemented here. This is due to sparse array restrictions on gpuArray objects and extra memory transfer overhead. The curvilinear codes here were not developed for parallel computing and remain un-optimized for large computational use.

A.3 Timestepping routines

A.3.1 Main Timestep Loop (LSERK)

With all preprocessing routines completed, then the timestepping loop can begin. The starting time begins at $\tau = 0$ which is denoted by `tau = 0`. The individual stages for the Runge-Kutta method are computed by the `rk4c` coefficients. We also use the `tic` command to use as a stopwatch for processing time estimates. This begins as:

```
for tstep = 1:Nsteps  % Begin timestep loop
    for RK = 1:5          % Begin LSERK loop
        % Current time for LSERK step
        tau = dtau*(tstep-1+rk4c(RK));
```

All subsequent lines are inside the Runge-Kutta 5 stage loop until otherwise noted. We begin the process by computing the jumps along the boundaries between elements. Recall that `vM` contains the node list for edges nodes for each element and `vP` contains the node list for the adjacent elements' edges corresponding to the nodes `vM`.

```
        % Compute jumps in field
        dEs(:) = Es(vM) - Es(vP);
        dEx(:) = Ex(vM) - Ex(vP);
        dEy(:) = Ey(vM) - Ey(vP);
        dHs(:) = Hs(vM) - Hs(vP);
        dHx(:) = Hx(vM) - Hx(vP);
        dHy(:) = Hy(vM) - Hy(vP);
```

Next, we set the boundary conditions for the global boundary. Since we have extended the straight segments of the chamber before and after the bend to be long enough for the simulation not to produce reflections at $s = s_{\min}$ or $s = s_{\max}$, we may set those boundaries with some freedom. We choose all boundaries to be of Dirichlet-type where we specify the normal components of $\mathbf{E}$ and tangential components of $\mathbf{B}$ to be zero. Again, this is effectively setting the entire domain to be enclosed by a perfectly conducting wall and since we only run the simulation to a time before any fields reach the end walls in s, the radiative or absorbing boundary condition is not needed.

The boundary conditions in the DG scheme are implemented by a mirror principle to enforce $\mathbf{n} \times \mathbf{E} = \mathbf{0}$. We need not actually impose a condition on $\mathbf{H}$ since $\mathbf{n} \cdot \mathbf{H} = 0$ will be automatically satisfied by setting `dHs(mB)`, `dHy(mB)` and `dHy(mB)` to zero. This is actually done in the previous step since for the nodes along the global boundary, edges with no adjacent element, the jumps are set to zero, i.e. `vM(mB) = vP(mB)`.

```
        % Set boundary fluxes to Dirichlet-type for E
        dEs(mB) = 2*Es(vB);
        dEx(mB) = 2*Ex(vB);
        dEy(mB) = 2*Ey(vB);
```

Next, we compute the normals components of along the edges using the normal maps and the corresponding fields.

```
        % Construct n*E and n*H normals
        ndotdE = ns.*dEs + nx.*dEx;
        ndotdH = ns.*dHs + nx.*dHx;
```

These are now used to compute the numerical flux terms from the formulation of the PDE in strong form. Recall that `alp` is usually set to `1` for an upwinding scheme.

```
        % Construct DG fluxes for each field
        fluxEs = -Z0*nx.*dHy - alp*(dEs - ndotdE.*ns);
        fluxEx = Z0*RoxpRE.*ns.*dHy - alp*(dEx - ndotdE.*nx);
        fluxEy = -Z0*RoxpRE.*ns.*dHx + Z0*nx.*dHs - alp*dEy;
        fluxHs = IoZ0*nx.*dEy - alp*(dHs - ndotdH.*ns);
        fluxHx = -IoZ0*RoxpRE.*ns.*dEy - alp*(dHx - ndotdH.*nx);
        fluxHy = -IoZ0*nx.*dEs + IoZ0*RoxpRE.*ns.*dEx - alp*dHy;
```

Another important step is to compute necessary derivatives which will be used in computing the PDE right-hand-sides. These are computed by use of the chain rule: calculating the derivative with respect to the reference coordinates `r` and `t` and multiplying by the corresponding Jacobian for `s` or `x` derivatives.

```
        % Construct derivatives for necessary fields
        DxHy = rx.*(Dr*Hy) + tx.*(Dt*Hy);
        DsHy = rs.*(Dr*Hy) + ts.*(Dt*Hy);
        DsHx = rs.*(Dr*Hx) + ts.*(Dt*Hx);
        DxHs = rx.*(Dr*Hs) + tx.*(Dt*Hs);
        DxEy = rx.*(Dr*Ey) + tx.*(Dt*Ey);
        DsEy = rs.*(Dr*Ey) + ts.*(Dt*Ey);
        DsEx = rs.*(Dr*Ex) + ts.*(Dt*Ex);
        DxEs = rx.*(Dr*Es) + tx.*(Dt*Es);
```

Next, the PDE right-hand-side is combined with the numerical flux in the following form:

```
        % Evaluate right-hand-sides for LSERK
        rhsEs = Z0*(DxHy + alpha*Hx) + ...
            Lift*(Fsc.*fluxEs/2);
        rhsEx = Z0*(-alpha*Hs - RoxpR.*(DsHy + ...
            lambdap(s-tau).*Ex_S)) + Lift*(Fsc.*fluxEx/2);
        rhsEy = Z0*(RoxpR.*DsHx - DxHs - IoxpR.*Hs) + ...
            Lift*(Fsc.*fluxEy/2);
        rhsHs = IoZ0*(-DxEy + alpha*Ex) + ...
            Lift*(Fsc.*fluxHs/2);
        rhsHx = IoZ0*(-alpha*Es + RoxpR.*DsEy) + ...
            Lift*(Fsc.*fluxHx/2);
        rhsHy = IoZ0*(DxEs - RoxpR.*DsEx + IoxpR.*Es + ...
            lambdap(s-tau).*Hy_S) + Lift*(Fsc.*fluxHy/2);
```

