The Pennsylvania State University
The Graduate School

RAYLEIGH STREAMING SIMULATION USING THE VORTICITY TRANSPORT EQUATION

A Thesis in
Acoustics
by
Debbie Sastrapradja

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2004

The thesis of Debbie Sastrapradja was reviewed and approved* by the following:

Victor W. Sparrow
Associate Professor of Acoustics
Thesis Adviser
Chair of Committee

Anthony A. Atchley
Professor of Acoustics
Head of the Graduate Program in Acoustics

Philip J. Morris
Boeing/A.D. Welliver Professor of Aerospace Engineering

Cengiz Camci
Professor of Aerospace Engineering

*Signatures are on file in the Graduate School.

Abstract

One part of understanding thermoacoustic devices involves studying a physical phenomenon called **acoustic streaming**, a steady fluid flow induced by oscillating acoustic waves. Current numerical calculation of acoustic streaming can involve major computing time and resources. In order to develop a quicker model, the vorticity transport equation (VTE) is used. The goal of using the VTE is to obtain a relatively fast solution with minimal computational resources, which in this case is a single PC. The intent of this method is that it is used in the early design stage of thermoacoustic devices where preliminary (although less detailed) fast results are desired. It is also preferred that the computing power be minimized as not to tie up other resources for the optimized design of thermoacoustic devices.

The most well known type of acoustic streaming, Rayleigh streaming, is simulated using the VTE method. A clustered grid is utilized to capture the boundary layer effect on the acoustic streaming. The governing equations used are the VTE, Poisson's equation, and an equation that relates the stream function with the velocity. The outline of the method of calculation involves (i) generating a clustered grid and ensuring there are enough points in the boundary layer, (ii) transforming the clustered grid into the uniform computational grid, (iii) transforming the governing equations to account for the clustering, (iv) calculating the vorticity and the stream function at each grid point using a Direct Method, and (v) calculating the acoustic streaming velocity using the stream function. Steps (iv) through (v) are repeated until the solution converges.

It is demonstrated that the VTE method to calculate Rayleigh streaming works well. There are two cases being simulated in the research, a parallel plate case and a cylindrical tube case. The numerical results agree with the analytical results for both cases, although there are some discrepancies in the cylindrical tube case. At this time, no numerical reason can be given to explain the discrepancies. The goal to perform the simulation fairly quickly with a single PC has been achieved.

Table of Contents

List of Figures

List of Tables

Acknowledgments

Work supported, in part, by Office of Naval Research Grant N00014-99-1-0921.

I would like to express my great appreciations to my adviser, Dr. Victor W. Sparrow, who have guided me throughout the entire process of the Ph.D. program. His help and patience is invaluable. His unwavering support, tolerance and attention to detail have made the journey seems a little easier. I would like to thank Dr. Philip J. Morris for his advice and suggestions in the research and the thesis writing process. His inputs have been very valuable. I would like to express my gratitude to Dr. Cengiz Camci for his immense help in solving the boundary condition problem and for his generosity in lending me the material needed to complete this research. I would like to thank Dr. Anthony A. Atchley for his time in serving as the committee member. For Dr. Said Boluriaan, I would like to extend my appreciation for his help in this research.

I am also very grateful to have the support of Efrem R. Reeves, who has been very patience, supportive, and encouraging towards the entire course of my study, academically and beyond. His encouragement has helped me a great deal during those challenging times. To him I am deeply indebted beyond words. I would like to express my gratitude for my parents for their endless support and encouragement. Without them I would have never imagined myself being at this stage in my life.

I also would like to thank my friends, Vernecia Sharae' McKay, Umesh Paliath, Preetham Rao, Anurag Agarwal, Yi Pin Liew, and Brian Tuttle for making my stay at the lab enjoyable.

Finally, thank you God, without whom nothing would be possible.

Chapter 1

Introduction

1.1 Acoustic Streaming

Acoustic streaming is defined as a second order nonlinear effect where the presence of sound waves generates mean mass flow. A quantitative knowledge of acoustic streaming is important for the design of thermoacoustic devices, and this will be explained later. This research focuses on the efficient calculation of acoustic Rayleigh streaming. Before the method is explained, it is important to review some concepts of acoustic streaming.

There are different types of acoustic streaming and also several different ways to categorize the different types. This introduction will look at the acoustic streaming categorized based on their driving mechanism. The few mechanisms that drive acoustic streaming are: (1) viscous stresses on boundaries, (2) acoustic energy dissipation, (3) jet-driven streaming, and (4) traveling wave streaming. A very thorough review of several categories of acoustic streaming from the time when Rayleigh first presented his analysis until the present day has been written by Boluriaan and Morris [4]. The next sections follow the acoustic streaming classification of that article.

1.1.1 Viscous stresses on boundaries-driven streaming

The earliest record on this type of streaming was found when Faraday [8] did an experiment with a vibrating plate in 1831. He found that there is a pattern

of steady vortices of the air in the vicinity of the plate. With the aid of very fine powder he was able to observe the movement of air from the maximum to the minimum displacement of the plate. Faraday explained that the vibration of the plate caused the existence of a boundary layer that caused the air to move. In another experiment Dvorak observed the gas motion inside a Kundt's tube. Although the air motion due to vibration phenomena had been observed by these individuals, there had been no mathematical explanation offered. Rayleigh [41] was the first to explain the underlying physical phenomena with the successive approximation method. He explained that the air motion is caused by a nonlinear second order effect. He analyzed acoustic streaming when a standing wave is present between parallel walls. This was the first mathematical explanation of the mass streaming phenomenon, and Rayleigh's analysis became the base of many analyses that followed. As a result, boundary layer generated streaming is often referred to as "Rayleigh streaming". This type of acoustic streaming originates from the tangential motion near the boundary area caused by the sound field. In his review of acoustic streaming, Nyborg [27] listed some of the analyses on mass streaming generated by viscous stresses on the boundaries. Table 1.1 lists the mass streaming sources and the individuals who studied the phenomena.

Streaming source	Investigators
Vibrating body (such as cylinders and spheres)	Andrade [2] in 1931, Schlichting [35, 36] in 1932 and 1955, Holtzmark et al. [14] in 1954, Raney et al. [30] in 1954, and Skavlem and Tjotta [38] in 1955
Kundt's tube	Schuster and Matz [37] in 1940, Thompson and Atchley in 2004 [44]
Orifice traversed by sound	Ingard and Labate [15] in 1950
Vibrating membranes	Kolb and Nyborg [20] in 1956, Jackson and Nyborg [17] in 1958
Vibrating gas bubbles	Elder [7] in 1959
The space between a plane solid boundary and the end of a vibrating bar	Jackson [16] in 1960

Table 1.1. Streaming sources and the people who studied them

An illustration of streaming near a cylinder is shown in Fig. 1.1.

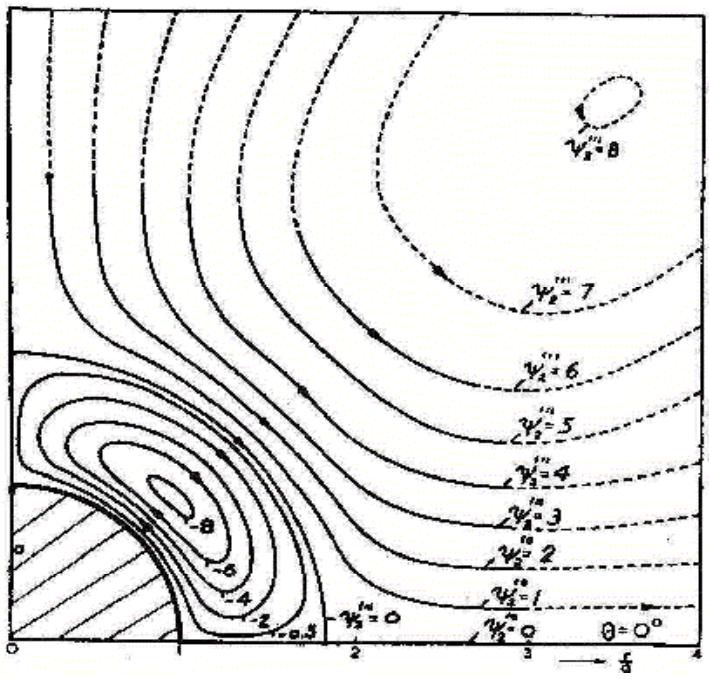

Figure 1.1. Streamlines in a quadrant of the field near a cylinder oscillating in a fluid. Oscillation occurs in the horizontal direction. From Holtzmark *et al.* [14].

Figure 1.2 shows the distribution of the first order (axial) acoustic velocity (u_1), the x component of the forcing function (F_x), and the axial streaming velocity (U) for a standing wave in a channel. "A" indicates the antinode and "N" indicates the node of the standing wave.

Westervelt [49] later corrected Rayleigh's solution for streaming near a boundary. He derived an expression called the velocity transform or Stokes drift that relates *Eulerian* streaming velocity to the *Lagrangian* streaming velocity [50]. The Eulerian velocity is the particle velocity observed at a fixed point. This is the velocity that would actually be measured in experiments. The Lagrangian velocity is the velocity experienced by a particular particle as it moves around in a medium. The Lagrangian velocity can be measured by flow visualization based on time-lapse photography recording the path of particles as they move along with the flow [50].

Away from the boundary, streaming in the main body is called *outer streaming*. Inside the boundary layer near the wall, there is another streaming cell type called

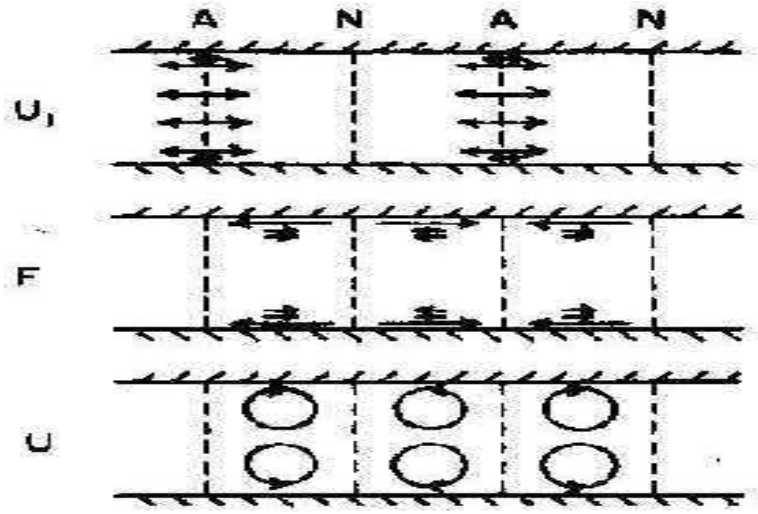

Figure 1.2. Distribution of u_1, F, and U for standing waves in channel. From Nyborg [27].

inner streaming. In his analysis, Rayleigh did not include the details of inner acoustic streaming near the boundary. Schlichting [35, 36] was the first to study inner streaming. Figure 1.3 shows the inner and outer streaming in a channel.

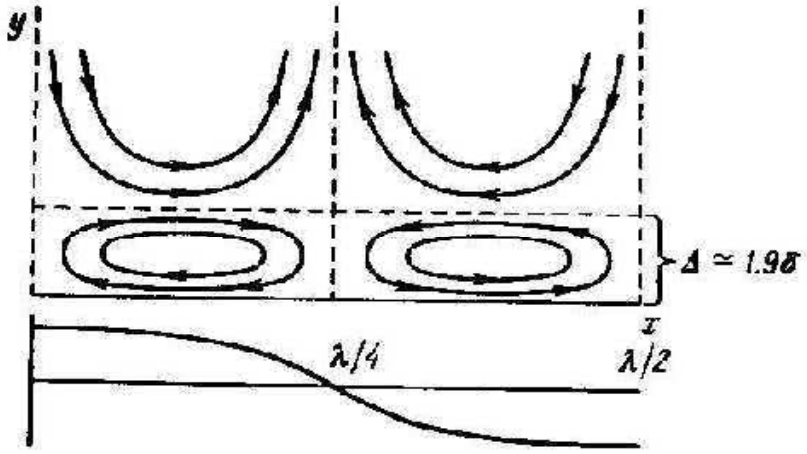

Figure 1.3. Inner and outer streaming in a channel. From Rudenko and Soluyan [34].

Lighthill [22] attempted to rectify misconceptions about acoustic streaming. He stated that the classical treatment of acoustic streaming (caused by Reynolds stress forcing resisted by viscosity), did not take into account the effect of the fluid's inertia on the streaming motion. He explained that the fundamental principle in most acoustic streaming is that the attenuation of acoustic energy flux makes momentum flux available to force streaming motion. The main intention of his lecture actually was to explain how turbulent jets are generated by sound. Along with his explanation about the turbulent jets phenomenon, he clarified the difference between the Lagrangian and Eulerian motion in the acoustic streaming. Riley [32] followed the path laid by Rayleigh and Lighthill. He and his research group developed a theory for streaming in incompressible flow which is valid for large values of a defined Strouhal number. The assumption of incompressible flow requires the acoustics condition to be $kl << 1$, where k is the wave number and l is a typical length. The flow region is divided into the outer and inner (or Stokes-layer) regions. The steady streaming in the Stokes-layer region is due to Reynolds stresses, and continues toward the edge (Rayleigh's law of streaming). In the outer region the streaming motion is governed by Helmholtz-type equations.

Rott [33], Waxler [48], Bailliet *et al.* [3], and Hamilton *et al.* [11], include some thermal effects in their streaming analyses. Rott [33] includes the effect of heat conduction, the dependence of viscosity on temperature, and the effect of a temperature gradient imposed on the wall on the streaming velocities. Waxler [48] includes the effect of heat conduction on streaming between parallel plates, while Bailliet *et al.* [3] include the heat conduction and the dependence of viscosity on temperature on streaming between parallel plates and cylindrical tubes. Both analyses impose a mean temperature gradient along the channel walls. Hamilton *et al.* [12] take into account the heat conduction and viscosity dependency on temperature effects, but leave out the effect of temperature gradient along the wall on the streaming.

1.1.2 Acoustic energy dissipation-driven streaming

The second type of acoustic streaming is generated by the dissipation of acoustic energy in the fluid. This type of streaming is also known as the "quartz wind" or

"Eckert streaming". Here, streaming is caused by an ultrasonic sound beam of high amplitude in a medium where the acoustic energy is dissipated in the main body of the fluid [31]. Riley [32] describes the acoustic energy dissipation streaming based on Lighthill [22]'s paper as:

> The dissipation of acoustic energy flux which permits the gradients in momentum flux that force the acoustic streaming motion.

Figure 1.4 illustrates the streamlines for the DC flow produced by a focused sound beam radiated by circular pistons.

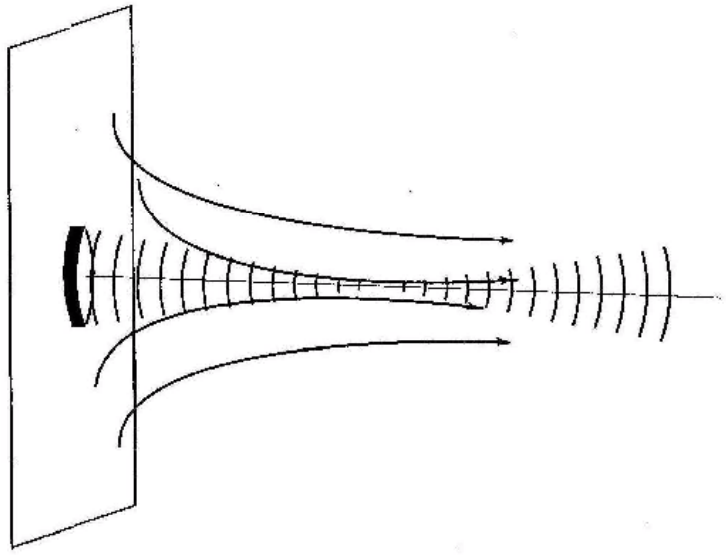

Figure 1.4. Illustration of streamlines for the DC flow produced by a focused sound beam. From Steven J. Younghouse M.S. Thesis [50].

Although he was not the one who made the first observation, Eckert [6] was the first one who gave a mathematical analysis for the quartz wind in 1948. He derived the solution for mass streaming that occurred in a closed tube when an intense sound beam is projected into a fluid. Eckert showed that quartz wind is caused by the sound absorption attributed to the fluid viscosity. Figure 1.5 illustrates the acoustic streaming pattern inside a closed volume where the sound beam radius is denoted by r_1 and the cylindrical tube radius is denoted by r_0. In Fig. 1.6 we can see the time progression of Eckert streaming in a channel. Markham [23]

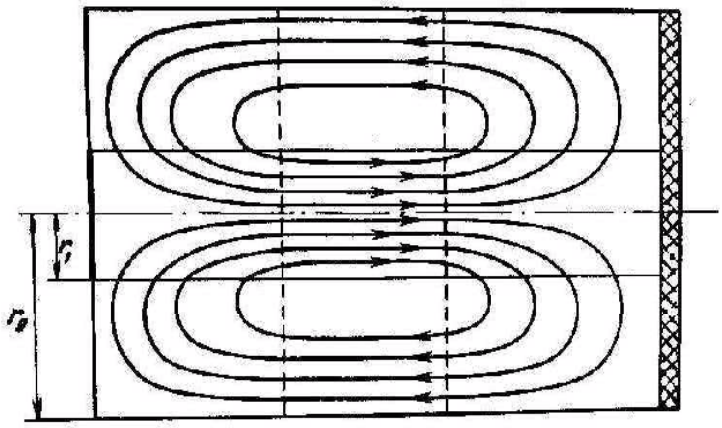

Figure 1.5. Streaming in a closed volume (with rigid walls). The region of Eckert streaming is indicated by the parallel dashed lines. From Rudenko and Soluyan [34].

later showed that absorption could be from viscosity alone or from viscosity and relaxation. According to Nyborg's review [27], the earlier observations of quartz wind were made by Meissner [25] in 1926 (in liquid) and Walker and Allen [47] in 1950 (in air). One year after Eckert published his analysis, Liebermann [21] came out with the report on his experimental results. In terms of theoretical and experimental works, the groups of researchers that presented their findings on quartz winds are listed in Table 1.2.

Type of observation	Investigators
Theoretical	Fox and Herzfeld [9] in 1950, Markham [23] in 1952, Nyborg [27] in 1953, Doak [5] in 1954, and Truesdell[46] in 1954
Experimental	Lieberman [21] in 1949, Piercy and Lamb [28] in 1954, Medwin [24] in 1954, and Johnsen and Tjotta [18] in 1957

Table 1.2. Theoretical and experimental studies of energy dissipation driven streaming.

Acoustic streaming where the velocity is considerably smaller than the fluid particle velocity can be categorized as slow streaming. Lighthill [22] explained that the assumption made by Rayleigh, Westervelt, Eckert and Nyborg, on the effect of the fluid's own inertia on the resultant streaming being negligibly small, is only valid for very slow streaming cases. To linearize the momentum equation

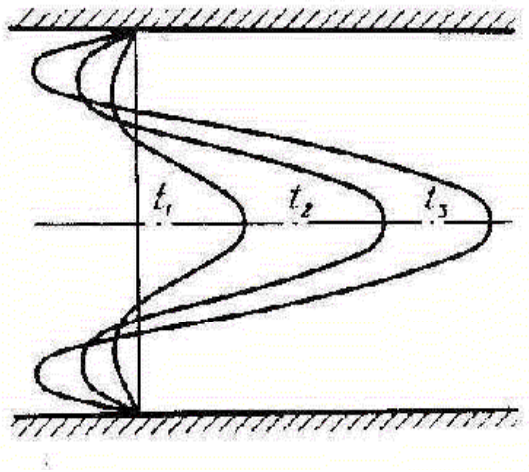

Figure 1.6. Temporal process of Eckert streaming establishment. Rudenko and Soluyan [34].

for the streaming calculation the Reynolds number corresponding to the DC flow must be less than unity [4]. Lighthill pointed out that it is the dissipation of acoustic energy flux which permits the gradients in momentum flux that force the acoustic streaming motions.

1.1.3 Jet-driven streaming

Swift [42] explained in his book, *"Thermoacoustics: A Unifying Perspective For Some Engines and Refrigerators"*, that if a tube has a small diameter entrance at one of its ends, a jet of air may blow into the tube when gas enters, driving time-averaged fluid motion inside the tube. Straightening the flow can break up the jets but it dissipates acoustic power. More research needs to be done to design flow straighteners that will optimize the efficiency of thermoacoustic devices. Figure 1.7 illustrates the suction and ejection periods of an orifice. In Fig. 1.7 (a) the streamlines shows how the fluid enters the orifice, while in (b) the flow separates to produce a jet [4].

Figure 1.8 shows a case where a tube has a small diameter entrance causing a jet to blow into the tube when gas enters the tube [39], driving a time-averaged convection within the tube. The downflow in the first illustration is concentrated in the center of the tube, while the upflow shown in the second illustration is

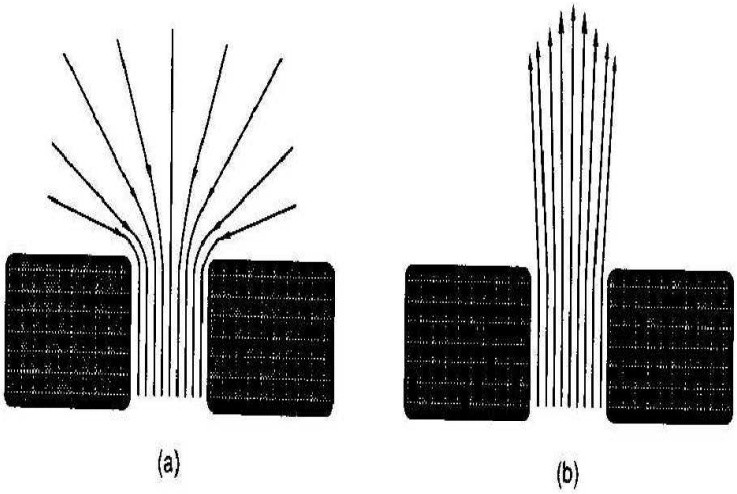

Figure 1.7. Oscillatory flow pattern in an orifice (a) suction period (b) ejection period. From Boluriaan and Morris [4].

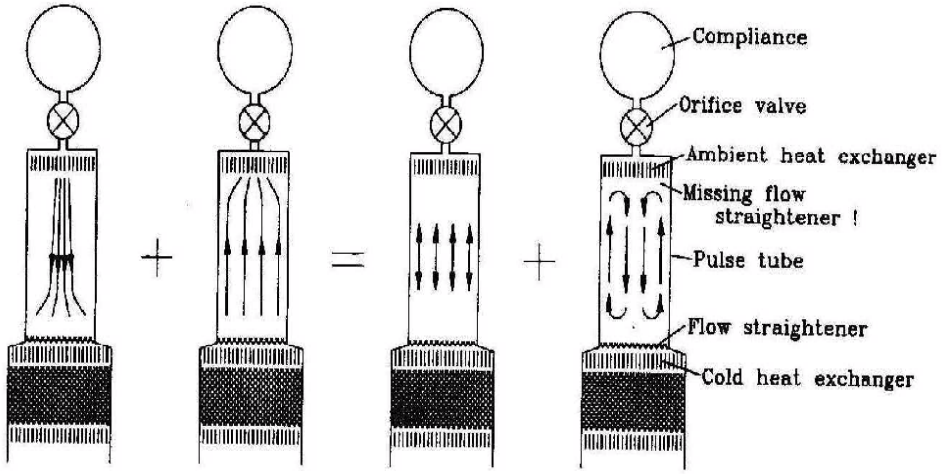

Figure 1.8. Jet-driven streaming in pulse tube caused by a missing flow straightener at the top of the pulse tube. From Swift [42].

broadly distributed. These two flows can be regarded as the superposition of a broadly distributed oscillating flow and a time-averaged toroidal circulation shown in third and fourth illustrations [42].

1.1.4 Traveling wave-driven streaming

Due to the phasing between the acoustic velocity and density, in an inviscid lossless flow, mass streaming can still occur. A traveling wave is a mechanism other than sound absorption that can generate acoustic streaming. Figure 1.9 shows the distribution of the forcing function F and the streaming velocity U when a plane traveling wave fills a closed channel. The boundary condition is assumed to be no-slip. Gedeon [10] analyzed and presented a theoretical calculation of acoustic streaming in Stirling and pulse tube thermoacoustic refrigerators. Streaming caused by a traveling wave is often referred to as "Gedeon streaming". Gedeon streaming is a net time-averaged mass flow along the axis of a regenerator, pulse tube, etc. This time-averaged mass flux is proportional to the acoustic intensity. This type of mass streaming can add a thermal load to the cold heat exchanger in a refrigerator or removes heat from the hot heat exchanger in a thermoacoustic engine. pulse tube, etc. It is important that the time-averaged mass flow M in the axial direction through a regenerator, pulse tube, stack, etc. should be near zero to prevent a large time-averaged convective enthalpy flux from flowing from hot to cold. In a refrigerator, such a steady energy flux adds an unwanted thermal load to the cold heat exchanger and in an engine, it wastefully removes high-temperature heat from the hot heat exchanger without creating acoustic power [42].

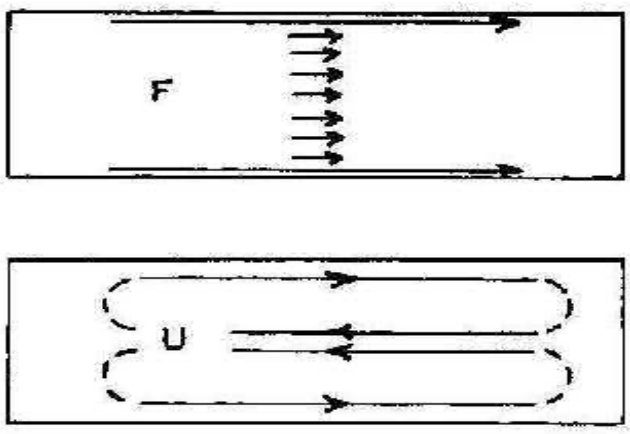

Figure 1.9. Distribution of F and U for plane traveling wave which fills closed channel (no-slip condition at the walls). From Nyborg [27].

1.2 Research Motivation

As discussed in section 1.1, quite a few researchers have performed acoustic streaming calculations both analytically and numerically since Rayleigh's first analysis. The focus of this thesis is on the numerical calculation of acoustic streaming due to a standing wave in a channel and a cylindrical tube. This research is motivated by the occurrence of acoustic streaming in thermoacoustic devices due to the high amplitude acoustics being used to drive those devices. The effect of acoustic streaming on such devices can either be detrimental or beneficial [42]. When streaming occurs in a thermoacoustic device, the fluid flow can carry heat away from the desirable heat transfer direction and decreases the performance of the engine. While in other cases, the flow can carry heat in the correct direction to aid the effectiveness of heat transfer in the engine. Figure 1.10 shows some types of streaming that can be harmful to thermoacoustic engines and refrigerators, while Fig. 1.11 shows the type of streaming that can be beneficial. The design of a thermoacoustic device can be optimized if the behavior of acoustic streaming in such a device is known a priori.

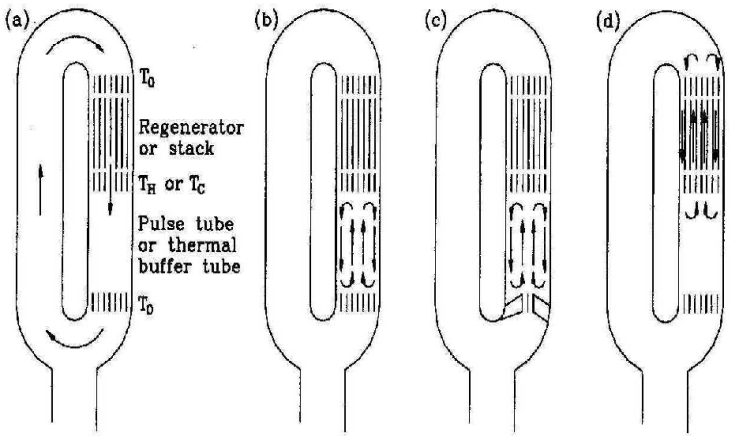

Figure 1.10. Types of mass streaming that are generally harmful to thermoacoustic engines and refrigerators. Arrows indicate the time-averaged mass-flux density, which is superimposed on the much larger oscillating flow. (a) Gedeon streaming. (b) Rayleigh streaming. (c) Jet-driven streaming. (d) Streaming within a regenerator or stack. From Swift [42].

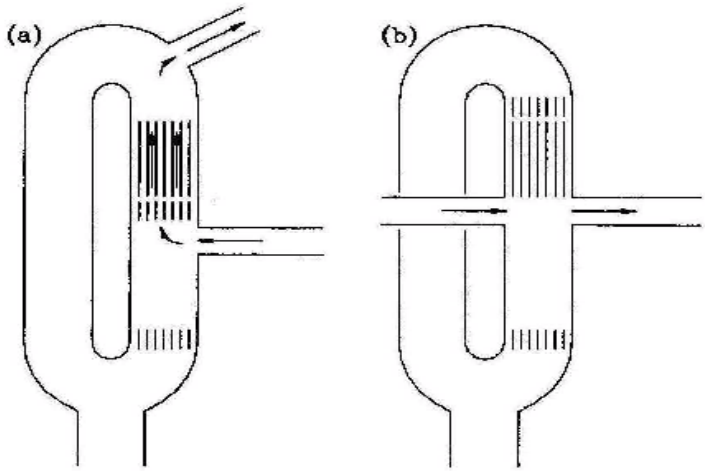

Figure 1.11. Types of mass streaming that can be beneficial for thermoacoustic engines and refrigerators. (a) Flow parallel to x, through a stack or regenerator. (b) Flow perpendicular to x, across one end of a stack or regenerator. From Swift [42].

There are several analytical calculations of acoustic streaming available. There are three analyses that are examined in the present research. One approach was taken by a group consisting Bailliet, Gusev, Raspet and Hiller [3]. Another group consisted of Hamilton, Ilinski and Zabolotskaya [11]. The third study is the streaming analysis performed by Rayleigh and reviewed by Nyborg [27]. In the subsequent chapters, they will be referred to as the "Bailliet *et al.*", "Hamilton *et al.*" and "Rayleigh/Nyborg" analyses. Like most streaming studies, these three analyses start with the successive approximation method where the acoustic variables are expanded to include second order terms and higher. The solution of the first order velocities (acoustic velocities) where a standing wave is present is derived first. Then the solution for the second order velocities (the streaming velocities) is worked out.

The numerical calculation of acoustic streaming usually involves the solution of the continuity equation, the Navier Stokes equation, and the energy equation. When a direct numerical simulation (DNS) is used to solve those equations, it requires considerable computing power and time. In this research, instead of using the three mentioned equations, the vorticity transport equation (VTE) is utilized to calculate the streaming behavior. This formulation allows the computation to be performed on a single PC and give relatively fast results. Although the predicted

streaming behavior is not as precise in detail as the DNS method, the speed can be beneficial when a fast calculation of streaming is needed as a part of the design stage of a thermoacoustic device.

The idea of using the VTE came from an article by Kamakura *et al.* *"Acoustic streaming induced in focused Gaussian beam"* [19]. In the article, Kamakura *et al.* numerically simulated Eckert streaming due to an ultrasound source of 1 cm radius with a 5 cm focal length with a Gaussian amplitude distribution, radiating 5 MHz ultrasound beams in water. They showed a buildup of acoustic streaming along and across the acoustic axis. The flow was calculated using the axisymmetric flow equations for a viscous incompressible fluid that are transformed into the vorticity and Poisson's equations. The equations were solved via a finite difference method with the appropriate initial and boundary conditions. They used a time-stepping method and an iteration method of SOR (Successive Over-Relaxation) within each time step to converge to the solution. Their computational domain had 200×200 grid points uniformly spaced. With a normalized time step of $\Delta\tau = 2 \times 10^{-3}$, the calculation can be done in about 5 minutes on a 400 MHz PC. Looking at the speed of the calculation, it was interesting to see if the method would also work to simulate Rayleigh streaming. So Kamakura *et al.* 's article became the starting point of this research.

1.3 Contribution

As mentioned in section 1.2, existing numerical calculations of acoustic streaming are made by solving the full Navier Stokes equations using direct numerical simulation (DNS), without making any assumption of the fluid characteristics. This type of calculation can give a very detailed view of the fluid flow features. Although the DNS method provides an in-depth insight of what is happening to the fluid flow inside a thermoacoustic engine, the computation power required is rather expensive (usually involving a beowolf cluster) and the computation time can be rather lengthy. In some cases, a less detailed calculation in a preliminary design process may be desired. The complexity of the numerical model may be lessened by making a set of assumptions about the flow. The current approach proposed in this research makes use of several assumptions about the flow and the use of

the vorticity transport equation as the primary equation to calculate the acoustic streaming velocity. It will be shown that the calculation can be performed on a single PC with a relatively fast computational time. The resulting flow patterns agree well with the analytical results. Provided the first order acoustic particle velocity inside a thermoacoustic device is known, one should be able to calculate the acoustic streaming velocity inside such devices fairly quickly. Proving that the current approach is a valid method to predict acoustic streaming behavior in a standing wave thermoacoustic device, it is then concluded that this method can be utilized to predict acoustic streaming behavior in other thermoacoustic devices having various geometries in a relatively quick and inexpensive manner.

It is worth mentioning that although the idea of using the VTE to obtain the acoustic streaming velocity came from Kamakura *et al.*'s article, during the span of this research it was discovered that their method cannot be implemented for the present problem. It has been necessary to modify their method for Rayleigh streaming. By using the appropriate numerical calculation techniques here, it was possible to get the correct streaming pattern results.

1.4 Thesis Overview

The subsequent chapters of the thesis will discuss the starting point of acoustic streaming analysis, the driving force of the acoustic streaming, and the results of the numerical calculation. **Chapter 2** discusses the successive approximation method as the starting point of acoustic streaming analysis and the governing equations used in this research to calculate the streaming velocity. In **Chapter 3** the analytical first order acoustic velocity that is used to derive the forcing function is presented. The first order velocities employed in this research are derived by the three research groups mentioned in section 1.2. Each group has a set of assumptions in their analysis that will be listed before discussing their approach in detail. **Chapter 4** talks about the physical and computational grids that are used for the calculation. The finite difference form of the governing equations will also be discussed in that chapter. In **Chapter 5** the numerical method utilized to solve the governing equations is presented in detail. The numerical results are then compared to the analytical results in **Chapter 6**. The results from using

the forcing functions of different research groups will also be compared. Finally in **Chapter 7** the findings in this research will be summarized and some suggestions for future work will be provided.

Chapter 2

Acoustic Streaming Calculation

There are two ways to predict the behavior of acoustic streaming: analytically or numerically. Each method has its own set of challenges. Analytical calculations are usually performed under certain assumptions to simplify the governing equations which may not be solved otherwise (under general conditions). Numerical prediction may be able to give results that cannot be calculated analytically. It may also enable the calculation of different operating conditions without starting the calculation over to account for the changes in the problem setup. The subsequent sections in this chapter will discuss each approach.

2.1 Method of Successive Approximation

Traditionally, most analytical streaming calculations were made using the method of successive approximation. This method starts by expanding acoustic variables to include higher order terms. Subsequently the excess pressure, density and velocity at any point in the fluid can be defined as:

$$\begin{aligned} p - p_0 &= p_1 + p_2 + \cdots \\ \rho - \rho_0 &= \rho_1 + \rho_2 + \cdots \\ \mathbf{u} &= \mathbf{u}_1 + \mathbf{u}_2 \end{aligned} \tag{2.1}$$

where p_1, ρ_1 and $\mathbf{u}_1$ are the first order approximations of the acoustic quantities which vary sinusoidally in time with frequency ω. The higher order terms are the

"correction" added to the first order approximations [27]. The corrections consist of time independent quantities and quantities that vary with frequency 2ω. In studying acoustic streaming the quantity of interest is $\mathbf{u}_2$, the *time-independent* second order velocity. So the method of successive approximation starts with the first order approximation $\mathbf{u}_1$. Knowing $\mathbf{u}_1$, the second order approximation $\mathbf{u}_2$ can be calculated and then added to the first order approximation as the correction.

To obtain the acoustic streaming velocity $\mathbf{u}_2$, the common method of solution is to apply the new expanded variables to the continuity and momentum equations. Coupled with the assumptions made prior to the calculation, one can find $\mathbf{u}_2$ by solving the momentum equation. This is the method employed by both Hamilton *et al.* and Bailliet *et al.* in calculating the acoustic streaming velocity analytically.

2.2 Vorticity Transport Equation

An alternative approach to obtain the analytical streaming velocity is through the use of the vorticity transport equation (VTE). To derive the VTE let us start with the momentum equation for incompressible fluid with no external body force:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla\mathbf{u}) \;= -\frac{1}{\rho}\nabla p + \nu\nabla^2\mathbf{u} \tag{2.2}$$

The following vector identities will be utilized:

$$\frac{1}{2}\nabla(\mathbf{u}\cdot\mathbf{u}) = (\mathbf{u}\cdot\nabla)\mathbf{u} + \mathbf{u}\times(\nabla\times\mathbf{u}) \tag{2.3}$$

$$\nabla\times(\mathbf{u}\times\boldsymbol{\omega}) = (\boldsymbol{\omega}\cdot\nabla)\mathbf{u} - (\mathbf{u}\cdot\nabla)\boldsymbol{\omega} + \mathbf{u}\nabla\cdot\boldsymbol{\omega} - \boldsymbol{\omega}\nabla\cdot\mathbf{u} \tag{2.4}$$

$$\nabla\cdot(\nabla\times\mathbf{u}) = 0 \tag{2.5}$$

$$\nabla\times\nabla\Phi = 0 \tag{2.6}$$

where Φ is any scalar quantity and $\boldsymbol{\omega} = \nabla\times\mathbf{u}$ is the vorticity. For an incompressible flow,

$$\nabla\cdot\mathbf{u} = 0 \tag{2.7}$$

From vector identity in Eq. 2.5,

$$\nabla\cdot(\nabla\times\mathbf{u}) = \nabla\cdot\boldsymbol{\omega} = 0 \tag{2.8}$$

Therefore for an incompressible flow, the vector identity in Eq. 2.4 can be expressed as:

$$\nabla \times (\mathbf{u} \times \boldsymbol{\omega}) = (\boldsymbol{\omega} \cdot \nabla)\mathbf{u} - (\mathbf{u} \cdot \nabla)\boldsymbol{\omega} \tag{2.9}$$

Using the vector identity in Eq. 2.3, the momentum equation can be written as:

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{1}{2}\nabla(\mathbf{u} \cdot \mathbf{u}) - \mathbf{u} \times (\nabla \times \mathbf{u}) = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} \tag{2.10}$$

Taking the curl of the left hand side of Eq. 2.10 and using the vector identities in Eqs. 2.6 and 2.9,

$$\begin{aligned} \nabla \times \text{LHS} &= \frac{\partial}{\partial t}(\nabla \times \mathbf{u}) + \frac{1}{2}\nabla \times \nabla(\mathbf{u} \cdot \mathbf{u}) - \nabla \times (\mathbf{u} \times \boldsymbol{\omega}) \\ &= \frac{\partial \boldsymbol{\omega}}{\partial t} - (\boldsymbol{\omega} \cdot \nabla)\mathbf{u} + (\mathbf{u} \cdot \nabla)\boldsymbol{\omega} \end{aligned} \tag{2.11}$$

Using vector identity in Eq. 2.6, the curl of the right hand side of Eq. 2.10 is:

$$\begin{aligned} \nabla \times \text{RHS} &= -\frac{1}{\rho}\nabla \times \nabla p + \nu \nabla^2 (\nabla \times \mathbf{u}) \\ &= \nu \nabla^2 \boldsymbol{\omega} \end{aligned} \tag{2.12}$$

Therefore the curl of the momentum equation can be expressed as:

$$\frac{\partial \boldsymbol{\omega}}{\partial t} - (\boldsymbol{\omega} \cdot \nabla)\mathbf{u} + (\mathbf{u} \cdot \nabla)\boldsymbol{\omega} = \nu \nabla^2 \boldsymbol{\omega} \tag{2.13}$$

or

$$\frac{\partial \boldsymbol{\omega}}{\partial t} + (\mathbf{u} \cdot \nabla)\boldsymbol{\omega} = (\boldsymbol{\omega} \cdot \nabla)\mathbf{u} + \nu \nabla^2 \boldsymbol{\omega} \tag{2.14}$$

Equation 2.14 is the general form of the **vorticity transport equation**. The physical interpretation of the terms in the VTE is:

- $\partial \boldsymbol{\omega}/\partial t$ is the rate of change of the vorticity with respect to time.
- $(\mathbf{u} \cdot \nabla)\boldsymbol{\omega}$ is the change of the vorticity due to spatial variations.
- $(\boldsymbol{\omega} \cdot \nabla)\mathbf{u}$ is the change of the vorticity due to stretching.
- $\nu \nabla^2 \boldsymbol{\omega}$ is the diffusion of the vorticity by viscosity

For a two-dimensional flow, the vorticity vector $\boldsymbol{\omega}$ is perpendicular to the plane of the flow. Therefore if the flow is in the x and y plane, the only nonzero vorticity component is in the z direction. For that reason, $(\boldsymbol{\omega} \cdot \nabla)\mathbf{u}$ is zero in a two-dimensional flow and the VTE becomes:

$$\frac{\partial \omega}{\partial t} + (\mathbf{u} \cdot \nabla)\omega = \nu \nabla^2 \omega \tag{2.15}$$

To clearly understand the concept of the forcing function that drives the VTE in this research, the VTE derivation will be done in a slightly different manner by taking the curl of the x and y momentum equations. The x component of the momentum equation for a two dimensional analysis (assuming constant density and viscosity and no external force) is:

$$\rho_0 \left(\frac{\partial u_x}{\partial t} + \frac{\partial u_x u_x}{\partial x} + \frac{\partial u_x u_y}{\partial y} \right) = -\frac{\partial p}{\partial x} + \mu \left(\frac{\partial^2 u_x}{\partial x^2} + \frac{\partial^2 u_x}{\partial y^2} \right) \tag{2.16}$$

And the y component is:

$$\rho_0 \left(\frac{\partial u_y}{\partial t} + \frac{\partial u_x u_y}{\partial x} + \frac{\partial u_y u_y}{\partial y} \right) = -\frac{\partial p}{\partial y} + \mu \left(\frac{\partial^2 u_y}{\partial x^2} + \frac{\partial^2 u_y}{\partial y^2} \right) \tag{2.17}$$

Here u_x and u_y represent the two Cartesian components of $\mathbf{u}$.

The VTE is obtained by taking the curl of momentum equation. Taking the x derivative of the left hand side of the y momentum equation (Eq. 2.17) we obtain:

$$\rho_0 \frac{\partial}{\partial x} \left[\frac{\partial u_y}{\partial t} + \frac{\partial u_x u_y}{\partial x} + \frac{\partial u_y u_y}{\partial y} \right] = \rho_0 \left[\frac{\partial}{\partial t} \frac{\partial u_y}{\partial x} + \left(\frac{\partial^2 u_x u_y}{\partial x^2} \right) + \left(\frac{\partial^2 u_y u_y}{\partial x \partial y} \right) \right] \tag{2.18}$$

and taking the x derivative of the right hand side of the y momentum equation we obtain

$$\frac{\partial}{\partial x} \left[-\frac{\partial p}{\partial y} + \mu \left(\frac{\partial^2 u_y}{\partial x^2} + \frac{\partial^2 u_y}{\partial y^2} \right) \right] = -\frac{\partial^2 p}{\partial x \partial y} + \mu \left(\frac{\partial^2}{\partial x^2} \frac{\partial u_y}{\partial x} + \frac{\partial^2}{\partial y^2} \frac{\partial u_y}{\partial x} \right) \tag{2.19}$$

Similarly by taking the y derivative of the left hand side of the x momentum equation (Eq. 2.16) we obtain

$$\rho_0 \frac{\partial}{\partial y} \left[\frac{\partial u_x}{\partial t} + \frac{\partial u_x u_x}{\partial x} + \frac{\partial u_x u_y}{\partial y} \right] = \rho_0 \left[\frac{\partial}{\partial t} \frac{\partial u_x}{\partial y} + \left(\frac{\partial^2 u_x u_x}{\partial x \partial y} \right) + \left(\frac{\partial^2 u_x u_y}{\partial y^2} \right) \right] \tag{2.20}$$

and taking the y derivative of the right hand side of the x momentum equation we obtain

$$\frac{\partial}{\partial y}\left[-\frac{\partial p}{\partial x}+\mu\left(\frac{\partial^2 u_x}{\partial x^2}+\frac{\partial^2 u_x}{\partial y^2}\right)\right]=-\frac{\partial^2 p}{\partial x \partial y}+\mu\left(\frac{\partial^2}{\partial x^2}\frac{\partial u_x}{\partial y}+\frac{\partial^2}{\partial y^2}\frac{\partial u_x}{\partial y}\right) \tag{2.21}$$

Subtracting Eqs. 2.18 and 2.20 we obtain the left hand side of the VTE:

$$\begin{aligned}\rho_0\frac{\partial}{\partial t}\left(\frac{\partial u_y}{\partial x}-\frac{\partial u_x}{\partial y}\right)+\rho_0\left[\frac{\partial}{\partial x}\left(\frac{\partial u_x u_y}{\partial x}\right)-\frac{\partial}{\partial y}\left(\frac{\partial u_x u_x}{\partial x}\right)\right]\\ +\rho_0\left[\frac{\partial}{\partial x}\left(\frac{\partial u_y u_y}{\partial y}\right)-\frac{\partial}{\partial y}\left(\frac{\partial u_x u_y}{\partial y}\right)\right]\end{aligned} \tag{2.22}$$

And the corresponding right hand side of the VTE is:

$$\begin{aligned}&\left[-\frac{\partial^2 p}{\partial x \partial y}+\mu\left(\frac{\partial^2}{\partial x^2}\frac{\partial u_y}{\partial x}+\frac{\partial^2}{\partial y^2}\frac{\partial u_y}{\partial x}\right)\right]-\left[-\frac{\partial^2 p}{\partial x \partial y}+\mu\left(\frac{\partial^2}{\partial x^2}\frac{\partial u_x}{\partial y}+\frac{\partial^2}{\partial y^2}\frac{\partial u_x}{\partial y}\right)\right]=\\ &\mu\left[\frac{\partial^2}{\partial x^2}\left(\frac{\partial u_y}{\partial x}-\frac{\partial u_x}{\partial y}\right)+\frac{\partial^2}{\partial y^2}\left(\frac{\partial u_y}{\partial x}-\frac{\partial u_x}{\partial y}\right)\right]\end{aligned} \tag{2.23}$$

The vorticity ω is defined as:

$$\omega=\frac{\partial u_y}{\partial x}-\frac{\partial u_x}{\partial y} \tag{2.24}$$

Therefore the left hand side of the VTE can be written as:

$$\begin{aligned}\rho_0\frac{\partial \omega}{\partial t}+\rho_0\left[\frac{\partial}{\partial x}\left(\frac{\partial u_x u_y}{\partial x}\right)-\frac{\partial}{\partial y}\left(\frac{\partial u_x u_x}{\partial x}\right)\right]\\ +\rho_0\left[\frac{\partial}{\partial x}\left(\frac{\partial u_y u_y}{\partial y}\right)-\frac{\partial}{\partial y}\left(\frac{\partial u_x u_y}{\partial y}\right)\right]\end{aligned} \tag{2.25}$$

and the right hand can be written as:

$$\mu\left[\frac{\partial^2 \omega}{\partial x^2}+\frac{\partial^2 \omega}{\partial y^2}\right] \tag{2.26}$$

Equating the left hand side with the right hand side, the VTE is expressed as:

$$\rho_0 \frac{\partial \omega}{\partial t} + \rho_0 \left[\frac{\partial}{\partial x} \left(\frac{\partial u_x u_y}{\partial x} \right) - \frac{\partial}{\partial y} \left(\frac{\partial u_x u_x}{\partial x} \right) \right]$$
$$+\rho_0 \left[\frac{\partial}{\partial x} \left(\frac{\partial u_y u_y}{\partial y} \right) - \frac{\partial}{\partial y} \left(\frac{\partial u_x u_y}{\partial y} \right) \right] = \mu \left[\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2} \right] \tag{2.27}$$

For this research, following Nyborg's [27] analysis, it is assumed that the wave number is much smaller then the inverse of the viscous penetration depth, i.e. $k << \beta$. The wave number and the inverse of the viscous penetration depth are defined as:

$$k = \frac{\omega}{c} \tag{2.28}$$

$$\beta = \sqrt{\frac{\omega \rho}{2\mu}} \tag{2.29}$$

where ω in Eqs. 2.28 and 2.29 is the angular frequency (and not the vorticity), c is the speed of sound, ρ is the density, and μ is the dynamic viscosity coefficient. It was found that when $k << \beta$, the absorption coefficient is much smaller then the wave number or $\alpha << k$ where $\alpha = bk^3/4\beta^2$. This assumption leads to the conditions of the first order velocity being the sum of an irrotational part and an incompressible part [27]:

$$\mathbf{u}_1 = \mathbf{u}_{1a} + \mathbf{u}_{1b} \tag{2.30}$$

where

$$\nabla \times \mathbf{u}_{1a} = 0 \tag{2.31}$$

$$\nabla \cdot \mathbf{u}_{1b} = 0 \tag{2.32}$$

The VTE further assumes the total field $\mathbf{u}_1 = \mathbf{u}_{1a} + \mathbf{u}_{1b}$ is incompressible.