Next, the temporary RK residual storage arrays are now upated.

```
        % Update residual storage arrays for LSERK
        Esu = rk4a(RK)*Esu + dtau*rhsEs;
        Exu = rk4a(RK)*Exu + dtau*rhsEx;
        Eyu = rk4a(RK)*Eyu + dtau*rhsEy;
        Hsu = rk4a(RK)*Hsu + dtau*rhsHs;
        Hxu = rk4a(RK)*Hxu + dtau*rhsHx;
        Hyu = rk4a(RK)*Hyu + dtau*rhsHy;
```

And finally, the fields are updated from the residual stages.

```
        % Update fields using LSERK
        Es = Es + rk4b(RK)*Esu;
        Ex = Ex + rk4b(RK)*Exu;
        Ey = Ey + rk4b(RK)*Eyu;
        Hs = Hs + rk4b(RK)*Hsu;
        Hx = Hx + rk4b(RK)*Hxu;
        Hy = Hy + rk4b(RK)*Hyu;
    end     % End of LSERK loop
end     % End of timestep loop
```

This marks the end of the Runge-Kutta loop for a single timestep. After all 5 stages, the fields have been properly evolved one `dt` in τ and can be either updated again to the next timestep or used for optional processes in the timestepping loop.

A.3.2 Optional Timestepping Routines

To implement the curvilinear element adjustments in vectorized form for increased performance, we can insert the following block of code immediately before updating the residual storage arrays, just after the right-hand-sides are computed. Again, this is not supported with the gpuArray implementation.

```
        % Fix the curved elements
        cEsi  = Es(:,cKs);
        cExi  = Ex(:,cKs);
        cEyi  = Ey(:,cKs);
        cHsi  = Hs(:,cKs);
        cHxi  = Hx(:,cKs);
        cHyi  = Hy(:,cKs);
        lambi = lambdap(s(:,cKs)-tau);
        RoxpRi = RoxpR(:,cKs);   % Can be placed outside timestep loop
        IoxpRi = IoxpR(:,cKs);   % Can be placed outside timestep loop
```

```
crhsEs = Z0*(cDx*cHyi(:) + alpha*cHxi(:));
crhsEx = Z0*(-alpha*cHsi(:) - ...
    RoxpRi(:).*(cDs*cHyi(:) + lambi(:).*Ex_Si(:)));
crhsEy = Z0*(RoxpRi(:).*(cDs*cHxi(:)) - ...
    cDx*cHsi(:) - IoxpRi(:).*cHsi(:));
crhsHs = (-cDx*cEyi(:) + alpha*cExi(:))/Z0;
crhsHx = (-alpha*cEsi(:) + RoxpRi(:).*(cDs*cEyi(:)))/Z0;
crhsHy = (cDx*cEsi(:) - RoxpRi(:).*(cDs*cExi(:)) + ...
    IoxpRi(:).*cEsi(:) + lambi(:).*Hy_Si(:))/Z0;

for f1 = 1:3
    cEso = Es(:,cKse(:,f1));
    cExo = Ex(:,cKse(:,f1));
    cEyo = Ey(:,cKse(:,f1));
    cHso = Hs(:,cKse(:,f1));
    cHxo = Hx(:,cKse(:,f1));
    cHyo = Hy(:,cKse(:,f1));

    gdEs = gVM{f1}*cEsi(:) - gVP{f1}*cEso(:);
    gdEx = gVM{f1}*cExi(:) - gVP{f1}*cExo(:);
    gdEy = gVM{f1}*cEyi(:) - gVP{f1}*cEyo(:);
    gdHs = gVM{f1}*cHsi(:) - gVP{f1}*cHso(:);
    gdHx = gVM{f1}*cHxi(:) - gVP{f1}*cHxo(:);
    gdHy = gVM{f1}*cHyi(:) - gVP{f1}*cHyo(:);

    gdEstmp = 2*gVM{f1}*cEsi(:);
    gdExtmp = 2*gVM{f1}*cExi(:);
    gdEytmp = 2*gVM{f1}*cEyi(:);
```

```
            gdEs(eflag{f1}) = gdEstmp(eflag{f1});
            gdEx(eflag{f1}) = gdExtmp(eflag{f1});
            gdEy(eflag{f1}) = gdEytmp(eflag{f1});
            gdHs(eflag{f1}) = 0*gdHs(eflag{f1});
            gdHx(eflag{f1}) = 0*gdHx(eflag{f1});
            gdHy(eflag{f1}) = 0*gdHy(eflag{f1});

            gndotdE = gns{f1}.*gdEs + gnx{f1}.*gdEx;
            gndotdH = gns{f1}.*gdHs + gnx{f1}.*gdHx;
            gRoxpRE = R./(R+gx{f1});

            gfluxEs = -Z0*gnx{f1}.*gdHy - ...
                alp*(gdEs - gndotdE.*gns{f1});
            gfluxEx = Z0*gRoxpRE.*gns{f1}.*gdHy - ...
                alp*(gdEx - gndotdE.*gnx{f1});
            gfluxEy = -Z0*gRoxpRE.*gns{f1}.*gdHx + Z0*gnx{f1}.*gdHs - ...
                alp*gdEy;
            gfluxHs = gnx{f1}.*gdEy/Z0 - ...
                alp*(gdHs - gndotdH.*gns{f1});
            gfluxHx = -gRoxpRE.*gns{f1}.*gdEy/Z0 - ...
                alp*(gdHx - gndotdH.*gnx{f1});
            gfluxHy = -gnx{f1}.*gdEs/Z0 + gRoxpRE.*gns{f1}.*gdEx/Z0 - ...
                alp*gdHy;

            crhsEs = crhsEs + glift{f1}*gfluxEs/2;
            crhsEx = crhsEx + glift{f1}*gfluxEx/2;
            crhsEy = crhsEy + glift{f1}*gfluxEy/2;
            crhsHs = crhsHs + glift{f1}*gfluxHs/2;
            crhsHx = crhsHx + glift{f1}*gfluxHx/2;
            crhsHy = crhsHy + glift{f1}*gfluxHy/2;
```

```
        end

        rhsEs(:,cKs) = reshape(crhsEs,Np,curv);
        rhsEx(:,cKs) = reshape(crhsEx,Np,curv);
        rhsEy(:,cKs) = reshape(crhsEy,Np,curv);
        rhsHs(:,cKs) = reshape(crhsHs,Np,curv);
        rhsHx(:,cKs) = reshape(crhsHx,Np,curv);
        rhsHy(:,cKs) = reshape(crhsHy,Np,curv);
```

It is important to note that these curvilinear adjustments add considerable computational complexity to each timestep and depending on the number of elements along the curved boundary. Future developments of this code may optimize this routine further.

Within the timestepping loop after the fields have been evolved by the RK procedure, a number of optional preprocessing routines can be run such as extracting the field at the probe points for time-series data or plotting.

To extract the data at the desired probe locations, we use the list `probeE` which contains the list of elements which contain the probe points to selectively extract data from the GPU for interpolation by use of the `gather` command. The storage array `probeF`, for the **E** fields, can be defined in the preprocessing section to be of size $(N_{steps} + 1) \times N_{probes} \times 3$. If the **H** fields are also desired, the `probeF` array can be augmented to size $(N_{steps} + 1) \times N_{probes} \times 6$ to store 3 more fields.

```
    for pr = 1:length(probeE);    % Start of probe loop
        sp = probe(pr,1);    xp = probe(pr,2);

        % Collect data from probe's element
        if GPUflag
```

```
            Esi = gather(Es(:,probeE(pr)));
            Exi = gather(Ex(:,probeE(pr)));
            Eyi = gather(Ey(:,probeE(pr)));
        else
            Esi = Es(:,probeE(pr));
            Exi = Ex(:,probeE(pr));
            Eyi = Ey(:,probeE(pr));
        end

        va = EToV(probeE(pr),1);  % Locate vertex 1 of element
        vb = EToV(probeE(pr),2);  % Locate vertex 2 of element
        vc = EToV(probeE(pr),3);  % Locate vertex 3 of element

        % Find s and x coordinates of element
        Vsx = [VS(va) VX(va); VS(vb) VX(vb); VS(vc) VX(vc)];

        % Interpolate field at probe point
        probeF(tstep+1,pr,1) = DG_Interp(Esi,sp,xp,N,Np,V,Vsx);
        probeF(tstep+1,pr,2) = DG_Interp(Exi,sp,xp,N,Np,V,Vsx);
        probeF(tstep+1,pr,3) = DG_Interp(Eyi,sp,xp,N,Np,V,Vsx);
    end    % End of probe loop
```

Again, if the number of probes: `length(probeE)` is large, it is useful to preprocess the interpolation matrix used in `DG_Interp` and the vertex trio for the element `Vsx`. However, if only a few probes are needed, this probe loop does not significantly impact performance. For a probe point which moves in time, the interpolation matrix and vertex list `Vsx` cannot be preprocessed. Adjustments must be made if the probe point lies in a curvilinear element.

Next, for debugging purposes, it can be useful to visualize the field as it evolves in time. This can be done by specifying a `stride` which indicates to plot the field every `stride` number of steps. In the following code example, we plot the E_x field component every `stride` steps. In the preprocessing section before the timestep loop, we must include:

```
s_i = interp*s;    % Interpolate s onto equally spaced grid
x_i = interp*x;    % Interpolate x onto equally spaced grid
if figflag     % Make empty figure for plotting
    figure;
end
```

These interpolated values of `s` and `x` on an equally spaced grid are used in the routine to plot the output field. This plotting routine can affect performance significantly so it is recommended that this be used for debugging purposes only and with a large enough `stride`.

```
    if ((tstep/stride) == floor(tstep/stride)) || tstep == Nsteps
        if figflag    % Display trisurf plot of field
            Ex_i = interp*gather(Ex);
            trisurf(tri,s_i,x_i,Ex_i)
            xlim([s_min s_max])
            ylim([-s_max/2 s_max/2])
            shading interp
            caxis([-max(abs(Ex_i(:))) max(abs(Ex_i(:)))])
            view(0,90)
            pause(0.01)
        end
        disp(['   Step ' num2str(tstep) ' of ' num2str(Nsteps) ...
            ' complete, elapsed time = ' num2str(toc) 's.'])
    end
```

At this point, all optional routines are completed and the timestepping continues to evolve τ to its final time $\tau_{\max}$ in a total of `Nsteps`. The loop is closed with another `end` statement.

A.4 Postprocessing Routines

In this final section, we discuss the closing routines used to save the data and analyze different aspects. To begin, we collect the necessary data, some of which lies may lie on the GPU, and save the necessary components to a temporary workspace file of extension `.mat`. Note, this will overwrite any previous data so be sure to rename data files after each run.

A.4.1 Cleanup and Save Data

This part is completed in a few short steps. Once the data is saved, it may be loaded with the `load` command for additional postprocessing or plotting. These extra routines may be part of another script as well.

```
if GPUflag    % Extract needed varibles in GPU memory
    Es = gather(Es);    Ex = gather(Ex);    Ey = gather(Ey);
    Hs = gather(Hs);    Hx = gather(Hx);    Hy = gather(Hy);
    Z0 = gather(Z0);    s = gather(s);      x = gather(x);
    alpha = gather(alpha);      dt = gather(dt);
    theta_x = gather(theta_x);
```

```
end

% Clear all variables except the following
clearvars -except Es Ex Ey Hs Hx Hy s x alpha beta ...
    s_min s_max x_in x_out R x_outf q Z0 c p tau_max Ntol ...
    tri interp s_i x_i GPUflag Nv N V Np VS VX EToV ...
    K probe probeF dt tmax figflag theta_x lambda

% Save workspace variables to .mat file
save('DG_recent_data.mat')

if GPUflag
    pause(0.01)
    gpuDevice([]);  % Reset GPU state and clear its memory
end
```