With those conditions in mind, let us now substitute Eq. 2.1 into Eq. 2.27 but keep terms only up to the second order. The resulting equation is a second order equation which means quantities on the left hand side and the right hand side are of second order. Notice that the velocities u_x and u_y are first order quantities and the vorticity ω is a second order quantity. Since u_x and u_y will later be identified as the axial and transverse acoustic velocities respectively, and their values are

known, the left hand side of the VTE drives the right hand side of the equation. Rearranging Eq. 2.27, it can be written as:

$$\rho_0 \frac{\partial \omega}{\partial t} + \rho_0 \left[-\frac{\partial}{\partial y} \left(\frac{\partial u_x u_x}{\partial x} + \frac{\partial u_x u_y}{\partial y} \right) + \frac{\partial}{\partial x} \left(\frac{\partial u_x u_y}{\partial x} + \frac{\partial u_y u_y}{\partial y} \right) \right] = \mu \left[\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2} \right] \tag{2.33}$$

We can define:

$$-F_x = \frac{\partial}{\partial x}(u_{1x} u_{1x}) + \frac{\partial}{\partial y}(u_{1x} u_{1y}) \tag{2.34}$$

$$-F_y = \frac{\partial}{\partial x}(u_{1x} u_{1y}) + \frac{\partial}{\partial y}(u_{1y} u_{1y}) \tag{2.35}$$

where $u_x = u_{1x}$ and $u_y = u_{1y}$.

u_{1x} and u_{1y} are the first order axial and transverse velocities respectively. Substituting Eqs. 2.34 and 2.35 into Eq. 2.33 and dividing through by ρ_0, the VTE can be written compactly as:

$$\frac{\partial \omega_2}{\partial t} - \left(\frac{\partial F_y}{\partial x} - \frac{\partial F_x}{\partial y} \right) = \nu \left(\frac{\partial^2 \omega_2}{\partial x^2} + \frac{\partial^2 \omega_2}{\partial y^2} \right) \tag{2.36}$$

where $\nu = \mu/\rho$ is the kinematic viscosity. The subscript "2" on ω is used to denote that the vorticity here is a second order quantity.

The **time-averaged** VTE can be obtained by integrating the time-dependent VTE with respect to time from time zero to infinity. The resulting equation is simply:

$$-\left(\frac{\partial F_y}{\partial x} - \frac{\partial F_x}{\partial y} \right) = \nu \left(\frac{\partial^2 \omega_2}{\partial x^2} + \frac{\partial^2 \omega_2}{\partial y^2} \right) \tag{2.37}$$

or written in a 2-D vector form,

$$-\nabla \times \mathbf{F} = \nu \nabla^2 \omega_2 \tag{2.38}$$

where $\mathbf{F} = F_x \,\hat{i} + F_y \,\hat{j}$ and $\omega_2 = (\partial u_{2y}/\partial x) - (\partial u_{2x}/\partial y)$. F_x and F_y are defined in Eqs. 2.34 and 2.35. It should be noted that since this analysis is in a planar (x, y) coordinate system, we interpret the vorticity ω as the rotating fluid in the (x, y) plane, the z component of the vector vorticity $\boldsymbol{\omega}$.

2.3 Poisson's Equation

In a two dimensional incompressible flow, a stream function can be defined through the relationships:

$$u_x = \frac{\partial \psi}{\partial y}\,, \qquad u_y = -\frac{\partial \psi}{\partial x} \tag{2.39}$$

Since the vorticity is

$$\omega = \frac{\partial u_y}{\partial x} - \frac{\partial u_x}{\partial y} \tag{2.40}$$

replacing the time-averaged quantities u_{2x} and u_{2y} for u_x and u_y in Eq. 2.39, and substituting them in Eq. 2.40 yields the expression:

$$\omega_2 = -\frac{\partial^2 \psi_2}{\partial x^2} - \frac{\partial^2 \psi_2}{\partial y^2} \tag{2.41}$$

or

$$-\omega_2 = \frac{\partial^2 \psi_2}{\partial x^2} + \frac{\partial^2 \psi_2}{\partial y^2} = \nabla^2 \psi_2 \tag{2.42}$$

Equation 2.42 is known as *Poisson's equation*. From Eq. 2.38 we know that:

$$-\nabla \times \mathbf{F}_2 = \nu \nabla^2 \omega_2 \tag{2.43}$$

Substituting ω_2 from Eq. 2.42 into Eq. 2.43 we get:

$$-\nabla \times \mathbf{F}_2 = \nu \nabla^2 (-\nabla^2 \psi_2) \tag{2.44}$$

or

$$\nabla \times \mathbf{F}_2 = \nu \nabla^4 \psi_2 \tag{2.45}$$

Notice that Eq. 2.45 is a second order equation in the acoustic quantities. Looking at the above equation and Eq. 2.39, one sees that: (1) Knowing $\mathbf{F}_2$ you can solve for the stream function ψ_2, and (2) knowing ψ_2 you can solve for the acoustic axial and transverse streaming velocities u_{2x} and u_{2y}.

Chapter 3

First Order Velocity And Analytical Acoustic Streaming Velocity

In the previous chapter we concluded that in order to calculate the acoustic streaming velocity we need to know the first order acoustic velocity. This research particularly looks at the acoustic streaming (Rayleigh streaming) velocity due to a standing wave between parallel walls. Therefore it is necessary to have an expression for the first order acoustic velocity in a channel when a standing wave is present. There are several first order acoustic velocity expressions used in this research. The first expression was derived by Rayleigh (and later corrected by Westervelt and then presented by Nyborg). The second expression was derived by Hamilton *et al.* and the third one was derived by Bailliet *et al.* Each of the expressions was derived under a set of assumptions. It is imperative that we look at the assumptions made in these analyses in order to make a fair comparison of the resulting first order velocities. These first order velocities are going to be the driving force of the VTE. All three methods will be discussed to understand their differences that lead to the differences in the calculated analytical streaming velocities. The differences in the first order velocities of course causes the differences in the driving forces, which in turn will give different calculated numerical streaming velocities. Now let us look at each of the methods closely.

3.1 Rayleigh/Nyborg

3.1.1 First order acoustic velocity

The details of the derivation of the first order velocity solution to a standing wave between parallel walls will not be presented here, but readers are encouraged to study Rayleigh's solution in his *Theory of Sound* [41]. His analysis assumed the following domain:

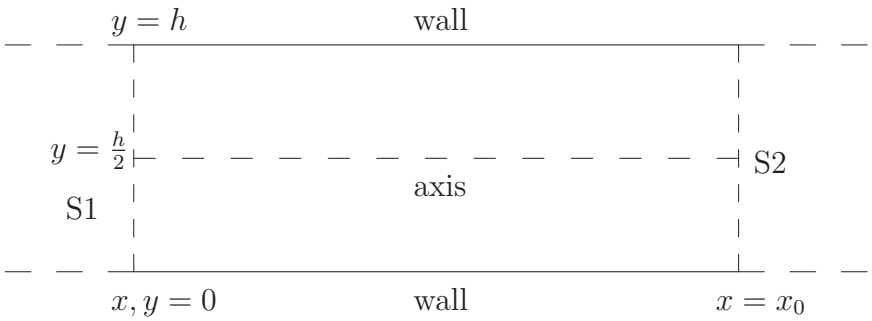

Figure 3.1. Rayleigh/Nyborg calculation domain

The channel is assumed to be an infinitely long channel. Rayleigh examined a section of the channel which is half a wavelength long from $x = 0$ to $x = x_0$. S1 and S2 are the left and right boundaries of the calculation domain. The top and bottom boundaries of the channel are located at $y = 0$ and $y = h$. The axis of symmetry is located at $y = h/2$ where h is the channel height.

Some important definitions in what follow include:

- Viscous penetration depth: $\delta_\nu = \sqrt{2\nu/\omega}$, where ν is the kinematic viscosity and ω is the radial frequency. δ_ν arises from the fact that in viscous flow, the fluid velocity is zero at the surface of solid bodies ("no-slip" condition). This creates a region called the *boundary layer* region, which is a region of fluid where the effect of viscosity causes the transverse gradient of the velocity to be larger compared to the axial variations. The viscous penetration depth can be defined as the distance in the transverse direction from the surface of the solid bodies to the point where the velocity is within 1% of the local free stream velocity [43].

- Inverse penetration depth: $\beta = 1/\delta_\nu$.
- Absorption coefficient: $\alpha = \frac{bk^3}{4\beta^2} = \frac{k^2\mu b}{2\rho_0 c}$, where $\mu b = \mu' + \frac{4}{3}\mu + \rho_0 R$.
  μ is the dynamic viscosity coefficient, μ' is the bulk viscosity coefficient, and R is a highly frequency dependent constant representing the effects of relaxation phenomena when pressure varies sinusoidally with time.
- First order axial and transverse acoustic velocities: u_1 and w_1 respectively.

The assumptions that Rayleigh made when he derived the first order particle velocity are:

1. The fluid is homogeneous and isotropic.
2. The shear viscosity μ and bulk viscosity μ' are independent of space and time.
3. The only forces acting on the fluid are surface stresses due to elasticity and viscosity.
4. The wavelength is much larger than viscous penetration depth ($\lambda >> \delta_\nu$).
5. $\lambda >> \delta_\nu$ means $k << \frac{1}{\delta_\nu}$ or $k << \beta$ where k is the wave number. If $k << \beta$ then $\alpha << \beta$ because $\frac{bk^3}{4\beta^2}$ is a small quantity so α is very small compared to k ($\alpha << k$)
6. When $\alpha << k$ the first order velocity field is irrotational ($\nabla \times \mathbf{u}_1 = 0$) and incompressible ($\nabla \cdot \mathbf{u}_1 = 0$).
7. Keep terms only up to the second order.
8. The second order velocity is incompressible ($\nabla \cdot \mathbf{u}_2 = 0$).

The solution was derived by solving the continuity and momentum equations under the above assumptions. The acoustic first order axial velocity inside a channel when a standing wave is present can then be expressed as [27]:

$$u_1 = u_0 \cos(kx) \left[\cos \omega t - e^{-\beta y} \cos(\omega t - \beta y)\right] \tag{3.1}$$

and the first order transverse velocity is:

$$w_1 = u_0 \frac{k}{\beta\sqrt{2}} \sin(kx) \left[\cos\left(\omega t - \frac{\pi}{4}\right) + e^{-\beta y} \cos\left(\omega t - \beta y - \frac{\pi}{4}\right) \right] \tag{3.2}$$

The first order axial velocity is plotted in Figs. 3.2 and 3.3 and the first order transverse velocity is plotted in Figs. 3.4 and 3.5.

The following physical geometry will be used in these and upcoming figures throughout the thesis. The channel and the cylindrical tube is half a wavelength long and the distance between the axis of symmetry and the top or bottom wall is 0.0232 m. The frequency assumed is 310 Hz, therefore the channel (and tube length) are 0.553 m. For air at 20°C, the speed of sound $c = 343$ m/s, the shear viscosity $\mu = 1.81 \times 10^{-5}$ Pa s, and the ambient density $\rho_0 = 1.21$ kg/m^3. These physical parameters are chosen based on Atchley and Thompson's [45] experiments to make it possible to compare the numerical results with their experimental results.

The forcing function that drives the VTE is calculated through Eqs. 2.34 and 2.35 where $u_{1x} = u_1$ and $u_{1y} = w_1$. Therefore,

$$-F_x = \frac{\partial}{\partial x}(u_1 u_1) + \frac{\partial}{\partial y}(u_1 w_1) \tag{3.3}$$

$$-F_y = \frac{\partial}{\partial x}(u_1 w_1) + \frac{\partial}{\partial y}(w_1 w_1) \tag{3.4}$$

The y component of $\mathbf{F}$ is usually negligible compared to the x component and the x component of $\mathbf{F}$ can be expressed as [1]:

$$F_x = \rho_0 \, k \, u_0^2 \, \sin(2kx) \left[\frac{1}{2} + \frac{1}{4} \left\{ e^{-2\beta y} - 3e^{-\beta y} \cos(\beta y) + e^{-\beta y} \sin(\beta y) \right\} \right] \tag{3.5}$$

Figures 3.6 and 3.7 show the x component of the Rayleigh/Nyborg forcing function. Notice in Fig. 3.7 that the amplitude of the forcing function has a "bump" near the wall. This indicates that the forcing function is the strongest in the area near the boundary (boundary layer area).

[1] Be aware that this expression is misprinted in [27]. The $\frac{1}{2}$ factor in Eq. 3.5 was printed as 1.

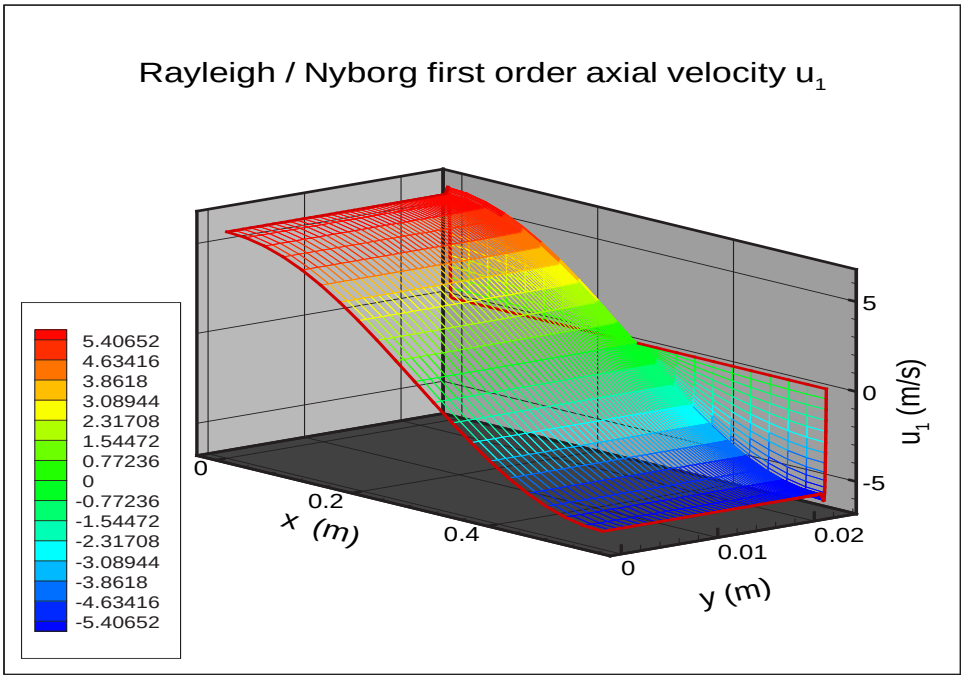

Figure 3.2. Rayleigh/Nyborg first order axial velocity u_1. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

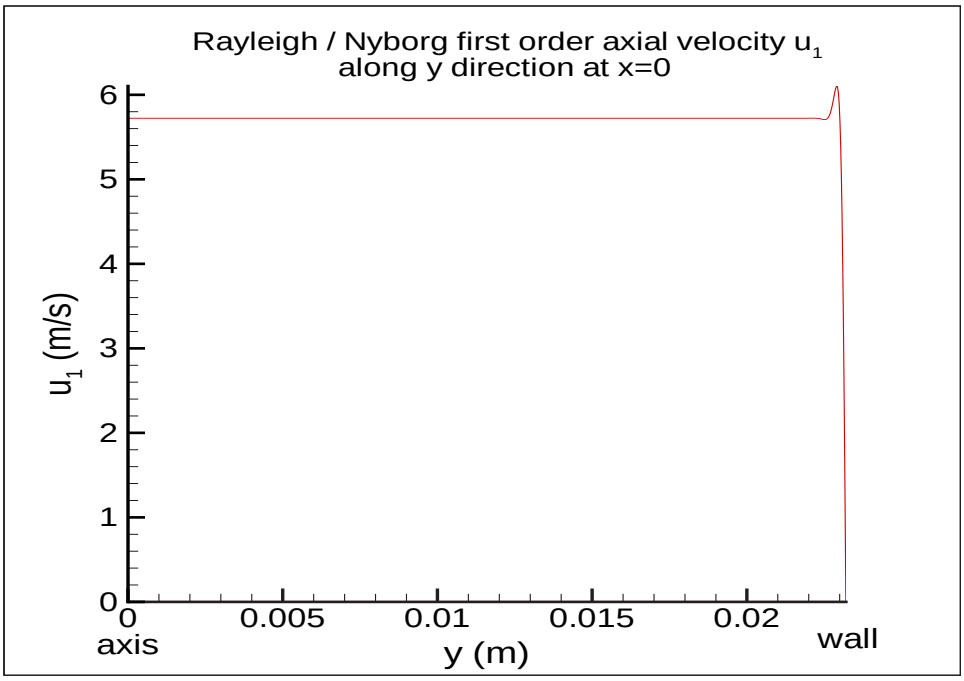

Figure 3.3. Rayleigh/Nyborg first order axial velocity u_1 along the y direction, next to boundary S1. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

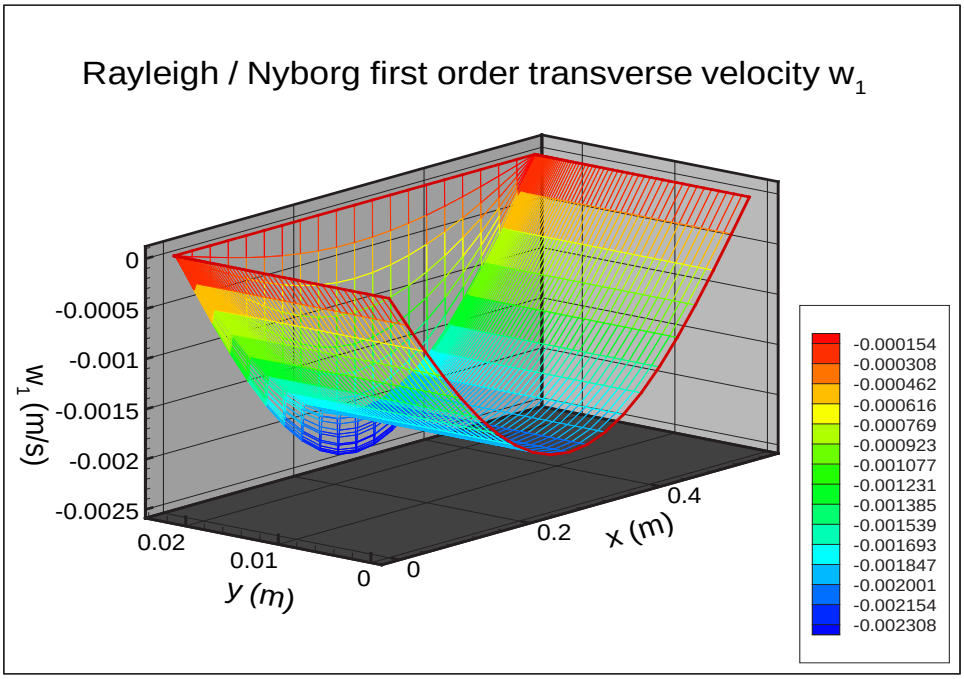

Figure 3.4. Rayleigh/Nyborg first order transverse velocity w_1. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

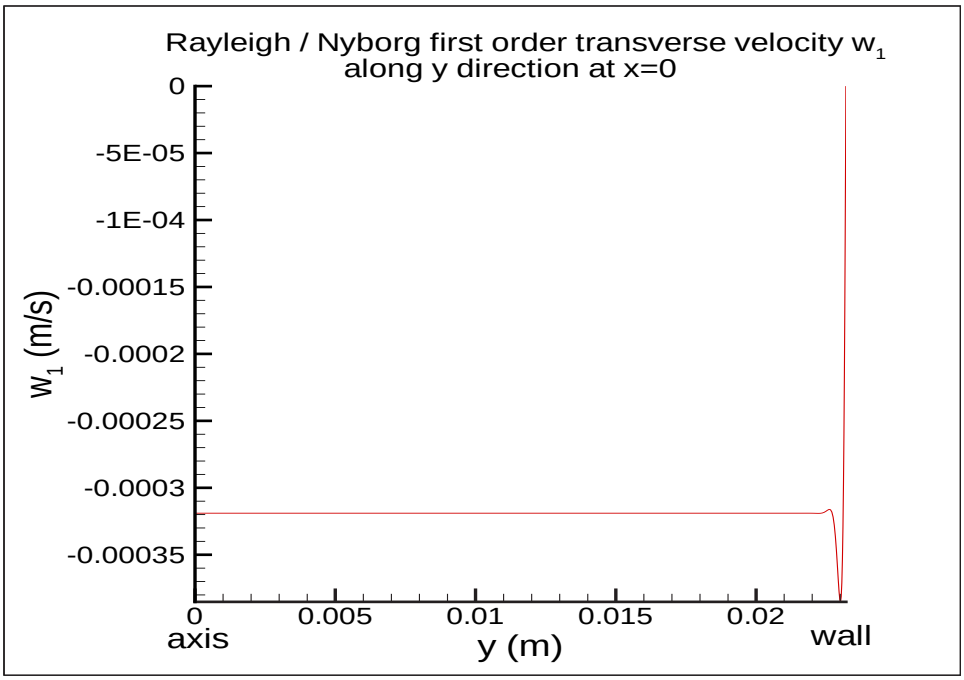

Figure 3.5. Rayleigh/Nyborg first order transverse velocity w_1 along the y direction, next to boundary S1. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

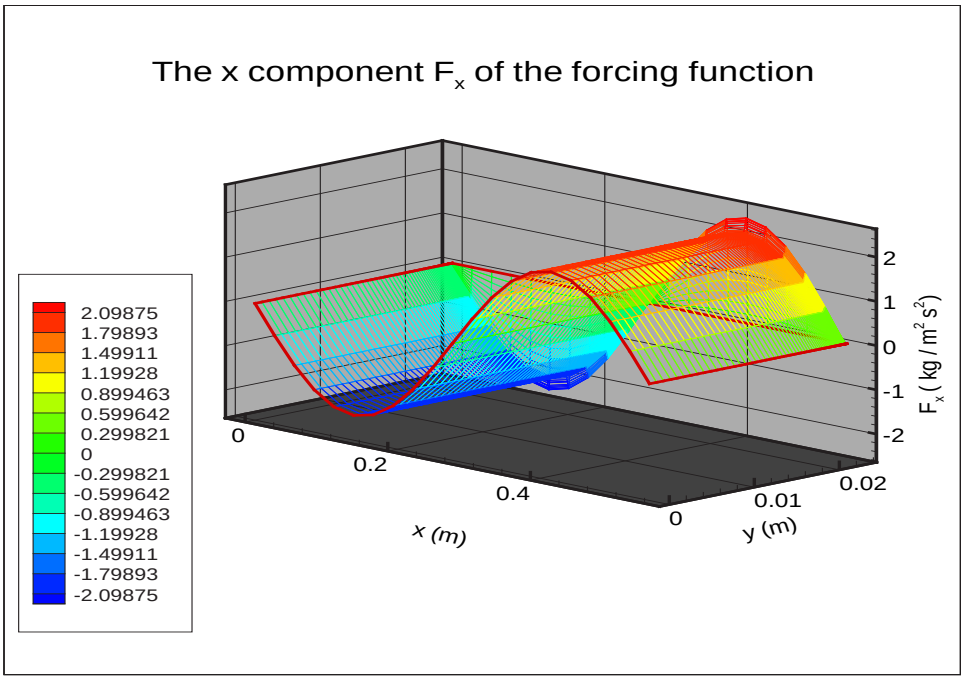

Figure 3.6. Rayleigh/Nyborg x component F_x of the forcing function. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

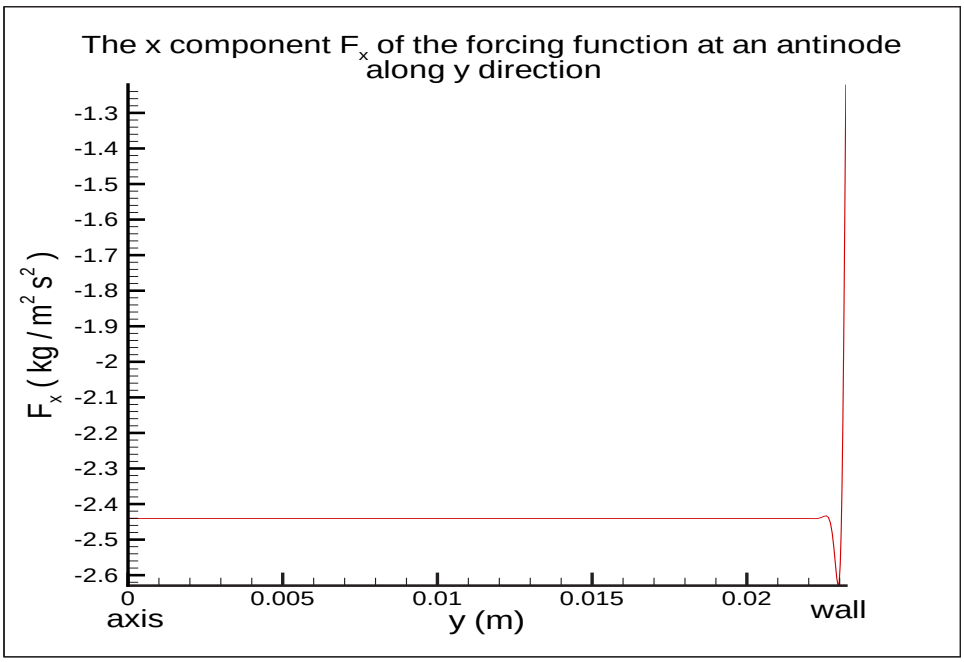

Figure 3.7. Rayleigh/Nyborg x component F_x of the forcing function at an antinode. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

3.1.2 Analytical acoustic streaming velocities

Rayleigh calculated the analytical acoustic streaming velocity with the assumption that the first order field is irrotational ($\nabla \times \mathbf{u}_1 = 0$) and divergence-free ($\nabla \cdot \mathbf{u}_1 = 0$). The second order velocity can be solved by solving the curl of the time-averaged momentum equation,

$$-\nabla \times \mathbf{F} = \nu \nabla^2 \omega \tag{3.6}$$

where $\omega = \nabla \times \mathbf{u}_2$. Using the relationship

$$\nabla^2 \psi_2 = -\omega_2 = -\nabla \times \mathbf{u}_2$$

Eq. 3.6 can be expressed as:

$$\nabla \times \mathbf{F}_2 = \nu \nabla^4 \psi_2 \tag{3.7}$$

Under the assumption that $\nabla \cdot \mathbf{u}_2$ is approximately zero, Nyborg presented the solution to Eq. 3.6. Since $\beta/k >> 1$,

$$\nabla^4 \psi_2 \cong \frac{d^4 \psi}{dy^4}$$

Substituting Eq. 3.5 into 3.7,

$$\frac{d^4 \psi}{dy^4} = -\frac{1}{2} \beta \rho_0 k u_0^2 \sin(2kx) \left\{ e^{-2\beta y} - 3e^{-\beta y} \cos(\beta y) + e^{-\beta y} \sin(\beta y) \right\} \tag{3.8}$$

Equation 3.8 has a particular solution of

$$\psi_2^p = \frac{u_0^2}{8\beta c} \sin(2kx) \left(4e^{-\beta y} \cos(\beta y) + 2e^{-\beta y} \sin(\beta y) + \frac{1}{2} e^{-2\beta y} \right) \tag{3.9}$$

and a homogeneous solution of

$$\psi_2^p = a_0 + a_1 y + a_2 y^2 + a_3 y^3 \tag{3.10}$$

The complete solution is the sum of the particular and the homogeneous solution. Boundary conditions are needed to obtain the coefficients a_0, a_1, a_2 and a_3. The

boundary conditions that satisfies the no-slip condition (where the velocities of the fluid in contact with the solid boundary has to be equal to that of the boundary) are:

1. The second order axial velocity u_2 is 0 at the wall:

$$u_2 = 0 \quad \text{at } y = 0 \tag{3.11}$$

2. The second order transverse velocity w_2 is 0 at the wall:

$$w_2 = 0 \quad \text{at } y = 0 \tag{3.12}$$

Since the flow is symmetrical about $y = \frac{h}{2}$, the boundary conditions on the axis of symmetry are:

1. u_2 must be an even function of y such that:

$$\frac{\partial u_2}{\partial y} = 0 \quad \text{at } y = \frac{h}{2} \tag{3.13}$$

2. w_2 must be an odd function of y such that it vanishes on the axis:

$$w_2 = 0 \quad \text{at } y = \frac{h}{2} \tag{3.14}$$

The complete solution is [27]:

$$\begin{aligned}\psi_2 = \frac{u_0^2}{8\beta c}\sin(2kx)\Bigg[\Bigg(4e^{-\beta y}\cos(\beta y) + 2e^{-\beta y}\sin(\beta y) + \frac{1}{2}e^{-2\beta y}\Bigg) \\ -\frac{9}{2} + 3\beta y\left(1 - \frac{y}{h}\right)\left(1 - 2\frac{y}{h}\right)\Bigg]\end{aligned} \tag{3.15}$$

The relationship between the stream function and the acoustic streaming velocity is:

$$u_2 = \frac{\partial \psi_2}{\partial y}, \qquad w_2 = -\frac{\partial \psi_2}{\partial x} \tag{3.16}$$

The axial and transverse streaming velocities u_2 and w_2 can now be calculated because ψ_2 is known. Substituting 3.15 into Eq. 3.16, the axial and transverse

acoustic streaming velocity components are:

$$u_2 = \frac{u_0^2}{8c}\sin(2kx)\left[-\left(e^{-2\beta y} + 2e^{-\beta y}\cos(\beta y) + 6e^{-\beta y}\sin(\beta y)\right)\right.$$
$$\left. +3 - 18\frac{y}{h}\left(1-\frac{y}{h}\right)\right] \tag{3.17}$$

$$w_2 = -\frac{u_0^2}{8\beta c}\cos(2kx)\left[\left(4e^{-\beta y}\cos(\beta y) + 2e^{-\beta y}\sin(\beta y) + \frac{1}{2}e^{-2\beta y}\right)\right.$$
$$\left. -\frac{9}{2} + 3\beta h\frac{y}{h}\left(1-\frac{y}{h}\right)\left(1-2\frac{y}{h}\right)\right] \tag{3.18}$$

Figures 3.8 through 3.10 show the analytical axial streaming velocity u_2. Figure 3.9 shows u_2 at an antinode in the cross section direction. The axial streaming velocity profile takes a parabolic shape. Zooming into the area near the wall it is noticeable that there is *inner streaming*. Inner streaming is a phenomenon where there are streaming cells inside the boundary layer. Figures 3.11 through 3.13 show the analytical transverse streaming velocity w_2. Figure 3.13 shows the cross sectional profile of w_2.

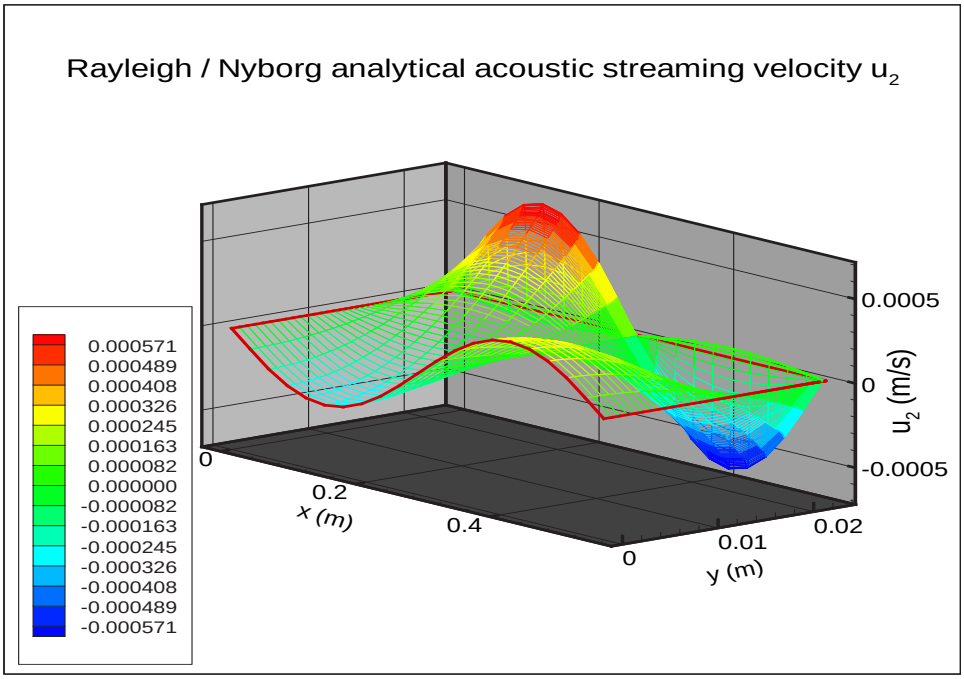

Figure 3.8. Rayleigh/Nyborg analytical axial streaming velocity u_2. $l = \lambda/2 = 0.553$ m, $r = 0.023$ m.

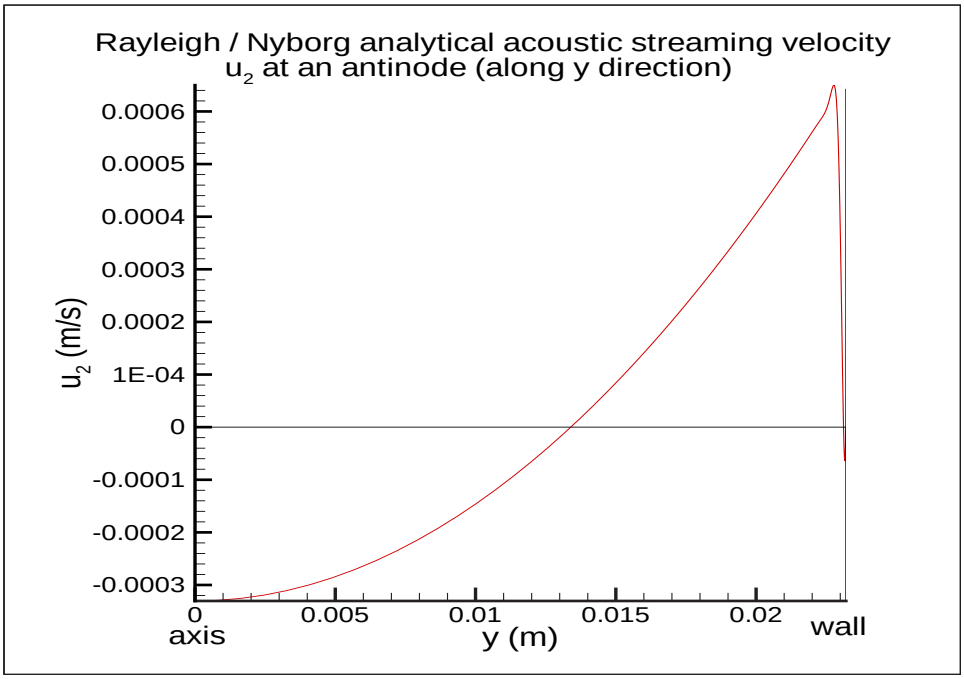

Figure 3.9. Rayleigh/Nyborg analytical axial streaming velocity u_2 at an antinode along the y direction. $l = \lambda/2 = 0.553$ m, $r = 0.023$ m.

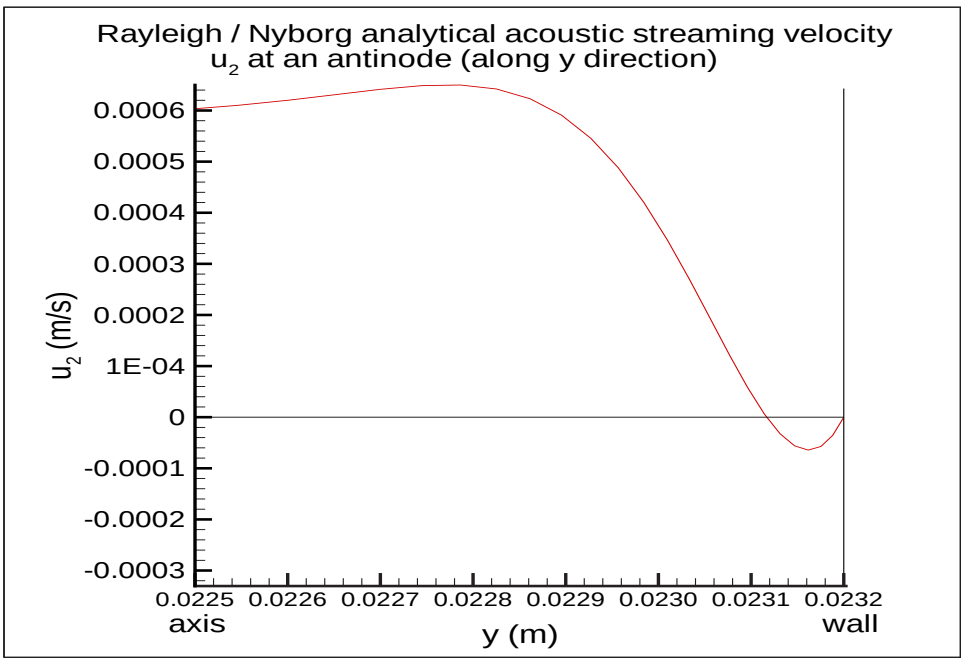

Figure 3.10. Rayleigh/Nyborg analytical axial streaming velocity u_2 near the wall at an antinode along the y direction. $l = \lambda/2 = 0.553$ m, $r = 0.023$ m.

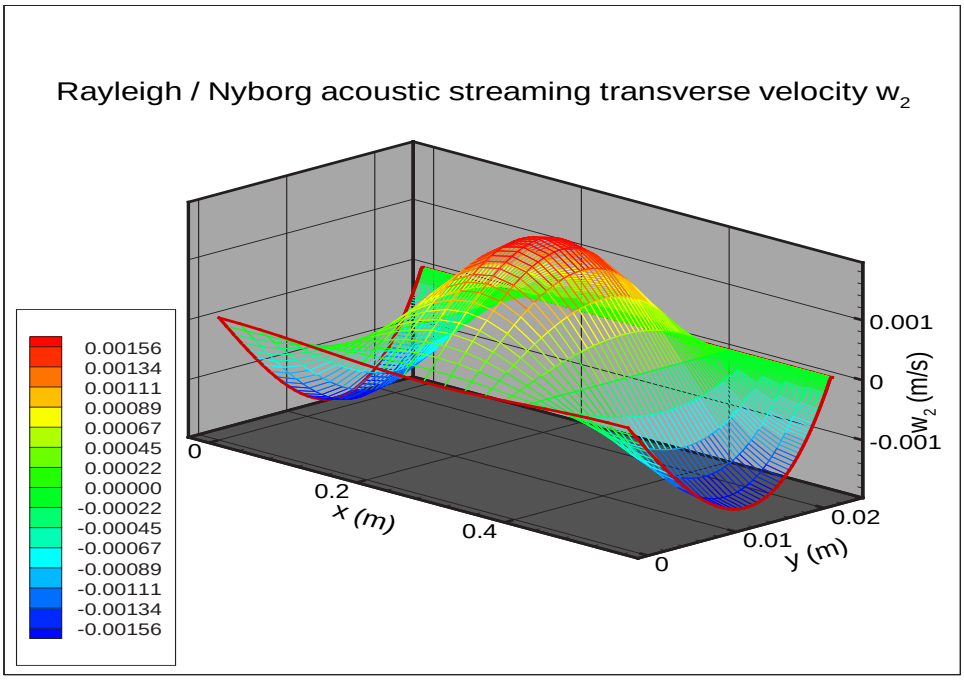

Figure 3.11. Rayleigh/Nyborg analytical transverse streaming velocity w_2. $l = \lambda/2 = 0.553$ m, $r = 0.023$ m.

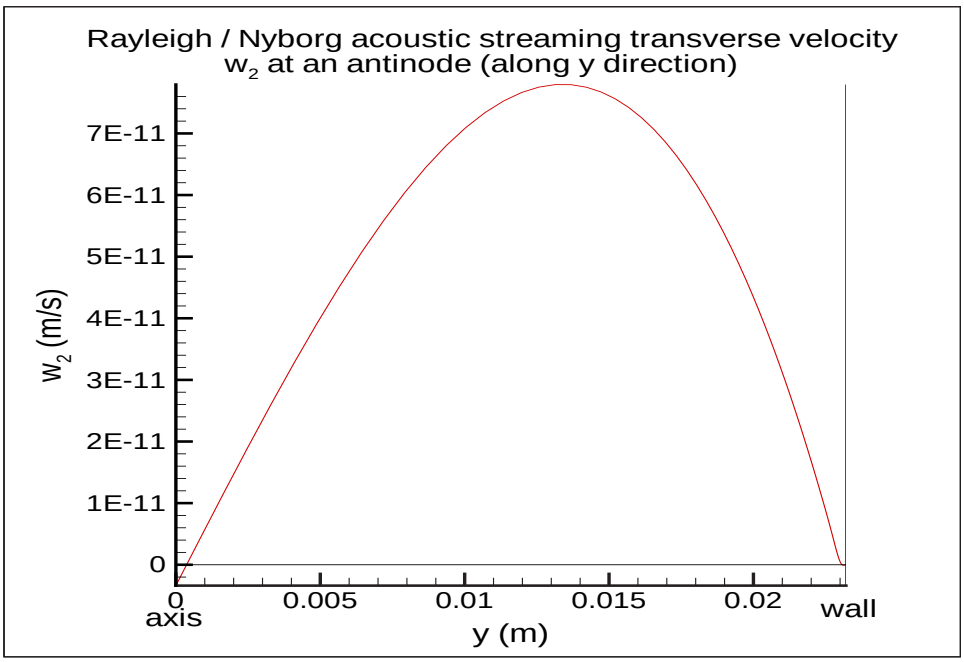

Figure 3.12. Rayleigh/Nyborg analytical transverse streaming velocity w_2 at an antinode along the y direction. $l = \lambda/2 = 0.553$ m, $r = 0.023$ m.

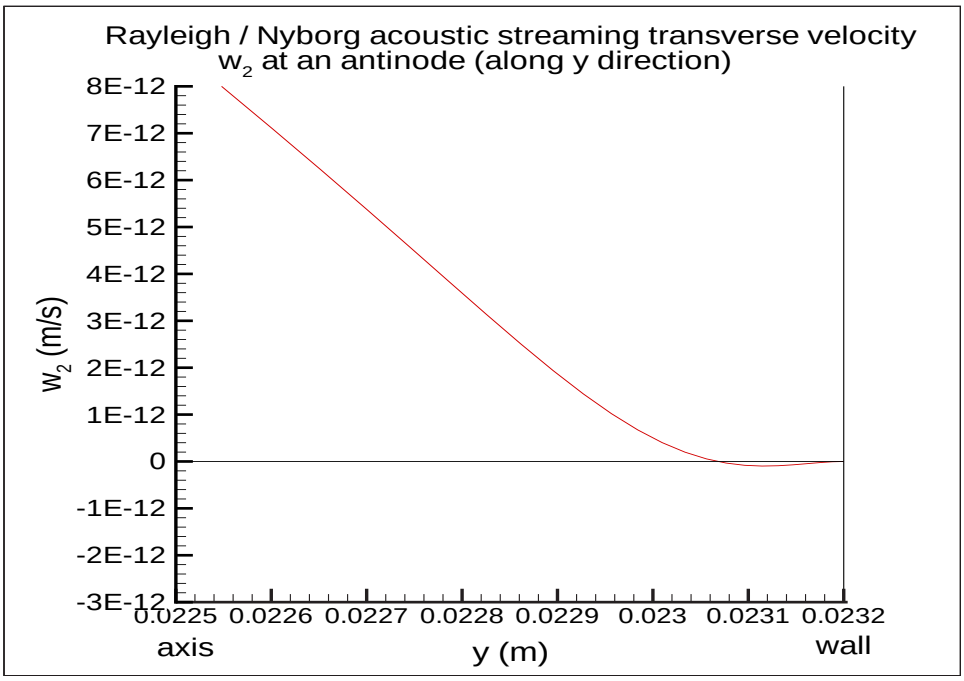

Figure 3.13. Rayleigh/Nyborg analytical transverse streaming velocity w_2 near the wall at an antinode along the y direction. $l = \lambda/2 = 0.553$ m, $r = 0.023$ m.

u_2 and w_2 are the axial and transverse components of the acoustic streaming velocity $\mathbf{u}_2$. In experiments, the measured quantity is actually the velocity of the mass instead of the streaming velocity. Because of this it is often desirable to know the time-average mass flow rate M across a given surface S, The following analysis is from Nyborg [27] pages 279 - 282. The rate at which mass flows across S is:

$$M = \int_S \rho \mathbf{u} \cdot \mathbf{dS} \tag{3.19}$$

where ρ is the fluid density and $\mathbf{u}$ is the fluid velocity. Using the method of successive approximation the acoustic variables can be expanded such as:

$$\begin{aligned} \rho &= \rho_0 + \rho_1 + \rho_2 + \cdots \\ \mathbf{u} &= \mathbf{u}_1 + \mathbf{u}_2 + \cdots \end{aligned} \tag{3.20}$$

Then Eq. 3.19 can be written as:

$$\begin{aligned} M &= \int_S (\rho_0 + \rho_1 + \rho_2 + \cdots)\,(\mathbf{u}_1 + \mathbf{u}_2 + \cdots) \cdot \mathbf{dS} \\ &= \int_S \rho_0\ \mathbf{u}_1 + \rho_1\ \mathbf{u}_1 + \rho_2\ \mathbf{u}_1 + \rho_0\ \mathbf{u}_2 + \rho_1\ \mathbf{u}_2 + \rho_2\ \mathbf{u}_2 + \cdots \mathbf{dS} \end{aligned} \tag{3.21}$$

Keeping terms only up to the second order,

$$M = \int_S \langle \rho_0 \mathbf{u}_2 + \rho_1 \mathbf{u}_1 \rangle \cdot \mathbf{dS} \tag{3.22}$$

where M now represents the time-averaged mass flow rate $< M >$. Defining new variables $\mathbf{U}$ and $\mathbf{u}_T$ [27] as

$$\mathbf{U} \equiv \mathbf{u}_2 + \mathbf{u}_T, \qquad \mathbf{u}_T \equiv \frac{1}{\rho_0} \langle \rho_1 \mathbf{u}_1 \rangle \tag{3.23}$$

enables Eq. 3.22 to be written in terms of $\mathbf{U}$:

$$\frac{M}{\rho_0} = \int_S \mathbf{U} \cdot \mathbf{dS} \tag{3.24}$$

Equation 3.23 can be interpreted as a net mass flow when acoustic streaming is present. This is analogous to a condition when there is no acoustic field present and the mass of an incompressible fluid of constant density ρ flows with steady velocity $\mathbf{U}$. Since in a steady state condition there is no net gain or loss of matter in any region bounded by any closed surface S, then

$$\oint \mathbf{U} \cdot \mathbf{dS} = 0 \tag{3.25}$$

and from the general vector theory,

$$\nabla \cdot \mathbf{U} = 0 \tag{3.26}$$

Equations 3.25 and 3.26 are valid within the second order approximation. $\mathbf{U}$ is sometimes called the *mass transport velocity*. This mass transport velocity is the quantity that is actually measured in experiments instead of the second order velocity $\mathbf{u}_2$ itself as mentioned earlier. The second order velocity $\mathbf{u}_2$ can be regarded

as the second order approximation of the Eulerian velocity, which is the average velocity of particles passing through a fixed point in space.