A.4.2 Optional Postprocessing

If we opted to generate a mesh of sub-triangles at to upscale the resolution for the plotting routines. We may plot the fields here using the `trisurf` command. In the code example, we plot the field $E_{yp}(s, x, \tau_{\max})$.

```
if figflag    % Display trisurf plot of field
    Ey_i = interp*Ey;
    figure
    trisurf(tri,s_i,x_i,Ey_i)
    xlim([s_min s_max])
    ylim([-(s_max-s_min)/2 (s_max-s_min)/2])
    shading interp
    caxis([-max(abs(Ex_i(:))) max(abs(Ex_i(:)))])
    view(0,90)
end
```

Another option to consider is to plot a field along a slice. For this example, we take the data for E_{yp} and slice it along the line $s = s_{\text{slice}}$ specified by the user. The x coordinates and the field along the slice are computed with the following routine.

```
function [x_slice,f_slice] = DG_Slice(s_s,xres,s,x,f,...
                                          N,Np,V,VS,VX,EToV,Ntol)
% This subroutine slices the field f at along the x-direction at a
% specific s-value. The routine calls the subroutine DG_ElementFind to
% locate the elements required for interpolation. The routine also
% assumes the domain is convex in x along the slice.
%
%  Inputs:
%  s_s  = slice location in s (1 x 1)
%  xres = resolution desired for slice (1 x 1)
%  s    = LGL node s-coordinates (Np x K)
%  x    = LGL node x-coordinates (Np x K)
%  F    = field values at (s,x) (Np x K)
%  N    = order of elements (1 x 1)
%  Np   = number of nodes per element (1 x 1)
%  V    = Vandermonde matrix (Np x Np)
%  VS   = vector of s-coordinates of vertices (1 x Nv)
%  VX   = vector of x-coordinates of vertices (1 x Nv)
%  EToV = element to vertex mapping matrix (K x 3)
%  Ntol = specified tolerance for node locating (1 x 1)
%
%  Outputs:
```

```
%  x_slice = vector of x-coordinates at s = s_s (xres x 1)
%  f_slice = field values at (s_s,x_slice,t_s) (xres x 1)

if (s_s < min(s(:))) || (s_s > max(s(:)))
    disp('Error: slice located outside domain')
    return
end

f_slice = zeros(xres,1);

% Generate initial x-grid to find x-domain
mwidth = (max(x(:))-min(x(:)))/100;
x_grid = linspace(min(x(:))-mwidth,max(x(:))+mwidth,101)';
x_probe = [ones(101,1)*s_s x_grid];

% Test each point if in interior/exterior of domain
probeE = DG_ElementFind(x_probe,VS,VX,EToV);

% Approximately locate boundaries in x
Lbound = find(probeE(1:end-1)==0 & probeE(2:end)~=0);
Ubound = find(probeE(1:end-1)~=0 & probeE(2:end)==0);

% Intialize intervals for bisection method boundary refine
x_L1 = x_grid(Lbound);
x_L2 = x_grid(Lbound+1);
x_U1 = x_grid(Ubound-1);
x_U2 = x_grid(Ubound);

% Iteration scheme to find lower boundary
while (x_L2-x_L1) > Ntol
```

```
    midp = (x_L1+x_L2)/2;
    test = DG_ElementFind([s_s midp],VS,VX,EToV);
    if test == 0
        x_L1 = midp;
    else
        x_L2 = midp;
    end
end
x_min = x_L2;

% Iteration scheme to find upper boundary
while (x_U2-x_U1) > Ntol
    midp = (x_U1+x_U2)/2;
    test = DG_ElementFind([s_s midp],VS,VX,EToV);
    if test == 0
        x_U1 = midp;
    else
        x_U2 = midp;
    end
end
x_max = x_U1;

% Generate slice domain with correct limits to given tolerance
x_slice = linspace(x_min,x_max,xres)';
x_probe = [ones(xres,1)*s_s x_slice];

% Find elements containing sample points
probeE = DG_ElementFind(x_probe,VS,VX,EToV);

% Sample fields at given coordinates based on elements which contain them
```

```
for pr = 1:xres
    va = EToV(probeE(q),1);
    vb = EToV(probeE(q),2);
    vc = EToV(probeE(q),3);
    sp = x_probe(pr,1);   xp = x_probe(pr,2);
    Vsx = [VS(va) VX(va); VS(vb) VX(vb); VS(vc) VX(vc)];
    f_slice(pr) = DG_Interp(f(:,probeE(pr)),sp,xp,N,Np,V,Vsx);
end


end % End of DG_Slice subroutine
```

This long routine will locate which elements lie along the slice line specified and interpolate the fields at `xres` number of points in x. The result is two vectors: `x_slice` which stores the x coordinates along the slice, and `f_slice` which stores the interpolated values of $f(s_{\text{slice}}, x_{\text{slice}})$. This routine is general enough in that it doesn't assume knowledge of the domain except that it be convex in x, i.e. x_{in} and $x_{\text{out}}(s)$ be single valued with $x_{\text{in}} < x_{\text{out}}(s)$.

Next, we can analyze the frequency content of time-series data with Fourier Transforms. In this example, we examine the frequency content of E_{yp} collected at a probe point `q`. This data was stored in the array `probeF` with the index `(:,q,3)` which appeared in the optional timestepping routine; the `3` corresponded to `Ey`. In the following example, we examine the frequency structure of `Ey` for probe number `q = 2` for frequencies $k = [0.05, 300]\text{m}^{-1}$. The Fourier transform of E_{yp} is stored in the `FTEy` variable. Optionally, the interferogram: the inverse Fourier transform of $|$`FTEy`$|^2$ is computed and stored in `IGEy`.

```
% Post-processing of time-domain signals
taus = (0:dtau:tau_max)';    % Time-domain of simulation
```

```
ks = 0.05:0.05:300;    % User specified frequency range
pmy = probeF(:,2,3);
FTEy = [];
for kk = 1:ceil(length(ks)/100);
    kp = ks(1+(kk-1)*100:min(kk*100,length(ks)));
    [Ks,Taus] = meshgrid(kp,ts);
    intp = exp(-2*pi*1i*(Ks.*Taus)).*repmat(pmy,1,length(kp));
    FTEy = horzcat(FTEy,trapz(taus,intp));
end
IGEy = [];
for tt = 1:ceil(length(ts)/1000);
    tp = taus(1+(tt-1)*1000:min(tt*1000,length(taus)));
    [Ks,Taus] = meshgrid(ks,tp);
    intp = exp(2*pi*1i*(Ks'.*Taus')).*repmat(abs(FTy').^2,1,length(tp));
    IGEy = horzcat(IGEy,trapz(ks,intp));
end
clear Ks Taus intp
```

The odd structure in `for` loops is used for memory limitations. If a large number of wave numbers were computed simultaneously, a memory overflow may occur as the arrays become arbitarily large. These loops limit the number of frequencies computed to 100 at a time. The numerical integration is done with the `trapz` command which uses the trapezoidal rule. The vectors `FTEy` or `IGEy` may be plotted using the `plot` command and `ks` or `taus` respectively as the dependent variable.

Lastly, additional plots can be formed with `contour` for level curves or `quiver` for direction fields. An example for ad-hoc plotting of the **E** direction field can be done as follows:

```
% Quiver plot of E field
Es_i = interp*Es;
Ex_i = interp*Ex;
NE_i = sqrt(Es_i.^2+Ex_i.^2);
figure
quiver(s_i,x_i,Es_i./NE_i,Ex_i./NE_i,'AutoScaleFactor',0.05);
axis equal
```

The `AutoScaleFactor` can be adjusted, but for clarity, if the factor is too large, the plot becomes too cluttered with possibly millions of arrows. It is recommended to only plot desired regions by cropping the arrays for appropriate values of `s_i` and `x_i` within a range.

A.5 Variable and Routine Dependencies

The entire code can be visually setup as a flowchart which lists the inputs and outputs of every routine and the chronological hierarchy of routines. Some routines may be run in parallel to others if they have no mutual dependencies.

A.5.1 List of Variables

We begin with a list of every workspace variable in used in the main code block, some of which are used within subroutines with the same name to avoid confusion. The following table lists every variable name used, its array size (if applicable), and a description with physical or relevant units in parenthesis. Postprocessing and optional variables are omitted; see relevant subroutines for more information.

Name	Size	Description
`alp`	(1×1)	DG Flux parameter (1 = upwinding, 0 = central)
`alpha`	(1×1)	Fourier series y-mode: α_p (m^{-1})
`c`	(1×1)	Speed of light (m/s)
`dEs`	$(3N_{fp} \times K)$	Jump in E_{sp} along element edges: $E_{sp}^- - E_{sp}^+$ (V/m)
`dEx`	$(3N_{fp} \times K)$	Jump in E_{xp} along element edges: $E_{xp}^- - E_{xp}^+$ (V/m)
`dEy`	$(3N_{fp} \times K)$	Jump in E_{yp} along element edges: $E_{yp}^- - E_{yp}^+$ (V/m)
`Dr`	$(N_p \times N_p)$	Differentiation matrix along r reference coordinate
`dHs`	$(3N_{fp} \times K)$	Jump in H_{sp} along element edges: $H_{sp}^- - H_{sp}^+$ (A/m)
`dHx`	$(3N_{fp} \times K)$	Jump in H_{xp} along element edges: $H_{xp}^- - H_{xp}^+$ (A/m)
`dHy`	$(3N_{fp} \times K)$	Jump in H_{yp} along element edges: $H_{yp}^- - H_{yp}^+$ (A/m)
`DsEx`	$(N_p \times K)$	Matrix product for $\partial E_{xp}/\partial s$ (V/m^2)
`DsEy`	$(N_p \times K)$	Matrix product for $\partial E_{yp}/\partial s$ (V/m^2)
`DsHx`	$(N_p \times K)$	Matrix product for $\partial H_{xp}/\partial s$ (A/m^2)
`DsHy`	$(N_p \times K)$	Matrix product for $\partial H_{yp}/\partial s$ (A/m^2)
`Dt`	$(N_p \times N_p)$	Differentiation matrix along t reference coordinate
`dtau`	(1×1)	Stable timestep $\Delta\tau$ for evolution (m)
`DxEs`	$(N_p \times K)$	Matrix product for $\partial E_{sp}/\partial x$ (V/m^2)
`DxEy`	$(N_p \times K)$	Matrix product for $\partial E_{yp}/\partial x$ (V/m^2)
`DxHs`	$(N_p \times K)$	Matrix product for $\partial H_{sp}/\partial x$ (A/m^2)
`DxHy`	$(N_p \times K)$	Matrix product for $\partial H_{yp}/\partial x$ (A/m^2)
`EToE`	$(K \times 3)$	Element-to-element connectivity map
`EToF`	$(K \times 3)$	Element-to-face (edge) connectivity map
`EToV`	$(K \times 3)$	Element-to-vertex connectivity map
`Es`	$(N_p \times K)$	Array storing values E_{sp} at nodes (V/m)