Nyborg defined $\mathbf{u}_T$ as

$$\mathbf{u}_T = \frac{1}{\rho_0} < \rho_1 \mathbf{u}_1 > = < \boldsymbol{\epsilon}_1 \cdot \nabla \mathbf{u}_1 > \tag{3.27}$$

where $|\boldsymbol{\epsilon}_1| = \epsilon_1$ is the displacement amplitude, i.e, $u_1 = \partial \epsilon_1 / \partial t$. Using the expression for $\mathbf{u}_1$ in Eq. 3.1 and $\epsilon_1 = \int u_1 dt$, $\mathbf{u}_T$ was calculated to be [27] :

$$u_T = \frac{u_0^2}{4c} \sin(2kx) \left[e^{-\beta y} \cos(\beta y) - e^{-2\beta y} \right] \tag{3.28}$$

The mass transport velocity is the sum of u_2 and u_T (as given in Eq. 3.23) so that:

$$\begin{aligned} U &= u_2 + u_T \\ &= -\frac{3u_0^2}{8c} \sin(2kx) \left[e^{-2\beta y} + 2e^{-\beta y} \cos(\beta y) - 1 + 6\frac{y}{h} \left(1 - \frac{y}{h} \right) \right] \end{aligned} \tag{3.29}$$

It is worth emphasizing that $\mathbf{u}_2$ is the acoustic streaming velocity which is a second order quantity added as a correction to the first order quantity, while $\mathbf{U}$ is the time-averaged mass transport velocity which is the quantity measured in experiments and is the sum of $\mathbf{u}_2$ and $\mathbf{u}_T$.

3.2 Hamilton, Illinskii, and Zabolotskaya [11]

Note that in this section, to follow the Rayleigh/Nyborg's notation, the transverse velocity is denoted as w (instead of the standard v).

3.2.1 First order particle velocities

Hamilton *et al.* also derived the first order velocities due to the presence of a standing wave [11]. Some of their assumptions are similar to Rayleigh/Nyborg's but additional assumptions are made. The computational domain is:

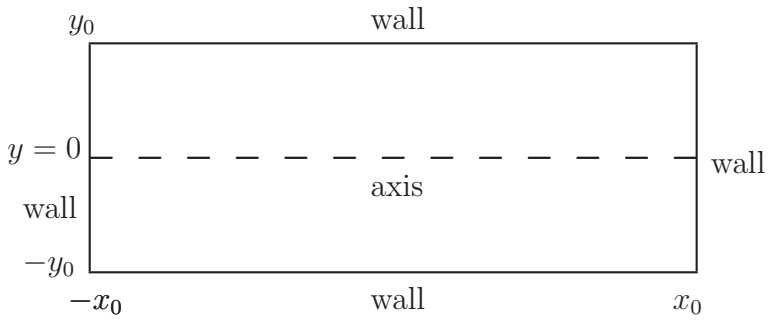

Figure 3.14. Hamilton *et al.* [11] calculation domain

The assumptions made by Hamilton *et al.* [27] are:

1. The fluid is viscous but not heat conducting.
2. The excitation is obtained by shaking the resonator along the x axis with velocity $v(t)$.
3. The resonator is excited harmonically.
4. Wave number is much larger than the viscous penetration depth ($k >> \delta_\nu$).
5. For a resonator driven at its lowest mode, the above conditions correspond to $x_0 >> \nu/c_0$.
6. The influence of the two ends of the resonator only affects the sound field to within a viscous penetration depth δ_ν.
7. δ_ν is negligible compared to the resonator length, so the boundary condition on u_1 at the tube ends have negligible effect on the field in the volume of the resonator.
8. Keep terms only up to the second order.
9. The model can take into account a specific source excitation.
10. There is no restriction to the channel width.

Since the acoustic streaming in the upper and lower half of the tube is symmetrical, the x component of the velocity, u_1, must be an even function of y such that $\partial u_1/\partial y = 0$ on the axis (i.e. at $y = 0$). The y component of the velocity, w_1, must be an odd function of y such that it vanishes at $y = 0$. The y component of the particle velocity w_1 does not vanish at the tube ends unlike u_1.

The x and y first order velocities derived by Hamilton *et al.* are [11]:

$$u_1 = -v_0 \left(1 - \frac{\cosh \alpha x}{\cosh \alpha x_0}\right) \left(1 - \frac{\cosh \beta y}{\cosh \beta y_0}\right) \tag{3.30}$$

$$w_1 = -v_0 y_0 \, \alpha \, f_\nu \, \frac{\sinh \alpha x}{\cosh \alpha x_0} \left(\frac{y}{y_0} - \frac{\sinh \beta y}{\sinh \beta y_0}\right) \tag{3.31}$$

where

$$\beta = \frac{1+i}{\delta_\nu} \tag{3.32}$$

$$\delta_\nu = \sqrt{\frac{2\nu}{\omega}} \tag{3.33}$$

$$f_\nu = \frac{\tanh \beta y_0}{\beta y_0} \tag{3.34}$$

$$\alpha = \frac{i\omega/c_0}{\sqrt{1 - f_\nu}} \tag{3.35}$$

Note that β here is not exactly the same as β defined in Rayleigh/Nyborg section. In the Rayleigh/Nyborg section, $\beta = 1/\delta_\nu$ and it is a real quantity. Here β is still the inverse of the viscous penetration depth but is a complex quantity. Figures 3.15 and 3.16 show the first order axial velocity derived by Hamilton *et al.* and Figs. 3.17 and 3.18 show the first order transverse velocity.

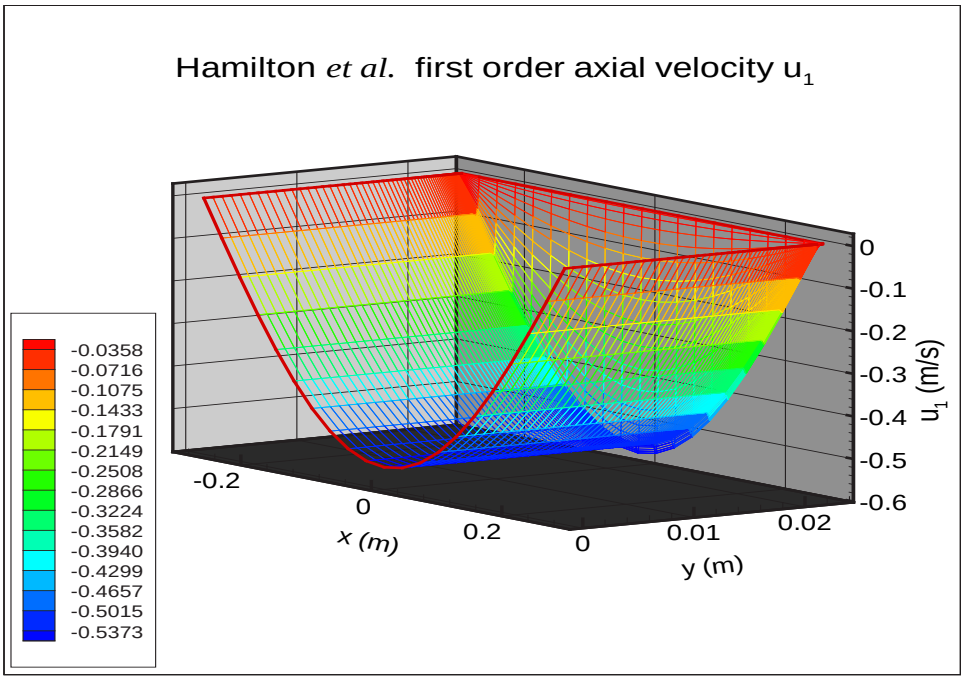

Figure 3.15. Hamilton *et al.* first order axial velocity u_1. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

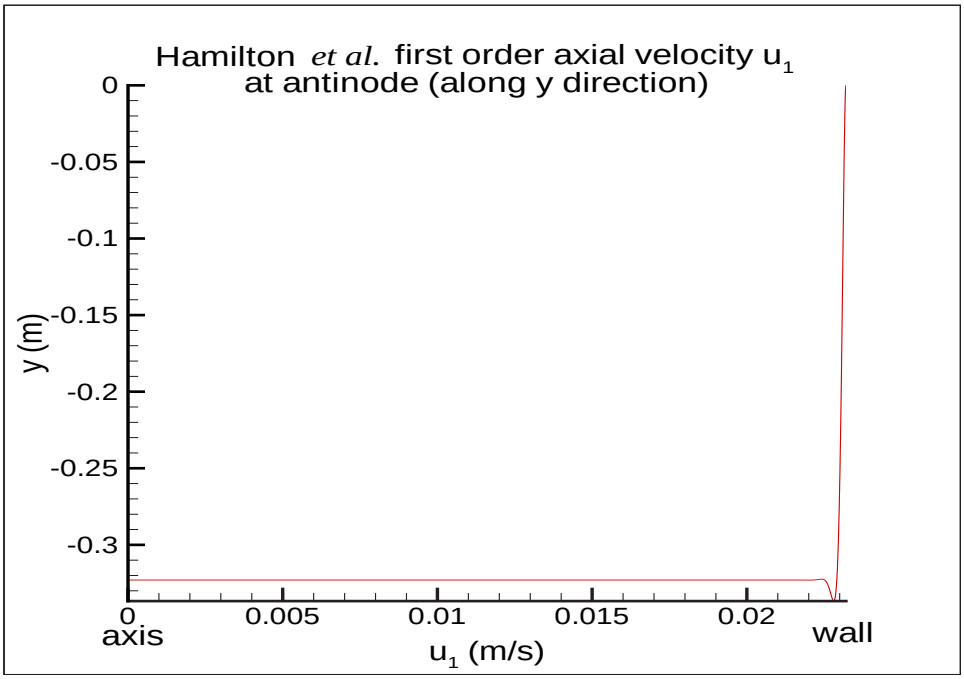

Figure 3.16. Hamilton *et al.* first order axial velocity u_1 along y direction. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

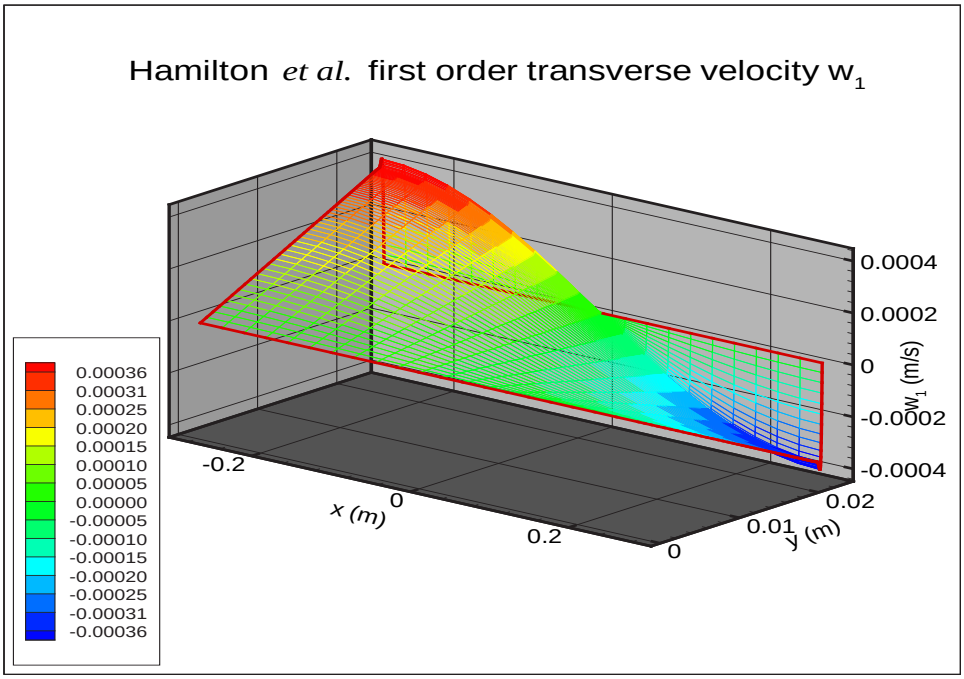

Figure 3.17. Hamilton *et al.* first order transverse velocity w_1. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

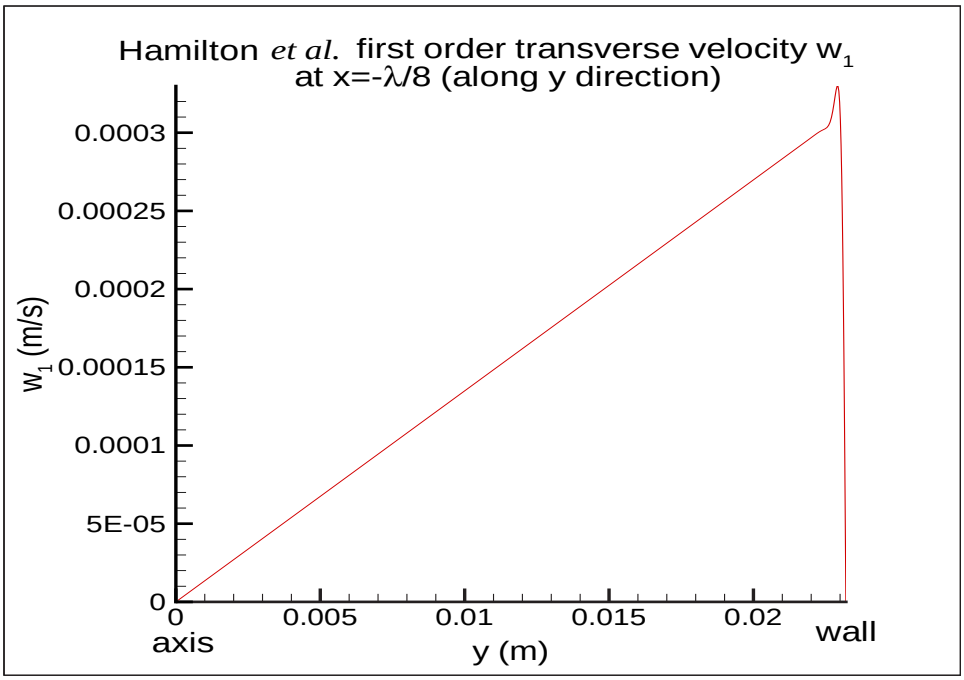

Figure 3.18. Hamilton *et al.* first order transverse velocity w_1 along y direction. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

The forcing function is calculated by substituting the first order velocities in Eqs. 3.30 and 3.31 into Eq. 3.3. The resulting forcing function is shown in Figs. 3.19 and 3.20. Notice that the forcing function has the same features as the one derived by Nyborg. In Fig. 3.20 the occurrence of the bump near the boundary on the forcing function can be observed.

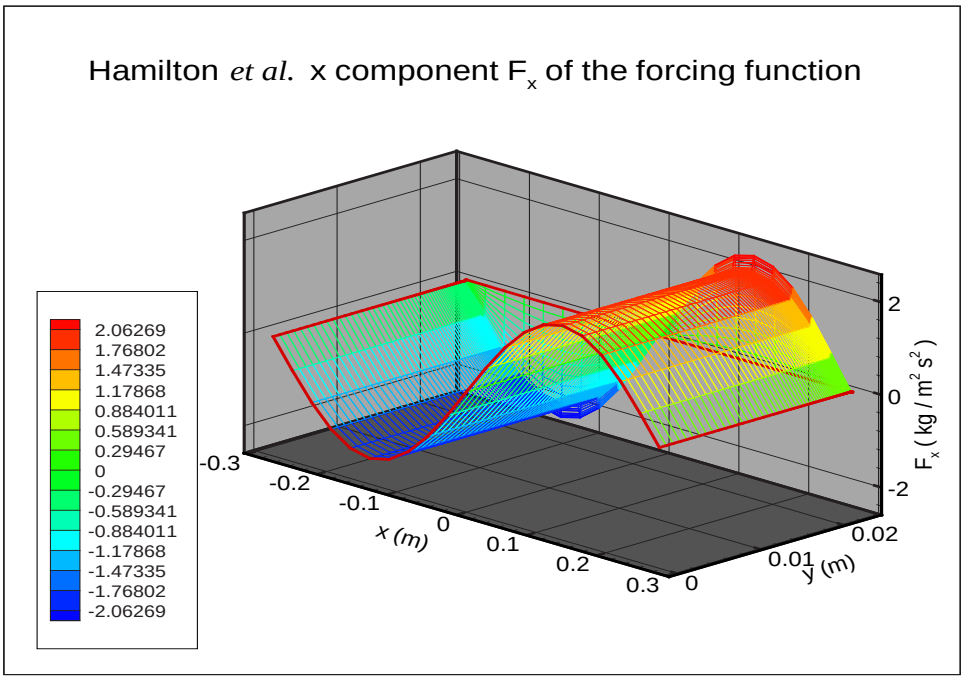

Figure 3.19. Hamilton *et al.* x component F_x of forcing function. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

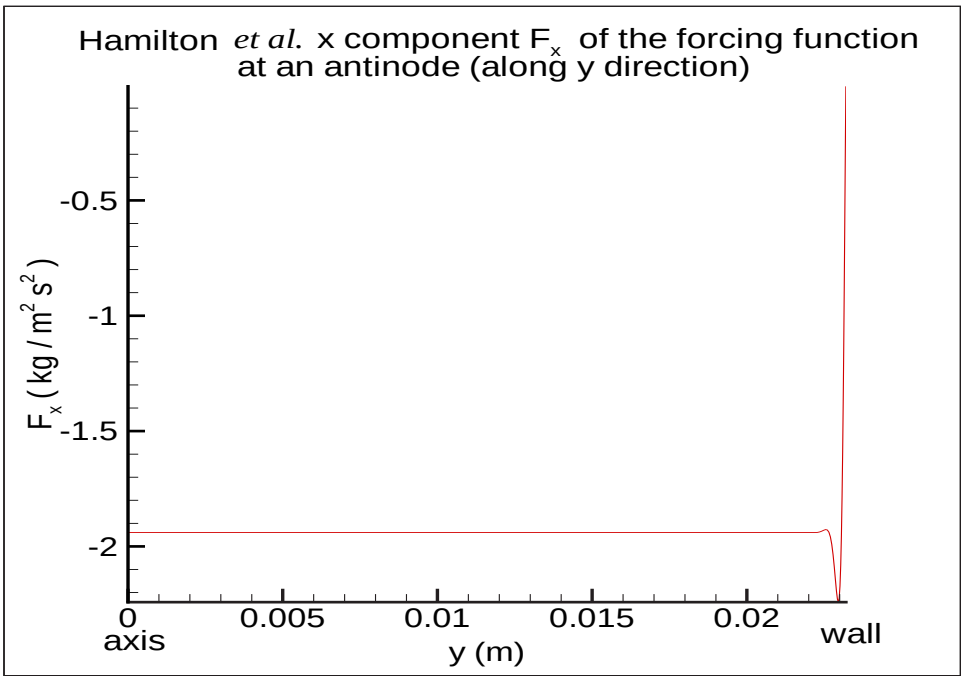

Figure 3.20. Hamilton *et al.* x component F_x of forcing function at an antinode along y direction. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

3.2.2 Analytical acoustic streaming velocities

Through the continuity and momentum equations Hamilton *et al.* [11] calculated the averaged mass transport velocity. The x component of the mass transport velocity is:

$$u_x^M = U_x^M + \frac{1}{y_0}\left[3A_3(x)\frac{y^2}{y_0^2} + A_1(x)\right] \tag{3.36}$$

where

$$U_x^M = V_0 \,\mathrm{Re}\left\{G(x)[H_1(y) + iH_2(y)] + \frac{1}{4}i(1-f_\nu)G^*(x)Y_x^*(y)\right\} \tag{3.37}$$

and

$$V_0 = \frac{2|v_0|^2}{x_0\omega} \tag{3.38}$$

is a characteristic velocity amplitude,

$$G(x) = \alpha^* x_0 \frac{\sinh \alpha^* x}{\cosh \alpha^* x_0}\left(\frac{\cosh \alpha x}{\cosh \alpha x_0} - 1\right) \tag{3.39}$$

$$H_1(y) = \frac{\cosh(2y/\delta_\nu) - \cos(2y/\delta_\nu)}{8|\cosh\beta y_0|^2} - \mathrm{Im}\frac{\cosh\beta y}{\cosh\beta y_0} \tag{3.40}$$

$$\begin{aligned} H_2(y) = f_\nu^* \frac{\cosh\beta y - \beta y \sinh\beta y}{4\cosh\beta y_0} + \frac{1}{4}\frac{\cosh\beta^* y}{\cosh\beta^* y_0} \\ - \frac{\cosh(2y/\delta_\nu) + i\cos(2y/\delta_\nu)}{8\beta\delta_\nu|\cosh\beta y_0|^2} \end{aligned} \tag{3.41}$$

$$Y_x(y) = 1 - \frac{\cosh\beta y}{\cosh\beta y_0} \tag{3.42}$$

$A_1(x)$ and $A_3(x)$ are determined from the no-slip boundary condition to be

$$A_1(x) = -\frac{3}{2}\Psi(x, y_0) + \frac{1}{2}\, y_0\, V(x, y_0) \tag{3.43}$$

$$A_3(x) = \frac{1}{2}\Psi(x, y_0) - \frac{1}{2}\, y_0\, V(x, y_0) \tag{3.44}$$

where

$$\Psi = V_0\delta_\nu \mathrm{Re}\left\{G(x)[H_3(y) + iH_4(y)] + \frac{1}{4}i\,(1 - f_\nu)\,G^*(x)H_5(y)\right\} \tag{3.45}$$

and

$$H_3(y) = \frac{\sinh(2y/\delta_\nu) - \sin(2y/\delta_\nu)}{16|\cosh\beta y_0|^2} - \mathrm{Im}\frac{\sinh\beta y}{\beta\delta_\nu\cosh\beta y_0} \tag{3.46}$$

$$\begin{aligned} H_4(y) = f_\nu^* \frac{2\sinh\beta y - \beta y\cosh\beta y}{4\beta\delta_\nu\cosh\beta y_0} + \frac{i}{4\beta\delta_\nu}\frac{\sinh\beta^* y}{\cosh\beta^* y_0} \\ - \frac{\sinh(2y/\delta_\nu) + i\sin(2y/\delta_\nu)}{16\beta\delta_\nu|\cosh\beta y_0|^2} \end{aligned} \tag{3.47}$$

$$H_5(y) = \frac{y}{\delta_\nu} - \frac{i}{\beta\delta_\nu}\frac{\sinh\beta^* y}{\cosh\beta^* y_0} \tag{3.48}$$

The mass transport velocity $\bar{\mathbf{u}}^M$ can be expressed in terms of the Eulerian streaming velocity $\bar{\mathbf{u}}$ and the particle velocity $\widetilde{\mathbf{u}}$:

$$\bar{\mathbf{u}}^M = \bar{\mathbf{u}} + \frac{\mathrm{Re}(\,\tilde{p}\,\tilde{\mathbf{u}}^*)}{2\rho_0 c_0^2} \tag{3.49}$$

Note that the calculated quantity is the time-averaged mass transport velocity (which is the equivalent of **U** in Nyborg's calculation) and **not** the acoustic streaming velocity. The acoustic streaming velocity $\mathbf{u}_2$ can be obtained by subtracting $\mathrm{Re}(\tilde{p}\,\tilde{\mathbf{u}}^*)/2\rho_0 c_0^2$ from $\bar{\mathbf{u}}^M$. Care must be taken when comparing the acoustic streaming velocity between the different groups as they present different velocities as their computation results. Nevertheless the streaming velocity $\mathbf{u}_2$ can usually be worked out from the presented results.

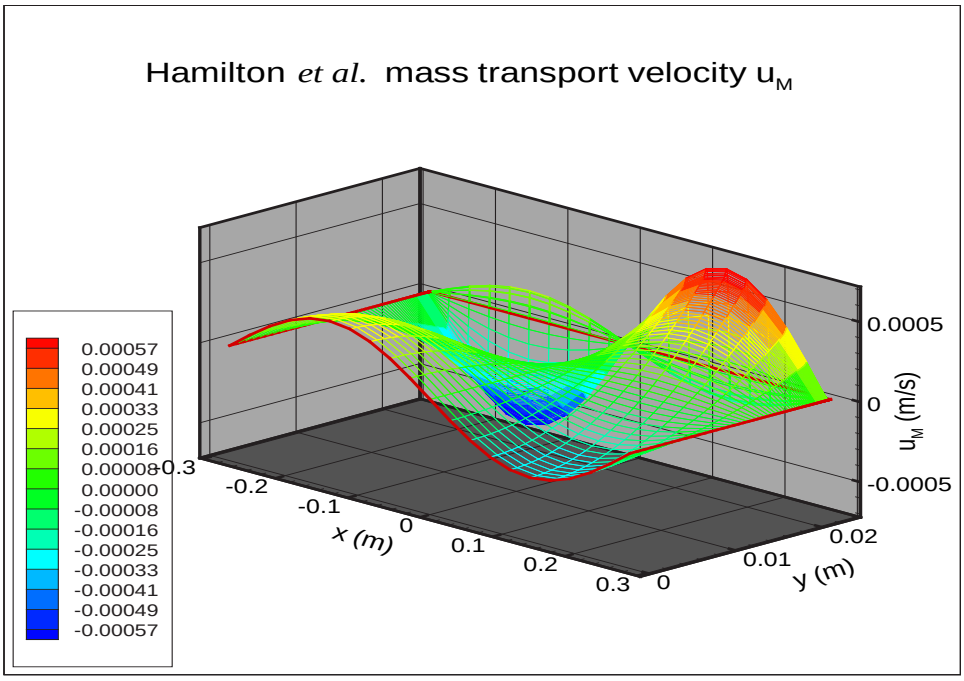

Figure 3.21. Hamilton *et al.* analytical mass transport velocity u_M. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

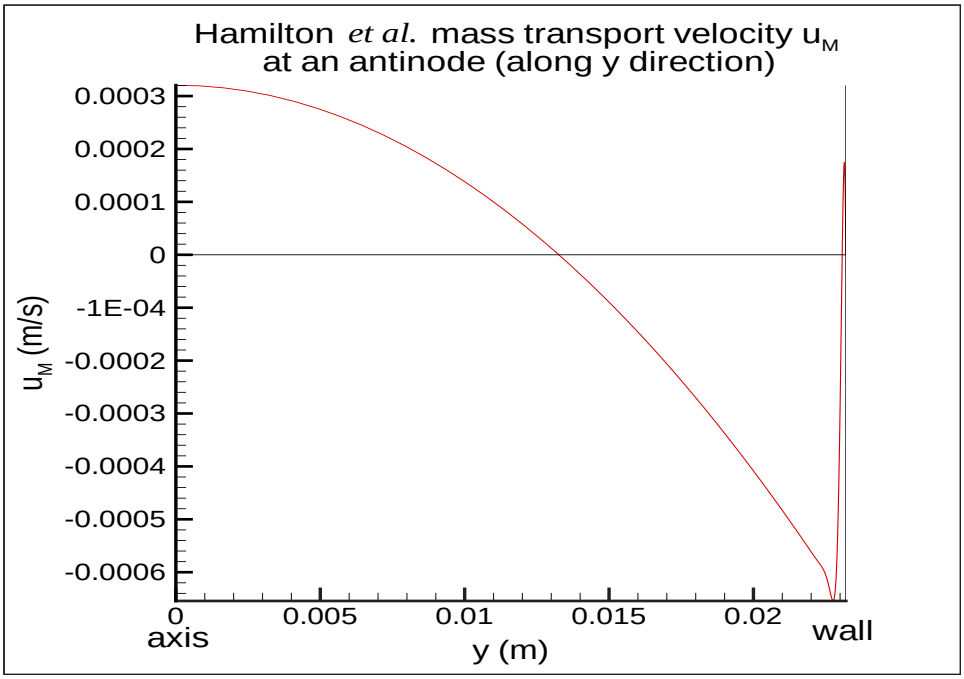

Figure 3.22. Hamilton *et al.* analytical mass transport velocity u_M on the axis along x direction. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

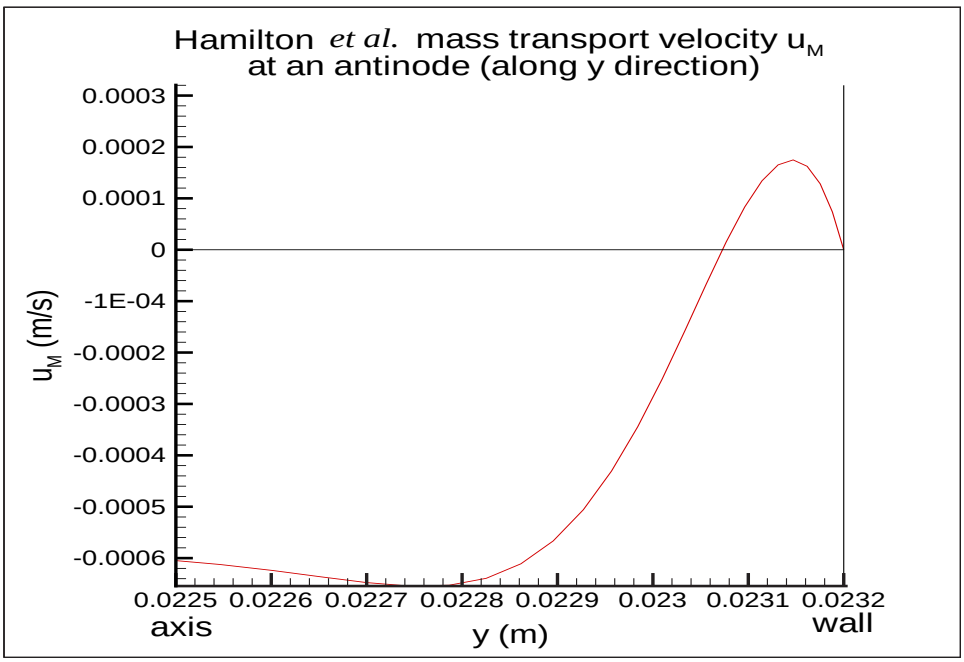

Figure 3.23. Hamilton *et al.* analytical mass transport velocity u_M at an antinode along y direction. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

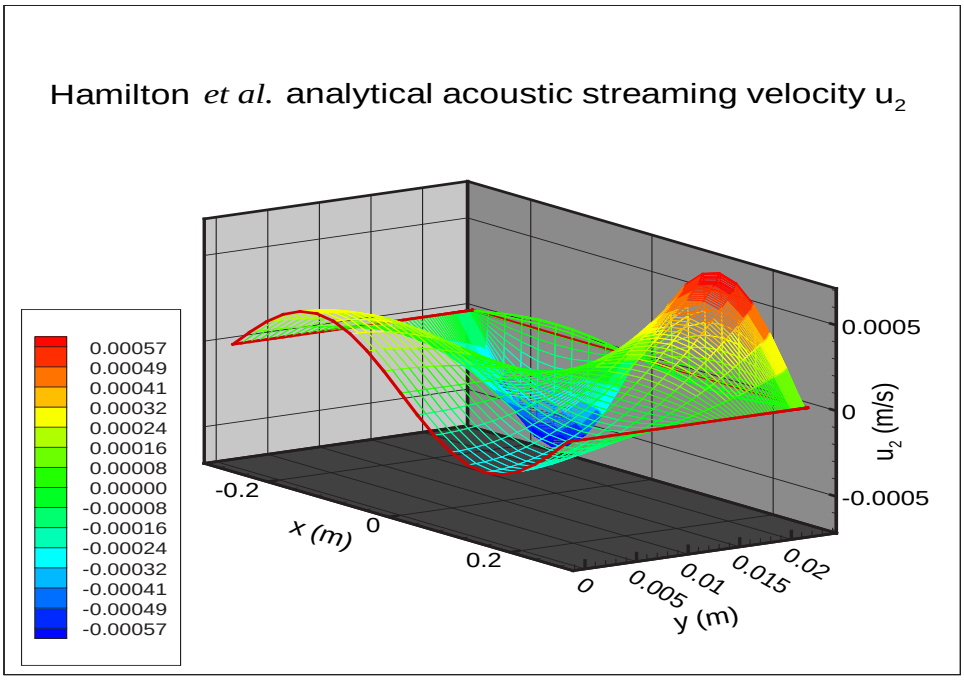

Figure 3.24. Hamilton *et al.* analytical acoustic streaming velocity u_2. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

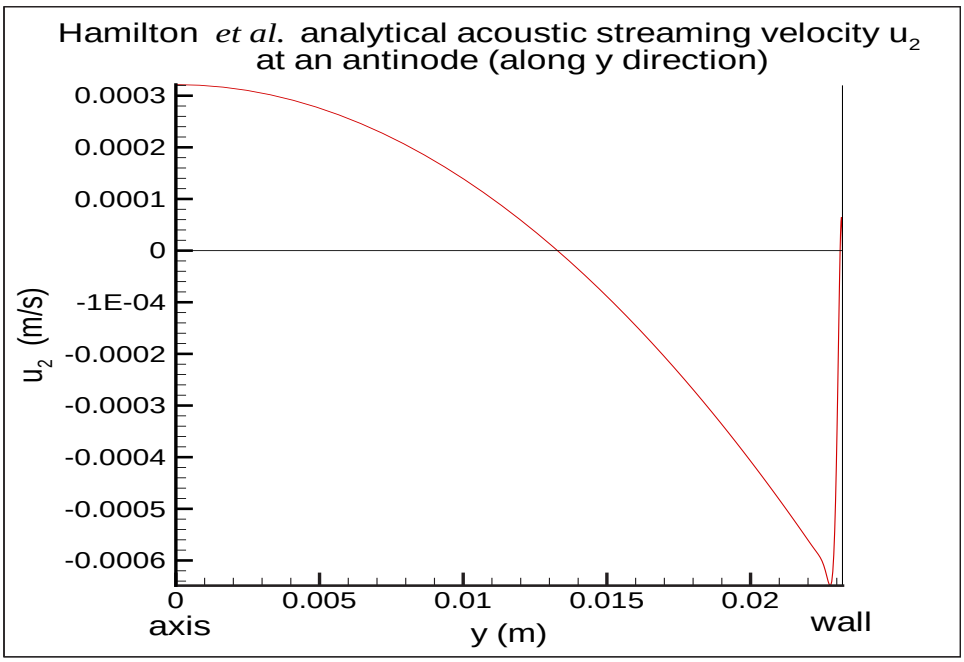

Figure 3.25. Hamilton *et al.* analytical acoustic streaming velocity u_2 at an antinode along y direction. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

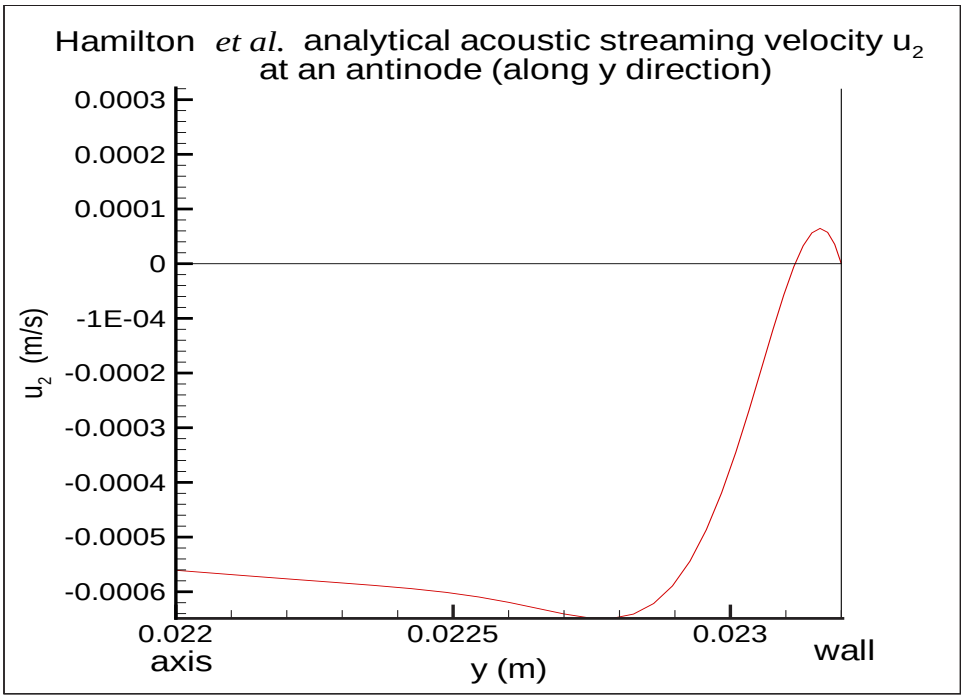

Figure 3.26. Hamilton *et al.* analytical acoustic streaming velocity u_2 at an antinode along y direction. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

3.3 Bailliet, Gusev, Raspet and Hiller [3]

Note that in this section, to follow Rayleigh/Nyborg's notation, the transverse velocity is denoted as w (instead of the standard v).

3.3.1 First order particle velocities

Another group that calculated the first order velocity of a standing wave in a channel and the acoustic streaming that resulted from it is the Bailliet *et al.* group. Their computational domain is:

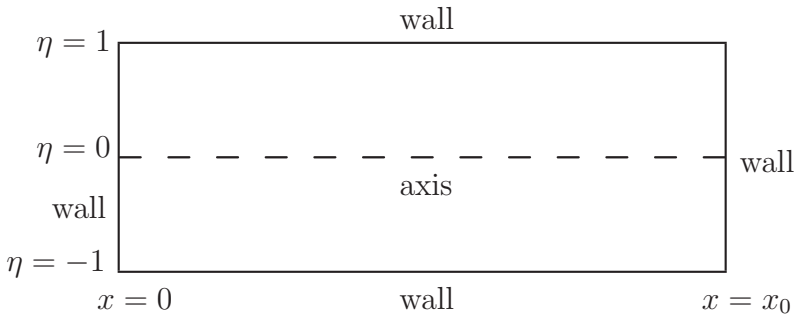

Figure 3.27. Bailliet *et al.* calculation domain

They assumed that:

1. The fluid is homogeneous.
2. The only forces acting on the fluid are surface stresses due to elasticity and viscosity.
3. The second viscosity coefficient is neglected.
4. An acoustic wave propagates laminarly in an ideal gas along the x axis between two *infinitely* wide rigid plates or in a cylindrical tube.
5. The transverse dimensions (τ) are supposed to be much smaller than the longitudinal (x) ones.

6. Since $\tau \ll x$, the ratio of the wavelength and the transverse dimension is large so that:
$$\frac{c_0}{R\omega} \gg 1$$
where R is the typical transverse dimension, c_0 is the adiabatic speed of sound, and ω is the angular frequency of the acoustic oscillation.

7. Transverse variations are much more rapid than the longitudinal ones.

8. The acoustic wave is a plane wave.

9. The acoustic streaming velocity is slow enough that it does not alter the first order variables.

10. There is no mean flow apart from acoustic streaming.

11. The transverse variation of acoustic pressure is neglected.

12. Axial particle velocity is much greater than transverse particle velocity: $u_x \gg u_\tau$

The x component of the first order velocity was calculated using the Navier Stokes equation (to the first order). In the presence of a standing wave between parallel walls the first order acoustic axial velocity is:

$$u_1 = \frac{\partial p_1/\partial x}{i\omega\rho_0} F_\eta \tag{3.50}$$

where

$$F_\eta = 1 - \frac{\cosh(b\eta)}{\cosh b} \tag{3.51}$$

and

$$b = \frac{(1-i)R}{\delta_\nu} = \frac{(1-i)R\sqrt{\omega}}{\sqrt{2\nu}} \tag{3.52}$$

The acoustic pressure p_1 is:

$$p_1(x) = \frac{p_1(0)}{2}\left(e^{-ikx} + e^{ikx}\right) \tag{3.53}$$

with the complex wave number

$$k = \frac{\omega}{c_0\sqrt{\frac{\gamma-(\gamma-1)F_t}{F}}}$$

The functions F and F_t are defined as:

$$F = 1 - \frac{\tanh b}{b} \tag{3.54}$$

$$F_t = 1 - \frac{\tanh(\sqrt{\sigma}b)}{\sqrt{\sigma}b} \tag{3.55}$$

The transverse component of the first order velocity can be obtain from the continuity equation,

$$\frac{D\rho}{Dt} + \rho\nabla\cdot\mathbf{u} = 0 \tag{3.56}$$

To the first order, the continuity equation gives:

$$-i\omega\rho_1 + \frac{\partial}{\partial x}(\rho_0 u_1) + \frac{\rho_0}{R}\frac{\partial}{\partial\eta}w_1 = 0 \tag{3.57}$$

u_1 is known and R is the distance from the axis of symmetry to the wall.
The first order transverse velocity neglecting the effect of temperature is:

$$w_1 = \frac{iR}{\omega\rho_0}\left[\frac{\partial^2 p_1}{\partial x^2}(\eta-\phi) + \frac{\omega^2}{c_0^2}p_1(\eta+(\gamma-1)\phi_t)\right] \tag{3.58}$$

with

$$\phi = \frac{\sinh(b\eta)}{b\cosh b} \tag{3.59}$$

$$\phi_t = \frac{\sinh(\sqrt{\sigma}b\eta)}{b\sqrt{\sigma}\cosh(\sqrt{\sigma}b)} \tag{3.60}$$

Figures 3.28 through 3.35 show the first order axial and transverse velocity as derived by Bailliet *et al.* for the parallel plate case. Figures 3.28 and 3.29 show the real part of the first order axial velocity. Figure 3.29 is the real part of the axial velocity profile in the cross sectional direction. Figures 3.30 and 3.31 show the imaginary part of the first order axial velocity. Figure 3.31 is the imaginary part of the axial velocity profile in the cross sectional direction. Figures 3.32 and

3.33 show the real part of the first order transverse velocity. Figure 3.33 is the real part of the transverse velocity profile in the cross sectional direction. Figures 3.34 and 3.35 show the imaginary part of the first order transverse velocity. Figure 3.35 is the imaginary part of the transverse velocity profile in the cross sectional direction. As with the other groups' first order axial velocity, there is the same feature where a bump occurs in the boundary.