Name	Size	Description
`Esu`	$(N_p \times K)$	Residual storage array for E_{sp} (V/m)
`Ex`	$(N_p \times K)$	Array storing values E_{xp} at nodes (V/m)
`Ex_0`	$(N_p \times K)$	Initial condition for E_{xp} at $\tau = 0$ at nodes (V/m)
`Ex_S`	$(N_p \times K)$	Source term for RHS of $dE_{xp}/d\tau$ (A)
`Exu`	$(N_p \times K)$	Residual storage array for E_{xp} (V/m)
`Ey`	$(N_p \times K)$	Array storing values E_{xp} at nodes (V/m)
`Ey_0`	$(N_p \times K)$	Initial condition for E_{yp} at $\tau = 0$ at nodes (V/m)
`Eyu`	$(N_p \times K)$	Residual storage array for E_{yp} (V/m)
`f1`	(1×1)	Multiplicative factor used in $\lambda(s)$ (m^{-1})
`f2`	(1×1)	Multiplicative factor used in $\lambda(s)$ and $\lambda'(s)$ (m^{2})
`f3`	(1×1)	Multiplicative factor used in $\lambda'(s)$ (m^{-3})
`figflag`	(1×1)	Flag for running option plotting routines (0 or 1)
`fluxEs`	$(3N_{fp} \times K)$	Numerical flux term for E_{sp} PDE (V/m)
`fluxEx`	$(3N_{fp} \times K)$	Numerical flux term for E_{xp} PDE (V/m)
`fluxEy`	$(3N_{fp} \times K)$	Numerical flux term for E_{yp} PDE (V/m)
`fluxHs`	$(3N_{fp} \times K)$	Numerical flux term for H_{sp} PDE (A/m)
`fluxHx`	$(3N_{fp} \times K)$	Numerical flux term for H_{xp} PDE (A/m)
`fluxHy`	$(3N_{fp} \times K)$	Numerical flux term for H_{yp} PDE (A/m)
`Fm`	$(N_{fp} \times 3)$	Reference element array of edge indices
`Fsc`	$(3N_{fp} \times K)$	Ratio of edge-to-area Jacobian by element
`Gp`	(1×1)	Fourier coefficient for $G(y)$ mode p (m^{-1})
`GPUflag`	(1×1)	Flag for using GPU processing (0 or 1)
`h`	(1×1)	Height of vacuum chamber in y (m)
`h0`	(1×1)	Scaled edge length for mesh generation (m)

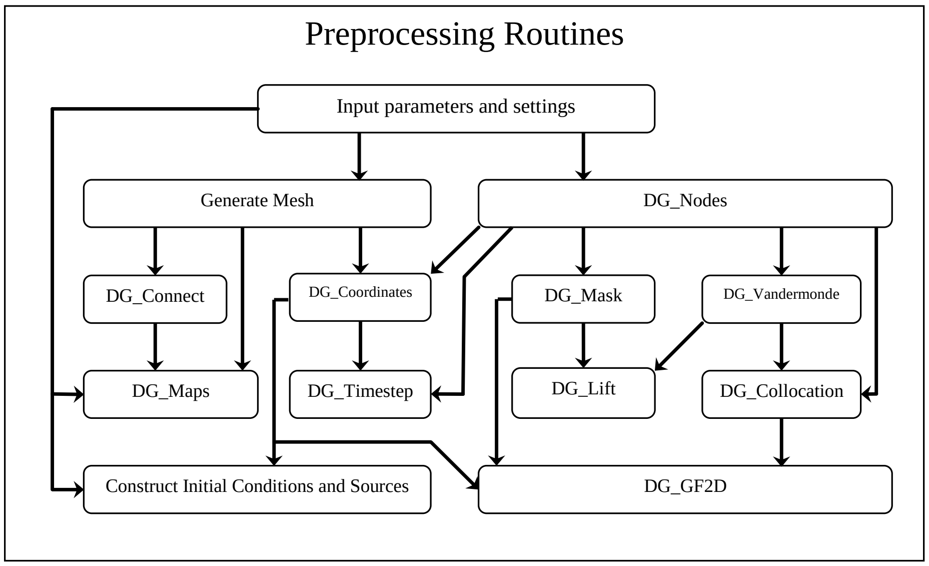
Name	Size	Description
`Hs`	$(N_p \times K)$	Array storing values H_{sp} at nodes (A/m)
`Hsu`	$(N_p \times K)$	Residual storage array for H_{sp} (A/m)
`Hx`	$(N_p \times K)$	Array storing values H_{xp} at nodes (A/m)
`Hxu`	$(N_p \times K)$	Residual storage array for H_{xp} (A/m)
`Hy`	$(N_p \times K)$	Array storing values H_{yp} at nodes (A/m)
`Hy_S`	$(N_p \times K)$	Source term for RHS of $dH_{yp}/d\tau$ (V)
`Hyu`	$(N_p \times K)$	Residual storage array for H_{yp} (A/m)
`interp`	$(N_{pout} \times N_p)$	Interpolation matrix to N_{pout} equally spaced nodes
`IoZ0`	(1×1)	Scalar quantity: $1/Z_0$ (Ω^{-1})
`IoxpR`	$(N_p \times K)$	Array quantity: $1/(x+R) = \kappa/(1+\kappa x)$ (m^{-1})
`K`	(1×1)	Total number of elements
`kap`	(function)	Function handle for curvature: $\kappa(s)$ (m^{-1})
`kap_s`	$(N_p \times K)$	Array evaluation of $\kappa(s)$ at nodes (m^{-1})
`lambda`	(function)	Function handle for $\lambda(s)$ distribution (m^{-1})
`lambda`	(function)	Function handle for $\lambda'(s)$ distribution (m^{-2})
`Lift`	$(N_p \times 3N_{fp})$	DG surface to volume operator
`mB`	(varies$\times$1)	Global edge nodal numbers along boundary
`N`	(1×1)	Polynomial order of elements N
`ndotdE`	$(3N_{fp} \times K)$	Array quantity $\mathbf{n} \cdot \mathbf{E}$ jumps along element edges
`ndotdH`	$(3N_{fp} \times K)$	Array quantity $\mathbf{n} \cdot \mathbf{H}$ jumps along element edges
`Nfp`	(1×1)	Number of nodes along faces (edges): N_{fp}
`Nout`	(1×1)	Output polynomial order for upscaling plots
`Np`	(1×1)	Number of nodes per element: N_p
`ns`	$(3N_{fp} \times K)$	s-component of normal vector $\mathbf{n}$ along edges

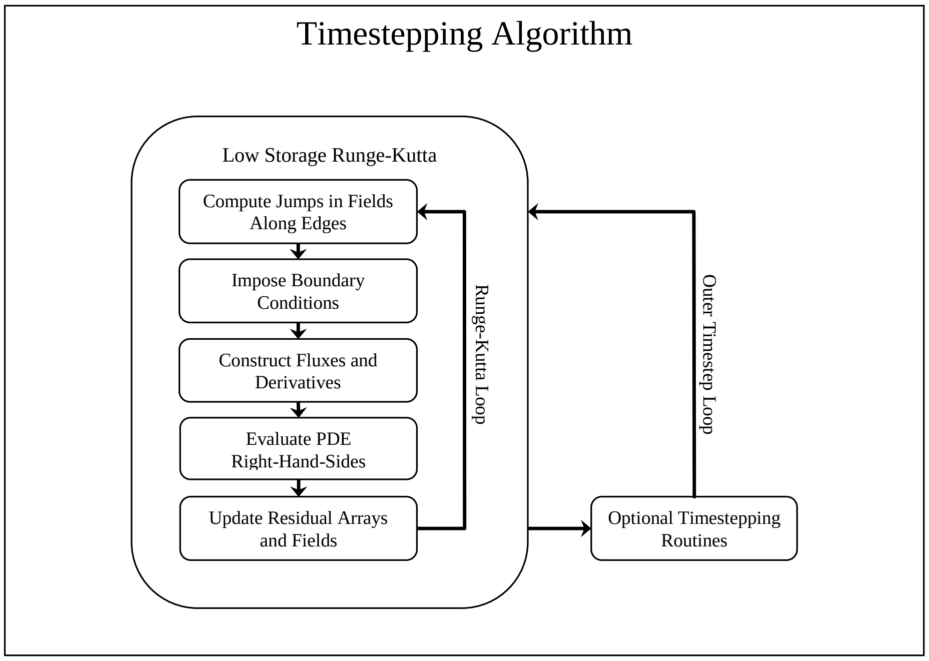
Name	Size	Description
`Nsteps`	(1×1)	Total number of timesteps: $\tau_{\max}/\Delta\tau$
`Ntol`	(1×1)	Tolerance for node locating algorithms
`Nv`	(1×1)	Total number of vertices in mesh: N_v
`nx`	$(3N_{fp} \times K)$	x-component of normal vector $\mathbf{n}$ along edges
`p`	(1×1)	Fourier mode number in y-direction: p
`r`	$(N_p \times 1)$	Reference triangle node coordinate: r
`rhsEs`	$(N_p \times K)$	Array for RHS of $dE_{sp}/d\tau$ (V/m/s)
`rhsEx`	$(N_p \times K)$	Array for RHS of $dE_{xp}/d\tau$ (V/m/s)
`rhsEy`	$(N_p \times K)$	Array for RHS of $dE_{yp}/d\tau$ (V/m/s)
`rhsHs`	$(N_p \times K)$	Array for RHS of $dH_{sp}/d\tau$ (A/m/s)
`rhsHx`	$(N_p \times K)$	Array for RHS of $dH_{xp}/d\tau$ (A/m/s)
`rhsHy`	$(N_p \times K)$	Array for RHS of $dH_{yp}/d\tau$ (A/m/s)
`rk4a`	(1×5)	Vector of LSERK butcher table coefficients a
`rk4b`	(1×5)	Vector of LSERK butcher table coefficients b
`rk4c`	(1×5)	Vector of LSERK butcher table coefficients c
`RoxpR`	$(N_p \times K)$	Array quantity: $R/(x+R) = 1/(1+\kappa x)$
`RoxpRE`	$(3N_{fp} \times K)$	Array quantity: $R/(x+R) = 1/(1+\kappa x)$ along edges
`rs`	$(N_p \times K)$	Array for Jacobian metric: $\partial r/\partial s$ (m^{-1})
`rx`	$(N_p \times K)$	Array for Jacobian metric: $\partial r/\partial x$ (m^{-1})
`s`	$(N_p \times K)$	Array of physical coordinates s at nodes (m)
`s_max`	(1×1)	Position of end of vacuum chamber: $s_{\max}$ (m)
`s_min`	(1×1)	Position of start of vacuum chamber: $s_{\min}$ (m)

Name	Size	Description
`sig_s`	(1×1)	Bunch length in s: σ_s (m)
`t`	$(N_p \times 1)$	Reference triangle node coordinate: t
`t_delay`	(1×1)	Shift offset for initial source position (m)
`tau`	(1×1)	Current time in timestepping loop: τ (m)
`tau_max`	(1×1)	Reference triangle node coordinate: $\tau_{\max}$ (m)
`theta_x`	$(N_p \times K)$	Evaluation of Heaviside function at nodes: $\Theta(x)$
`ts`	$(N_p \times K)$	Array for Jacobian metric: $\partial t/\partial s$ (m^{-1})
`tx`	$(N_p \times K)$	Array for Jacobian metric: $\partial t/\partial x$ (m^{-1})
`V`	$(N_p \times N_p)$	Vandermonde matrix for LGL polynomial basis
`vB`	(varies$\times$1)	Global nodal numbers along boundary
`vM`	$(3N_{fp}{\cdot}K{\times}1)$	Global nodal numbers along interior element edges
`vP`	$(3N_{fp}{\cdot}K{\times}1)$	Global nodal numbers along exterior element edges
`VS`	$(1 \times N_v)$	Vector of s-coordinates of mesh vertices (m)
`VX`	$(1 \times N_v)$	Vector of x-coordinates of mesh vertices (m)
`x`	$(N_p \times K)$	Array of physical coordinates x at nodes (m)
`x_in`	(1×1)	Inner wall position of vacuum chamber: x_{in} (m)
`x_max`	(1×1)	Maximum of $x_{\text{out}}(s)$ in vacuum chamber (m)
`x_out`	(1×1)	Initial outer wall position: $x_{\text{out}}(0)$ (m)
`Z0`	(1×1)	Impedance of free space: Z_0 (Ω)

A.5.2 Flowchart of Routines

In this section we visually outline the hierarchy of the steps outlined in sections 2–3 with optional routines omitted. The variable dependencies can be viewed in the respective header comments in each routine.