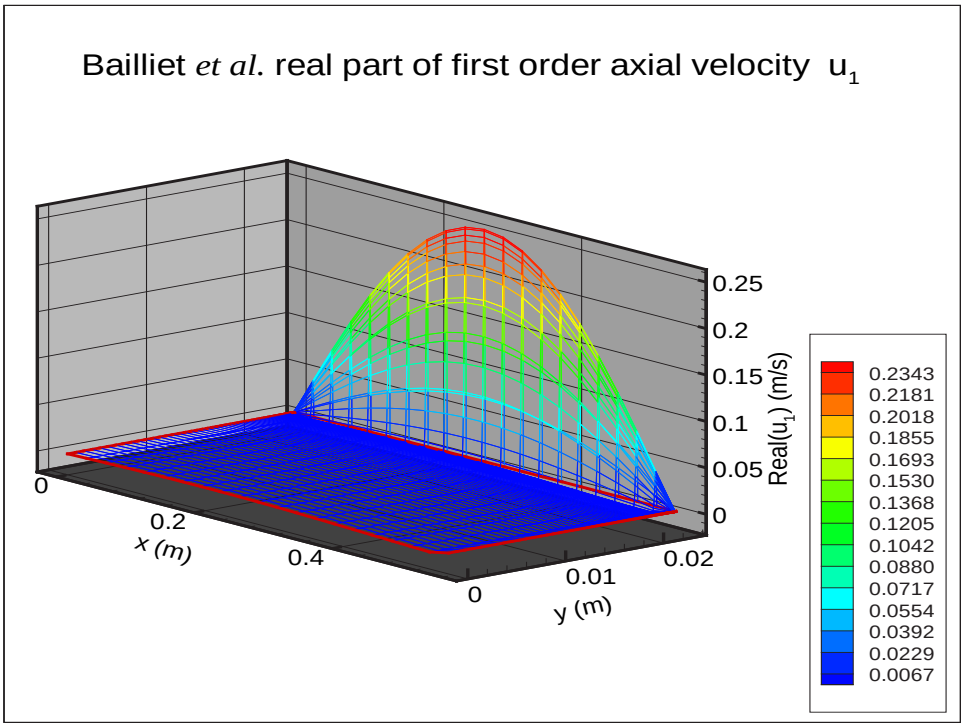

Figure 3.28. Bailliet *et al.* real part of first order axial velocity u_1. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

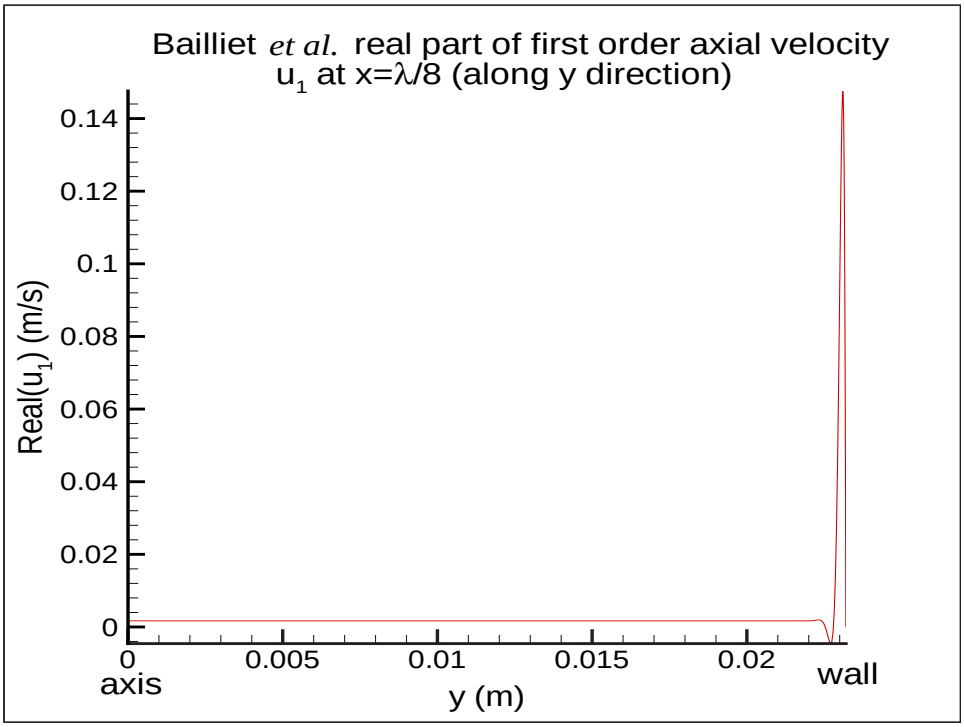

Figure 3.29. Bailliet *et al.* real part of first order axial velocity u_1 along y direction. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

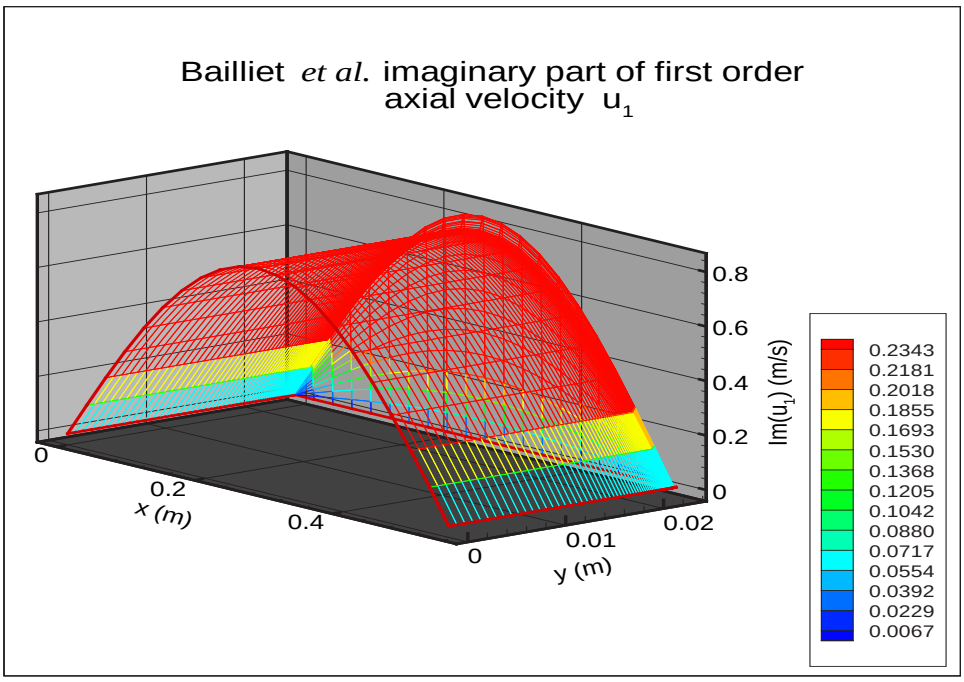

Figure 3.30. Bailliet *et al.* imaginary part of first order axial velocity u_1. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

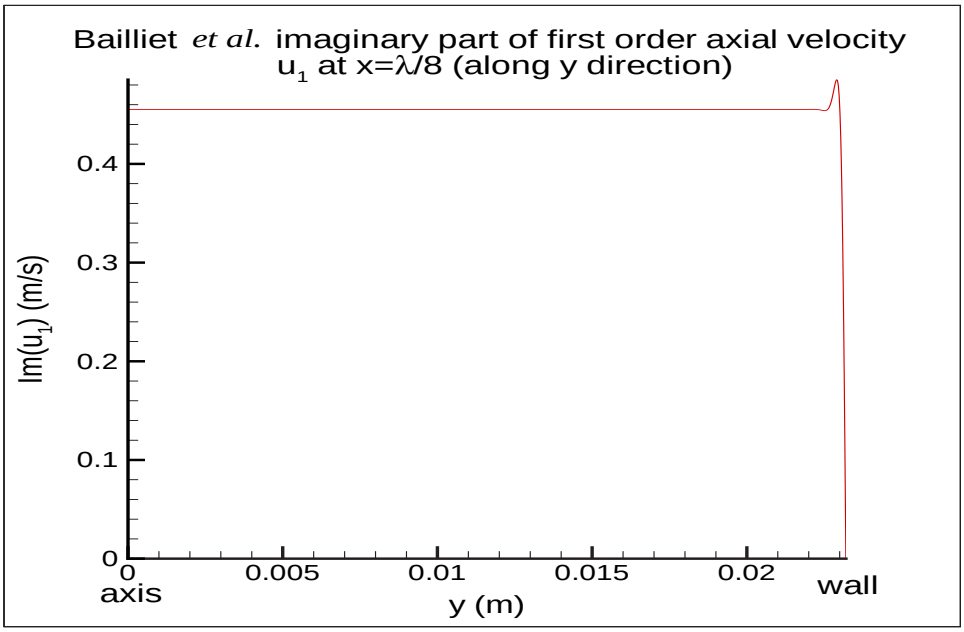

Figure 3.31. Bailliet *et al.* imaginary part first order axial velocity u_1 along y direction. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

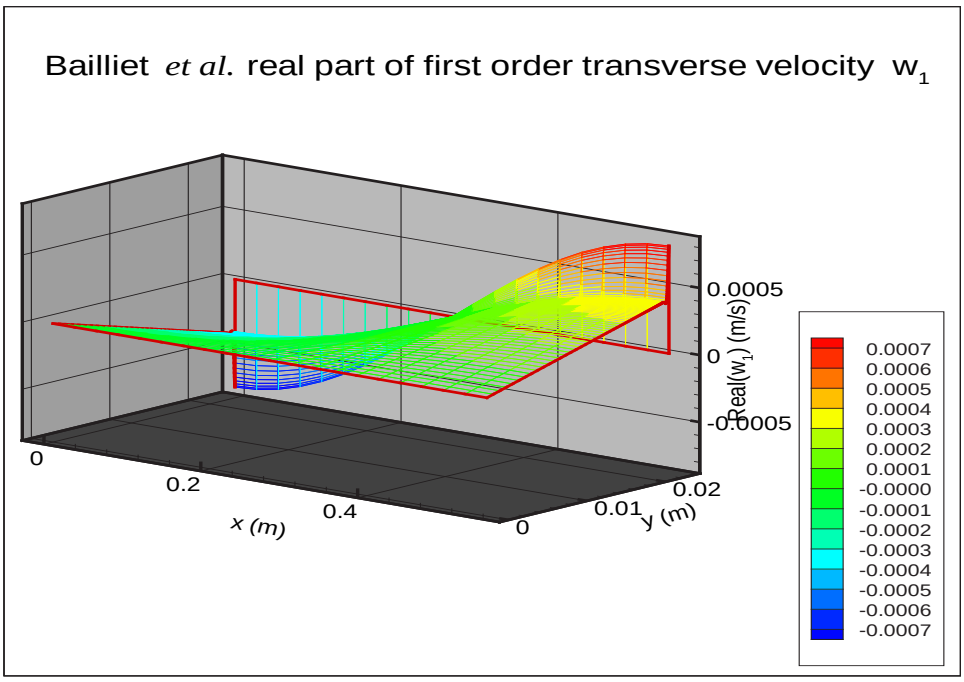

Figure 3.32. Bailliet *et al.* real part of first order transverse velocity w_1. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

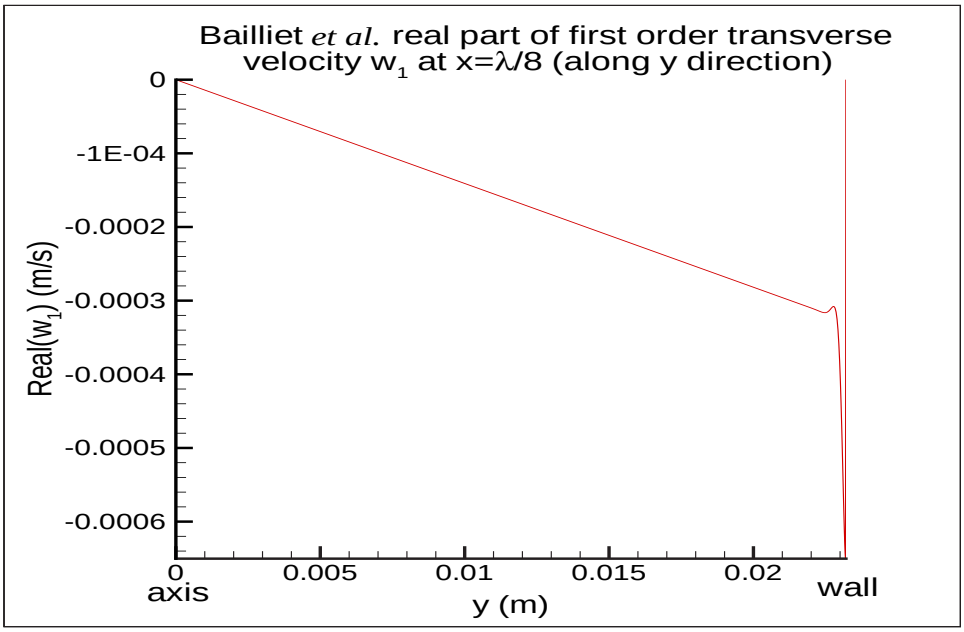

Figure 3.33. Bailliet *et al.* real part of first order transverse velocity w_1 along y direction. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

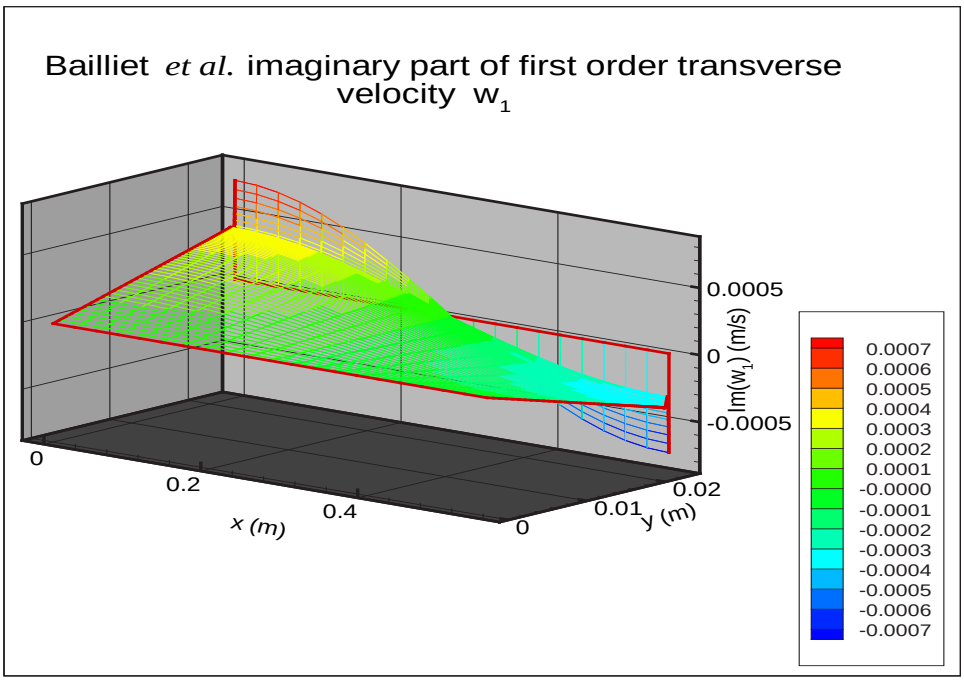

Figure 3.34. Bailliet *et al.* imaginary part first order transverse velocity w_1. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

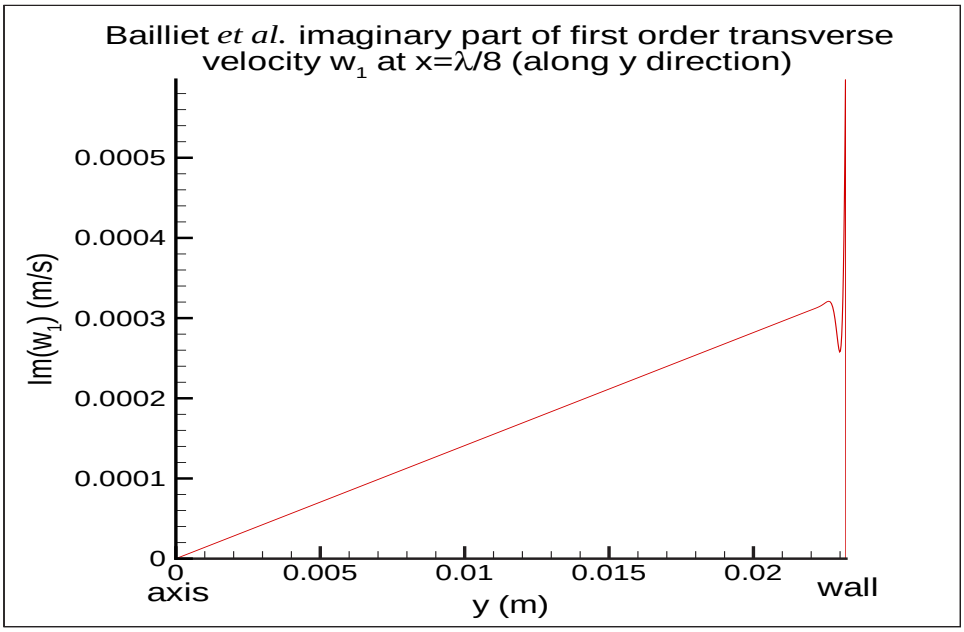

Figure 3.35. Bailliet *et al.* imaginary part first order transverse velocity w_1 along y direction. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

The forcing function is calculated by substituting the first order velocities in Eqs. 3.50 and 3.58 into Eq. 3.3. The resulting forcing function is shown in Figs. 3.36 and 3.37. Notice that the forcing function has the same feature as the one derived by Rayleigh/Nyborg and Hamilton *et al.* Figure 3.37 shows the occurrence of the bump near the boundary on the forcing function.

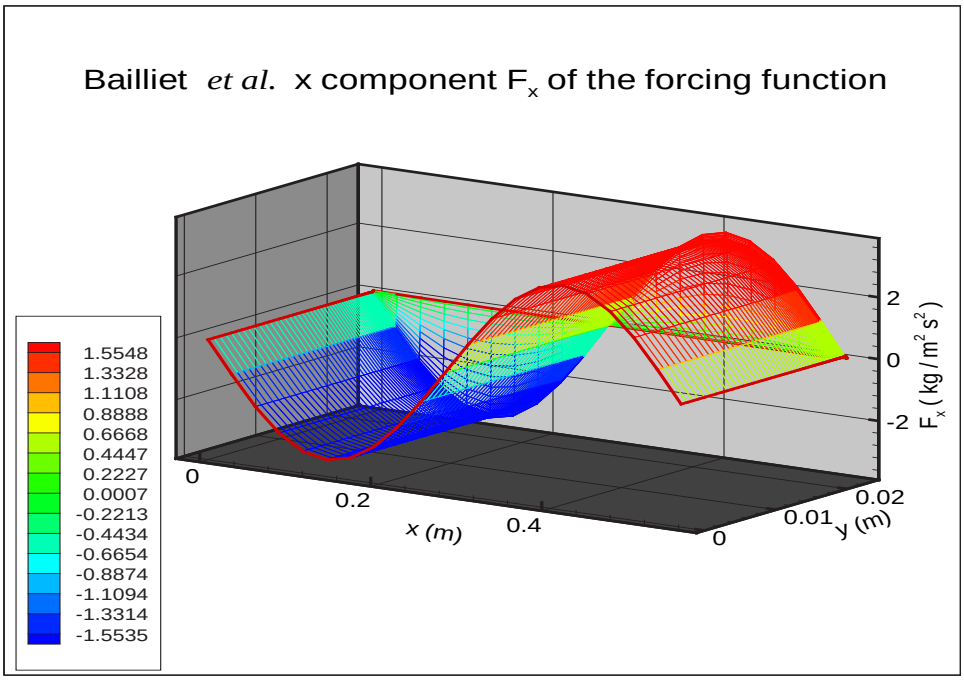

Figure 3.36. Bailliet *et al.* x component of forcing function. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

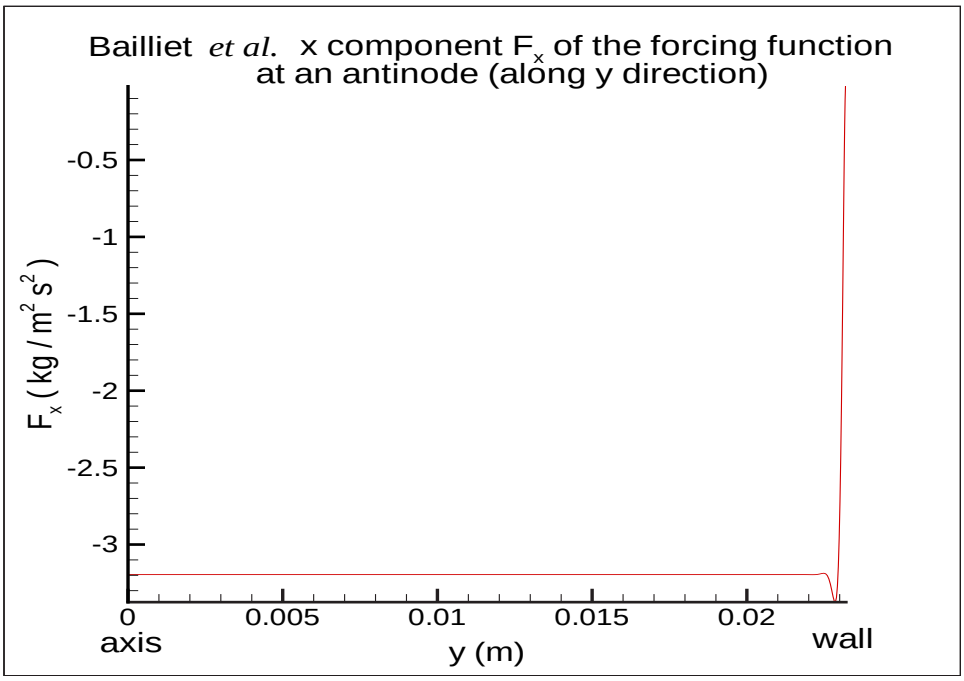

Figure 3.37. Bailliet *et al.* x component of forcing function at an antinode along y direction. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

3.3.2 Analytical acoustic streaming velocities

For a channel with parallel plates located at $y = \pm R$ and $\eta = y/R$, by solving the momentum and continuity equation they obtained the analytical axial streaming velocity (neglecting the effect of temperature):

$$\begin{aligned}\langle u_2 \rangle = &-\frac{3}{8\omega\rho_0^2 c_0^2}\left(\eta^2 - 1\right)\mathrm{Re}\left(ip_1\frac{\partial p_1^*}{\partial x}\Psi_1\right) \\ &-\frac{1}{2\omega\rho_0^2 c_0^2}\mathrm{Re}\left(ip_1\frac{\partial p_1^*}{\partial x}\Psi_2\right)\end{aligned} \tag{3.61}$$

The functions Ψ_1 and Ψ_2 are:

$$\begin{aligned}\Psi_1 = {} & \beta(\gamma-1)\left[\frac{2\sigma}{\sigma+1} - 2(1-F^*)\left(1+\frac{\sigma-1}{(\sigma+1)^2}\right)\right. \\ & \left. + \frac{4\sigma(1-F_t)}{(\sigma+1)^2} + 2b^2\sigma(1-F_t)\frac{1-F^*}{\sigma+1}\right] \\ & + 2(\gamma-1)(1-F_t)\left(\frac{1}{\sigma}+\frac{1-\sigma}{(\sigma+1)^2}-\frac{1}{\sigma+1}\right) \\ & - \frac{2(\gamma-1)(1-F^*)}{\sigma+1}\left(\sigma+\frac{2}{\sigma+1}\right) \\ & + 2(\gamma-2)(1-F^*) + 2 + 2b^2(1-F^*) - \frac{2(\gamma-1)}{\sigma(\sigma+1)} \\ & + 2b^2(\gamma-1)\frac{(1-F^*)(1-F_t)}{\sigma+1} \\ & - \frac{1+(\gamma-1)(1-F_t)}{F}\left[3(1-F) - 8(1-F^*) + b^2|1-F|^2 + 5 + 2b^2(1-F^*)\right]\end{aligned} \tag{3.62}$$

$$\begin{aligned}\Psi_2 = {} & \beta(\gamma-1)\left[\frac{\sigma}{\sigma+1} - (1-F_\eta^*) + (1-F_\eta^*)\frac{1-F_{\eta t}}{\sigma+1} - b^2\phi^*\phi_t\frac{\sigma}{\sigma+1}\right. \\ & \left. + \frac{\sigma b^2(1-F_t)(1-F)}{\sigma+1}\right] + F_\eta^* + (1-F^*)b^2 - \eta\phi^* b^2 - \frac{\gamma-1}{\sigma(1+\sigma)} \\ & + \frac{(\gamma-1)(1-F_{\eta t})}{\sigma} - \frac{\gamma-1}{\sigma+1}(1-F_\eta^*)(1-F_{\eta t}) + \frac{b^2(\gamma-1)(1-F^*)(1-F_t)}{(\sigma+1)} \\ & - b^2\phi_t\phi^*\frac{\gamma-1}{\sigma+1} - \frac{1+(\gamma-1)(1-F_t)}{F}\times \\ & \left[(1-F^*)b^2 + 3F_\eta^* - \eta\phi^* b^2\frac{1}{2} + \frac{|1-F_\eta|^2}{2} - b^2\frac{|\phi|^2}{2} + \frac{b^2|1-F|^2}{2} - F_\eta\right]\end{aligned} \tag{3.63}$$

Figures 3.38 through 3.40 show the analytical acoustic streaming velocity as derived by Bailliet *et al.* Figures 3.39 and 3.40 are the cross sectional profile of the streaming velocity at an antinode. From Fig. 3.40 it can be seen that there is no inner streaming in the boundary layer area unlike the other groups' calculations.

The analytical acoustic streaming obtained by the three groups will be compared in Chapter 6.

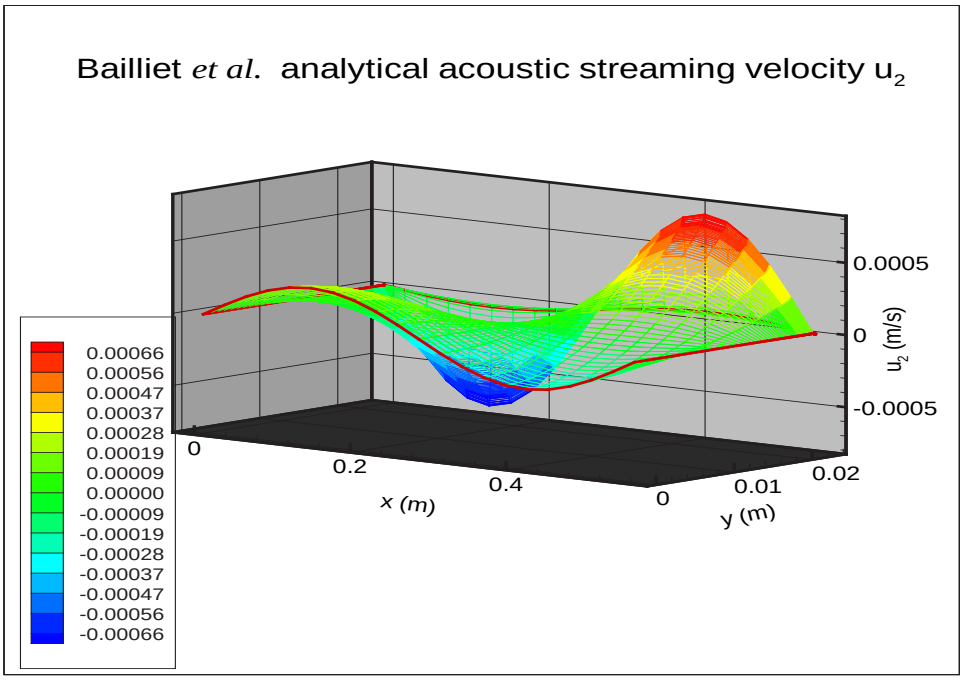

Figure 3.38. Bailliet *et al.* analytical acoustic streaming velocity u_2. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

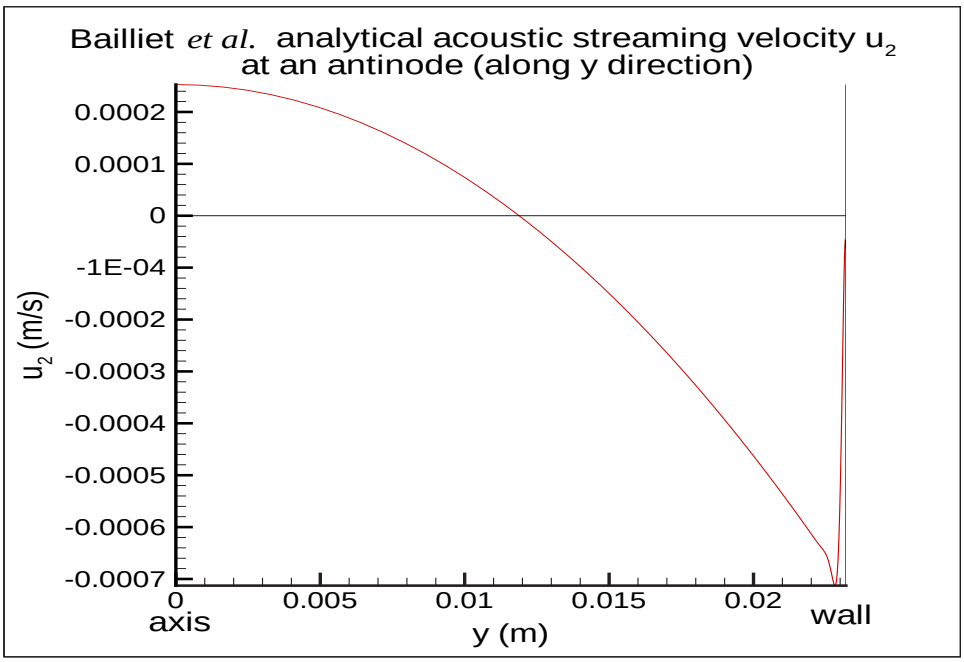

Figure 3.39. Bailliet *et al.* analytical acoustic streaming velocity u_2 at an antinode along y direction. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

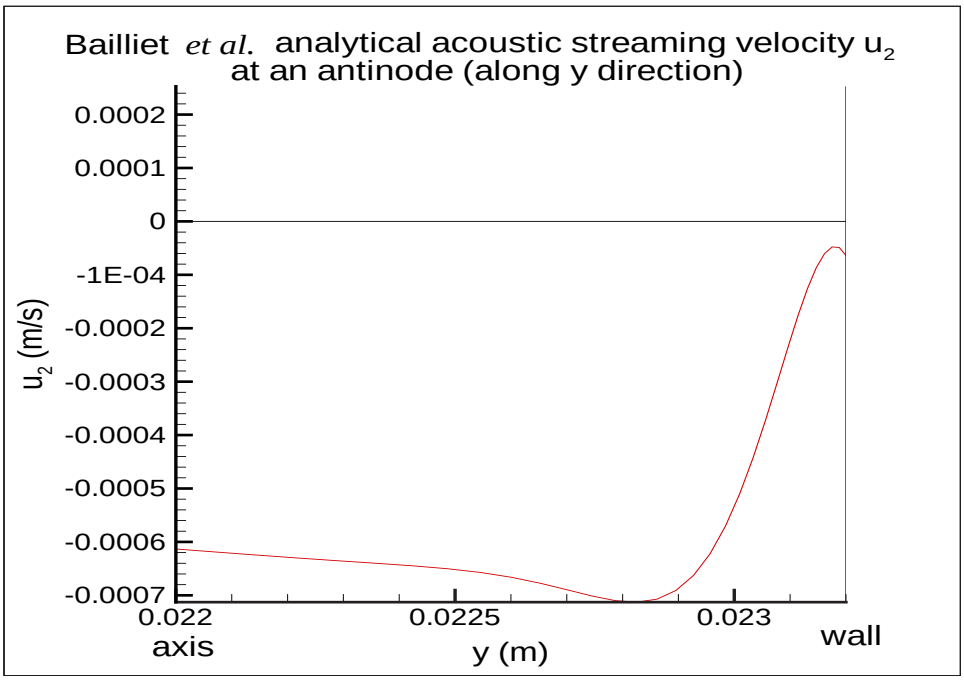

Figure 3.40. Bailliet *et al.* analytical acoustic streaming velocity u_2 at an antinode along y direction. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

3.4 Chapter 3 Discussion

By observing Figs. 3.6, 3.7, 3.19, 3.20, 3.36, 3.37, it can be concluded that although the forcing functions derived using the different groups' first order velocities look similar in their overall profiles, the details are different in the region near the wall. Since Rayleigh streaming originates from viscous stresses on the boundaries, slight discrepancies in the forcing functions near the boundaries lead to variations in the resulting streaming velocities. The assumptions made when deriving the first order velocities play a significant part in determining the behavior of the first order velocities and hence the forcing functions. The differences in the forcing functions will be discussed in more detail in Chapter 6 to emphasize their importance in causing the differences in the analytical and numerical streaming velocities.

So far in Chapter 3 the analyses performed by the three different groups have been discussed. Their assumptions, the first order acoustic velocities, the forcing functions, and the analytical streaming velocities have been presented. The forcing functions derived from the three groups' first order velocities will be used to drive

the VTE. In the next two chapters the computational grid and the numerical method utilized to solve the VTE will be explained. Then the resulting numerical streaming velocities as well as the analytical velocities will be compared in Chapter 6.

Chapter 4

Computational Grid

4.1 Nonuniform Physical Domain

The numerical calculation in this research is performed on only one half of a channel due to the fact that acoustic streaming is symmetrical on the upper and lower halves of the channel. The simulation can model either the upper or lower half. Here the computational domain is chosen to be the upper half of the channel as shown in Fig. 4.1.

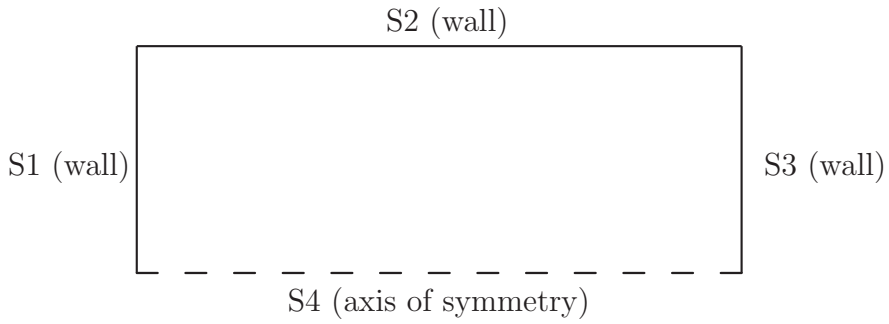

Figure 4.1. Computational domain

The top boundary of the computational domain ($S2$) represents the top wall of the channel while the bottom boundary ($S4$) represents the axis of symmetry (i.e. the middle of the channel). The left ($S1$) and right ($S3$) boundaries of the computational domain represent rigid ends.

As seen in Chapter 3, the forcing function's highest amplitude is concentrated at the area near the wall. Because of this, the computational grid spacing needs to be fine enough to capture this feature. Further, having an accurate forcing function is critical for the calculation. The need for adequate grid points in the vicinity of the wall results in the need to have a large number of grid points in the cross section of the channel if the points are uniformly spaced. This would lead to undesirable additional computation time and memory requirements. In actuality, we only need to have grid points concentrated near the boundary $S2$ instead of having a large number of points uniformly spaced in the computational domain. Therefore it is possible to use a grid that is clustered near the boundary instead of a uniformly spaced grid. That way, with a fixed amount of grid points, we can have more points in the vicinity of the boundary and less everywhere else.

A *physical domain* is a domain that represents the physical dimensions of the actual device being simulated. The grid points in the physical domain represent "probes" where sources are located or measurements are taken. Because we need more probes near the wall to act as the "source" representing the forcing function, the grid points are clustered near the wall and as a consequence the spacing between grid points in the cross sectional direction will vary.

Unequal grid spacings can pose some problems in the numerical computation. Not only can this add to the complexity of the program, but it can lead to instability in the computation. To overcome these problems, the calculation itself is going to be done in a *computational domain* that has uniform spacing between grid points. A computational domain is a domain where the actual computation is performed. It does not necessarily represent the physical dimensions of the simulated device but it is related to the physical domain through a transformation step. The transformation of the physical domain into the computational domain is performed by mapping the points in the physical domain to the computational domain. There are a few requirements that must be met in the transformation process [13]:

1. The mapping must be one to one which means grid lines of the same family cannot cross each other.

2. The grid point distribution is smooth (with minimum grid line skewness).

3. The grid lines must have orthogonality or near orthogonality.

4. The grid points are concentrated where there are high flow gradients.

Since we only need to cluster the grid points in the area near the wall we can use the simplest technique of grid generation which is the *algebraic method*. The following steps outline and give an example of how to generate a physical domain grid:

1. Determine the desired number of grid points in the x and y directions. As an example, let the number of points in the x direction $nx = 25$ and the number of grid points in the y direction $ny = 25$.

2. Generate a uniform grid points in the x direction. In Fortran this can be done by the following:

$$\begin{aligned} \text{do} \quad i &= 1,\ nx \\ x(i) &= \frac{i-1}{nx-1} \times l \\ \text{end do} & \end{aligned} \tag{4.1}$$

   where l is the length of the channel (or tube).

3. Generate a clustered grid in the y direction. The hyperbolic tangent function was chosen to cluster the grid in one direction. In Fortran this can be done in the following manner:

$$\begin{aligned} \text{do} \quad j &= 1,\ ny \\ y(j) &= \frac{\tanh\left(\pi\ cf\ \frac{j-1}{ny-1}\right)}{\tanh(\pi\ cf)} \times h \\ \text{end do} & \end{aligned} \tag{4.2}$$

   where cf is the clustering factor and h is the distance between the axis of symmetry and the top boundary $S2$. Notice that the argument of the hyperbolic tangent function is a uniform grid multiplied by the clustering factor and π. The reason for this is because $\left(\frac{j-1}{ny-1}\right)$ has a maximum value of 1 at $j = ny$ and the Tanh function has a maximum value of 1 for argument value of π and above (see Fig. 4.2). To control the amount of clustering, the

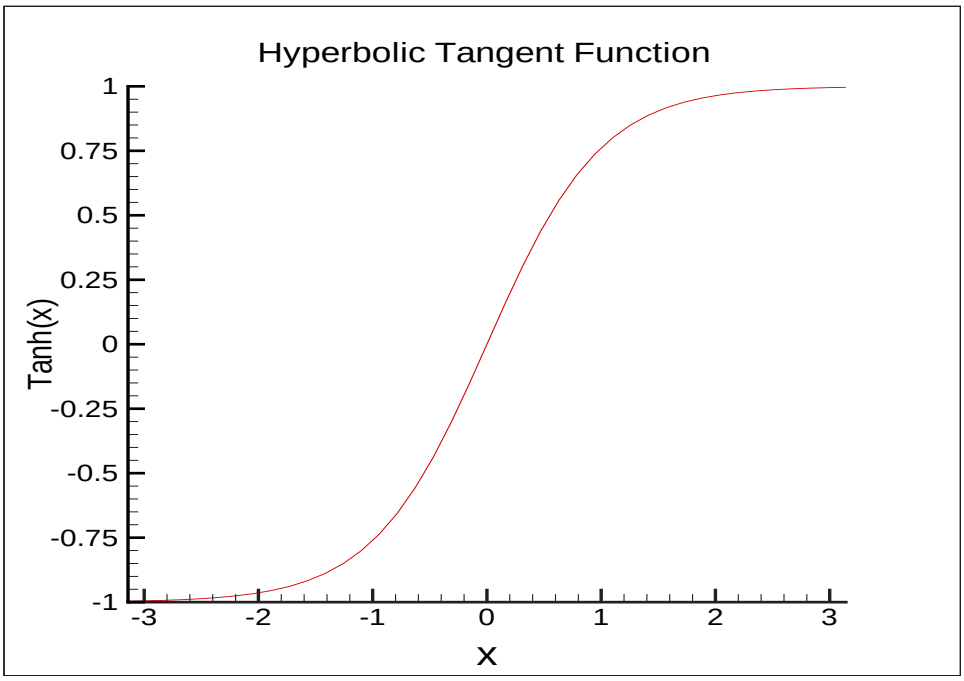

Figure 4.2. Hyperbolic tangent function

argument can be multiplied by a factor (in this case it is called the *clustering factor* cf) whose value is less than 1. The clustering factor is chosen to be less than or equal to 1 because $|\tanh(x)| \leq 1$ for $|x| \geq \pi$. The higher the value of the argument (but still less than π), the more clustered the grid is. For example, let x be a uniform grid, and if $0 \leq x \leq \pi$ then the grid points are mapped as in Fig. 4.3. When $0 \leq x \leq \pi/2$ the grid points are mapped as in Fig. 4.4.

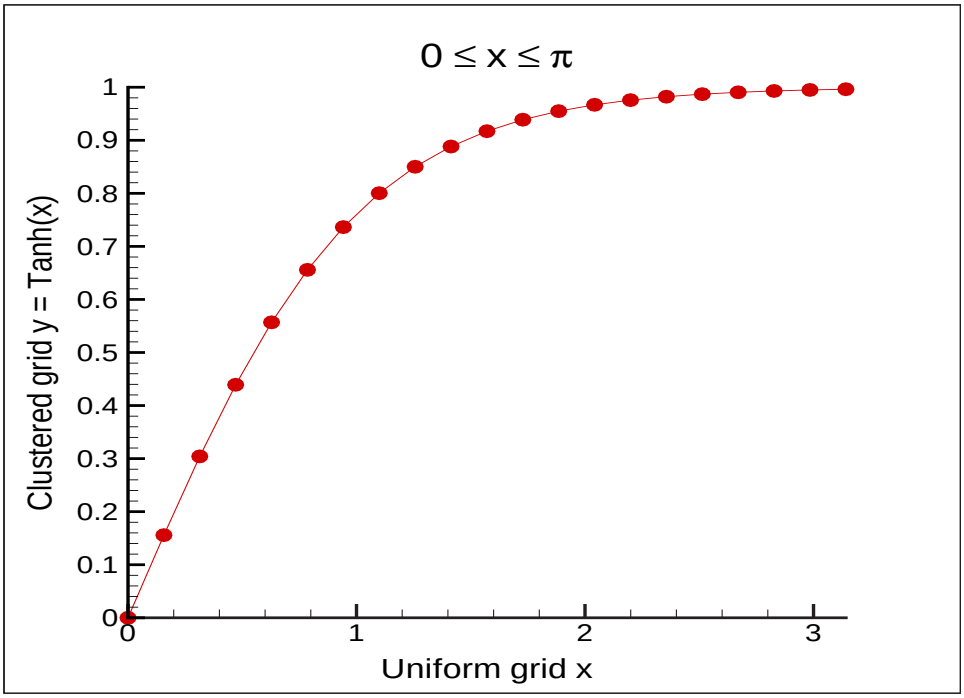

Figure 4.3. Clustered grid y mapped from uniform grid x

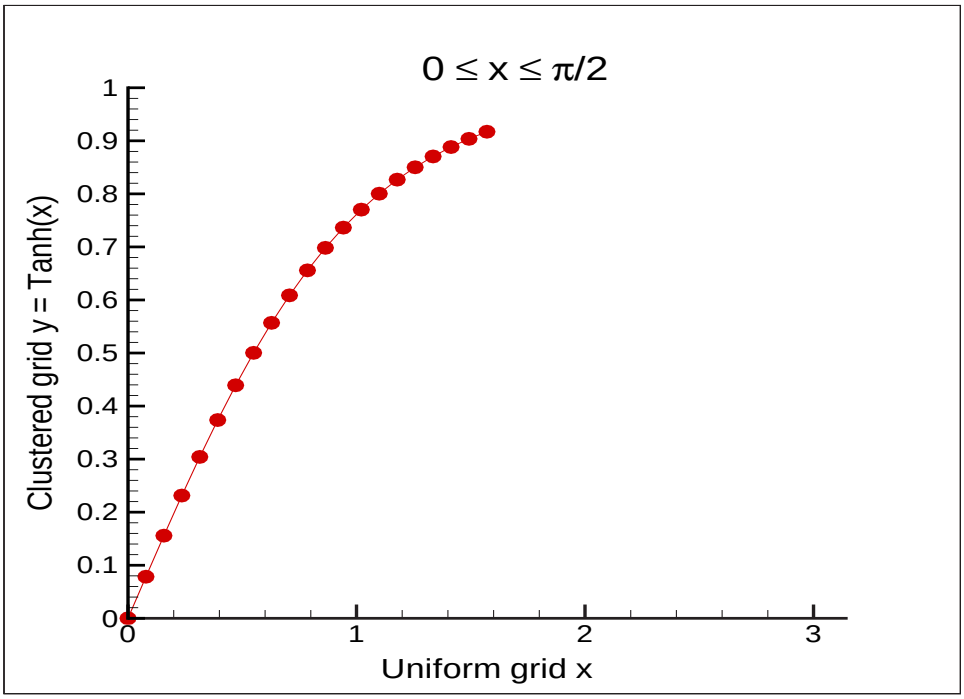

Figure 4.4. Clustered grid y mapped from uniform grid x

Since the same number of grid points are distributed in a shorter distance, they will be clustered more as the maximum value of the argument of the Tanh function is reduced. Notice that as the grid is more clustered, the maximum value of the Tanh function is no longer 1. To take this into account when y is calculated, it is divided by the maximum value of $\tanh(x)$ to make $\left|\tanh\left(\pi\, cf\, \frac{j-1}{ny-1}\right) / \tanh(\pi\, cf)\right| = 1$ (see Eq. 4.2). Then the result can be scaled by h to get the correct crosswise dimension. Figure 4.5 shows a typical result for a 25×25 clustered grid.

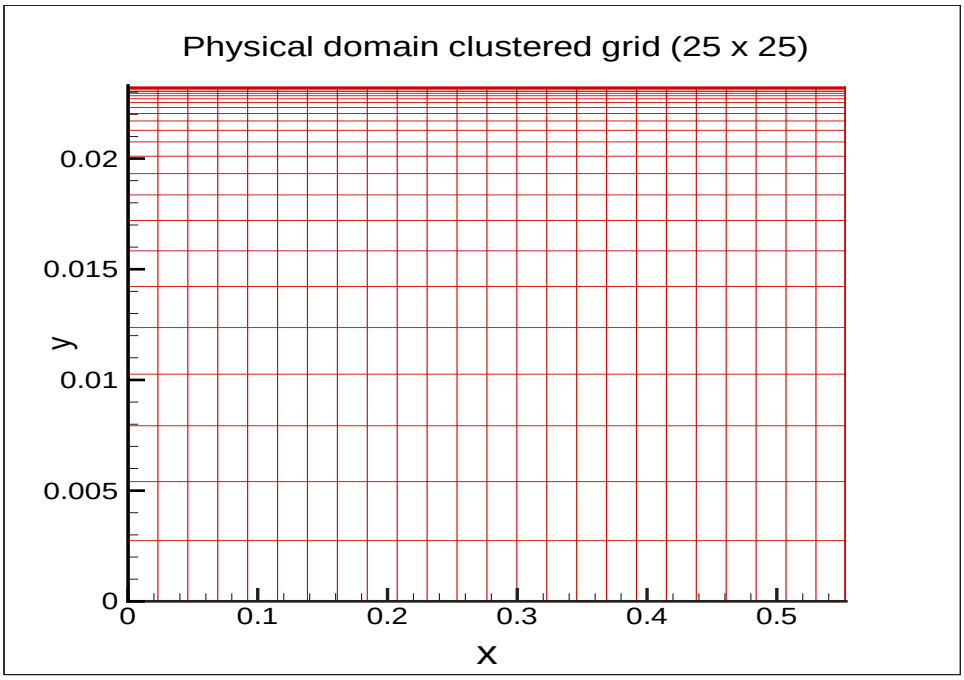

Figure 4.5. Physical domain clustered grid with 25×25 points. $l = 0.553$ m, $h = 0.0232$ m

4.2 Uniform Computational Domain

The next step to be done is to transform the clustered physical grid into a uniform computational grid. In order to do this one will need the metrics of transformation.

4.2.1 Metrics of Transformation

When finite differencing is applied to a partial differential equation, one will need the distance between neighboring grid points to calculate the spatial derivatives. If there is some clustering in the computational grid, the distance between adjacent grid points will not be the same for all points (in the same direction) in the grid. This can lead to instability in the calculation because the largest distance between two points may not satisfy the stability condition. In order to avoid such instability, the computational grid needs to be uniform in space. This can be obtained by transforming the clustered physical grid into a uniform computational grid through the use of the metrics in the partial differential equations to be solved.

Let x and y be the coordinates in the physical domain and ξ and η be the coordinates in the computational domain. Then the relationship between the physical and computational domains can be defined as:

$$\xi = \xi(x, y) \tag{4.3}$$

$$\eta = \eta(x, y) \tag{4.4}$$

Partial derivatives with respect to x and y are then:

$$\frac{\partial}{\partial x} = \frac{\partial \xi}{\partial x}\frac{\partial}{\partial \xi} + \frac{\partial \eta}{\partial x}\frac{\partial}{\partial \eta} \tag{4.5}$$

$$\frac{\partial}{\partial y} = \frac{\partial \xi}{\partial y}\frac{\partial}{\partial \xi} + \frac{\partial \eta}{\partial y}\frac{\partial}{\partial \eta} \tag{4.6}$$

Using the notation $\frac{\partial \xi}{\partial x} = \xi_x$, the above equations can be written as:

$$\frac{\partial}{\partial x} = \xi_x\frac{\partial}{\partial \xi} + \eta_x\frac{\partial}{\partial \eta} \tag{4.7}$$

$$\frac{\partial}{\partial y} = \xi_y\frac{\partial}{\partial \xi} + \eta_y\frac{\partial}{\partial \eta} \tag{4.8}$$

ξ_x, ξ_y, η_x, and η_y are called the *metrics* of transformation. Because of approximation such as:

$$\xi_x = \frac{\partial \xi}{\partial x} \cong \frac{\Delta \xi}{\Delta x} \tag{4.9}$$

it can be said that the metrics of transformation give the ratio of arc lengths in

the computational domain to that of the physical domain.

For this research, the uniform computational domain was chosen to be: $0 \leq \xi \leq 1$ and $0 \leq \eta \leq 1$. Since the grid is already uniformly spaced in the x direction it only has to be scaled so that the maximum value of ξ is 1:

$$\xi = \frac{x}{l} \tag{4.10}$$

The grid has to be "unstretched" to make it uniformly spaced in the y direction and scaled at the same time to make the maximum value of η to be 1:

$$\eta = \frac{1}{\pi\, cf}\ \tanh^{-1}\left(\frac{y \times \tanh(\pi\, cf)}{h}\right) \tag{4.11}$$

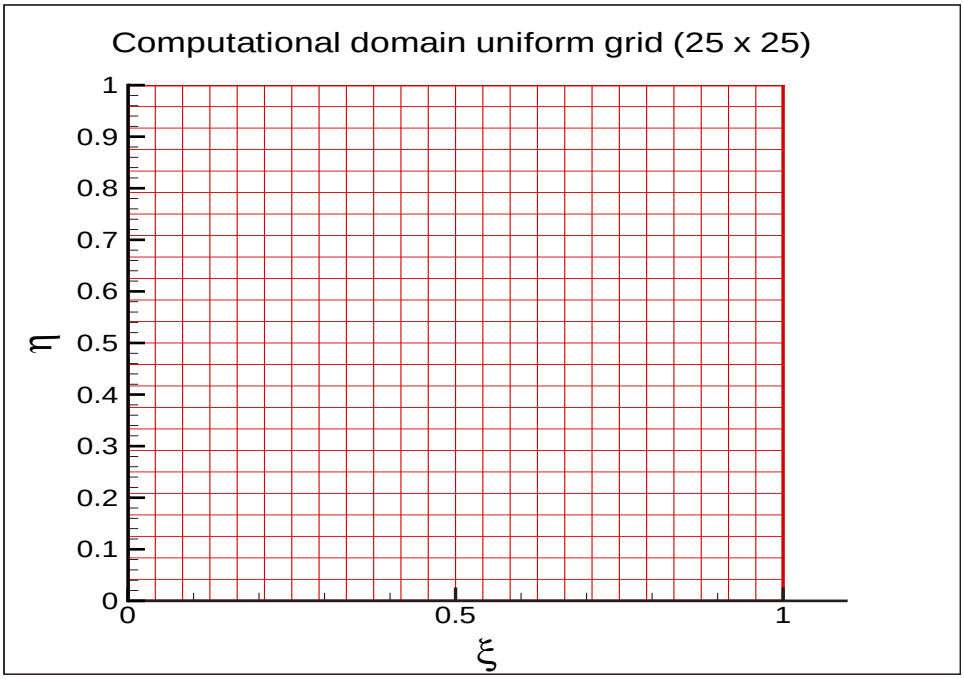

Figure 4.6. Computational domain uniform grid with 25×25 points

The metrics are calculated by:

$$\xi_x = \frac{\partial \xi}{\partial x} = \frac{\partial (x/l)}{\partial x} = \frac{1}{l} \tag{4.12}$$

and

$$\begin{aligned}\eta_y = \frac{\partial \eta}{\partial y} =& \frac{\partial}{\partial y}\left[\frac{1}{\pi\, cf}\ \tanh^{-1}\left(\frac{y \times \tanh(\pi\, cf)}{h}\right)\right] \\ =& \frac{1}{\pi\, cf}\ \frac{\tanh(\pi\, cf)/h}{1-(\tanh(\pi\, cf)/h)^2\ y^2}\end{aligned} \tag{4.13}$$

Also

$$\begin{aligned}\eta_{yy} =& \frac{\partial^2 \eta}{\partial y^2} \\ =& \frac{1}{\pi\, cf}\ \frac{2\ (\tanh(\pi\, cf)/h)^3\ y}{[1-(\tanh(\pi\, cf)/h)^2\ y^2]^2}\end{aligned} \tag{4.14}$$

These metrics are going to be utilized in the calculation to transform the PDE.

The numerical scheme used in this research is a second order finite difference method. Since the computational domain involves the clustering of the grid near a boundary, the governing equations have to be transformed to take account of the grid clustering. The metrics in Eqs. 4.12, 4.13, and 4.14 will take into account the nonuniformity of the grid in the PDE.

Recall Eqs. 4.7 and 4.8, and the second derivatives with respect to x and y are:

$$\frac{\partial^2}{\partial x^2} = \xi_{xx}\frac{\partial}{\partial \xi} + \xi_x^2\frac{\partial^2}{\partial \xi^2} + \eta_{xx}\frac{\partial}{\partial \eta} + \eta_x^2\frac{\partial^2}{\partial \eta^2} + 2\ \xi_x\eta_x\frac{\partial^2}{\partial \xi \partial \eta} \tag{4.15}$$

$$\frac{\partial^2}{\partial y^2} = \xi_{yy}\frac{\partial}{\partial \xi} + \xi_y^2\frac{\partial^2}{\partial \xi^2} + \eta_{yy}\frac{\partial}{\partial \eta} + \eta_y^2\frac{\partial^2}{\partial \eta^2} + 2\ \xi_y\eta_y\frac{\partial^2}{\partial \xi \partial \eta} \tag{4.16}$$

Using the metrics of transformation, the transformed time-averaged VTE is:

$$-\left(\xi_x\frac{\partial F_y}{\partial \xi} - \eta_y\frac{\partial F_x}{\partial \eta}\right) = \nu\left[\xi_x^2\frac{\partial^2 \omega}{\partial \xi^2} + \eta_{yy}\frac{\partial \omega}{\partial \eta} + \eta_y^2\frac{\partial^2 \omega}{\partial \eta^2}\right] \tag{4.17}$$

Recall from Chapter 2 that the stream function ψ is related to the vorticity ω by the Poisson's equation:

$$-\omega = \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} \tag{4.18}$$

The transformed Poisson's equation can be expressed as:

$$-\omega = \xi_x^2\frac{\partial^2 \psi}{\partial \xi^2} + \eta_{yy}\frac{\partial \psi}{\partial \eta} + \eta_y^2\frac{\partial^2 \psi}{\partial \eta^2} \tag{4.19}$$

Equations 4.17 and 4.19 clearly show that the clustering is taken into account in the PDE through the metrics of transformation ξ_x, η_y, and η_{yy}.

4.2.2 Boundary Conditions

Vorticity

Let P be a point at the boundary $S2$ and Q be a point adjacent to P. The distance between them is Δy. Suppose ψ is sufficiently smooth [26], then:

$$\psi_Q = \psi_P + \frac{\partial \psi_P}{\partial y}\Delta y + \frac{1}{2}\left.\frac{\partial^2 \psi}{\partial y^2}\right|_P \Delta y^2 + \frac{1}{6}\left.\frac{\partial^3 \psi}{\partial y^3}\right|_P \Delta y^3 + O\left(\Delta^4\right) \tag{4.20}$$

If the wall is not penetrable by the fluid, $u_{2y} = 0$ at P. Recall:

$$\omega = \frac{\partial u_y}{\partial x} - \frac{\partial u_x}{\partial y}$$

and

$$u_{2x} = \frac{\partial \psi}{\partial y}, \qquad u_{2y} = -\frac{\partial \psi}{\partial x}$$

Since $u_{2y} = 0$, at point P:

$$\omega = -\frac{\partial u_{2x}}{\partial y} \tag{4.21}$$

or

$$\omega_P = -\frac{\partial}{\partial y}\left(\frac{\partial \psi}{\partial y}\right) = -\left.\frac{\partial^2 \psi}{\partial y^2}\right|_P \tag{4.22}$$

Taking the derivative of Eq. 4.22,

$$\frac{\partial \omega_P}{\partial y} = -\left.\frac{\partial^3 \psi}{\partial y^3}\right|_P \tag{4.23}$$

Substituting Eqs. 4.22 and 4.23 to Eq. 4.20:

$$\psi_Q = \psi_P + \frac{\partial \psi_P}{\partial y}\Delta y - \frac{\omega_P}{2}\Delta y^2 - \frac{1}{6}\frac{\partial \omega_P}{\partial y}\Delta y^3 \tag{4.24}$$

The first order derivative can be approximated as:

$$\frac{\partial \omega_P}{\partial y} = \frac{\omega_Q - \omega_P}{\Delta y} \tag{4.25}$$

Therefore Eq. 4.24 can be expressed as:

$$\begin{aligned}\psi_Q &= \psi_P + \frac{\partial \psi_P}{\partial y}\Delta y - \frac{\omega_P}{2}\Delta y^2 - \frac{1}{6}\frac{\omega_Q - \omega_P}{\Delta y}\Delta y^3 \\ &= \psi_P + \frac{\partial \psi_P}{\partial y}\Delta y - \frac{\omega_P}{2}\Delta y^2 - \frac{1}{6}\left(\omega_Q - \omega_P\right)\Delta y^2\end{aligned} \tag{4.26}$$

Placing ω_P on the left hand side and all other terms on the right hand side,

$$\omega_P \frac{\Delta y^2}{3} = -\psi_Q + \psi_P + \frac{\partial \psi_P}{\partial y}\Delta y - \frac{1}{6}\omega_Q \Delta y^2 \tag{4.27}$$

or

$$\omega_P = -\frac{3}{\Delta y^2}\left(\psi_Q - \psi_P - \frac{\partial \psi_P}{\partial y}\Delta y\right) - \frac{\omega_Q}{2} \tag{4.28}$$

Applying Eq. 4.28 to the uniform computational domain, the boundary condition for the vorticity is:

$$\omega_P = -\frac{3}{\Delta \eta^2}\left(\psi_Q - \psi_P - \frac{\partial \psi_P}{\partial \eta}\Delta \eta\right) - \frac{\omega_Q}{2} \tag{4.29}$$

Stream Function

The boundary condition is assumed to be a no-slip condition so that velocities are zero at the wall. Due to the fact that $u = \partial\psi/\partial y$ and $w = -\partial\psi/\partial x$, the no-slip condition enables the stream function ψ to be set to any constant value, which is zero in this case. The value of zero is chosen for convenience, although any constant value will work. Therefore the stream function boundary condition is $\psi = 0$ at the walls. On the axis of symmetry, ψ is set to zero because there the transverse streaming velocity u_{2y} is zero.

Chapter 5

Numerical Calculation Through A Direct Method

There are different ways to numerically solve the VTE and Poisson's equation. One way is to solve them through an *iterative method* and another way is through a *direct method*. In this research we use the direct method because here the VTE is a time-averaged equation and the method itself is unconditionally stable. The direct method involves solving a set of linear algebraic equations. To do this, we first need to form a matrix of the coefficients of the vorticity (in the case of VTE) or the stream function (in the case of Poisson's equation). We then have to decompose each matrix into its lower and upper matrices, taking advantage of the fact that the solution to a triangular set of equations is quite trivial. The solution is then obtained through *backsubstitution*.

Although Lower-Upper (LU) decomposition is a well known technique in the numerical calculation area, it is worthwhile to discuss how it works in the following sections because it is the backbone of the calculation in this research.

5.1 Lower Upper Decomposition

The following explanation is taken from *Numerical Recipes* Chapter 2 pages 34 through 39 [29].

A matrix $\mathbf{A}$ can be written as a product of two matrices such that

$$\mathbf{L} \cdot \mathbf{U} = \mathbf{A} \tag{5.1}$$

where $\mathbf{L}$ is the lower triangular matrix (with elements only on the diagonal and below it), and $\mathbf{U}$ is the upper triangular matrix (with elements only on the diagonal and above it). A set of algebraic equations can be written in a vector form as:

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{5.2}$$

Here $\mathbf{x}$ is the unknown vector and $\mathbf{b}$ is the known vector. In analogy with Hooke's Law from mechanics, $\mathbf{A}$ is called the *stiffness matrix* and $\mathbf{b}$ is the *load vector*. $\mathbf{A}$ can be decomposed into lower and upper matrices such that:

$$\mathbf{A} \cdot \mathbf{x} = (\mathbf{L} \cdot \mathbf{U}) \cdot \mathbf{x} = \mathbf{L} \cdot (\mathbf{U} \cdot \mathbf{x}) = \mathbf{b} \tag{5.3}$$

Let $\mathbf{U} \cdot \mathbf{x} = \mathbf{y}$ then

$$\mathbf{L} \cdot \mathbf{y} = \mathbf{b} \tag{5.4}$$

Vector $\mathbf{y}$ can be obtained through *forward substitution* and $\mathbf{x}$ is obtained by solving

$$\mathbf{U} \cdot \mathbf{x} = \mathbf{y} \tag{5.5}$$

through *backsubstitution*. Another advantage of the LU decomposition method is that the method can be done with as many right hand sides $\mathbf{b}$ as necessary.

To illustrate LU decomposition, let us look at the following example (from *Numerical Recipes* page 45 [29]). A 4×4 matrix $\mathbf{A}$ can be written as a product of its lower and upper triangular matrices such as

$$\mathbf{A} = \mathbf{L} \cdot \mathbf{U}$$

or

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} = \begin{bmatrix} \alpha_{11} & 0 & 0 & 0 \\ \alpha_{21} & \alpha_{22} & 0 & 0 \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{bmatrix} \cdot \begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\ 0 & \beta_{22} & \beta_{23} & \beta_{24} \\ 0 & 0 & \beta_{33} & \beta_{34} \\ 0 & 0 & 0 & \beta_{44} \end{bmatrix} \tag{5.6}$$

Then

$$\mathbf{L} \cdot \mathbf{y} = \mathbf{b}$$

is

$$\begin{bmatrix} \alpha_{11} & 0 & 0 & 0 \\ \alpha_{21} & \alpha_{22} & 0 & 0 \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \tag{5.7}$$

The equations to be solved are:

$$\alpha_{11} y_1 = b_1 \rightarrow y_1 = \frac{b_1}{\alpha_{11}} \tag{5.8}$$

$$\alpha_{21} y_1 + \alpha_{22} y_1 = b_2 \rightarrow y_2 = \frac{1}{\alpha_{22}} \left(b_2 - \alpha_{21} y_1 \right) \tag{5.9}$$

$$\alpha_{31} y_1 + \alpha_{32} y_2 + \alpha_{33} y_3 = b_3 \rightarrow y_3 = \frac{1}{\alpha_{33}} \left(b_3 - \alpha_{31} y_1 - \alpha_{32} y_2 \right) \tag{5.10}$$

$$\alpha_{41} y_1 + \alpha_{42} y_2 + \alpha_{43} y_3 + \alpha_{44} y_4 = b_4 \rightarrow y_4 = \frac{1}{\alpha_{44}} \left(b_4 - \alpha_{41} y_1 - \alpha_{42} y_2 - \alpha_{43} y_3 \right) \tag{5.11}$$

or

$$y_i = \frac{1}{\alpha_{ii}} \left[b_i - \sum_{j=1}^{i-1} \alpha_{ij} y_j \right] \qquad i = 2, 3, \ldots, N \tag{5.12}$$

This is the forward substitution procedure. The next step is to solve:

$$\mathbf{U} \cdot \mathbf{x} = \mathbf{y}$$

or

$$\begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\ 0 & \beta_{22} & \beta_{23} & \beta_{24} \\ 0 & 0 & \beta_{33} & \beta_{34} \\ 0 & 0 & 0 & \beta_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} \tag{5.13}$$

The equations to be solved are:

$$\beta_{11}x_1+\beta_{12}x_2+\beta_{13}x_3+\beta_{14}x_4 = y_1 \rightarrow x_1 = \frac{1}{\beta_{11}}\left(y_1 - \beta_{12}x_2 - \beta_{13}x_3 - \beta_{14}x_4\right) \tag{5.14}$$

$$\beta_{22}x_2 + \beta_{23}x_3 + \beta_{24}x_4 = y_2 \rightarrow x_2 = \frac{1}{\beta_{22}}\left(y_2 - \beta_{23}x_3 - \beta_{24}x_4\right) \tag{5.15}$$

$$\beta_{33}x_3 + \beta_{34}x_4 = y_3 \rightarrow x_3 = \frac{1}{\beta_{33}}\left(y_3 - \beta_{34}x_4\right) \tag{5.16}$$

$$\beta_{44}x_4 = y_4 \rightarrow x_4 = \frac{y_4}{\beta_{44}} \tag{5.17}$$

or

$$x_i = \frac{1}{\beta_{ii}}\left[y_i - \sum_{j=i+1}^{N} \beta_{ij}x_j\right] \qquad i = N-1, N-2, \ldots, 1 \tag{5.18}$$

This is the backward substitution procedure. From Eqs. 5.12 and 5.18, it can be seen that divisions are done with the diagonal elements as the divisor (i.e., α_{ii} and β_{ii}). Those elements are called the *pivots*. When the pivots are small with respect to the other elements in the matrix the results can be inaccurate. To overcome this problem, rows in the matrix are permutated so that the pivots can be replaced by other larger elements. Exchanging rows in a matrix does not change the linear algebraic equation itself as long as the matching rows in **b** are also exchanged.

To decompose matrix **A** into its lower and upper matrices let us first look at Eq. 5.6. If Eq. 5.6 is written out as its ith and jth component, the first row equations are:

$$\alpha_{11}\beta_{11} = a_{11} \tag{5.19}$$

$$\alpha_{11}\beta_{12} = a_{12} \tag{5.20}$$

$$\alpha_{11}\beta_{13} = a_{13} \tag{5.21}$$

$$\alpha_{11}\beta_{14} = a_{14} \tag{5.22}$$

The second row equations are:

$$\alpha_{21}\beta_{11} = a_{21} \tag{5.23}$$
$$\alpha_{21}\beta_{12} + \alpha_{22}\beta_{22} = a_{22} \tag{5.24}$$
$$\alpha_{21}\beta_{13} + \alpha_{22}\beta_{23} = a_{23} \tag{5.25}$$
$$\alpha_{21}\beta_{14} + \alpha_{22}\beta_{24} = a_{24} \tag{5.26}$$

The third row equations are:

$$\alpha_{31}\beta_{11} = a_{31} \tag{5.27}$$
$$\alpha_{31}\beta_{12} + \alpha_{32}\beta_{22} = a_{32} \tag{5.28}$$
$$\alpha_{31}\beta_{13} + \alpha_{32}\beta_{23} + \alpha_{33}\beta_{33} = a_{33} \tag{5.29}$$
$$\alpha_{31}\beta_{14} + \alpha_{32}\beta_{24} + \alpha_{33}\beta_{34} = a_{34} \tag{5.30}$$

The fourth row equations are:

$$\alpha_{41}\beta_{11} = a_{41} \tag{5.31}$$
$$\alpha_{41}\beta_{12} + \alpha_{42}\beta_{22} = a_{42} \tag{5.32}$$
$$\alpha_{41}\beta_{13} + \alpha_{42}\beta_{23} + \alpha_{43}\beta_{33} = a_{43} \tag{5.33}$$
$$\alpha_{41}\beta_{14} + \alpha_{42}\beta_{24} + \alpha_{43}\beta_{34} + \alpha_{44}\beta_{44} = a_{44} \tag{5.34}$$

In terms of i and j, they can be written as

$$\alpha_{i1}\beta_{1j} + \cdots = a_{i,j}$$

The number of terms in the sum depends on whether i is smaller than j or vice versa. There are three cases:

$$i < j: \quad \alpha_{i1}\beta_{1j} + \alpha_{i2}\beta_{2j} + \cdots + \alpha_{ii}\beta_{ij} = a_{ij} \tag{5.35}$$
$$i = j: \quad \alpha_{i1}\beta_{1j} + \alpha_{i2}\beta_{2j} + \cdots + \alpha_{ii}\beta_{jj} = a_{ij} \tag{5.36}$$
$$i > j: \quad \alpha_{i1}\beta_{1j} + \alpha_{i2}\beta_{2j} + \cdots + \alpha_{ij}\beta_{jj} = a_{ij} \tag{5.37}$$

One procedure to decompose a matrix into its lower and upper matrices is *Crout's algorithm*. The steps that are taken for the LU decomposition are:

1. Set the diagonal of the lower triangular matrix $\alpha_{ii} = 1$, where $i = 1, \ldots, N$
2. For each $j = 1, 2, 3, \ldots, N$:
   (a) For $i = 1, 2, \ldots, j$ use 5.35, 5.36 and 5.37 to calculate β_{ij}

$$\beta_{ij} = a_{ij} - \sum_{k=1}^{i-1} \alpha_{ik}\beta_{kj} \tag{5.38}$$

   (b) For $i = j+1, j+2, \ldots, N$ use 5.37 to calculate α_{ij}

$$\alpha_{ij} = \frac{1}{\beta_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} \alpha_{ik}\beta_{kj} \right) \tag{5.39}$$

Since $\alpha_{ii} = 1$ (i.e. $\alpha_{11} = 1$) Eqs. 5.19 through 5.22 are now:

$$\beta_{11} = a_{11} \tag{5.40}$$
$$\beta_{12} = a_{12} \tag{5.41}$$
$$\beta_{13} = a_{13} \tag{5.42}$$
$$\beta_{14} = a_{14} \tag{5.43}$$

Then we can calculate Eqs. 5.23 through 5.26 to get:

$$\alpha_{21} = \frac{a_{21}}{\beta_{11}} \tag{5.44}$$
$$\beta_{22} = \frac{1}{\alpha_{22}}(a_{22} - \alpha_{21}\beta_{12}) = a_{22} - \alpha_{21}\beta_{12} \tag{5.45}$$
$$\beta_{23} = \frac{1}{\alpha_{22}}(a_{23} - \alpha_{21}\beta_{13}) = a_{23} - \alpha_{21}\beta_{13} \tag{5.46}$$
$$\beta_{24} = \frac{1}{\alpha_{22}}(a_{24} - \alpha_{21}\beta_{14}) = a_{24} - \alpha_{21}\beta_{14} \tag{5.47}$$

Equations 5.24 through 5.34 can be solved in the same manner. So once a matrix is decomposed into its upper and lower matrices, the set of algebraic equations can be solved using the forward and backward substitution technique as explained

before.

5.1.1 Storing Band Diagonal Matrix Elements Compactly

When the set of equations that resulted from taking the finite difference of the vorticity transport equation and Poisson's equation are put in a matrix form as in Eq. 5.1, it yields banded matrices. A banded matrix is a matrix with nonzero elements only along a few diagonal lines above and below the main diagonal. The advantage of banded matrices is the fact that only the nonzero elements need to be stored. This can reduce the memory allocation requirements significantly. A banded matrix can be stored compactly by tilting the matrix 45° to make the nonzero elements lie in a long narrow matrix. The new compact matrix will have $(m_1 + 1 + m_2)$ columns and N rows. Here m_1 is the number of nonzero elements in the diagonals below the main diagonal (*subdiagonal*) and m_2 is the number of nonzero elements in the diagonals above the main diagonal (*superdiagonal*). The example below is from *Numerical Recipes* page 44 [29]. It illustrates the formation of a compact matrix from a banded diagonal matrix. A 7×7 banded diagonal matrix:

$$\begin{pmatrix} 3 & 1 & 0 & 0 & 0 & 0 & 0 \\ 4 & 1 & 5 & 0 & 0 & 0 & 0 \\ 9 & 2 & 6 & 5 & 0 & 0 & 0 \\ 0 & 3 & 5 & 8 & 9 & 0 & 0 \\ 0 & 0 & 7 & 9 & 3 & 2 & 0 \\ 0 & 0 & 0 & 3 & 8 & 4 & 6 \\ 0 & 0 & 0 & 0 & 2 & 4 & 4 \end{pmatrix}$$

has $N = 7$ rows, $m_1 = 2$ subdiagonal and $m_2 = 1$ superdiagonal. It can be stored

compactly as a 7×4 matrix:

$$\begin{pmatrix} \# & \# & 3 & 1 \\ \# & 4 & 1 & 5 \\ 9 & 2 & 6 & 5 \\ 3 & 5 & 8 & 9 \\ 7 & 9 & 3 & 2 \\ 3 & 8 & 4 & 6 \\ 2 & 4 & 4 & \# \end{pmatrix}$$

The # denotes the wasted space that will not be referenced by any manipulation and can have arbitrary values. LU decomposition of the compact matrix can then be conducted. This research uses Fortran subroutines provided by *Numerical Recipes* to do a banded matrix decomposition along with the forward and backward substitution process.

The techniques we discussed so far will be applied to solve the VTE and Poisson's equation. The transformed time-averaged VTE for a clustered grid from Eq. 4.17 was:

$$-\left(\xi_x \frac{\partial F_y}{\partial \xi} - \eta_y \frac{\partial F_x}{\partial \eta}\right) = \nu \left(\xi_x^2 \frac{\partial^2 \omega}{\partial \xi^2} + \eta_{yy} \frac{\partial \omega}{\partial \eta} + \eta_y^2 \frac{\partial^2 \omega}{\partial \eta^2}\right) \tag{5.48}$$

The second order accurate finite difference approximation for the above equation is:

$$\begin{aligned} -\left(\xi_{x_{i,j}} \frac{F_{(y)i+1,j} - F_{(y)i-1,j}}{2\Delta\xi} - \eta_{y_{i,j}} \frac{F_{(x)i,j+1} - F_{(x)i,j-1}}{2\Delta\eta}\right) = \\ \nu \left(\xi_{x_{i,j}}^2 \frac{\omega_{i+1,j} - 2\omega_{i,j} + \omega_{i-1,j}}{(\Delta\xi)^2} + \eta_{yy_{i,j}} \frac{\omega_{i,j+1} - \omega_{i,j-1}}{2\Delta\eta} + \right. \\ \left. \eta_{y_{i,j}}^2 \frac{\omega_{i,j+1} - 2\omega_{i,j} + \omega_{i,j-1}}{(\Delta\eta)^2}\right) \end{aligned} \tag{5.49}$$

$\Delta\xi$ is the distance between adjacent grid points in the x direction and $\Delta\eta$ is the distance between adjacent grid points in the y direction. Rewriting Eq. 5.49 by placing the *unknown* quantities (in this case ω) on the left hand side and the *known*

quantities (in this case the forcing function) on the right hand side:

$$\begin{aligned} -\left(\nu\frac{\xi_{x_{i,j}}^2}{(\Delta\xi)^2}\right)\omega_{i+1,j} - \left(\nu\frac{\xi_{x_{i,j}}^2}{(\Delta\xi)^2}\right)\omega_{i-1,j} - \nu\left(\frac{\eta_{yy_{i,j}}}{2\Delta\eta} + \frac{\eta_{y_{i,j}}^2}{(\Delta\eta)^2}\right)\omega_{i,j+1} \\ -\nu\left(-\frac{\eta_{yy_{i,j}}}{2\Delta\eta} + \frac{\eta_{y_{i,j}}^2}{(\Delta\eta)^2}\right)\omega_{i,j-1} + 2\,\nu\left(\frac{\xi_{x_{i,j}}^2}{(\Delta\xi)^2} + \frac{\eta_{y_{i,j}}^2}{(\Delta\eta)^2}\right)\omega_{i,j} \\ = \frac{\xi_{x_{i,j}}}{2\Delta\xi}F_{(y)i+1,j} - \frac{\xi_{x_{i,j}}}{2\Delta\xi}F_{(y)i-1,j} - \frac{\eta_{y_{i,j}}}{2\Delta\eta}F_{(y)i,j+1} + \frac{\eta_{y_{i,j}}}{2\Delta\eta}F_{(y)i,j-1} \end{aligned} \tag{5.50}$$

The boundary conditions from Chapter 4 are:

1. Boundary $S1$.

$$\omega_{i,j} = -\frac{3}{\Delta\xi^2}\left(\psi_{i+1,j} - \psi_{i,j} + \frac{\partial\psi_{i,j}}{\partial\xi}\Delta\xi\right) - \frac{\omega_{i,j}}{2} \tag{5.51}$$

$$\text{where} \quad \frac{\partial\psi_{i,j}}{\partial\xi} = \frac{\xi_x}{2\Delta\xi}\left(-\psi_{i+2,j} + 4\psi_{i+1,j} - 3\psi_{i,j}\right), \qquad \text{and} \quad i = 1$$

2. Boundary $S3$.

$$\omega_{nx,j} = -\frac{3}{\Delta\xi^2}\left(\psi_{nx-1,j} - \psi_{nx,j} - \frac{\partial\psi_{nx,j}}{\partial\xi}\Delta\xi\right) - \frac{\omega_{nx-1,j}}{2} \tag{5.52}$$

$$\text{where} \quad \frac{\partial\psi_{nx,j}}{\partial\xi} = \frac{\xi_x}{2\Delta\xi}\left(3\psi_{nx,j} - 4\psi_{nx-1,j} + \psi_{nx-2,j}\right)$$

3. Boundary $S2$.

$$\omega_{i,ny} = -\frac{3}{\Delta\eta^2}\left(\psi_{i,ny-1} - \psi_{i,ny} - \frac{\partial\psi_{i,ny}}{\partial\eta}\Delta\eta\right) - \frac{\omega_{i,ny-1}}{2} \tag{5.53}$$

$$\text{where} \quad \frac{\partial\psi_{ny,j}}{\partial\eta} = \frac{\eta_y}{2\Delta\eta}\left(3\psi_{i,ny} - 4\psi_{i,ny-1} + \psi_{i,ny-2}\right)$$

4. Axis of symmetry.

$$\omega_{i,1} = 0 \tag{5.54}$$

There are a total of $(nx \times ny)$ equations in 5.50 and $(nx-2) \times (ny-2)$ unknowns $\omega_{i,j}$ where $i = 2, \ldots, nx-1$ and $j = 2, \ldots, ny-1$. nx and ny are the total number of grid points in the x and y directions, respectively. Equation 5.50 can be put in a matrix form $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$. The stiffness matrix $\mathbf{A}$ is sparse and banded with equal left and right bandwidths of $(nx+1)$. It holds the entries for the coefficients of the unknown values (ω) and has a dimension of $(nx \times ny)$ rows by $(m_1 + m_2 + 1)$ columns. The coefficients are stored as follows:

1. The diagonal elements are stored in $A(1:n, m_1+1)$, where $n = nx \times ny$.
2. The subdiagonal elements are stored in $A(j:n, 1:m_1), j > 1$ according to the number of elements on each subdiagonal.
3. The superdiagonal elements are stored in $A(1:j, m_1+2:m_1+m_2+1)$, $j < n$ according to the number of elements on each superdiagonal.

The inner points coefficients of the computational grid are stored first. The boundary points are stored next. The rows corresponding to the boundary points only have one nonzero entry equal to **unity**.

The finite difference of the VTE (Eq. 5.50) written in a generalized form is:

$$a_{i,j}\,\omega_{i+1,j} + b_{i,j}\,\omega_{i-1,j} + c_{i,j}\,\omega_{i,j+1} + d_{i,j}\,\omega_{i,j-1} + e_{i,j}\,\omega_{i,j} = (\nabla \times \mathbf{F})_{i,j} \tag{5.55}$$

with

$$\begin{aligned}
a_{i,j} &= -\left(\nu \frac{\xi_x^2}{(\Delta\xi)^2}\right) \\
b_{i,j} &= -\left(\nu \frac{\xi_x^2}{(\Delta\xi)^2}\right) \\
c_{i,j} &= -\,\nu \left(\frac{\eta_{yy}}{2\Delta\eta} + \frac{\eta_y^2}{(\Delta\eta)^2}\right) \\
d_{i,j} &= -\,\nu \left(-\frac{\eta_{yy}}{2\Delta\eta} + \frac{\eta_y^2}{(\Delta\eta)^2}\right) \\
e_{i,j} &= 2\,\nu \left(\frac{\xi_x^2}{(\Delta\xi)^2} + \frac{\eta_y^2}{(\Delta\eta)^2}\right)
\end{aligned} \tag{5.56}$$

and

$$(\nabla \times \mathbf{F})_{i,j} = \frac{\xi_x}{2\Delta\xi} F_{(y)i+1,j} - \frac{\xi_x}{2\Delta\xi} F_{(y)i-1,j} - \frac{\eta_y}{2\Delta\eta} F_{(y)i,j+1} + \frac{\eta_y}{2\Delta\eta} F_{(y)i,j-1} \quad (5.57)$$

For illustration let the grid be a 5×5 rectangular grid. The equations for the inner grid points are:

$$\begin{array}{lllll}
a_{2,2}\,\omega_{3,2}+ & b_{2,2}\,\omega_{1,2}+ & c_{2,2}\,\omega_{2,3}+ & d_{2,2}\,\omega_{2,1}+ & e_{2,2}\,\omega_{2,2} = \quad -(\nabla \times \mathbf{F})_{2,2} \\
a_{3,2}\,\omega_{4,2}+ & b_{3,2}\,\omega_{2,2}+ & c_{3,2}\,\omega_{3,3}+ & d_{3,2}\,\omega_{3,1}+ & e_{3,2}\,\omega_{3,2} = \quad -(\nabla \times \mathbf{F})_{3,2} \\
a_{4,2}\,\omega_{5,2}+ & b_{4,2}\,\omega_{3,2}+ & c_{4,2}\,\omega_{4,3}+ & d_{4,2}\,\omega_{4,1}+ & e_{4,2}\,\omega_{4,2} = \quad -(\nabla \times \mathbf{F})_{4,2} \\
a_{2,3}\,\omega_{3,3}+ & b_{2,3}\,\omega_{1,3}+ & c_{2,3}\,\omega_{2,4}+ & d_{2,3}\,\omega_{2,2}+ & e_{2,3}\,\omega_{2,3} = \quad -(\nabla \times \mathbf{F})_{2,3} \\
a_{3,3}\,\omega_{4,3}+ & b_{3,3}\,\omega_{2,3}+ & c_{3,3}\,\omega_{3,4}+ & d_{3,3}\,\omega_{3,2}+ & e_{3,3}\,\omega_{3,3} = \quad -(\nabla \times \mathbf{F})_{3,3} \\
a_{4,3}\,\omega_{5,3}+ & b_{4,3}\,\omega_{3,3}+ & c_{4,3}\,\omega_{4,4}+ & d_{4,3}\,\omega_{4,2}+ & e_{4,3}\,\omega_{4,3} = \quad -(\nabla \times \mathbf{F})_{4,3} \\
 & & \vdots & &
\end{array}$$

Then the coefficients of ω of the inner grid points are stored in the stiffness matrix $\mathbf{A}$, while $\nabla \times \mathbf{F}$ is stored in the right hand side as vector $\mathbf{b}$ such that:

$$\begin{bmatrix}
e_{2,2} & a_{2,2} & 0 & 0 & c_{2,2} & 0 & 0 & 0 & 0 & 0 & \cdots \\
b_{3,2} & e_{3,2} & a_{3,2} & 0 & 0 & c_{3,2} & 0 & 0 & 0 & 0 & \cdots \\
0 & b_{4,2} & e_{4,2} & a_{4,2} & 0 & 0 & c_{4,2} & 0 & 0 & 0 & \cdots \\
0 & 0 & b_{5,2} & e_{5,2} & 0 & 0 & 0 & c_{5,2} & 0 & 0 & \cdots \\
d_{2,3} & 0 & 0 & 0 & e_{2,3} & a_{2,3} & 0 & 0 & c_{2,3} & 0 & \cdots \\
0 & d_{3,3} & 0 & 0 & b_{3,3} & e_{3,3} & a_{3,3} & 0 & 0 & c_{3,3} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots
\end{bmatrix}
\begin{bmatrix}
\omega_{2,2} \\ \omega_{3,2} \\ \omega_{4,2} \\ \omega_{2,3} \\ \omega_{3,3} \\ \omega_{4,3} \\ \vdots
\end{bmatrix}
= -\boldsymbol{\nabla} \times \mathbf{F}$$

where

$$-\boldsymbol{\nabla} \times \mathbf{F} = \begin{bmatrix}
-(\nabla \times \mathbf{F})_{2,2} \\
-(\nabla \times \mathbf{F})_{3,2} \\
-(\nabla \times \mathbf{F})_{4,2} \\
-(\nabla \times \mathbf{F})_{2,3} \\
-(\nabla \times \mathbf{F})_{3,3} \\
-(\nabla \times \mathbf{F})_{4,3} \\
\vdots
\end{bmatrix}$$

Then we have to put in the coefficients for the boundary points whose values are all equal to unity so that:

$$\mathbf{A} \cdot \boldsymbol{\omega} = -\boldsymbol{\nabla} \times \mathbf{F} \tag{5.58}$$

where

$$\mathbf{A} = \begin{bmatrix}
\cdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots \\
\cdots & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
\cdots & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
\cdots & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
\cdots & 0 & 0 & b_{2,2} & e_{2,2} & a_{2,2} & 0 & 0 & c_{2,2} & 0 & 0 & 0 & \cdots \\
\cdots & 0 & 0 & 0 & b_{3,2} & e_{3,2} & a_{3,2} & 0 & 0 & 0 & c_{3,2} & 0 & \cdots \\
\cdots & 0 & 0 & 0 & 0 & b_{4,2} & e_{4,2} & a_{4,2} & 0 & 0 & 0 & c_{4,2} & \cdots \\
\cdots & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & \cdots \\
\cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \cdots \\
\cdots & 0 & 0 & 0 & d_{2,3} & 0 & 0 & 0 & b_{2,3} & e_{2,3} & a_{2,3} & 0 & \cdots \\
\cdots & 0 & 0 & 0 & 0 & d_{3,3} & 0 & 0 & 0 & b_{3,3} & e_{3,3} & a_{3,3} & \cdots \\
\cdots & 0 & 0 & 0 & 0 & 0 & d_{4,3} & 0 & 0 & 0 & b_{4,3} & e_{4,3} & \cdots \\
\cdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots
\end{bmatrix} \tag{5.59}$$

$$\boldsymbol{\omega} = \begin{bmatrix}
\vdots \\
\boldsymbol{\omega_{4,1}} \\
\boldsymbol{\omega_{5,1}} \\
\boldsymbol{\omega_{1,2}} \\
\omega_{2,2} \\
\omega_{3,2} \\
\omega_{4,2} \\
\boldsymbol{\omega_{5,2}} \\
\boldsymbol{\omega_{1,3}} \\
\omega_{2,3} \\
\omega_{3,3} \\
\omega_{4,3} \\
\vdots
\end{bmatrix} \qquad -\boldsymbol{\nabla} \times \mathbf{F} = \begin{bmatrix}
\vdots \\
-(\nabla \times \mathbf{F})_{4,1} \\
-(\nabla \times \mathbf{F})_{5,1} \\
-(\nabla \times \mathbf{F})_{1,2} \\
-(\nabla \times \mathbf{F})_{2,2} \\
-(\nabla \times \mathbf{F})_{3,2} \\
-(\nabla \times \mathbf{F})_{4,2} \\
-(\nabla \times \mathbf{F})_{5,2} \\
-(\nabla \times \mathbf{F})_{1,3} \\
-(\nabla \times \mathbf{F})_{2,3} \\
-(\nabla \times \mathbf{F})_{3,3} \\
-(\nabla \times \mathbf{F})_{4,3} \\
\vdots
\end{bmatrix} \tag{5.60}$$

Note, in this section only, that the $\boldsymbol{\omega}$'s elements in bold are the boundary points.

We can store the stiffness matrix $\mathbf{A}$ compactly by taking the diagonal along with the $(nx+1)$ subdiagonal and superdiagonal, and tilt them 45° clockwise. The resulting matrix dimension is $(nx \times nx)$ rows by $(2\,nx+1)$ columns:

$$\mathbf{A}_{\text{compact}} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\
d_{2,2} & 0 & 0 & 0 & b_{2,2} & e_{2,2} & a_{2,2} & 0 & 0 & c_{2,2} \\
d_{3,2} & 0 & 0 & 0 & b_{3,2} & e_{3,2} & a_{3,2} & 0 & 0 & c_{3,2} \\
d_{4,2} & 0 & 0 & 0 & b_{4,2} & e_{4,2} & a_{4,2} & 0 & 0 & c_{4,2} \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\
d_{2,3} & 0 & 0 & 0 & b_{2,3} & e_{2,3} & a_{2,3} & 0 & 0 & c_{2,3} \\
d_{3,3} & 0 & 0 & 0 & b_{3,3} & e_{3,3} & a_{3,3} & 0 & 0 & c_{3,3} \\
d_{4,3} & 0 & 0 & 0 & b_{4,3} & e_{4,3} & a_{4,3} & 0 & 0 & c_{4,3} \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\
d_{2,4} & 0 & 0 & 0 & b_{2,4} & e_{2,4} & a_{2,4} & 0 & 0 & c_{2,4} \\
d_{3,4} & 0 & 0 & 0 & b_{3,4} & e_{3,4} & a_{3,4} & 0 & 0 & c_{3,4} \\
d_{2,4} & 0 & 0 & 0 & b_{2,4} & e_{2,4} & a_{2,4} & 0 & 0 & c_{2,4} \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0
\end{bmatrix}$$

Once the compact form of matrix $\mathbf{A}$ is established, LU decomposition is performed. Since the LU decomposition of $\mathbf{A}$ produces additional nonzero fill-ins, it cannot be stored as compactly as $\mathbf{A}$ itself [29]. A subroutine from *Numerical*

Recipes provides a storage scheme that returns the upper triangular matrix in the space that was previously occupied by **A** with the diagonal elements of **A** located in the first column of the storage. The lower triangular matrix is placed into a separate compact matrix of size $N \times m_1$. Pivoting is exercised to avoid instability caused by divisions by small numbers. Since Crout's method of LU decomposition is employed, only partial pivoting (interchanging rows only) can be efficiently implemented [29]. In essence, the LU decomposition is not done on matrix **A** itself but rather the **rowwise permutation** of **A**.

5.1.2 Boundary Points

The boundary points are known quantities. Because of this their coefficients are unity in the stiffness matrix as explained in the numerical calculation chapter.

5.1.3 Right Hand Side Vector Formation

The right hand side vector holds the known quantities, which is the forcing function and the boundary conditions in this case. By storing the boundary conditions in the right hand side vector, it enables us to set the boundary points values to unity in the stiffness matrix as mentioned above.

The stream function can be solved by the same method as the vorticity. Poisson's equation in the transformed domain is:

$$\xi_x^2 \frac{\partial^2 \psi}{\partial x^2} + \eta_{yy} \frac{\partial \psi}{\partial \eta} + \eta_y^2 \frac{\partial^2 \psi}{\partial y^2} = -\omega \tag{5.61}$$

The finite difference form is:

$$\begin{aligned} \xi_x^2 \frac{\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j}}{(\Delta\xi)^2} + \eta_{yy} \frac{\psi_{i,j+1} - \psi_{i,j-1}}{2\Delta\eta} \\ \eta_y^2 \frac{\psi_{i,j+1} - 2\psi_{i,j} + \psi_{i,j-1}}{(\Delta\eta)^2} = -\omega_{i,j} \end{aligned} \tag{5.62}$$

Placing the unknown values on the left hand side and the known values on the

right hand side:

$$\begin{aligned}\left(\frac{\xi_x^2}{(\Delta\xi)^2}\right)\psi_{i+1,j}+\left(\frac{\xi_x^2}{(\Delta\xi)^2}\right)\psi_{i-1,j}+\left(\frac{\eta_{yy}}{2\Delta\eta}+\frac{\eta_y^2}{(\Delta\eta^2)^2}\right)\psi_{i,j+1}+\\ \left(-\frac{\eta_{yy}}{2\Delta\eta}+\frac{\eta_y^2}{(\Delta\eta^2)^2}\right)\psi_{i,j-1}-2\left(\frac{\xi_x^2}{(\Delta\xi)^2}+\frac{\eta_y^2}{(\Delta\eta)^2}\right)\psi_{i,j}=-\omega_{i,j}\end{aligned} \tag{5.63}$$

The coefficients for the stiffness matrix are then:

$$\begin{aligned}a_{i,j}&=\left(\frac{\xi_x^2}{(\Delta\xi)^2}\right)\\ b_{i,j}&=\left(\frac{\xi_x^2}{(\Delta\xi)^2}\right)\\ c_{i,j}&=\left(\frac{\eta_{yy}}{2\Delta\eta}+\frac{\eta_y^2}{(\Delta\eta^2)^2}\right)\\ d_{i,j}&=\left(-\frac{\eta_{yy}}{2\Delta\eta}+\frac{\eta_y^2}{(\Delta\eta^2)^2}\right)\\ e_{i,j}&=-2\left(\frac{\xi_x^2}{(\Delta\xi)^2}+\frac{\eta_y^2}{(\Delta\eta)^2}\right)\\ \mathbf{F}_{i,j}&=-\omega_{i,j}\end{aligned} \tag{5.64}$$

The LU decomposition of the compact stiffness matrix and the forward and backward substitutions are performed in the same manner as explained in the VTE solution section. Once the stream function is computed, the axial and transverse acoustic streaming velocities are calculated by:

$$\begin{aligned}u_2=&\frac{\partial\psi}{\partial y}=\eta_y\,\frac{\partial\psi}{\partial\eta}=\eta_y\,\frac{\psi_{i,j+1}-\psi_{i,j-1}}{2\Delta\eta}\\ w_2=&-\frac{\partial\psi}{\partial x}=-\xi_x\,\frac{\partial\psi}{\partial\eta}=\eta_y\,\frac{\psi_{i+1,j}-\psi_{i-1,j}}{2\Delta\xi}\end{aligned} \tag{5.65}$$

with boundary conditions:

$$\begin{aligned}u_2,w_2&=0 \quad \text{on tube ends and at the wall}\\ w_2&=0 \quad \text{on the axis of symmetry}\\ \frac{\partial u_2}{\partial y}&=0 \quad \text{on the axis of symmetry}\end{aligned} \tag{5.66}$$

5.2 Under-Relaxation

The calculation steps as explained thus far, are as follows:

1. Calculate the vorticity ω, using the VTE with first order acoustic velocities as the driving force.

2. Knowing the values of ω, calculate the stream function ψ by solving Poisson's equation.

3. Calculate the axial and transverse acoustic streaming velocities by taking the y and x derivatives of ψ.

It was found that if these steps were performed just one time to calculate the streaming velocities, the results will be incorrect. Steps 1 through 3 have to be executed many times until the solution converges. And in order to reach convergence, an under-relaxation method has to be utilized when calculating ψ to prevent divergence [40]. This is because for large sets of linear equations, roundoff errors can accumulate and grow to the extent that the matrix is close to singular [29]. Therefore to restore the full machine precision, iterative improvement (in this case the under-relaxation method) can be done to the solution. For each iteration, an "intermediate" step is performed between step 2 and 3 such that:

$$\psi^{n+1} = \varepsilon\psi + (1-\varepsilon)\psi^n \tag{5.67}$$

where ψ^{n+1} is the "under-relaxed" stream function, ψ is the "newly" calculated value of the stream function (from step 2), and ψ^n is the value of stream function from the previous iteration. The new under-relaxed value of the stream function ψ^{n+1} then has to saved as the "old" value so that it can be used as ψ^n for the next iteration. ε is the relaxation factor where $0 \leq \varepsilon \leq 1$. Hence ψ^{n+1} is the weighted average of the old (ψ^n) and the new (ψ) solutions. It was found that for this research the value of ε has to be sufficiently small for the solution to converge correctly. A typical value of ε used in this research is 10^{-4}. This means that the iteration number is large and if the size of the computational grid is large, the calculation time can be long.

5.3 Outline of Computation

The numerical computation consists of the following steps:

1. Define the physical parameters to be used in the calculation such as the dimensions of the simulated device, the speed of sound, viscosity, driving velocity amplitude, etc.
2. Make the physical grid and the computational grid.
3. Define the metrics of transformation.
4. Calculate the forcing function from the first order velocities.
5. Store the coefficients of $\boldsymbol{\omega}$ of the inner grid points in the compact stiffness matrix.
6. Store the boundary values in the compact stiffness matrix.
7. Perform LU decomposition for the banded matrix.
8. Solve the set of algebraic equations for the VTE through back substitution.
9. Store the newly calculated values of $\boldsymbol{\omega}$ to be used as the forcing function for Poisson's equation
10. Store the coefficients of $\boldsymbol{\psi}$ of the inner grid points in the compact stiffness matrix.
11. Store the boundary values in the compact stiffness matrix.
12. Perform LU decomposition for the banded matrix.
13. Solve the set of algebraic equations for Poisson's equation through back substitution.
14. Perform the under-relaxation process on the stream function $\boldsymbol{\psi}$ using the newly calculated and the old (from previous iteration) values of $\boldsymbol{\psi}$.
15. Calculate the acoustic streaming velocity by taking the derivative of the stream function with respect to x and y.

16. Repeat steps 5 through 15 until the solution converges.

Chapter 6

Analytical And Numerical Acoustic Streaming Comparison

In the following sections, the analytical and numerical streaming velocities are going to be compared. First the forcing functions computed by the three groups (Rayleigh/Nyborg [27], Hamilton *et al.* [11] and Bailliet *et al.* [3]) will be examined. Then the analytical streaming velocities from each group will be evaluated. After that the analytical and numerical streaming velocities are going to be compared. Finally the numerical streaming velocities resulting from the different forcing functions (i.e. from the different groups) will be investigated. All of the results presented in this chapter were calculated using the parameters given in Chapter 3.