Timestepping Algorithm
Low Storage Runge-Kutta
Compute Jumps in Fields Along Edges
Impose Boundary Conditions
Construct Fluxes and Derivatives
Evaluate PDE Right-Hand-Sides
Update Residual Arrays and Fields
Runge-Kutta Loop
Outer Timestep Loop
Optional Timestepping Routines

References

[1] T. Agoh, *Dynamics of Coherent Synchrotron Radiation by Paraxial Approximation*, Ph.D. Dissertation, University of Tokyo, December (2004).

[2] T. Agoh and K. Yokoya, "Calculation of coherent synchrotron radiation using mesh", Phys. Rev. ST Accel. Beams **7**, 054403 (2004).

[3] G. Bassi, J. A. Ellison, K. Heinemann, R. Warnock, "Microbunching Instability in a Chicane: Two-Dimensional Mean Field Treatment", Phys. Rev. ST Accel. Beams **12**, 080704 (2009).

[4] B. E. Billinghurst, J. C. Bergstrom, C. Baribeau, T. Batten, L. Dallin, T. E. May, J. M. Vogt, W. A. Wurtz, R. Warnock, D. A. Bizzozero, and S. Kramer, "Observation of Wakefields and Resonances in Coherent Synchrotron Radiation", Phys. Rev. Lett. **114**, 204801 (2015).

[5] D. Bizzozero, J. A. Ellison, K. A. Heinemann, S. R. Lau, "Paraxial Approximation in CSR Modeling Using the Discontinuous Galerkin Method", Proceedings of FEL13, New York, August 2013. Paper available at: `http://accelconf.web.cern.ch/AccelConf/FEL2013/papers/mopso06.pdf`

[6] D. Bizzozero, J. A. Ellison, R. Warnock "Modeling CSR in a Vacuum Chamber by Partial Fourier Analysis and the Discontinuous Galerkin Method", Proceedings of FEL14, Basel, Switzerland, August 2014. Paper available at: `http://accelconf.web.cern.ch/AccelConf/FEL2014/papers/tup023.pdf`

[7] B. Cockburn, "Discontinuous Galerkin Methods", ZAMM, Journal of Applied Mathematics and Mechanics, Volume 83, Issue 11, p. 731–754, November (2003).

[8] R. Cools, "Monomial Cubature Rules Since 'Stroud': a Compilation – Part 2", J. Comput. Appl. Math., 112(1-2), 21–27 (1999).

[9] L. Evans, *Partial Differential Equations* (American Mathematical Society, 2010).

[10] K. Heinemann, D. Bizzozero, J. A. Ellison, S. R. Lau, G. Bassi, "Rapid integration over history in self-consistent 2D CSR modeling", Proceedings of ICAP2012, Rostock-Warnemunde, Germany, August 2012. See jacow.org for the paper and slides at TUSDC2, i.e., `http://accelconf.web.cern.ch/AccelConf/ICAP2012/papers/tusdc2.pdf`

[11] J. Hesthaven and T. Warburton, *Nodal Discontinuous Galerkin Methods* (New York: Springer, 2008).

[12] J. Jackson, *Classical Electrodynamics, Third Edition* (New York: Wiley, 1998).

[13] C. Johnson, *Numerical Solution of Partial Differential Equations by the Finite Element Method* (Mineola N.Y.: Dover, 2009).

[14] A. Klöckner, T. Warburton, J. Bridge, J. S. Hesthaven, "Nodal discontinuous Galerkin methods on graphics processors", J. Comput. Phys. 228, Issue 21, 7863–7882 (2009).

[15] A. Klöckner, T. Warburton, J. S. Hesthaven, "High-Order Discontinuous Galerkin Methods by GPU Metaprogramming" in *GPU Solutions to Multi-scale Problems in Science and Engineering*, edited by D. A. Yuen, L. Wang, X. Chi, L. Johnsson, W. Ge, and Y. Shi (Springer, 2013). Also available as `arXiv:1211.0582 [cs.MS]`.

[16] R. McOwen *Partial Differential Equations – Methods and Applications*, (Upper Saddle River, NJ: Pearson Education, 2003).

[17] P. -O. Persson, G. Strang, "A Simple Mesh Generator in MATLAB", SIAM Review, Volume 46 (2), 329–345, (2004).

[18] B. Rivière, *Discontinuous Galerkin Methods for Solving Elliptic and Parabolc Equations* (Philadelphia: SIAM, 2008).

[19] J. Strikwerda, *Finite Difference Schemes and Partial Differential Equations* (Pacific Grove, CA: Wadsworth & Brooks/Cole Advanced Books & Software, 1989).

[20] G.V. Stupakov and I.A. Kotelnikov, "Calculation of coherent synchrotron radiation impedance using the mode expansion method", Phys. Rev. ST Accel. Beams **12**, 104401 (2009).

[21] Y. Xu and C. Shu, "Local Discontinuous Methods for Nonlinear Schrödinger Equations", J. Comput. Phys. 205, Issue 1, 72–97 (2005).

[22] D. Zhou, *Coherent Synchrotron Radiation and Microwave Instability in Electron Storage Rings*, Ph.D. Dissertation, The Graduate University for Advanced Studies, September (2011).

[23] D. Zhou, K. Ohmi, K. Oide, L. Zang, and G. Stupakov, "Calculation of Coherent Synchrotron Radiation Impedance for a Beam Moving in a Curved Trajectory", Jpn. J. Appl. Phys. 51, 016401 (2012).