6.1 Forcing Functions And Their Gradients Comparison

Figure 6.1 shows the x component of the forcing function for all three groups. Away from the boundary, the Hamilton *et al.* and Bailliet *et al.* forcing functions are very close in amplitude while Rayleigh/Nyborg's has a slightly lower amplitude. It is 0.065 kg/m^{2}s^2 (3.2%) lower than Hamilton *et al.* and 0.06 kg/m^{2}s^2 (2.94%) lower than Bailliet *et al.* Their differences are more pronounced in the area near the wall as seen in Fig. 6.2. The Hamilton *et al.* forcing function has the highest peak near the wall of 2.357 kg/m^{2}s^2 compared to the two other groups. The highest

amplitude in the area near the wall for the Rayleigh/Nyborg forcing function is 2.281 kg/m^{2}s^2 (3.23% lower than Hamilton *et al.*) and for Bailliet *et al.* it is 2.148 kg/m^{2}s^2 (8.87% lower than Hamilton *et al.*). The forcing function amplitudes on the axis and near the boundary are tabulated on Table 6.1.

Research group	On axis (kg/m^{2}s^2)	Positive peak near boundary (kg/m^{2}s^2)
Rayleigh/Nyborg	1.975	2.281
Hamilton *et al.*	2.039	2.357
Bailliet *et al.*	2.034	2.148

Table 6.1. Rayleigh/Nyborg, Hamilton *et al.*, and Bailliet *et al.* forcing function on the axis and near the boundary for a 21×81 grid.

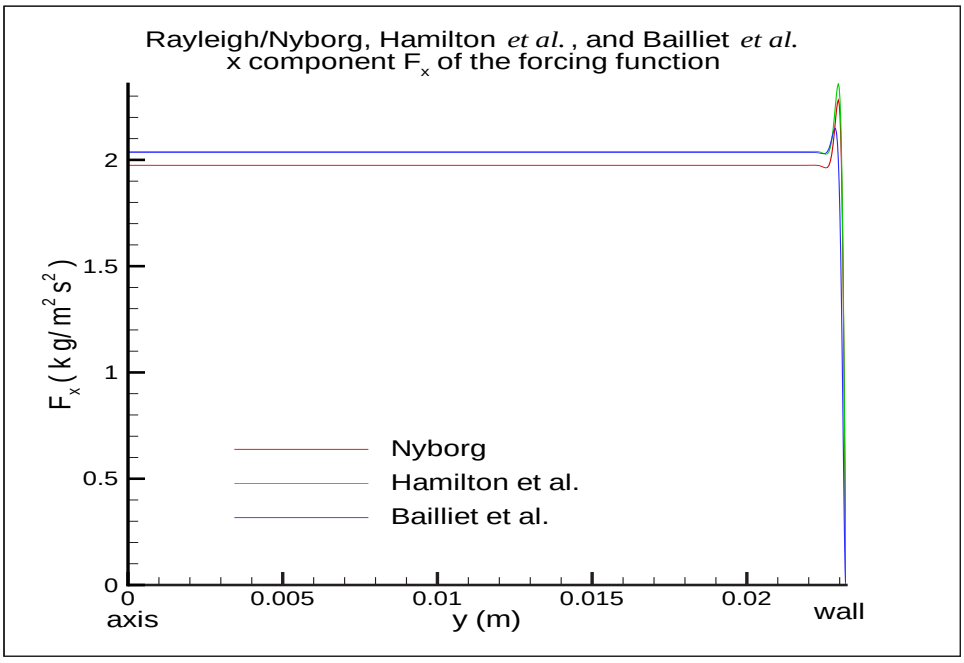

Figure 6.1. Rayleigh/Nyborg, Hamilton *et al.*, and Bailliet *et al.* forcing functions at an antinode along y direction. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

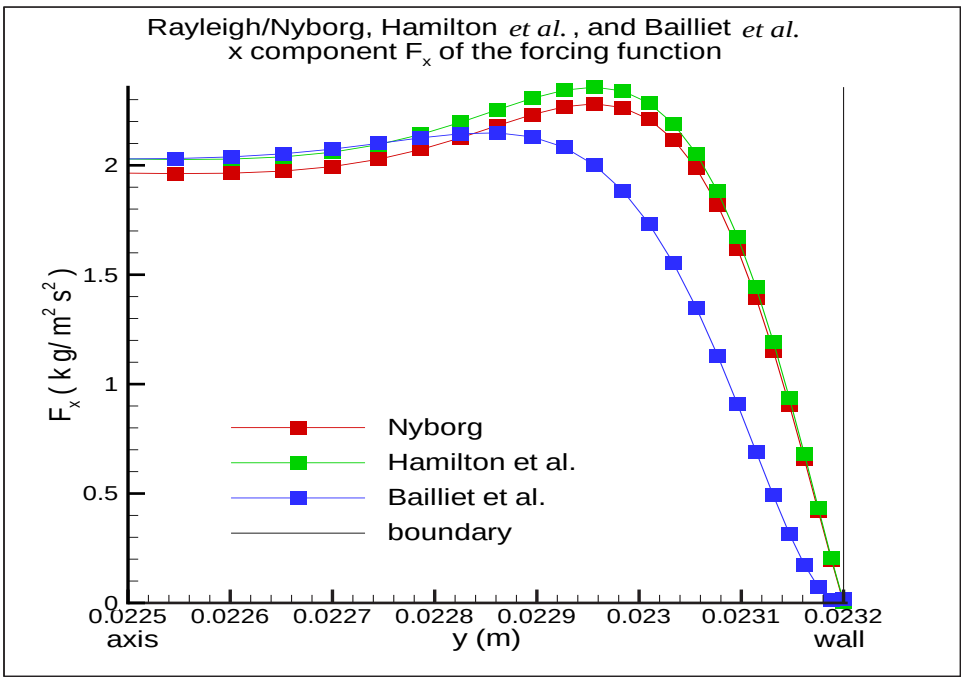

Figure 6.2. Rayleigh/Nyborg, Hamilton *et al.*, and Bailliet *et al.* forcing functions at an antinode along y direction. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

Since in the time-averaged VTE the gradient of the forcing function is the driving force of the equation, it is essential that one observes the gradient of the forcing function along with the forcing function itself. For this reason, the gradients of the forcing functions of the three groups are plotted in Figs. 6.3 and 6.4. In Fig. 6.4, it can be observed that away from the wall, the differences between the gradients of the forcing functions are not significant while in the area near the wall, the differences are more considerable. Notice that the difference in the gradients between Rayleigh/Nyborg and the Hamilton *et al.* is significantly smaller than the difference between the two groups' gradients compared to the Bailliet *et al.*'s gradients shown in Fig. 6.4.

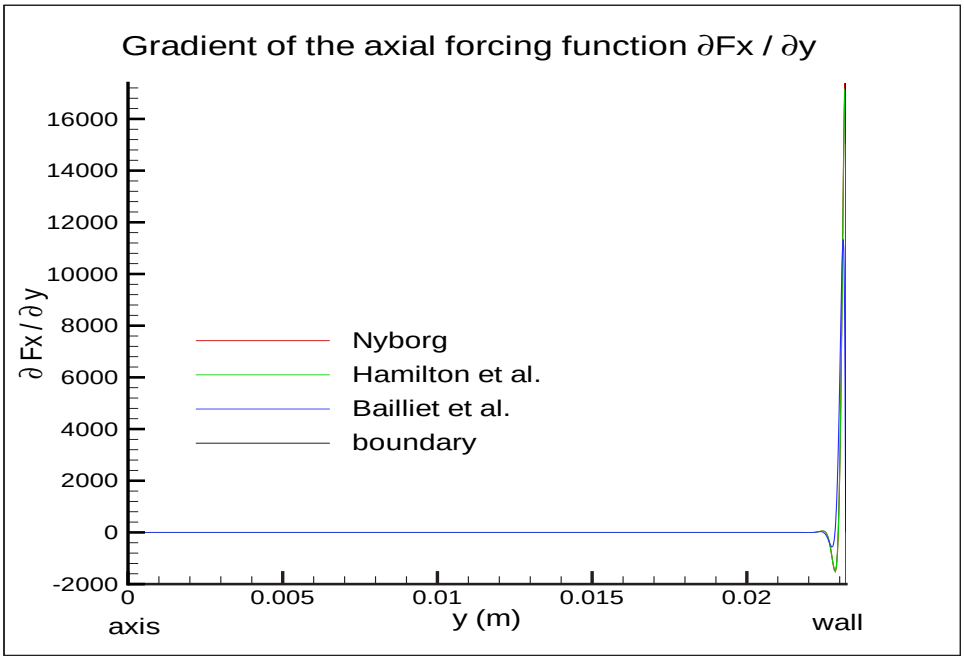

Figure 6.3. Rayleigh/Nyborg, Hamilton *et al.*, and Bailliet *et al.* gradients of the forcing functions along y direction. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

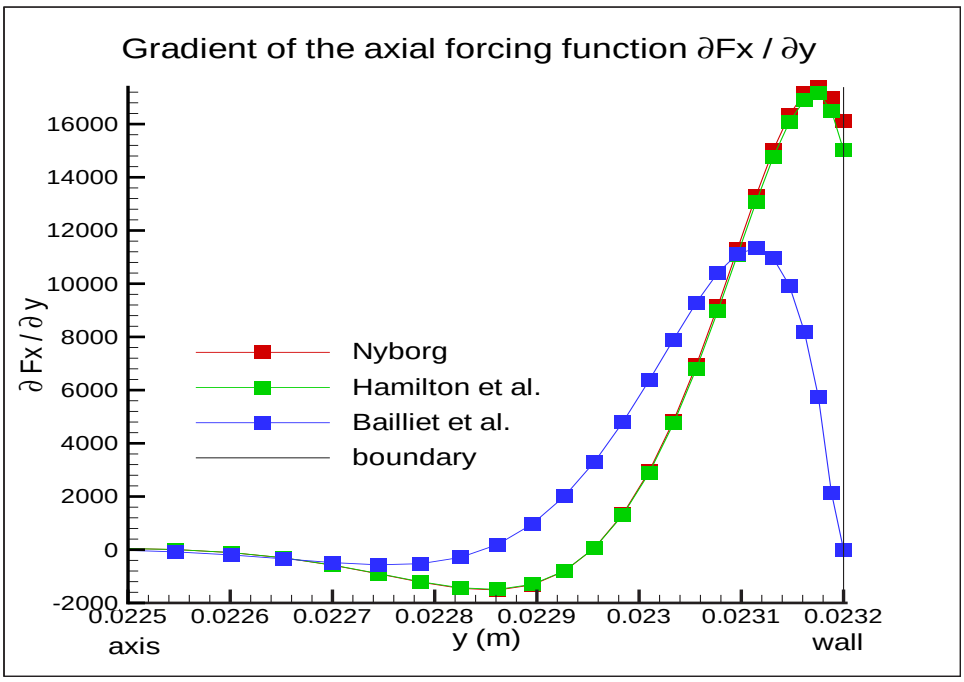

Figure 6.4. Rayleigh/Nyborg, Hamilton *et al.*, and Bailliet *et al.* gradients of the forcing functions along y direction. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

6.2 Analytical Streaming Comparison

Figure 6.5 shows the analytical streaming velocities of all three groups at an antinode. Towards the axis, the Rayleigh/Nyborg and Hamilton *et al.* analytical streaming velocities are fairly close to each other. Hamilton *et al.*'s streaming velocity is 7.5×10^{-6} m/s (2.3%) higher than Rayleigh/Nyborg's. The Bailliet *et al.* analytical streaming velocity is noticeably higher than the other two. Compared to Rayleigh/Nyborg, Bailliet *et al.*'s u_2 is 2.63×10^{-5} m/s (7.7%) higher. Compared to Hamilton *et al.*, it is 1.89×10^{-5} m/s (5.6%) higher. In the area next to the wall, it can be observed from Fig. 6.6 that the Rayleigh/Nyborg and Hamilton *et al.* streaming velocities agree well with each other. The values of their positive peaks only differ by 3.3×10^{-6} m/s (5.1%). Their negative peaks differ by 3.1×10^{-5} m/s (4.7%). The Bailliet *et al.*'s positive peak is lower than both Rayleigh/Nyborg and Hamilton *et al.* by 4.8×10^{-5} m/s (74%), while its negative peak is higher by 5.4×10^{-5} m/s (8.5%).

Research group	On axis (m/s)	Positive peak (m/s)	Negative peak (m/s)
Rayleigh/Nyborg	3.14×10^{-4}	6.12×10^{-5}	6.18×10^{-4}
Hamilton *et al.*	3.21×10^{-4}	6.45×10^{-5}	6.49×10^{-4}
Bailliet *et al.*	3.41×10^{-4}	1.67×10^{-5}	7.04×10^{-4}

Table 6.2. Rayleigh/Nyborg, Hamilton *et al.*, and Bailliet *et al.* analytical streaming velocities on the axis and near the boundary for a 21×81 grid.

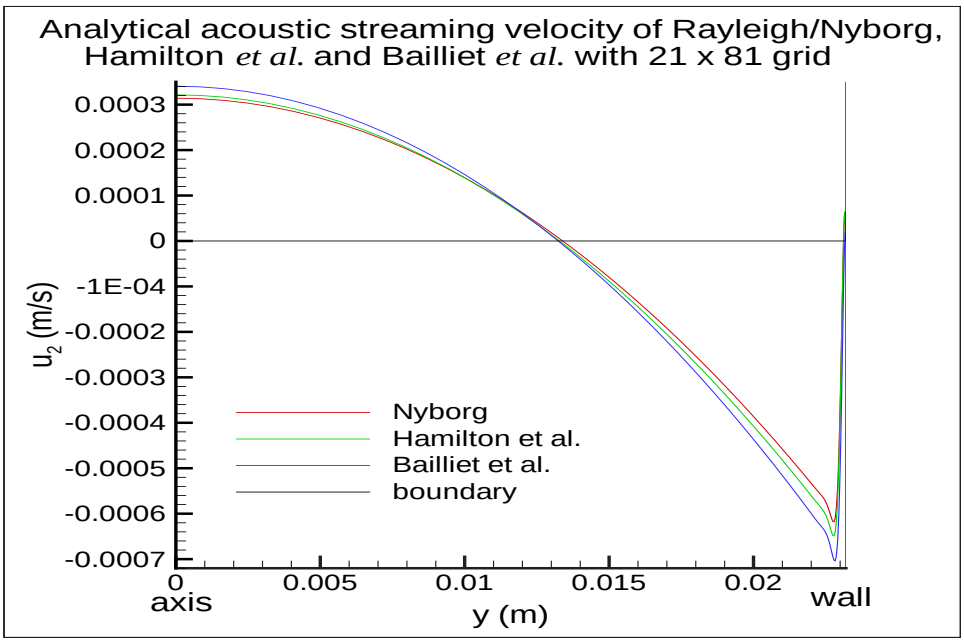

Figure 6.5. Rayleigh/Nyborg, Hamilton *et al.*, and Bailliet *et al.* analytical streaming velocities at an antinode along y direction. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

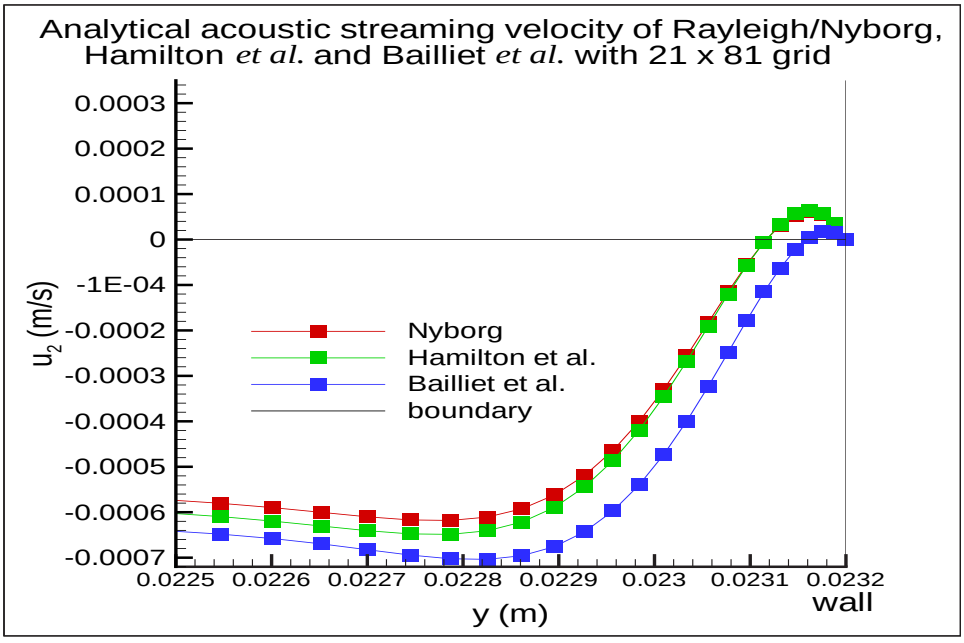

Figure 6.6. Rayleigh/Nyborg, Hamilton *et al.*, and Bailliet *et al.* analytical streaming velocities at an antinode along y direction. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

6.3 Analytical vs. Numerical Streaming: Parallel Plate Case

6.3.1 Rayleigh/Nyborg

Figures 6.7 through 6.12 show the numerical streaming velocity at an antinode calculated using the Rayleigh/Nyborg forcing function. In Figs. 6.7 and 6.8 the calculation was made using 21×31 grid points, in Figs. 6.9 and 6.10 the grid was 21×61, and 21×81 in Figs. 6.11 and 6.12. It can be observed from Figs. 6.7, 6.9, and 6.11 that away from the boundary the numerical streaming velocity is not significantly affected by the number of grid points. But Figs. 6.8, 6.10, and 6.12 show that in the boundary layer the number of grid points used in the calculation affects the accuracy of the results. The finer the grid spacing is near the boundary, the closer the numerical results are to the analytical values. For this calculation, 21×81 grid points give velocity results which agree very well with the analytical velocities.

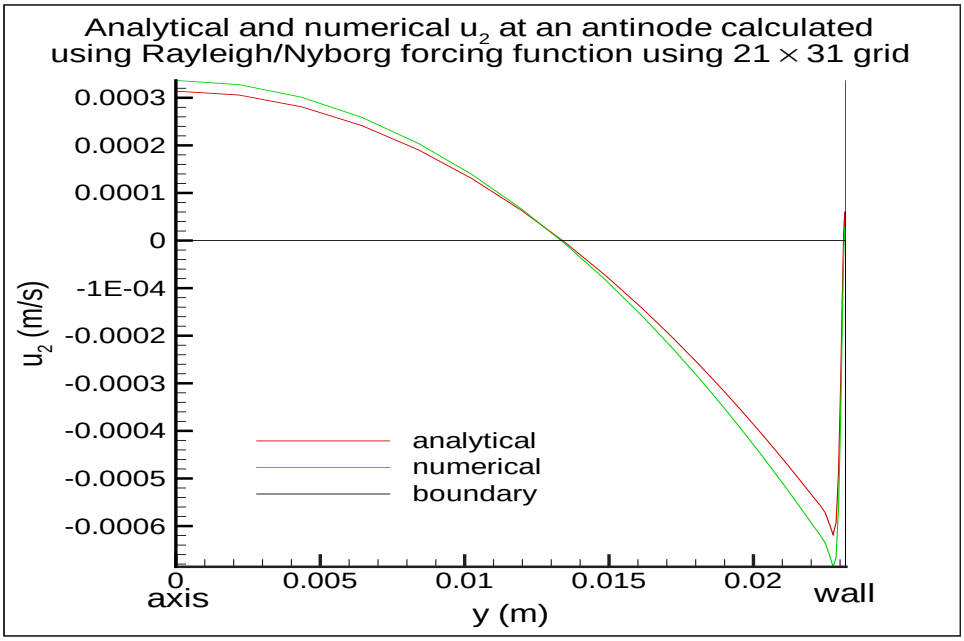

Figure 6.7. Numerical streaming velocity calculated using the Rayleigh/Nyborg forcing function at an antinode along y direction with 21×31 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

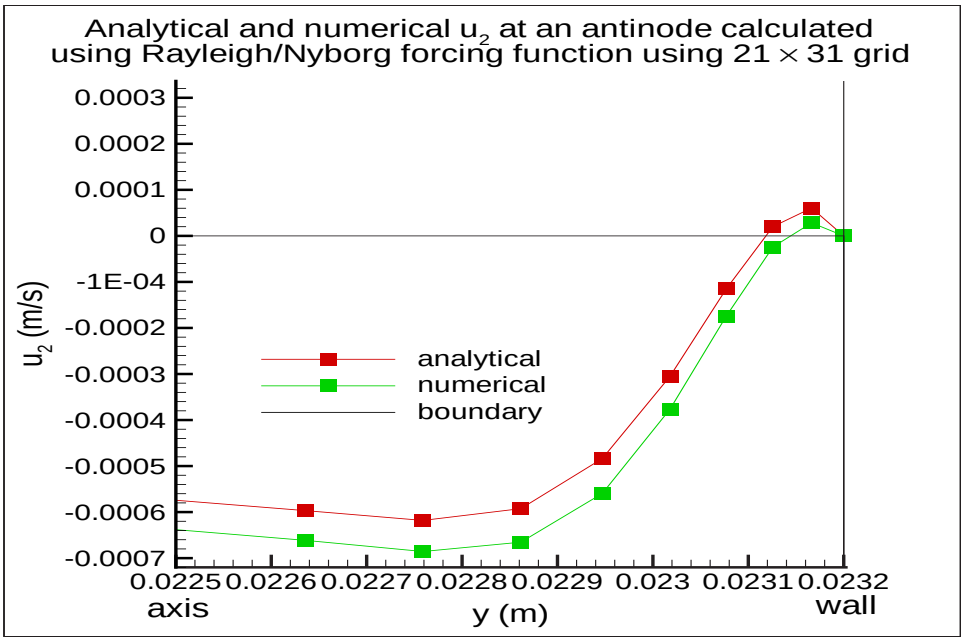

Figure 6.8. Numerical streaming velocity calculated using the Rayleigh/Nyborg forcing function at an antinode along y direction with 21×31 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

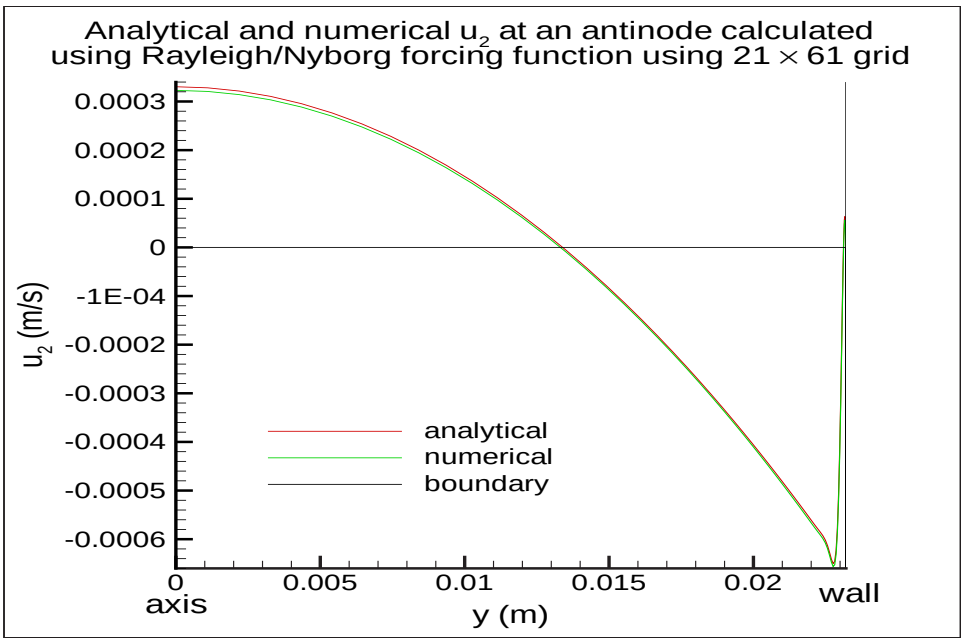

Figure 6.9. Numerical streaming velocity calculated using the Rayleigh/Nyborg forcing function at an antinode along y direction with 21×61 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

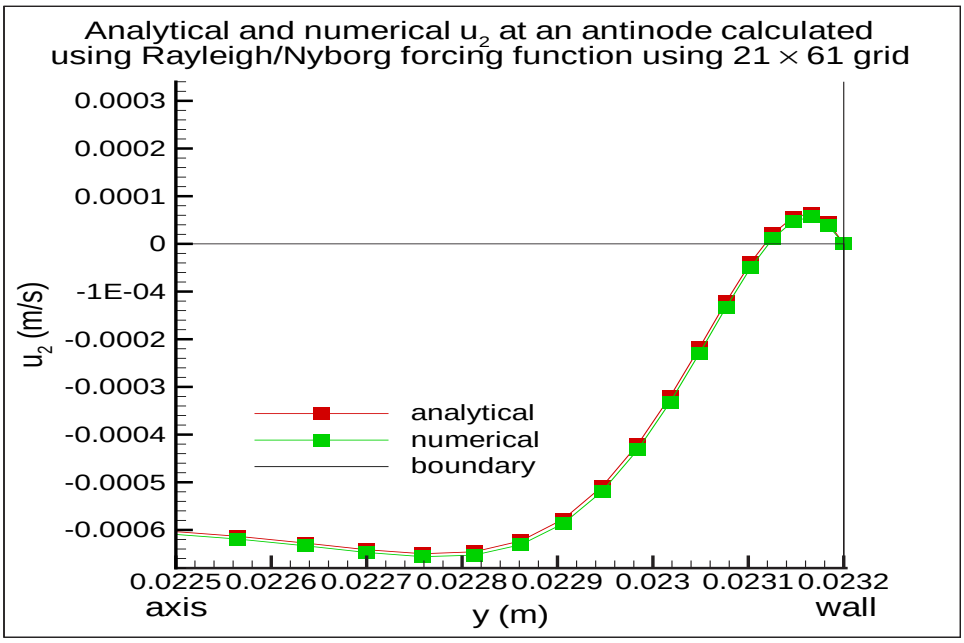

Figure 6.10. Numerical streaming velocity calculated using the Rayleigh/Nyborg forcing function at an antinode along y direction with 21×61 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

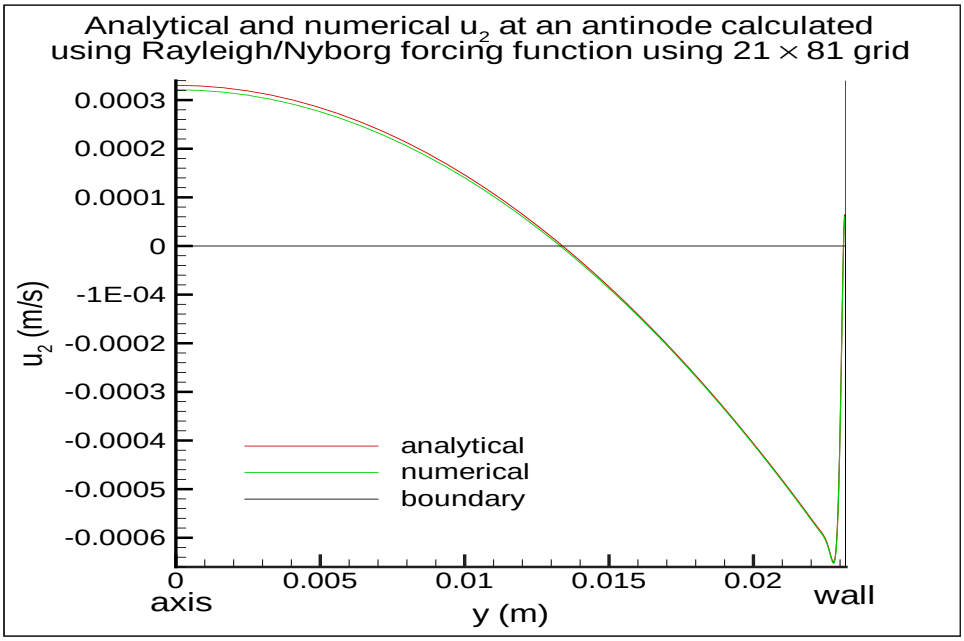

Figure 6.11. Numerical streaming velocity calculated using the Rayleigh/Nyborg forcing function at an antinode along y direction with 21×81 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

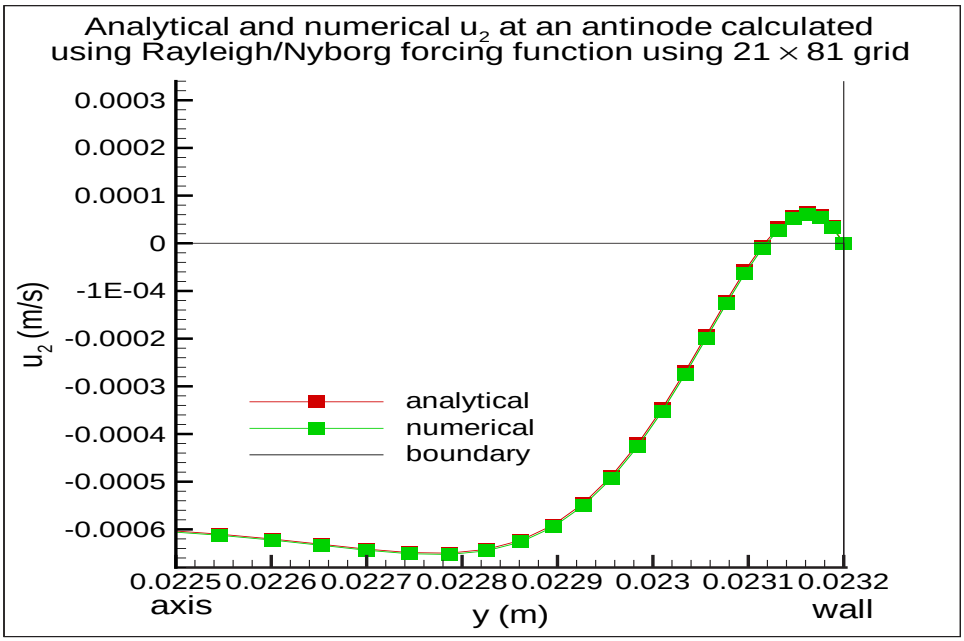

Figure 6.12. Numerical streaming velocity calculated using the Rayleigh/Nyborg forcing function at an antinode along y direction with 21×81 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

6.3.2 Hamilton *et al.*

In Figs. 6.13 through 6.18 the numerical streaming velocities calculated using the Hamilton *et al.* forcing function are presented. The calculation was performed using three grid sizes. Figures 6.13 and 6.14 are the results with 21×31 grid points, Figs. 6.15 and 6.16 with 21×61 grid points, and Figs. 6.17 and 6.18 with 21×81 grid points. As in the Rayleigh/Nyborg's case, the size of the grid makes a difference when it comes to the streaming velocity near the boundary. A 21×81 grid will suffice for this calculation. The numerical velocity agrees well with the analytical velocity as observed in Figs. 6.17 and 6.18.

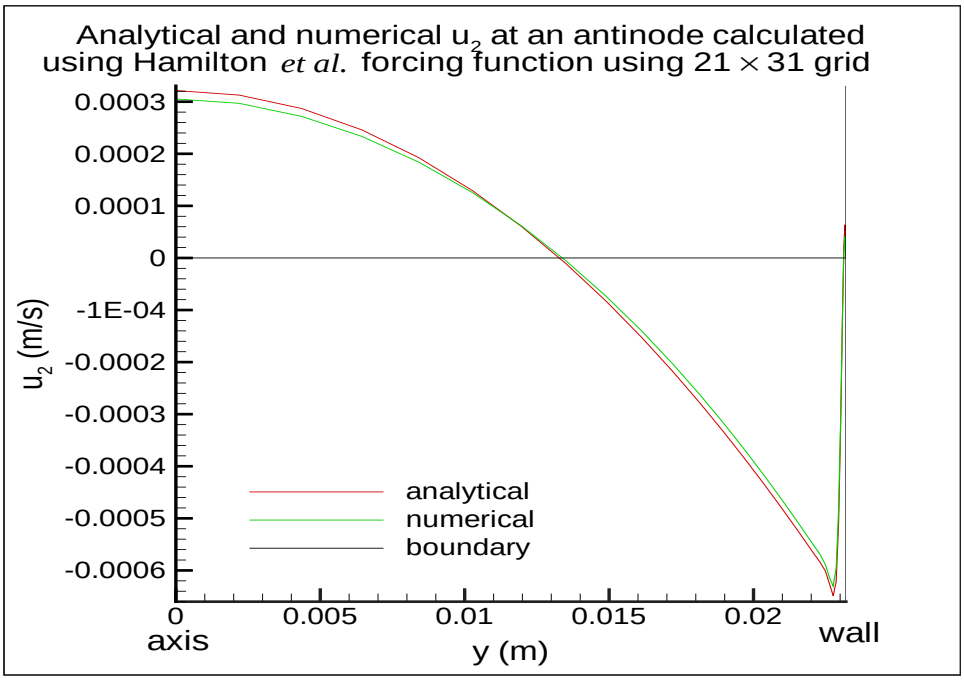

Figure 6.13. Numerical streaming velocity calculated using Hamilton *et al.* forcing function at an antinode along y direction with 21×31 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

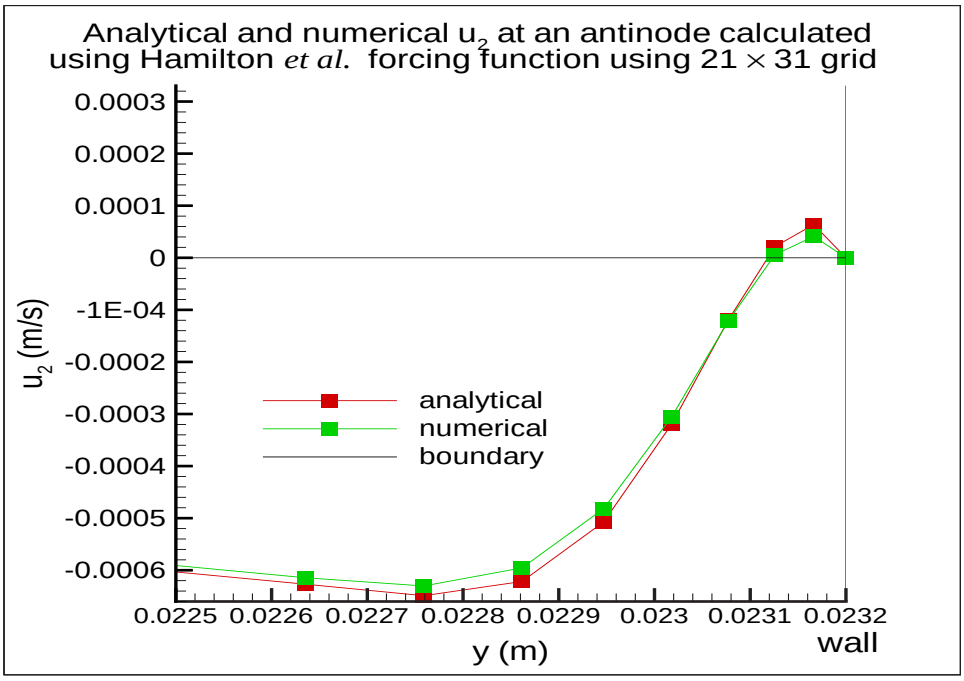

Figure 6.14. Numerical streaming velocity calculated using Hamilton *et al.* forcing function at an antinode along y direction with 21×31 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

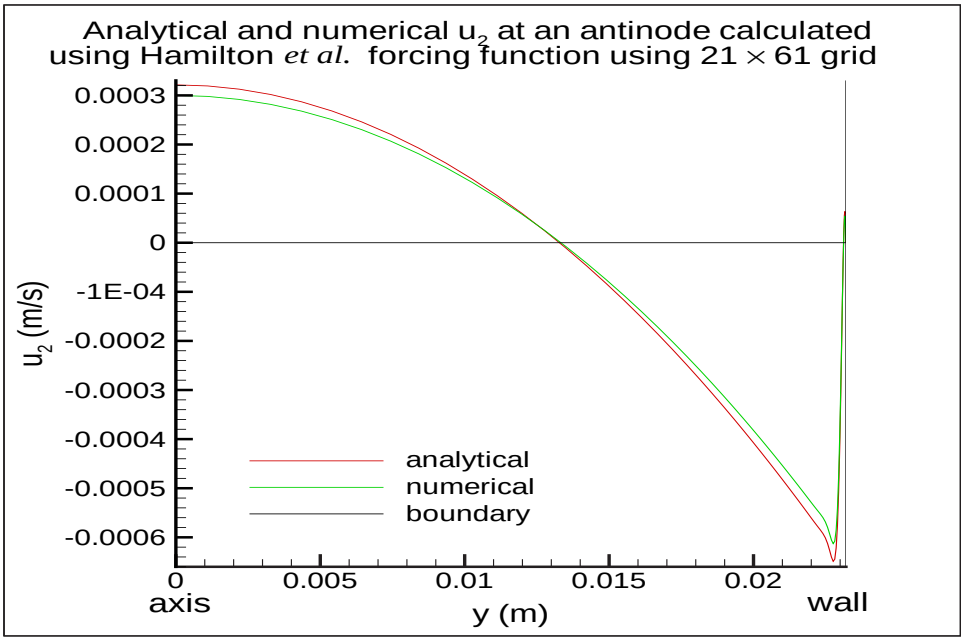

Figure 6.15. Numerical streaming velocity calculated using Hamilton *et al.* forcing function at an antinode along y direction with 21×61 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

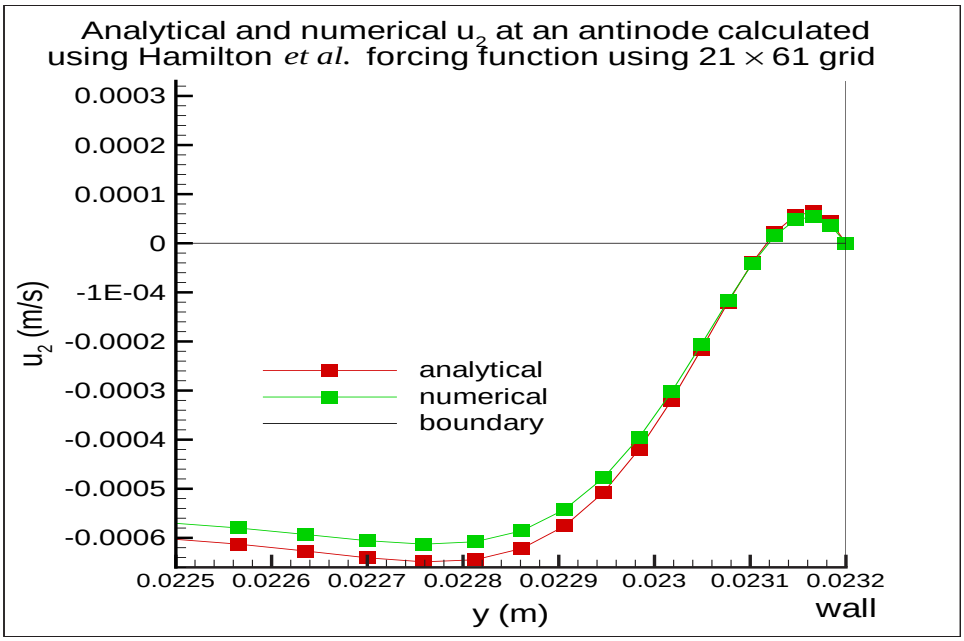

Figure 6.16. Numerical streaming velocity calculated using Hamilton *et al.* forcing function at an antinode along y direction with 21×61 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

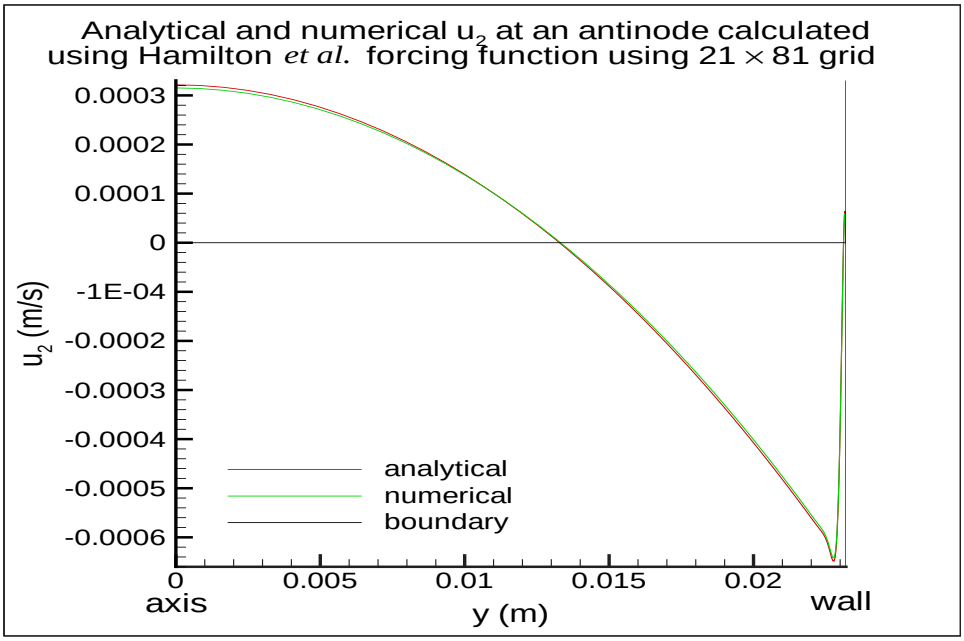

Figure 6.17. Numerical streaming velocity calculated using Hamilton *et al.* forcing function at an antinode along y direction with 21×81 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

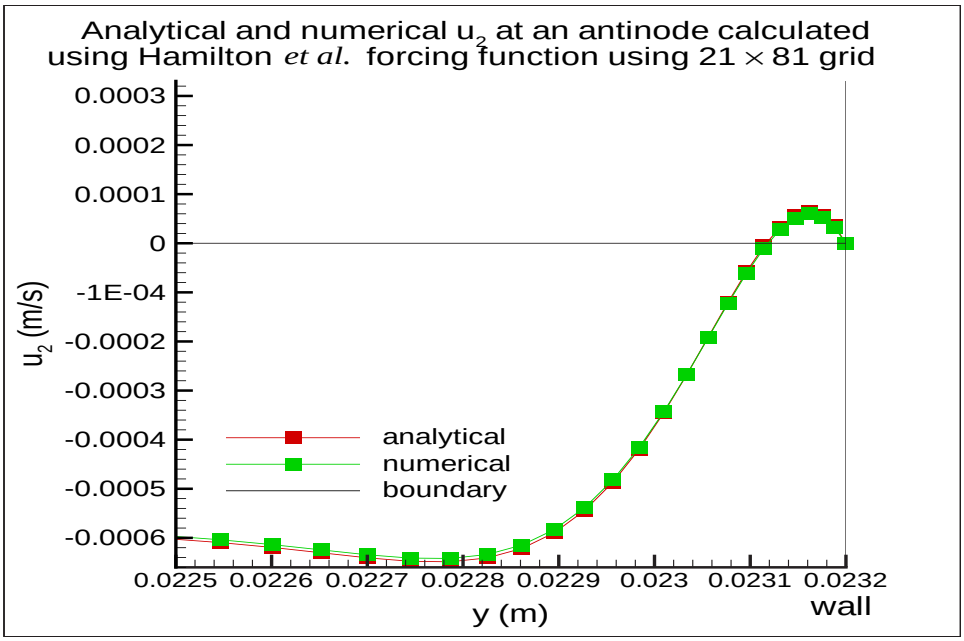

Figure 6.18. Numerical streaming velocity calculated using Hamilton *et al.* forcing function at an antinode along y direction with 21×81 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

6.3.3 Bailliet *et al.*

The numerical streaming velocities calculated using the Bailliet *et al.* forcing function are presented in Figs. 6.19 through 6.24. Figures 6.19 and 6.20 are the results with 21×31 grid points, Figs. 6.21 and 6.22 with 21×61 grid points, and Figs. 6.23 and 6.24 with 21×81 grid points. From Figs. 6.19, 6.21, and 6.23 it can be observed that the numerical streaming velocity is higher than the analytical value. The difference is especially substantial in the boundary layer. On the axis, the difference between the numerical and analytical streaming velocities for 21×31, 21×61, and 21×81 grid points are 1.303×10^{-5} m/s (3.7 %), 1.833×10^{-5} m/s (5.1 %), and 1.896×10^{-5} m/s (5.3 %) respectively. The difference in the streaming velocity amplitude near the boundary (the positive peak) is 2.93×10^{-4} m/s (97 %), 2.88×10^{-4} m/s (94 %), and 2.88×10^{-4} m/s (95 %) for 21×31, 21×61, and 21×81 grid points respectively. For the negative peak, the difference is 2.23×10^{-5} m/s (3.1 %), 4.14×10^{-5} m/s (5.6 %), and 4.26×10^{-5} m/s (5.7 %) for 21×31, 21×61, and 21×81 grid points respectively.

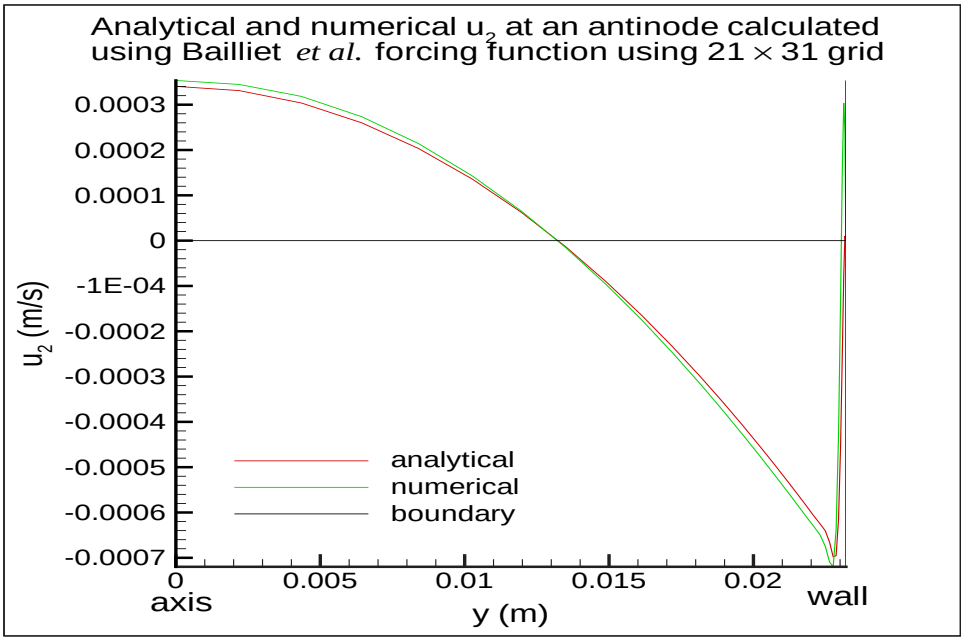

Figure 6.19. Numerical streaming velocity calculated using Bailliet *et al.* forcing function at an antinode along y direction with 21×31 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

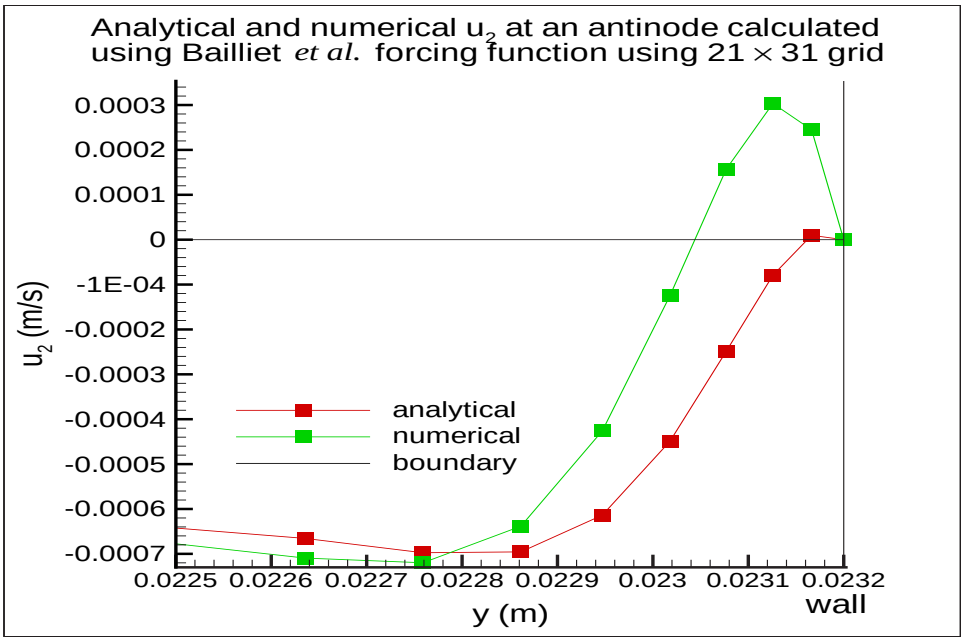

Figure 6.20. Numerical streaming velocity calculated using Bailliet *et al.* forcing function at an antinode along y direction with 21×31 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

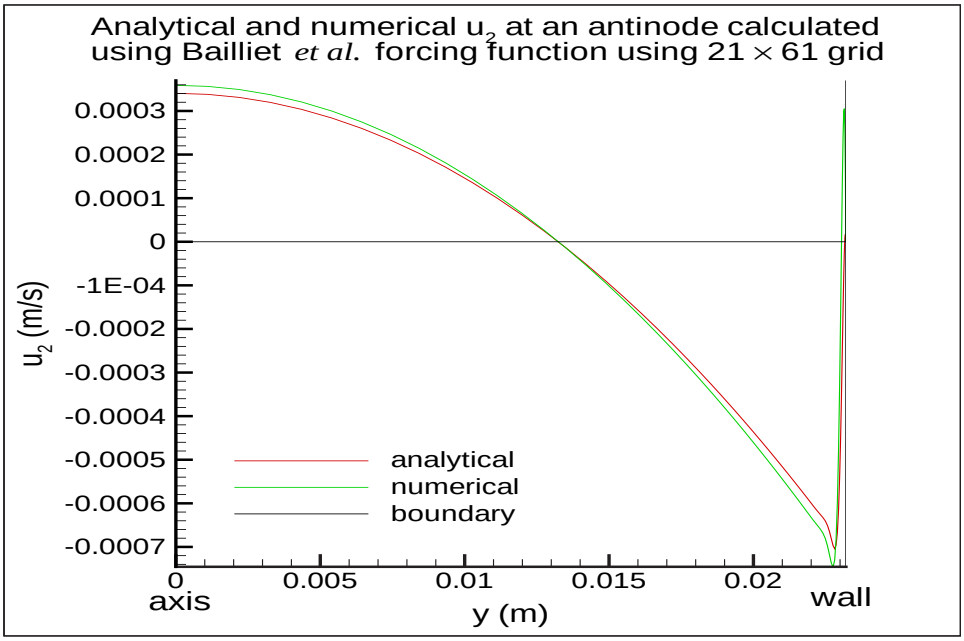

Figure 6.21. Numerical streaming velocity calculated using Bailliet *et al.* forcing function at an antinode along y direction with 21×61 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

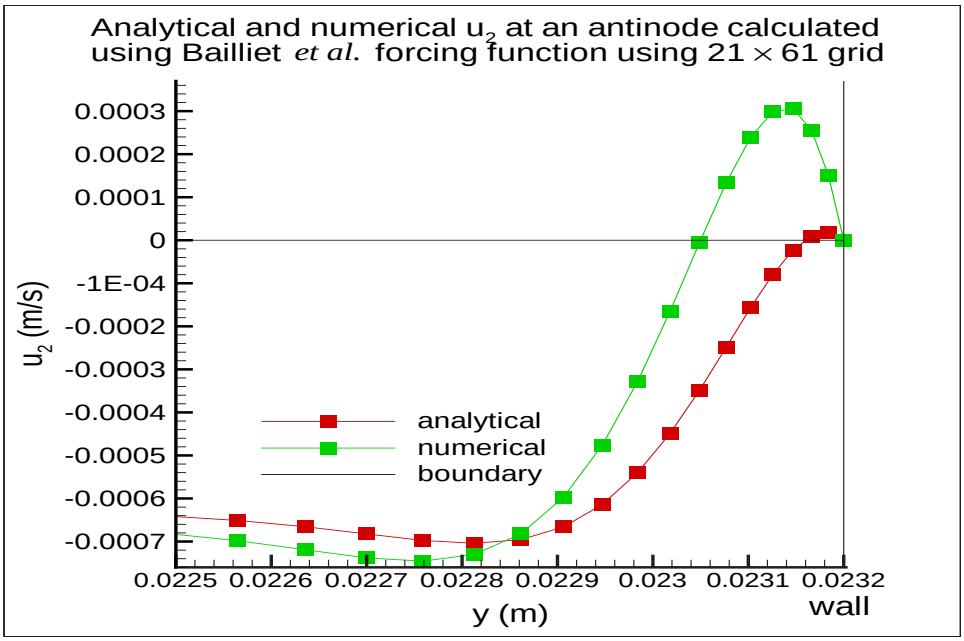

Figure 6.22. Numerical streaming velocity calculated using Bailliet *et al.* forcing function at an antinode along y direction with 21×61 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

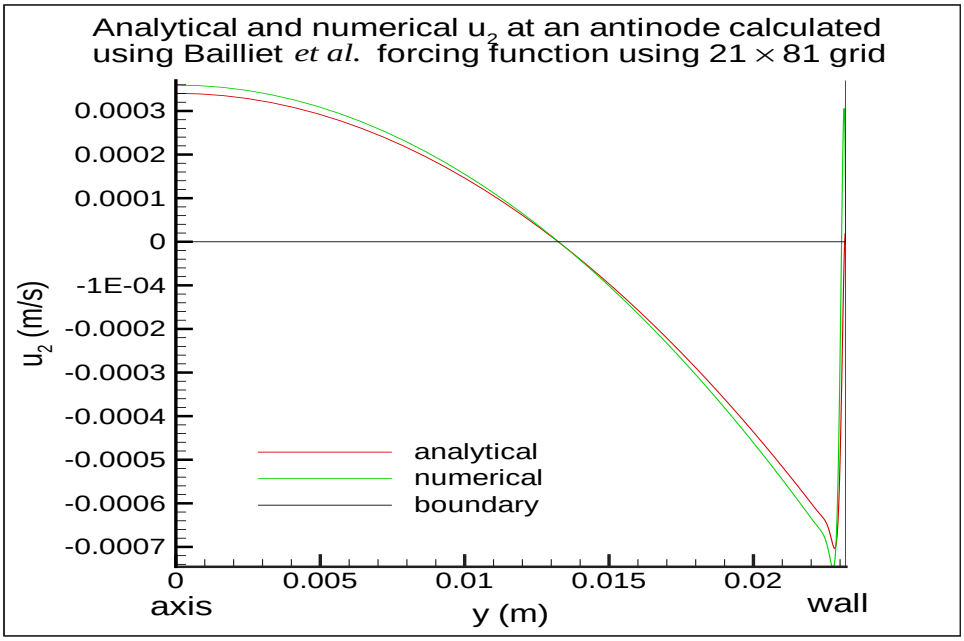

Figure 6.23. Numerical streaming velocity calculated using Bailliet *et al.* forcing function at an antinode along y direction with 21×81 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

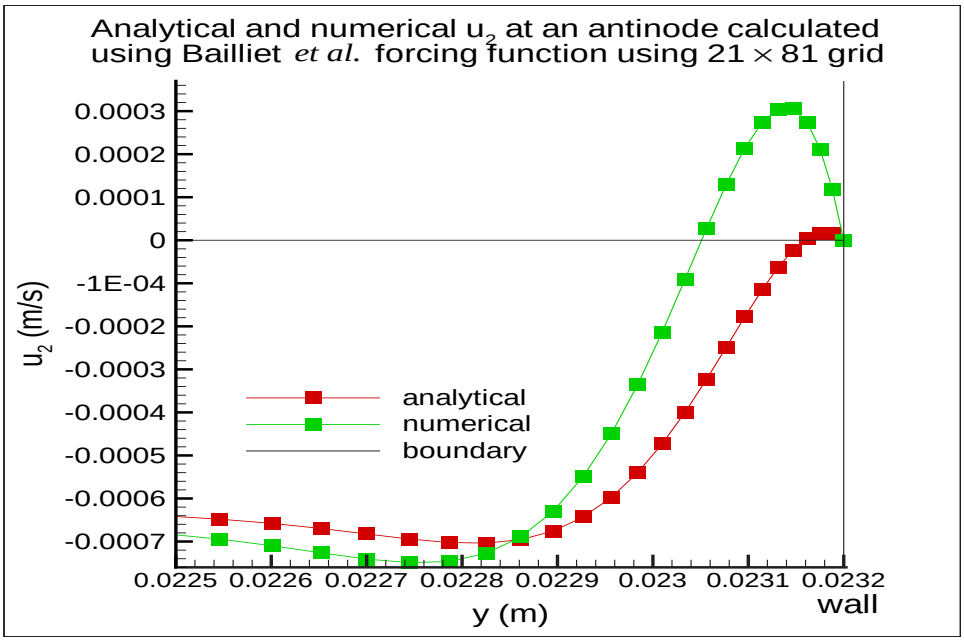

Figure 6.24. Numerical streaming velocity calculated using Bailliet *et al.* forcing function at an antinode along y direction with 21×81 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

6.3.4 Numerical Streaming Velocity Comparison

The numerical streaming velocities calculated using the Rayleigh/Nyborg vs. the Hamilton *et al.* vs. the Bailliet *et al.* forcing functions will be compared in this section. Figures 6.25 and 6.26 show the streaming velocities with the 21×31 grid, Figs. 6.27 and 6.28 show the streaming velocities with the 21×61 grid, and Figs. 6.29 and 6.30 show the streaming velocities with the 21×81 grid. The calculated streaming velocities with the Rayleigh/Nyborg and Hamilton *et al.* forcing functions agree very well with each other. This is true for the streaming away from the boundary as well as near the boundary. It can be observed from those figures that the more points the grid has in the y direction, the better the agreement in the results. The streaming velocity calculated with the Bailliet *et al.* forcing function is significantly higher than the other two group. On the axis it is 3.82×10^{-5} m/s (11%) higher than the Rayleigh/Nyborg and 5.97×10^{-5} m/s (17%) higher than the Hamilton *et al.* velocities (based on the 21×81 grid). Near the boundary the difference is substantially larger, which is about 1×10^{-4} m/s (13%) on the negative peak and 2.45×10^{-4} m/s (80%) on the positive peak (based on the 21×81 grid) higher than the Rayleigh/Nyborg. Compared to the Hamilton *et al.* it is about 1.4×10^{-4} m/s (18%) on the negative peak and 2.5×10^{-4} m/s (81%) on the positive peak (based on the 21×81 grid) higher. The streaming velocities on the axis and near the boundary (positive and negative peaks) are tabulated on Table 6.3 for the 21×31 grid, Table 6.4 for the 21×61 grid, and Table 6.5 for the 21×81 grid.

Research group	On axis u_2 (m/s)	Positive peak near boundary (m/s)	Negative peak near boundary (m/s)
Rayleigh/Nyborg	3.36×10^{-4}	2.84×10^{-5}	-6.9×10^{-4}
Hamilton *et al.*	3.05×10^{-4}	4.1×10^{-5}	-6.3×10^{-4}
Bailliet *et al.*	3.54×10^{-4}	3.04×10^{-4}	-7.2×10^{-4}

Table 6.3. Numerical streaming velocities at various locations along the cross-sectional direction for 21×31 grid points.

Research group	On axis u_2 (m/s)	Positive peak near boundary (m/s)	Negative peak near boundary (m/s)
Rayleigh/Nyborg	3.23×10^{-4}	5.64×10^{-5}	-6.6×10^{-4}
Hamilton *et al.*	3×10^{-4}	5.46×10^{-5}	-6.1×10^{-4}
Bailliet *et al.*	3.59×10^{-4}	3.05×10^{-4}	-7.5×10^{-4}

Table 6.4. Numerical streaming velocities at various locations along the cross-sectional direction for 21×61 grid points.

Research group	On axis u_2 (m/s)	Positive peak near boundary (m/s)	Negative peak near boundary (m/s)
Rayleigh/Nyborg	3.21×10^{-4}	6.02×10^{-5}	-6.5×10^{-4}
Hamilton *et al.*	3×10^{-4}	5.67×10^{-5}	-6.1×10^{-4}
Bailliet *et al.*	3.59×10^{-4}	3.05×10^{-4}	-7.5×10^{-4}

Table 6.5. Numerical streaming velocities at various locations along the cross-sectional direction for 21×81 grid points.

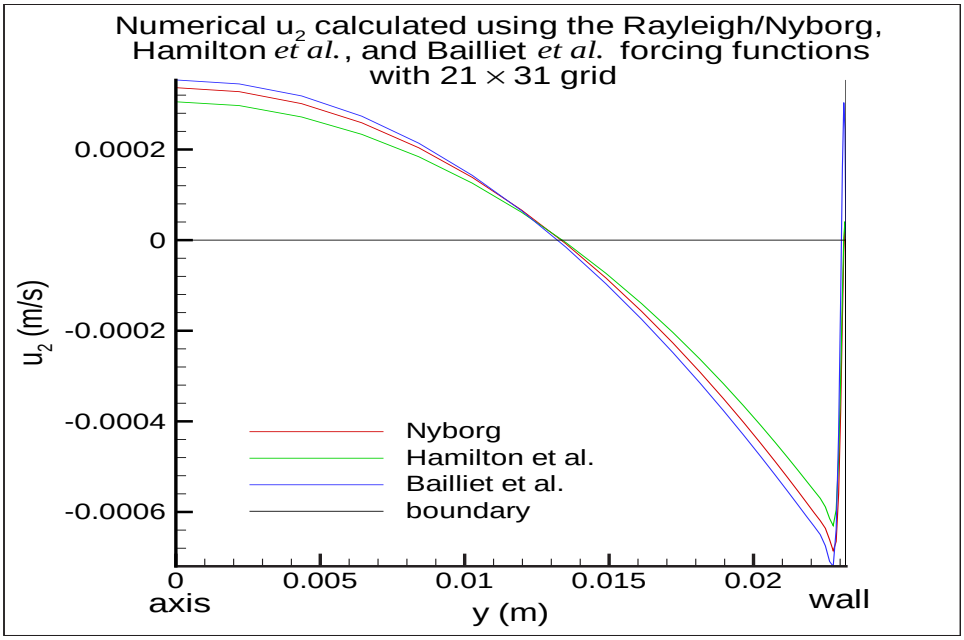

Figure 6.25. Numerical streaming velocities calculated using the Rayleigh/ Nyborg, Hamilton *et al.*, and Bailliet *et al.* forcing functions at an antinode along y direction with 21×31 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

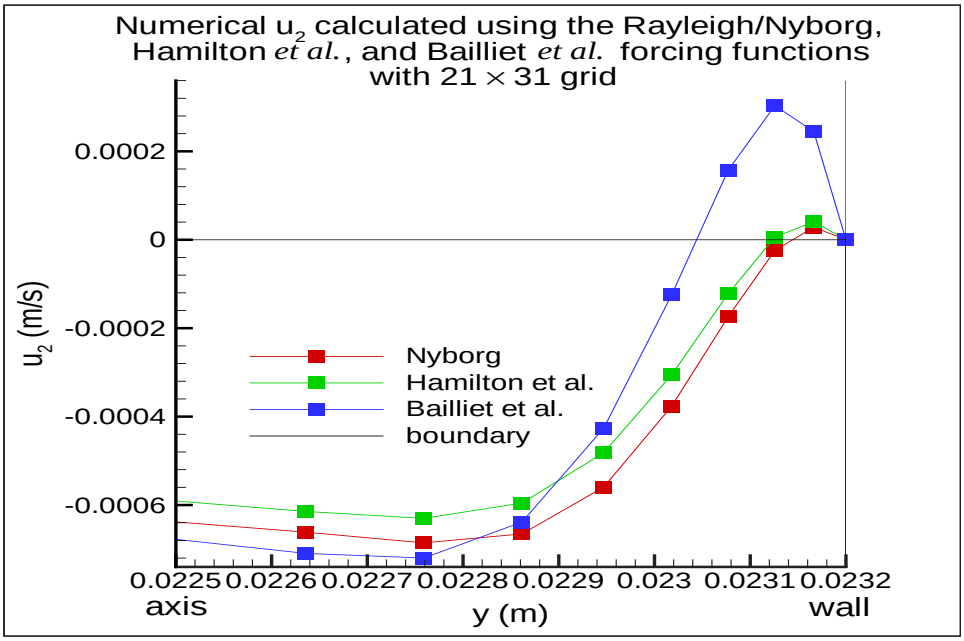

Figure 6.26. Numerical streaming velocities calculated using the Rayleigh/ Nyborg, Hamilton *et al.*, and Bailliet *et al.* forcing functions at an antinode along y direction with 21×31 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

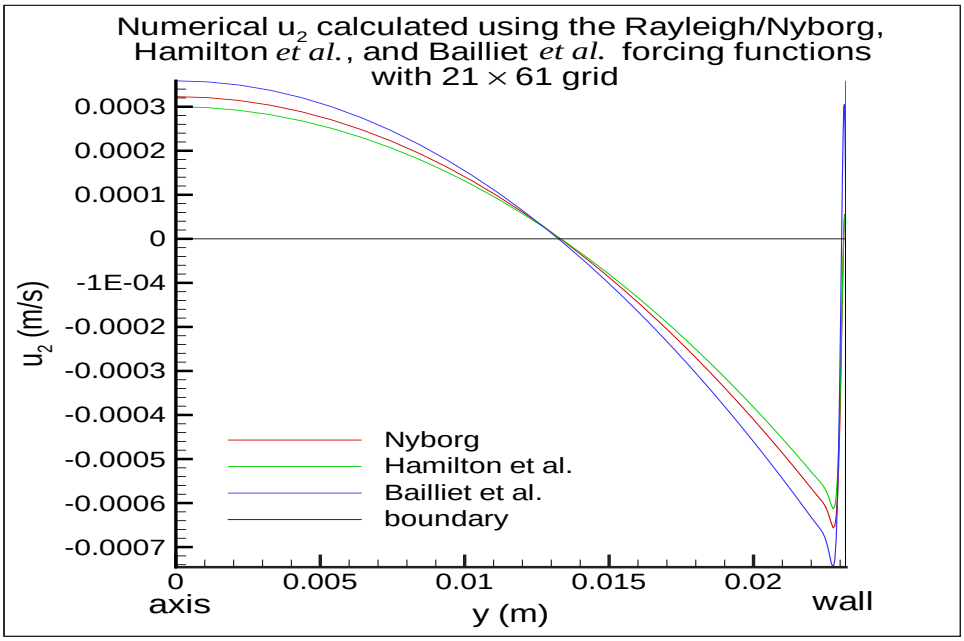

Figure 6.27. Numerical streaming velocities calculated using the Rayleigh/ Nyborg, Hamilton *et al.*, and Bailliet *et al.* forcing functions at an antinode along y direction with 21×61 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

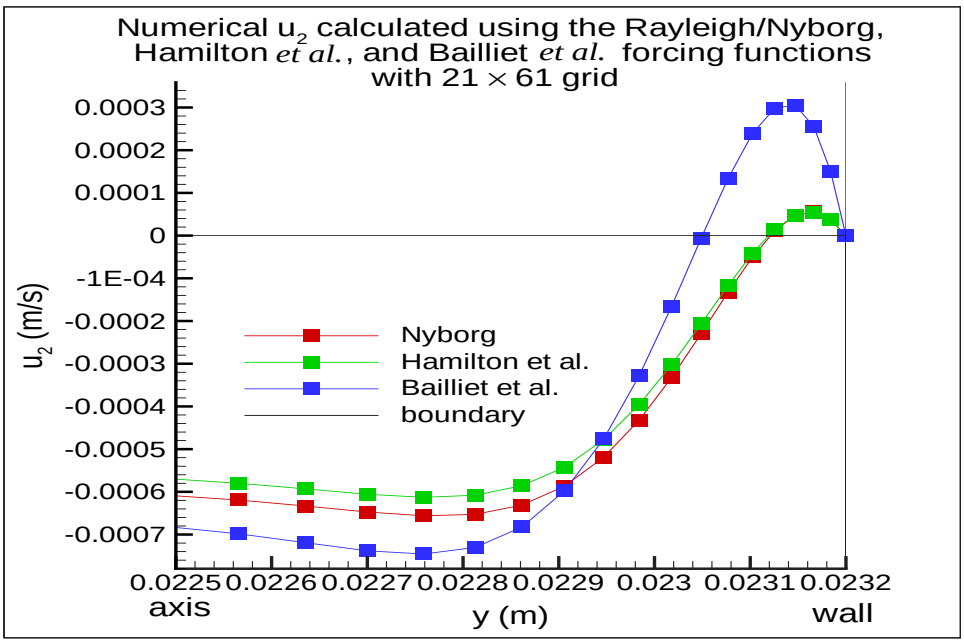

Figure 6.28. Numerical streaming velocities calculated using the Rayleigh/ Nyborg, Hamilton *et al.*, and Bailliet *et al.* forcing functions at an antinode along y direction with 21×61 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

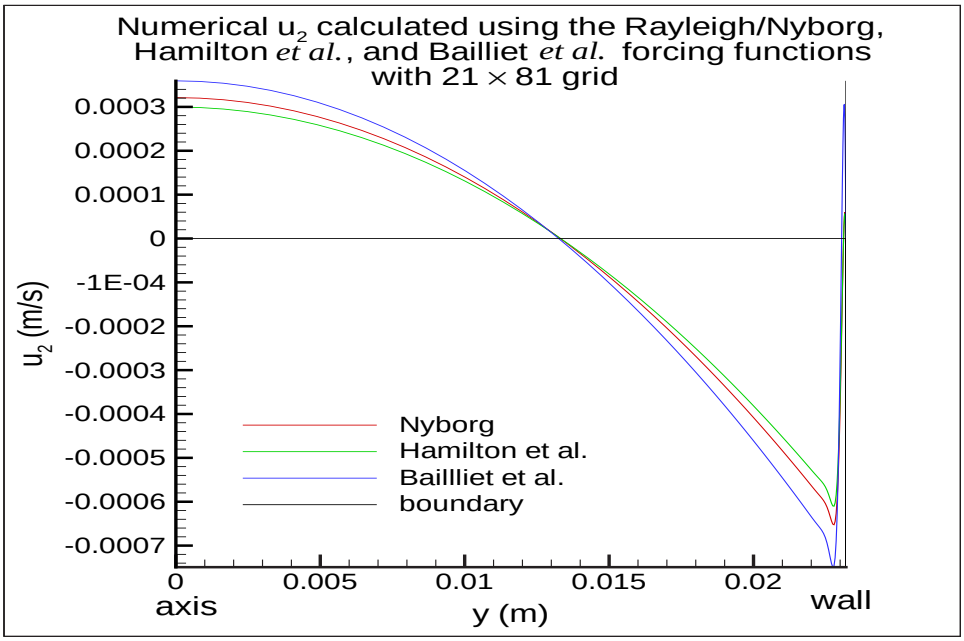

Figure 6.29. Numerical streaming velocities calculated using the Rayleigh/ Nyborg, Hamilton *et al.*, and Bailliet *et al.* forcing functions at an antinode along y direction with 21×81 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

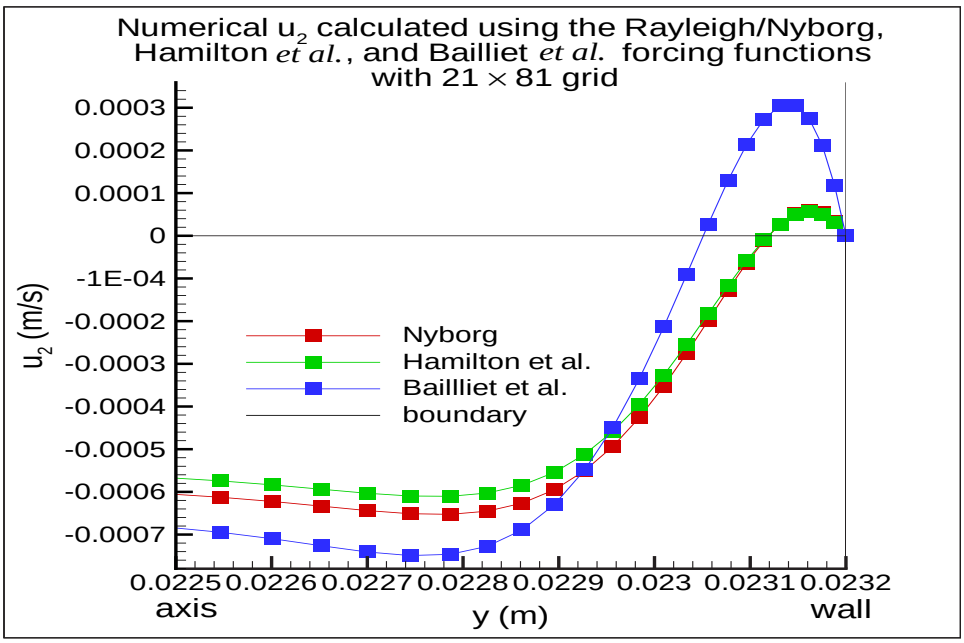

Figure 6.30. Numerical streaming velocities calculated using the Rayleigh/ Nyborg, Hamilton *et al.*, and Bailliet *et al.* forcing functions at an antinode along y direction with 21×81 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

6.4 Analytical vs. Numerical Streaming: Cylindrical Tube Case

6.4.1 Bailliet *et al.* vs. Schuster and Matz

For the cylindrical tube case, Bailliet *et al.* did not present a closed form solution for the streaming velocity in their paper. In another paper, Hamilton *et al.* [12] also presented a solution for streaming in a cylindrical tube. Like Bailliet *et al.*'s solution, Hamilton *et al.*'s solution involves the integration of Bessel functions. As a consequence, both groups did some numerical integrations to approximate the streaming velocity for a cylindrical tube. The streaming velocity in a cylindrical tube solution derived by Hamilton *et al.* [12] is presented in Appendix A. Due to the lack of a closed form solution for streaming in a cylindrical tube, the VTE numerical streaming of Chapter 5 is compared to the analytical streaming in a cylindrical tube derived by Schuster and Matz. The expression for streaming in a cylindrical tube derived by Schuster and Matz [37] is:

$$u_2 = -\left(\frac{3}{8}\frac{u_0^2}{c_0}\right)\left(1 - 2\frac{r^2}{r_0^2}\right)\sin 2kx \tag{6.1}$$

Figures 6.31 through 6.36 show the VTE numerical streaming velocities calculated using the Bailliet *et al.* forcing function for a cylindrical tube and the analytical streaming velocity obtained by Schuster and Matz. On the axis, the numerical streaming velocity is 7.9×10^{-5} m/s (12%) higher than the analytical streaming velocity (based on the 21×81 grid). Near the boundary, Schuster and Matz's calculation does not show the inner streaming unlike the calculated numerical streaming. Schuster and Matz's negative peak value near the boundary is 1×10^{-4} m/s (17%) lower than Bailliet *et al.*'s value (based on the 21×81 grid). The streaming velocities on the axis and the negative peak are tabulated on Table 6.6 for the 21×31 grid, Table 6.7 for the 21×61 grid, and Table 6.8 for the 21×81 grid.

Research group	On axis u_2 (m/s)	Negative peak near boundary (m/s)
Bailliet *et al.*	6.9×10^{-4}	-7×10^{-4}
Schuster and Matz	6.3×10^{-4}	-6.24×10^{-4}

Table 6.6. Bailliet *et al.* numerical vs. Schuster and Matz analytical streaming velocities on the axis and near the boundary for 21×31 grid points.

Research group	On axis u_2 (m/s)	Negative peak near boundary (m/s)
Bailliet *et al.*	7×10^{-4}	-7.3×10^{-4}
Schuster and Matz	6.3×10^{-4}	-6.3×10^{-4}

Table 6.7. Bailliet *et al.* numerical vs. Schuster and Matz analytical streaming velocities on the axis and near the boundary for 21×61 grid points.

Research group	On axis u_2 (m/s)	Negative peak near boundary (m/s)
Bailliet *et al.*	7.4×10^{-4}	-7.7×10^{-4}
Schuster and Matz	6.6×10^{-4}	-6.6×10^{-4}

Table 6.8. Bailliet *et al.* numerical vs. Schuster and Matz analytical streaming velocities on the axis and near the boundary for 21×81 grid points.

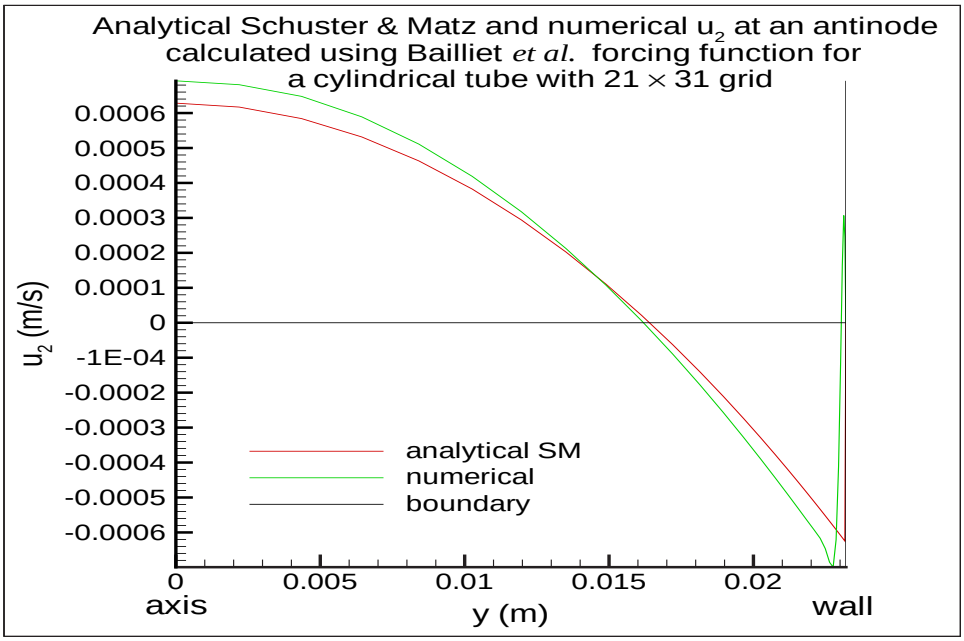

Figure 6.31. Bailliet *et al.* numerical streaming velocity vs. the Schuster and Matz analytical streaming velocity at an antinode along y direction with 21×31 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

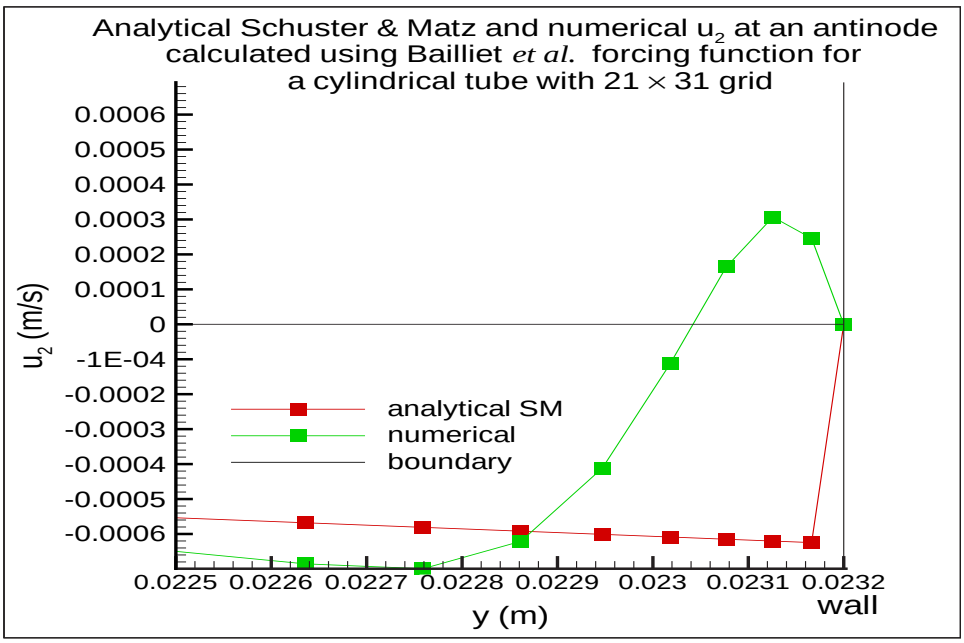

Figure 6.32. Bailliet *et al.* numerical streaming velocity vs. the Schuster and Matz analytical streaming velocity at an antinode along y direction with 21×31 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

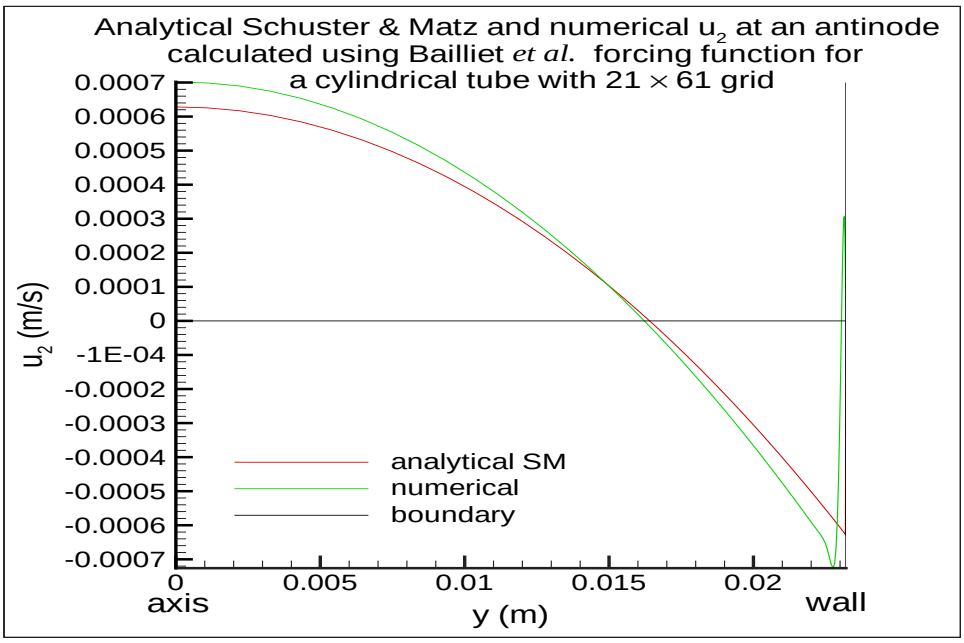

Figure 6.33. Bailliet *et al.* numerical streaming velocity vs. the Schuster and Matz analytical streaming velocity at an antinode along y direction with 21×61 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

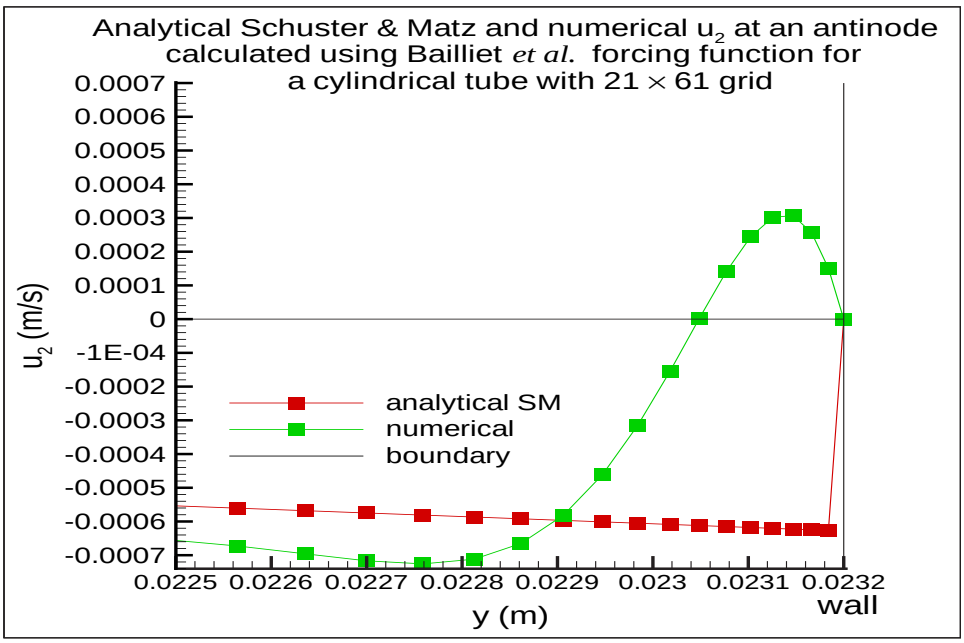

Figure 6.34. Bailliet *et al.* numerical streaming velocity vs. the Schuster and Matz analytical streaming velocity at an antinode along y direction with 21×61 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

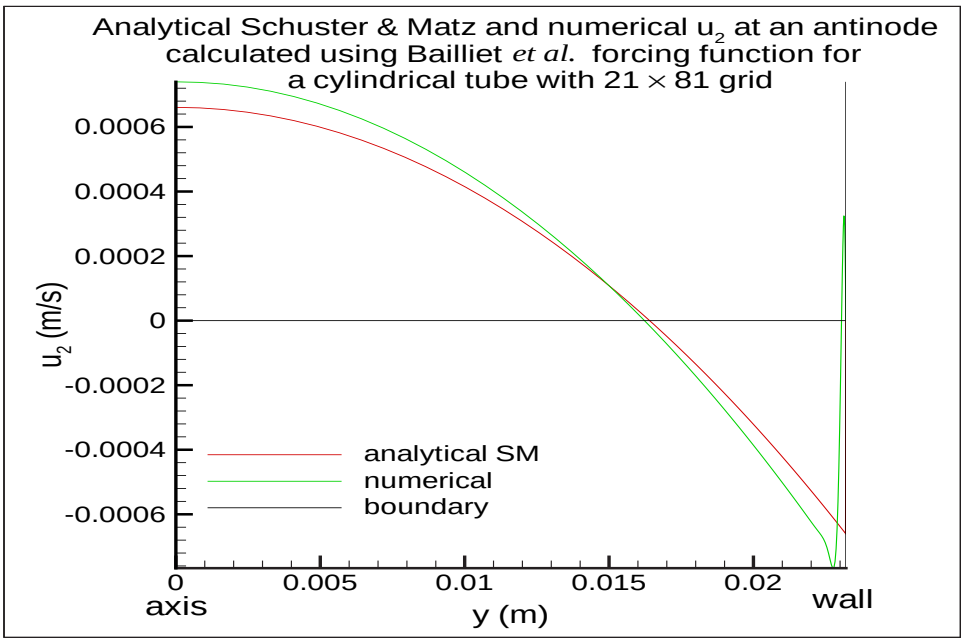

Figure 6.35. Bailliet *et al.* numerical streaming velocity vs. the Schuster and Matz analytical streaming velocity at an antinode along y direction with 21×81 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

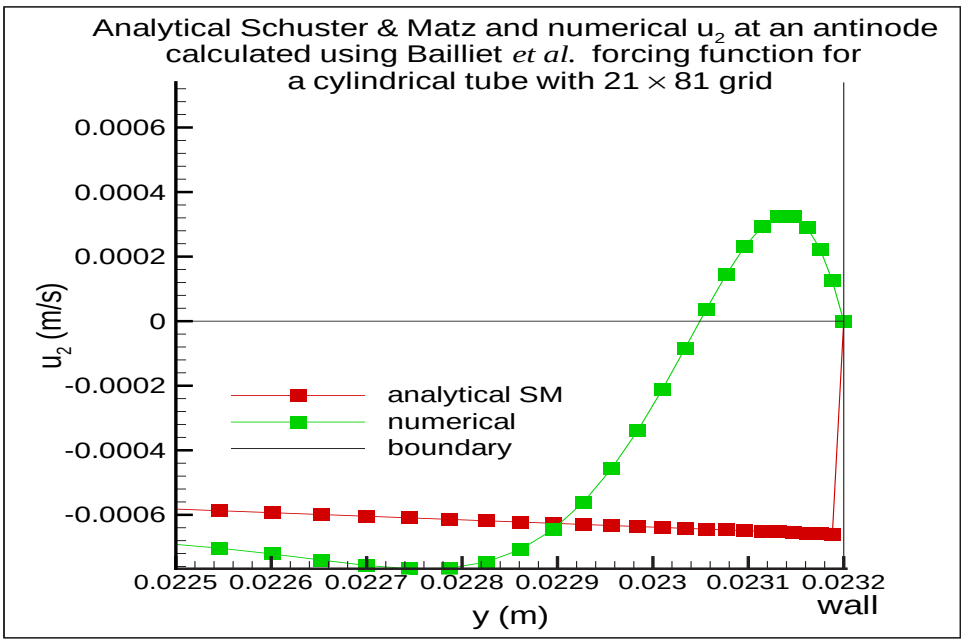

Figure 6.36. Bailliet *et al.* numerical streaming velocity vs. the Schuster & Matz analytical streaming velocity at an antinode along y direction with 21×81 grid points. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

6.5 Acoustic streaming numerical velocities vector plots

Figures 6.37 and 6.38 show the acoustic streaming velocity vector plots from the Rayleigh/Nyborg forcing function calculation. Note that in this section the channel dimension is not to scale. Figures 6.37, 6.39, 6.41, and 6.43 show the upper half of the channel for the entire length (one half of a wavelength) of the channel. Figure 6.38, 6.40, and 6.42, and 6.44 show the region near the top wall where inner streaming occurs in the boundary layer. Figures 6.39 and 6.40 show the acoustic streaming velocities vector plots from the Hamilton *et al.* forcing function calculation and Figs. 6.41 and 6.42 from the Bailliet *et al.* forcing function calculation. Figures 6.37 through 6.42 are for the parallel plate cases. The cylindrical tube cases are shown in Figs. 6.43 and 6.44.

Comparing the inner streaming cells from Figs. 6.38, 6.40 and 6.42, it can be observed that the inner streaming cells are a little bit larger (in the y direction) for the Bailliet *et al.* than Rayleigh/Nyborg and Hamilton *et al.* cases. It appears that the Bailliet *et al.*'s streaming cells are about twice as thick (in the cross sectional direction) as the Rayleigh/Nyborg and Hamilton *et al.*'s. This can be confirmed from Fig. 6.30. The zero crossing of the Bailliet *et al.*'s axial streaming velocity is at about $y = 2.302$ cm from the axis of symmetry, while the zero crossing of Rayleigh/Nyborg and Hamilton *et al.*'s axial streaming velocity is at about $y = 2.312$ cm. The zero crossing marks the change in the direction (or the "turn-around" point) of the streaming velocity. It is the location where the inner streaming ends and the outer streaming begins. Figure 6.44 shows that the cylindrical tube case also has inner streaming cells.

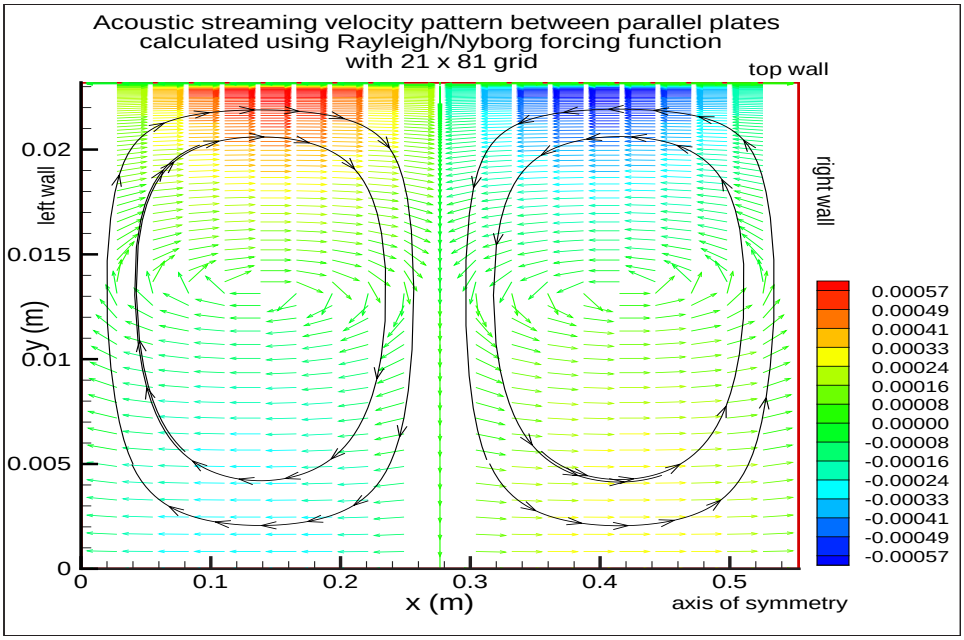

Figure 6.37. Rayleigh/Nyborg streaming cells. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

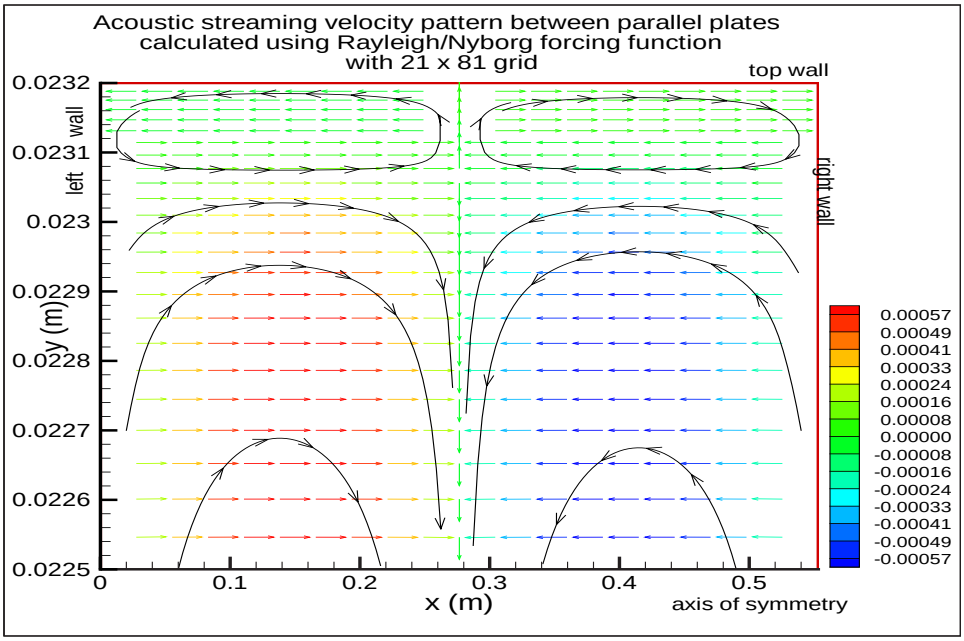

Figure 6.38. Rayleigh/Nyborg streaming cells. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

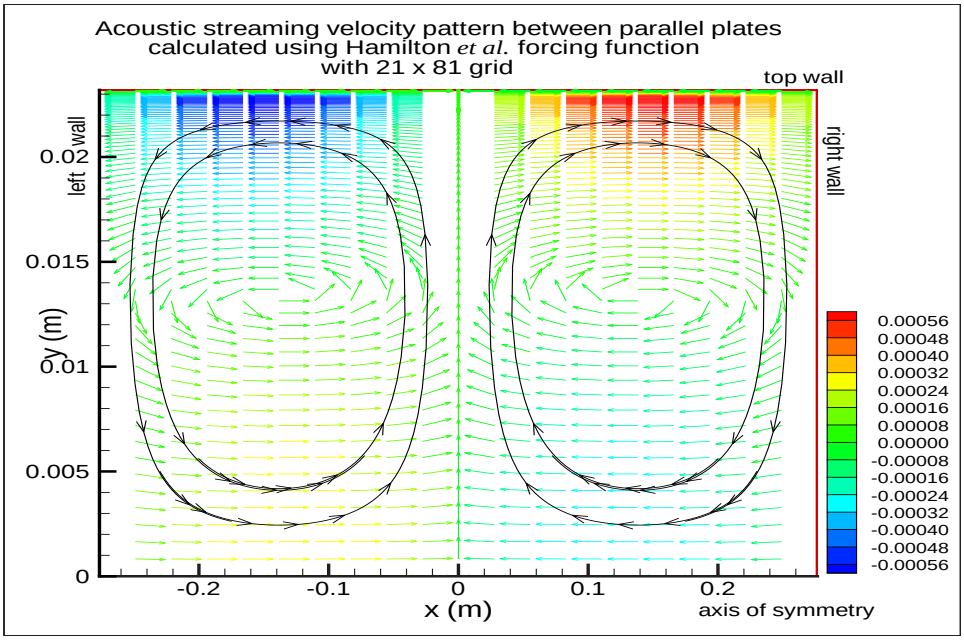

Figure 6.39. Hamilton *et al.* streaming cells. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

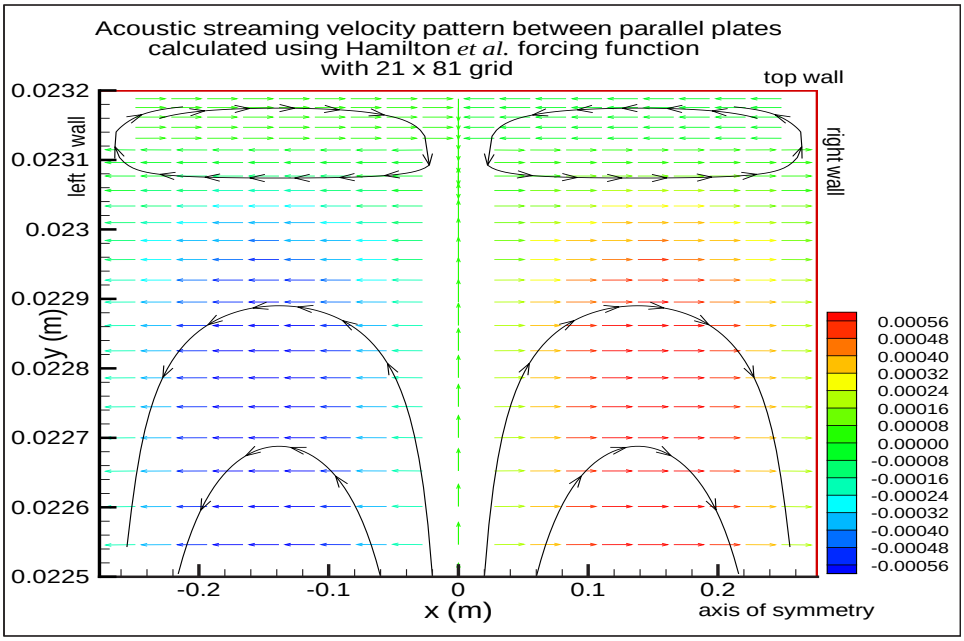

Figure 6.40. Hamilton *et al.* streaming cells. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

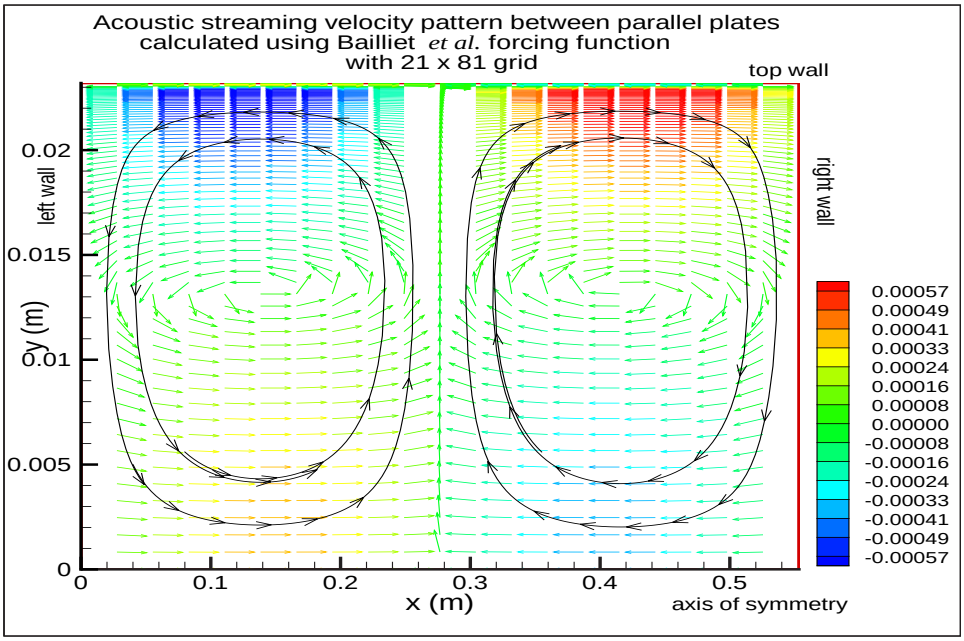

Figure 6.41. Bailliet *et al.* streaming cells in a channel. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

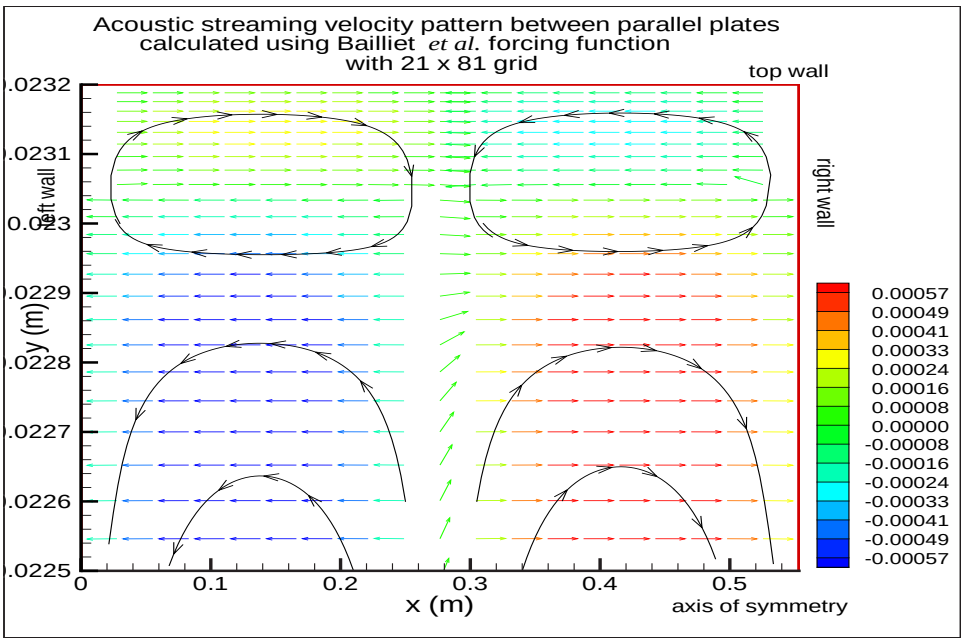

Figure 6.42. Bailliet *et al.* streaming cells in a channel. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

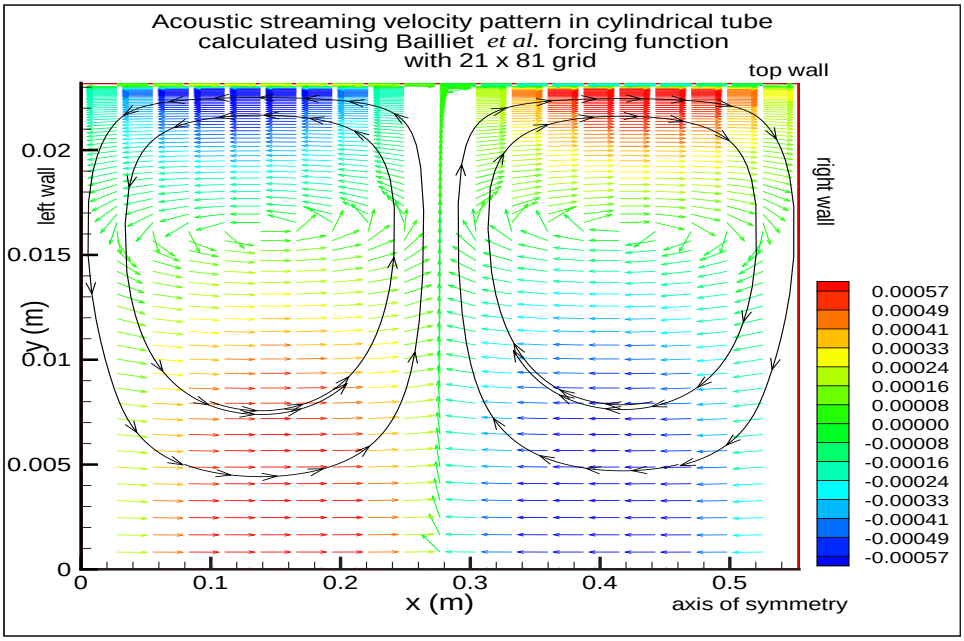

Figure 6.43. Bailliet *et al.* streaming cells in a cylindrical tube. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

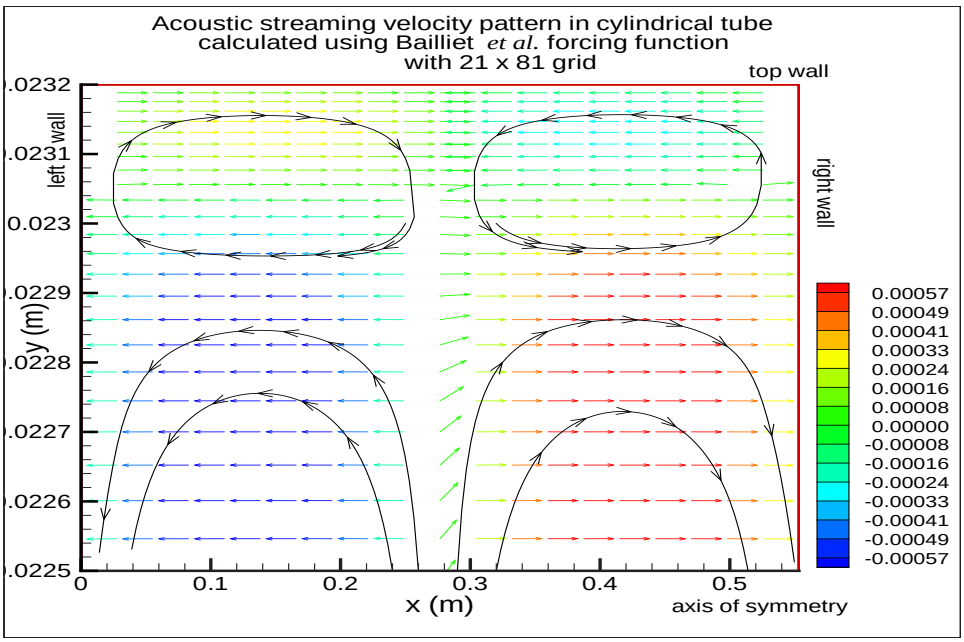

Figure 6.44. Bailliet *et al.* streaming cells in a cylindrical tube. $l = \lambda/2 = 0.553$ m, $r = 0.0232$ m.

6.6 Results Summary

Let us summarize the results presented in the previous sections.

1. **Forcing function**.
   The forcing functions derived by Rayleigh/Nyborg, Hamilton *et al.*, and Bailliet *et al.* are very close to each other in amplitudes with discrepancies that are more pronounced near the boundary.

2. **Analytical streaming velocities**.
   Similar to the forcing functions, the analytical streaming velocities calculated by Rayleigh/Nyborg, Hamilton *et al.*, and Bailliet *et al.* are reasonably close to each other in amplitudes with the difference being more pronounced near the boundary. The analytical streaming velocities calculated by Rayleigh/Nyborg and Hamilton *et al.* are closer to each other compared to Bailliet *et al.*'s streaming velocities.

3. **Analytical vs. numerical streaming velocities**.
   The more points there are near the boundary (in the boundary layer region), the more the numerical results agree with the analytical results. In the case of the calculation using the Rayleigh/Nyborg and Hamilton *et al.* forcing functions, the numerical results agree very well with the analytical results when the finest grid was used. In the case of Bailliet *et al.*, the results agree fairly well except in the region of inner streaming (right next to the boundary). For the cylindrical tube case, the numerical results were compared to the analytical results derived by Schuster and Matz. The discrepancies in the results are considerable although the correct streaming pattern was attained from the numerical calculation. Schuster and Matz's solution does not include the inner streaming cell, therefore no comparison can be made for the inner streaming.

4. **Numerical streaming velocities**.
   The numerical streaming velocities resulting from using the Rayleigh/Nyborg and Hamilton *et al.* forcing functions agree well with each other both in the regions away from and in the vicinity of the boundary. The numerical

streaming velocities calculated using the Bailliet *et al.* forcing function is slightly higher than the other two away from the boundary. However, they are significantly higher in amplitude in the inner streaming region.

5. **Inner and outer streaming**.
   From the streaming velocity vector plots, it can be observed that the inner streaming due to using the Bailliet *et al.* forcing function is about twice as thick in the cross sectional direction as those for the Rayleigh/Nyborg and Hamilton *et al.* forcing functions.

Chapter 7

Conclusions and Future Work Suggestions

7.1 Review and Discussion

Rayleigh/Nyborg, Hamilton *et al.*, and Bailliet *et al.* derived the first order acoustic velocities that have been used to calculate the forcing functions that drive the vorticity transport equation. They all started the calculation with the successive approximation method, but use different assumptions in the calculations. The different assumptions yield similar yet somewhat different forcing functions. The overall features of the forcing functions are alike, but the details are dissimilar. Recall that the acoustic streaming generated by a standing wave is caused by the shear forces in the viscous boundary layer along the boundaries. That is why the forcing function is the greatest in the region near the boundary as was discussed in Chapter 3. This particular feature is very important in the acoustic streaming due to the origin of the streaming itself. The differences in the forcing functions (especially in the boundary layer region) give way to different streaming velocities. It is explicable because as mentioned, the forcing functions are the ones that drive the VTE. Hence different forcing functions will result in different streaming velocities. Another way to look at it is, acoustic streaming is a second order nonlinear effect generated by sound waves. So the first order acoustic velocities characteristics will determine the resulting streaming velocities. Differences in the acoustic velocities can yield dissimilar streaming velocities. This explains why Rayleigh/Nyborg's

streaming velocities are more in agreement with the Hamilton *et al.* streaming velocities than with the Bailliet *et al.* streaming velocities. The Rayleigh/Nyborg forcing function is closer in value to the Hamilton *et al.* forcing function compared to the Bailliet *et al.* forcing function.

In the case of a cylindrical tube, the difference in the streaming velocities calculated using the Bailliet *et al.* and the analytical values derived by Schuster and Matz are more significant than for the parallel plate case. No explanation can be offered at this time on the cause of their discrepancies.

It is noteworthy that the correct vorticity boundary condition is critical for the calculation. Kamakura *et al.* [19] set their vorticity boundary condition to be zero. It is not applicable in this research. Setting the vorticity boundary condition to zero yielded the correct overall streaming pattern but the wrong amplitude (which is several orders of magnitude higher). Care must be taken when setting the vorticity boundary condition as different streaming types may require different boundary conditions in the simulation. Also in Kamakura *et al.*'s calculation, a time-stepping method is used to converge to the solution. It is also not applicable in this research. A direct method is used instead, iterated until the solution converges.

7.2 Conclusions

Typical acoustic streaming calculations using a DNS method require considerable computing time and resources [1, 4]. Although the results are very detailed, the long computing time can sometimes be a problem if results are needed fairly quickly (such as in the early design stage). The current method requires a relatively short computing time and it can be done on a single PC. The finest grid used in this research is 21×81 grid points and it requires about 1 hour and 15 minutes of computing time on a 2 GHz Intel Pentium 4, 512 RAM PC using an Absoft compiler. This reduces the amount of computing time significantly as DNS sometimes requires a few days to finish the calculation on multiple processors. The resulting numerical solutions agreed fairly well with the analytical solutions. This is proof that this method is a valid method to be used as a tool in thermoacoustic design. As long as one has a knowledge of the first order acoustic velocities inside a thermoacoustic device, then this method can be used to predict the Rayleigh streaming

in that device.

The number of grid points near the boundary will determine the accuracy of the resulting streaming velocities. The more grid points there are near the boundary, the better the agreement is between the numerical and analytical results. A balance must be made between the number of grid points (which means larger grid size) and a reasonable computing time. After a certain point, adding more grid points will not affect the numerical results anymore. The assumptions made in deriving the analytical first order velocities are vital. Different assumptions can give different details in the first order velocities. Even little discrepancies in the first order velocities (i.e. the forcing functions) can result in a fairly significant difference in the streaming velocities. When making a comparison, one has to keep in mind all of the assumptions made in order to make a fair comparison.

Besides the forcing function, the other crucial aspect of the calculation is establishing the correct boundary conditions. Different types of streaming may require different types of boundary conditions. For example, a zero vorticity boundary condition worked for the Eckert streaming calculation performed by Kamakura *et al.* But when the same boundary condition was applied to Rayleigh streaming calculation, the results were incorrect in amplitude (although the correct streaming pattern was obtainable).

In a recent paper, Hamilton *et al.* extended their analysis to include the effect of heat conduction and the dependence of the fluid's viscosity on temperature [12]. However they did not include the temperature gradient along the channel walls in their model. They found that for a wide channel the thermal effects change the streaming velocity by only a few percent. Their findings are in agreement with Rott [33]. This means the exclusion of heat conduction and the viscosity dependence on temperature in the current research's model does not undermine the model's validity in making quick estimates of Rayleigh streaming in thermoacoustic devices.

7.3 Future Work

Although analytical results show that heat conduction and the dependence of viscosity on temperature affect the acoustic streaming velocities only by a few percent [12], they did not include the effect of the mean temperature gradient along the

channel wall. It is known and proven through experiment that the temperature gradient along the resonator's inner wall can have significant effect on the acoustic streaming velocities. The streaming velocity profiles depart from the theoretical results for high streaming Reynolds number due to fluid inertia and the temperature gradient [45]. The current method does not provide a way to include the effects of temperature gradient or the fluid inertia. It will be interesting to see how the numerical results vary with the inclusion of the thermal effects (heat conduction, viscosity dependency on temperature, and temperature gradient along resonator's wall) and the fluid inertia.

When one wants to calculate the Rayleigh streaming inside thermoacoustic devices, it is likely that the geometries may not be a straight parallel plate or a cylindrical tube. In such cases, one will need the expression for the first order velocities inside the devices. If no analytical expressions are available, a possible approach is to use acoustics software such as SYSNOISE to numerically calculate the first order velocity and use then use the SYSNOISE results to calculate the forcing function. Modeling the geometries of a thermoacoustic device in SYSNOISE and doing the simulation to obtain the acoustic field inside the device is an interesting project in its own. Then one may be able to predict the behavior of acoustic streaming in any thermoacoustic device which can be modeled.

Another interesting research topic in acoustic streaming is the suppression of Rayleigh streaming by using a small angle tapered tube. The geometry of the tapered tube is relatively simple, and no analytical acoustic velocity expression is available currently. If one is able to model the tapered tube in SYSNOISE and obtain the values of the linear acoustic velocity inside the tube, by using the present method then one can possibly see the effect of the angle of the tube on acoustic streaming suppression.

The comparisons made have all been between the analytical and numerical velocities. It would be beneficial if comparison could be made with experimental data. This is especially crucial when analytical results are not available. The analytical results for streaming velocities are very limited in terms of the type of geometry used in the calculation. Usually the calculations have been performed for a channel or a cylindrical tube. The calculation can be extremely complicated when a more complex geometry is involved. Therefore experimental data will be

very useful in benchmarking the method when it is utilized for geometries other than a channel or a cylindrical tube.

It would also be interesting to see how the finite difference scheme affects the results of the calculation. It may be possible to use other finite difference methods to get a more accurate solution more quickly.

Appendix A

Acoustic Streaming In A Cylindrical Tube

A.1 Hamilton, Illinskii, and Zabolotskaya's solution

The calculation domain used by Hamilton *et al.* [12] to calculate the acoustic streaming in a cylindrical tube is as follows:

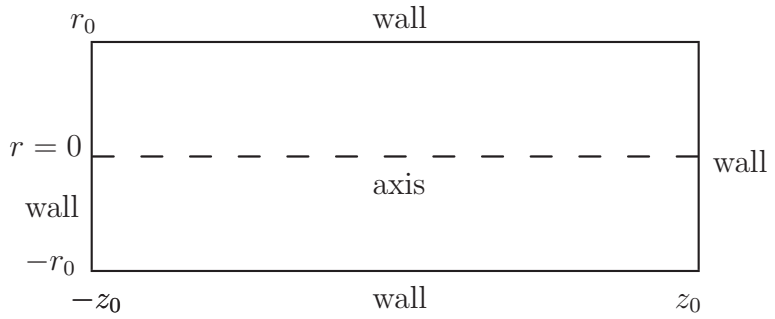

Figure A.1. Hamilton *et al.* [12] calculation domain for streaming in a cylindrical tube.

The x component of the Eulerian streaming velocity, $\bar{u}_z$ is:

$$\bar{u}_z = \bar{u}_z^M - \mathrm{Re}\left\{\frac{d\tilde{u}_{z0}}{dz}\tilde{u}_{z0}^* Q_5(r)\right\} \tag{A.1}$$

where

$$Q_5(r) = -\frac{f_1}{2i\omega} R_\rho(r) R_z^*(r) \tag{A.2}$$

and

$$\begin{aligned}
\tilde{u}_{z0}(z) &= v_0\left(\frac{\cosh \alpha z}{\cosh \alpha z_0} - 1\right) && \text{(A.3)}\\
\alpha^2 &= -\left(\frac{1+(\gamma-1)f_\chi}{1-f_\nu}\right)\frac{\omega^2}{c_0^2} && \text{(A.4)}\\
f_1 &= \frac{1-f_\nu}{1+(\gamma-1)f_\chi} && \text{(A.5)}\\
f_{\nu,\chi} &= \frac{2J_1(\beta_{\nu,\chi} r_0)}{\beta_{\nu,\chi} r_0 J_0(\beta_{\nu,\chi} r_0)} && \text{(A.6)}\\
\beta_{\nu,\chi} &= \frac{i-1}{\delta_{\nu,\chi}} && \text{(A.7)}\\
\delta_{\nu,\chi} &= \sqrt{\frac{2\nu}{\omega}} && \text{(A.8)}\\
\delta_{\nu,\chi} &= \sqrt{\frac{2\chi}{\omega}} && \text{(A.9)}\\
\chi &= \frac{\kappa_0}{\rho_0 c_p} && \text{(A.10)}
\end{aligned}$$

r_0 is the radius of the tube, κ_0 is the equilibrium thermal conductivity (it is 0 when there is no heat conduction), ρ_0 is the equilibrium density, c_p is the specific heat at constant pressure, ν is the equilibrium kinematic viscosity, and γ is the ratio of specific heats.

$\bar{u}_z^M$ is the x component of the mass transport velocity:

$$\bar{u}_z^M = \bar{U}_z^M(z,r) + \frac{2}{r_0}\left[2A_3(z)\frac{r^2}{r_0^2} + A_1(z)\right] \tag{A.11}$$

where

$$\bar{U}_z^M = \mathrm{Re}\left\{\frac{d\tilde{u}_{z0}}{dz}\tilde{u}_{z0}^*[Q_1(r)+Q_2(r)+Q_3(r)+Q_5(r)] + \frac{d\tilde{u}_{z0}}{dz}v_0^* Q_4(r)\right\} \tag{A.12}$$

The functions $A_1(z)$ and $A_3(z)$ are determined by the condition that the motion must vanish at the walls. A_1 and A_3 are:

$$A_1(z) = -2\Psi(z,r_0) + \frac{r_0}{2}\bar{U}_z^M(z,r_0) \tag{A.13}$$

$$A_3(z) = \Psi(z,r_0) - \frac{r_0}{2}\bar{U}_z^M(z,r_0) \tag{A.14}$$

where

$$\Psi(z,r) = \mathrm{Re}\left\{\frac{d\tilde{u}_{z0}}{dz}\tilde{u}_{z0}^*[P_1(r)+P_2(r)+P_3(r)+P_5(r)] + \frac{d\tilde{u}_{z0}}{dz}v_0^*P_4(r)\right\} \tag{A.15}$$

and

$$P_i(r) = \frac{1}{r}\int_0^r Q_i(r')\,r'dr'\,, \qquad 1\le i\le 5 \tag{A.16}$$

$$Q_i(r) = \int_0^r T_i(r')\,dr'\,, \qquad 1\le i\le 4 \tag{A.17}$$

$$T_1(r) = \frac{1}{\nu r}\int_0^r |R_z(r')|^2\,r'\,dr' \tag{A.18}$$

$$T_2(r) = -\frac{r_0 f_\nu}{4\nu}R_r(r)R_z^*(r) \tag{A.19}$$

$$T_3(r) = (\gamma-1)\frac{b\,r_0 f_1 f_\nu^*}{4\nu}R_T(r)\frac{J_1^*(\beta_\nu r)}{J_1^*(\beta_\nu r_0)} \tag{A.20}$$

$$T_4(r) = -(\gamma-1)\frac{f_1}{2\nu r}\int_0^r R_T(r')\,r'dr' \tag{A.21}$$

The functions $R_z(r), R_r(r), R_T(r)$ and $R_\rho(r)$ are:

$$R_z(r) = 1 - \frac{J_0(\beta_\nu r)}{J_0(\beta_\nu r_0)} \tag{A.22}$$

$$R_r(r) = \frac{r}{r_0} - \frac{J_1(\beta_\nu r)}{J_1(\beta_\nu r_0)} + \frac{(\gamma-1)(1-f_\nu)f_\chi}{[1+(\gamma-1)f_\chi]f_\nu}\times\left(\frac{r}{r_0} - \frac{J_1(\beta_\chi r)}{J_1(\beta_\chi r_0)}\right) \tag{A.23}$$

$$R_T(r) = 1 - \frac{J_0(\beta_\chi r)}{J_0(\beta_\chi r_0)} \tag{A.24}$$

$$R_\rho(r) = 1 + (\gamma-1)\frac{J_0(\beta_\chi r)}{J_0(\beta_\chi r_0)} \tag{A.25}$$

A.2 Bailliet, Gusev, Raspet, and Hiller solution

The analysis for a cylindrical tube was performed using polar coordinates, such that $\eta = r/R$, where r is the radial direction and R is the radius of the tube. The calculation domain used by Bailliet *et al.* [3] to calculate acoustic streaming in a cylindrical tube is as follows:

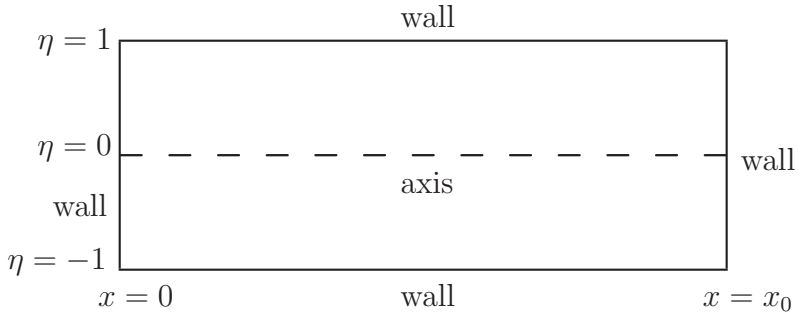

Figure A.2. Bailliet *et al.* calculation domain for streaming in a cylindrical tube

The x component of the streaming velocity is:

$$\begin{aligned}\langle u_2 \rangle =& \frac{4}{\rho_0}\left(\eta^2-1\right)\int_0^1 \langle \rho_1 u_{1x}\rangle\, \eta\, d\eta \\ &+ \frac{4R^2}{\mu_0}(\eta^2-1)\times\int_0^1 \eta\int_{-1}^{\eta}\frac{1}{\eta'}\int_0^{\eta'}\partial_x(\rho_0\left\langle u_{1x}^2\right\rangle)\eta''d\eta\, d\eta' d\eta'' \\ &+ \frac{4R}{\nu_0}(\eta^2-1)\int_0^1 \eta\int_{-1}^{\eta}\langle u_{1x}u_{1\eta}\rangle\, d\eta\, d\eta' \\ &- \frac{4\beta}{T_0}(\eta^2-1)\int_0^1\int_{-1}^{\eta}\langle T_1\partial_\eta u_{1x}\rangle\, d\eta\, d\eta' \\ &+ \frac{R^2}{\mu_0}\int_{-1}^{\eta}\frac{1}{\eta'}\int_0^{\eta'}\partial_x(\rho_0\left\langle u_{1x}^2\right\rangle)\eta''d\eta' d\eta'' \\ &+ \frac{R}{\nu_0}\int_{-1}^{\eta}\langle u_{1x}u_{1\eta}\rangle\, d\eta' - \frac{\beta}{T_0}\int_{-1}^{\eta}\langle T_1\partial_\eta u_{1x}\rangle\, d\eta'\end{aligned} \tag{A.26}$$

The first order axial velocity u_{1x} is:

$$u_{1x} = \frac{\partial_x p_1}{i\omega\rho_0} F_\eta \tag{A.27}$$

where the acoustic pressure varies along x:

$$p_1(x) = \frac{p_1(0)}{2}\left(e^{-ikx} + e^{ikx}\right) \tag{A.28}$$

$p_1(0)$ is the rms first order pressure. ω is the radial frequency, c_0 is the speed of sound, and γ is the ratio of specific heat. The complex wave number $k = \omega/c_0\sqrt{[\gamma-(\gamma-1)F_t]/F}$. The coefficients that describe the viscous and thermal boundary layer effects are:

$$F_\eta = 1 - \frac{J_0(b\eta)}{J_0(b)} \tag{A.29}$$

$$b = \frac{(1+i)R}{\delta_\nu}, \qquad \delta_\nu = \frac{2\mu}{\omega\rho} \tag{A.30}$$

$$\rho_1 = \frac{\gamma-(\gamma-1)F_{\eta t}}{c_0^2} p_1 - \frac{\partial_x T_0}{T_0\omega^2}\partial_x p_1 \left[\frac{\sigma F_\eta - F_{\eta t}}{\sigma - 1}\right] \tag{A.31}$$

$$T_1 = \frac{\gamma-1}{\rho_0 c_0^2} T_0 F_{\eta t} p_1 - \frac{\partial_x T_0}{\rho_0\omega^2}\partial_x p_1 \left[\frac{\sigma F_\eta - F_{\eta t}}{\sigma - 1}\right] \tag{A.32}$$

$$F_{\eta t} = 1 - \frac{J_0(\sqrt{\sigma}b\eta)}{J_0(\sqrt{\sigma}b)} \tag{A.33}$$

$$\phi = \frac{J_1(b\eta)}{bJ_0(b)}, \qquad \phi_t = \frac{J_1(\sqrt{\sigma}b\eta)}{\sqrt{\sigma}bJ_0(\sqrt{\sigma}b)} \tag{A.34}$$

$$F = 1 - \frac{2J_1(b)}{bJ_0(b)}, \qquad F_t = 1 - \frac{2J_1(\sqrt{\sigma}b)}{\sqrt{\sigma}bJ_0(\sqrt{\sigma}b)} \tag{A.35}$$

μ is the shear viscosity, ρ is the fluid density, σ is the Prandtl number, T_0 is the equilibrium temperature in the absence of oscillation, and β is the coefficient that expresses the viscosity and heat conduction as functions of the mean temperature of the fluid approximated around a reference temperature.

The first order radial velocity $u_{1\eta}$ is:

$$\begin{aligned} u_{1\eta} =& \frac{iR}{\omega\rho_0}\left[\frac{\partial^2 p_1}{\partial x^2}\left(\frac{\eta}{2}-\phi\right)+\frac{\partial_x T_0}{T_0}\partial_x p_1\left\{\frac{\eta}{2}+\frac{-\sigma\phi+\phi_t}{\sigma-1}\right\}\right. \\ &+\frac{\omega^2}{c_0^2}p_1\left(\frac{\eta}{2}+(\gamma-1)\phi_t\right)+\partial_x p_1\frac{\beta+1}{2}\frac{\partial_x T_0}{T_0} \\ &\left.\times\left\{\eta\left(1-F_\eta\right)-\phi+\frac{\phi b^2(1-F)}{2}\right\}\right] \end{aligned} \tag{A.36}$$

Appendix B

Running The Code

The program is structured in a way that the main program calls the subroutines where the codes for the calculation are written and a module is used to define the global variables. In the following sections the module, main program and subroutines will be listed (in the order of their calling by the main program).

1. **Main program**
   In the main program the frequency, wavelength, radial frequency, period, wave number, viscous penetration depth, tube length, tube height and acoustic velocity are defined. These variables are defined in the main program because given a certain frequency then the other variables can be calculated. Therefore, to change the model, according to the thermoacoustic working frequency, one only has to change the value of the frequency in the main program. The other variables will be changed accordingly.

2. **Make the grid**
   The first thing that needs to be done after defining the physical variables is to make the grid for the calculation. First a uniform grid is generated, then the physical stretched grid is generated. After that, the uniform computational grid is generated.

3. **Define the metrics**
   The metrics of transformation are calculated in this subroutine.

4. **Calculate analytical streaming velocity**

Having defined the physical variables and grids, the analytical streaming velocity can then be calculated in this subroutine.

5. **Calculate dummy variables**
   Some dummy variables are calculated in this subroutine. This is done so that the variables do not have to be calculated inside the loops. This saves a few calculation operations. This is especially important if many grid points are used in the calculation. Defining the dummy variables can potentially save some calculation time.

6. **Calculate the forcing function**
   Before the iteration step is performed, the forcing function is calculated using the first order axial and tangential (or radial) acoustic velocity in this subroutine.

7. **Calculate coefficients of vorticity solution**
   The coefficients of the vorticity solution as defined in Eq. 5.56 are calculated in this subroutine.

8. **Setting up the stiffness matrix**
   The stiffness matrix for the coefficients of the vorticity solution at the inner points is then set up in this subroutine. The right hand side vector containing the forcing function is also set up in this subroutine.

9. **Setting up the boundary condition for the vorticity**
   This subroutine sets up the boundary condition for the vorticity solution.

10. **Banded matrix decomposition**
    The banded matrix can now be decomposed into its upper and lower matrices.

11. **Backsubstitution process**
    The solution is found through the backsubstitution process.

12. **Put results in an array**
    The results from the backsubstitution process (the vorticity) are put in a vector. To be used in the next step as the forcing function of the VTE, they

need to be put back in an array form. This subroutine converts the vector form into the array form.

13. **Calculate coefficients of the stream function solution**
    The coefficients of the stream function solution as defined in Eq. 5.64 are calculated in this subroutine.

14. **Setting up the boundary condition for the stream function**
    This subroutine sets up the boundary conditions for the stream function solution.

15. **Calculate acoustic streaming velocity**
    Once the stream function is computed, the acoustic streaming velocity can be calculated in this subroutine.

16. **Saving data to files**
    The data can be saved every few thousands iterations or as desired. The data is saved in a format that can be read by Tecplot. The vorticity and stream function are also saved so in case the calculations process is interrupted, there will be restart files available.

The two subroutines to do the banded matrix decomposition and backsubstitution (subroutine BANDEC and BANBKS in appendix B) were taken from the corresponding subroutine of Numerical Recipes [29] and were modified for use in the present work.

Appendix C

Acoustic Streaming Calculation Codes

C.1 Main Program

```
PROGRAM MAIN_TIME_MARCHING

   USE PHYSICAL_PARAMETERS
   USE ARRAYSIZEINFO
   USE COMPUTATIONAL_PARAMETERS
   USE COEF_FOR_MATRIX
   USE STIFFNESS_MATRIXandRHS_VECTOR
   USE WRITING_TO_FILES

IMPLICIT NONE

   Integer i, j, rownumber, m1, m2, nooftime
   Integer iter_chi

   Real calc_time1, calc_time2, myseconds

   Double precision distfromwall
   Double precision percentdiff
   Double precision integral_result

   Real*8, dimension (array_size_x, array_size_y) :: residue_diff

   f = 310.              ! frequency = 310 Hz.
                         ! This is to match Mike Thompson's experiment
   lambda = c_0/310.     ! wavelength
   period = 1./f
   omegafreq = 2.0*pi*f  ! radial frequency (rad/sec)
   k = omegafreq/c_0     ! swave number
```

```
print*, "f= ",f, ",period=",period
print*, "omega= ",omegafreq
print*, "k= ",k

nu = mu/rho_0         ! kinematic viscosity
print*, "nu= ",nu

betha_sqr = (omegafreq * rho_0) / (2.0 * mu)
betha = sqrt(betha_sqr)
acboundlayerthick = 1.0/betha    ! viscous penetration depth
print*, "acboundary layer thickness= ",acboundlayerthick

!--- thermal penetration depth ---!
thermalpendepth = acboundlayerthick/sqrt(prandtl)
print*,"thermal penetration depth = ",thermalpendepth

tube_length = 0.0   ! initialize variable tube_length for the channel length
tube_height = 0.0   ! initialize variable tube_height for the channel height

tube_length = lambda/2.0   ! length of channel is half a wavelength
tube_height = 0.0232       ! radius = 2.32 cm.
                           ! This parameter is from
                           ! Mike Thompson's experiment
print*,"tube length = ", tube_length
print*,"tube height = ", tube_height
print*,"It is", ceiling(tube_height/acboundlayerthick), "times bl thickness"

!*************************************************************************!
! Calculate u_0 from nonlinear Reynolds number                            !
! Nonlinear Re no = (U1^2 * R^2 * rho0 * omega) / (c_0^2 * mu)            !
! So u_0 = sqrt( (Re_nl * c_0^2 * mu)/(R^2 * rho0 * omega)                !
! Mike Thompson's use Re_nl = 0.36, 5.7, and 19                           !
!*************************************************************************!
u_0 = sqrt( (Re_nl*(c_0**2)*mu)/((tube_height**2)*rho_0*omegafreq) )
                          ! u_0 here is the peak acs velocity amplitude
                          ! at the antinode
Ppeak = u_0*rho_0*c_0     ! Peak pressure
Mach = u_0 / c_0          ! Mach number

print*,"Nonlinear Reynold's number = ",RE_nl
print*,"u_0=",u_0, "m/s,  u(rms) = ", u_0/sqrt(2.0), "m/s"
print*,"Ppeak=",Ppeak,
print*,"Pa or in SPL =",20 * log10((Ppeak/sqrt(2.))/Pref),"dB"
print*,"Mach number = ",Mach

l = tube_length   ! put tube_length variable into l for later use
h = tube_height   ! put tube_height variable into h for later use

!--- nondimensional Reynold's number ---!
Re_nondim = (rho_0*u_0*tube_length)/(mu)
```

```
   print*,"Nondimensional Reynold's number = ",Re_nondim

   !--- Make the physical and computational grid for the calculation ---!
   print*,"Calling subroutine MAKE_GRID"
   call MAKE_GRID

   !--- Calculate the metrics of transformation ---!
   print*,"Calling subroutine DEFINE_METRICS"
   call DEFINE_METRICS

   !--- Calculate the analytical streaming velocity ---!
   print*,"Calling subroutine CALCULATE_VEL_ANALYTIC"
   call CALCULATE_VEL_ANALYTIC

   !--- Calculate some dummy variables to save some operations ----!
   print*,"Calling subroutine CALCULATE_SPEEDUP_CONSTANTS"
   call CALCULATE_SPEEDUP_CONSTANTS

!----------------------------------------------------------------------!
! Initialize variables
!----------------------------------------------------------------------!
   Omega = 0.
   Omega_old = 0.
   Chi = 0.
   Chi_old = 0.

   forcing_function = 0.

   m1 = array_size_x   ! subdiagonal bandwidth
   m2 = array_size_x   ! superdiagonal bandwidth

!**********************************************!
! Start iterations                             !
!**********************************************!

    ! --- Initialize ---!
    iteration_no = 0
    time_print = 0
    tau  = 0.0
    tau_start = 0.0
    tau_end = 0.0
    cycles_tobe_calc = 0.0
    cycle_no = 0.0
    saveeveryNperiod = 0.0
    total_for_averaging = 0
    nooftime = 0

    calc_time1 = 0.0
    calc_time2 = 0.0

   !--- It was found that solution already converges at 60,000 iterations ---!
```

```
   total_no_iteration = 60000
   print*," Total number of iteration for the program: ", total_no_iteration

   !--- define how often to save the data ---!
   saveeveryiteration = 4000  ! save every 4,000 iterations
   print*,"save every", saveeveryiteration, "iteration"

   !--- calculate the number of files for the data ---!
   nooftime = total_no_iteration/saveeveryiteration
   print*,"No of files to be written",nooftime

   !--- Calculate the forcing function from 1st order velocity ---!
   call CALCULATE_FORCE_RAYLEIGH

   Ux = 0.
   Uy = 0.

   !---- Use this when restarting the computation ----!
   !call CHIOMEGA_RESTART_FILE(Omega_old, Chi)
   !Chi_old = Chi

time_marching_1 : do tau_iter =1, total_no_iteration

   iteration_no = iteration_no + 1
   cycle_no = (iteration_no*delta_tau)/period
   modulo = mod(iteration_no,saveeveryiteration)
   mod_int = nint(modulo)

   tau = (delta_tau*(iteration_no-1))

   call OMEGA_COEFF_SETUP         ! set up the coefficients of vorticity

   call INNERPOINTS_MATRIX_SETUP(a_coef_omega, b_coef_omega, c_coef_omega, &
       d_coef_omega,e_coef_omega, forcing_function, array_size_x, array_size_y)
                                  ! set up the stiff matrix

   call OMEGA_BOUNDARY_CONDITION  ! set up the coefficients of vorticity

   lowertriang = 0.0              ! initialize

   call BANDEC(array_size_x,array_size_y)   ! banded matrix decomposition

   call BANBKS(array_size_x,array_size_y)   ! backsubstitution process

   !--- The rhs_vector now holds the solution to Omega. ---!
   !--- Put it back in the original form (array) --- !
   call RESULTINVECTOR_TO_ARRAY(rhs_vector, Omega_old, array_size_x, &
                                array_size_y)

   !--- Calculate Chi first ---!
   call CHI_COEFF_SETUP   ! set up the coefficients of stream function
```

```
! --- (min forcing_function because ...=-omega) is the forcing   ---!
! --- function now. The Stiffmatrix & rhs_vector will be cleared ---!
! --- inside the INNERPOINTS_MATRIX_SETUP                        ---!
call INNERPOINTS_MATRIX_SETUP(a_coef_chi, b_coef_chi, c_coef_chi, &
     d_coef_chi,e_coef_chi, -forcing_function, array_size_x, array_size_y)

call CHI_BOUNDARY_CONDITION   ! set up the coefficients of stream function

lowertriang = 0.0             ! initialize

call BANDEC(array_size_x,array_size_y)   ! banded matrix decomposition

call BANBKS(array_size_x,array_size_y)   ! backsubstitution process

!--- The rhs_vector now holds the solution to Chi. ---!
!--- Put it back in the original form (array)  ---!
call RESULTINVECTOR_TO_ARRAY(rhs_vector, Chi, array_size_x, array_size_y)

!======= Under Relaxation Process. Have to be done for convergence =======!
Chi_new = 1.e-4*Chi + (1.-1.e-4)*Chi_old
          ! Put the "under-relaxed" value of Chi at Chi_new
          ! Chosen relaxation factor is 1e-4. Too little value will
          ! result in a too long of computation. Too large of a value
          ! will cause instability

Chi_old = Chi_new  ! Now put the under-relaxed value into the old variable
                   ! for the use in next iteration
Chi = Chi_new

!========== Calculate the numerical acoustic streaming velocity ==========!
call CALCULATE_VELOCITY

!--- if the remainder = 0 then print ---!
if ( (iteration_no>(saveeveryiteration-1)) .AND. (mod_int == 0) ) then

    time_print = time_print + 1
    print*,"printing at cycle no", cycle_no," time_print= ", time_print
    print*,"Max streaming velocity = ",maxval(Ux), "m/s at",maxloc(Ux)

    if (time_print < 10) then
       write(file_ext,'(i1,a)') time_print,'.dat'
    else if (10 <= time_print .AND. time_print < 100) then
       write(file_ext,'(i2,a)') time_print,'.dat'
    else if (100 <= time_print .AND. time_print < 1000) then
       write(file_ext,'(i3,a)') time_print,'.dat'
    end if

    myfile='Ux_100dB'//file_ext
    call tecout(Ux, x, y, array_size_x, array_size_y, myfile)
```

```
        myfile='Fx'//file_ext
        call tecout(Fx, x, y, array_size_x, array_size_y, myfile)

        myfile='Omega'//file_ext
        call tecout(Omega_old, x, y, array_size_x, array_size_y, myfile)
        myfile='Chi'//file_ext
        call tecout(Chi, x, y, array_size_x, array_size_y, myfile)

        myfile='restart.dat'
        call RESTART_FILE(Omega_old, Chi, myfile)

        myfile='Uy_100dB'//file_ext
        call tecout(Uy, x, y, array_size_x, array_size_y, myfile)

        myfile='UxUycombo_100dB'//file_ext
        call tecout_combo(Ux, Uy, x, y,  array_size_x, array_size_y, myfile)

   end if

end do time_marching_1

   print*,"No of file to be read:",nooftime
   call READ_VERTICAL_LINE(nooftime)
   call READ_Fx(nooftime)

STOP
END PROGRAM MAIN_TIME_MARCHING
```

C.2 Subroutines

C.2.1 Subroutine to make physical and computational grid

```
SUBROUTINE MAKE_GRID

   USE PHYSICAL_PARAMETERS
   USE COMPUTATIONAL_PARAMETERS

IMPLICIT NONE

   Double precision stretchlimit

   Integer i, j

   !---------------------------------------------------------------------!
   ! The physical domain grid is clustered near the wall
   !---------------------------------------------------------------------!

   ! --- Allocating space for the grid. This is temporary storage. --- !
```

```
 ! --- It will be deallocated at the end of the subroutine ---!
 Allocate(xgen(array_size_x, array_size_y))
 Allocate(ygen(array_size_x, array_size_y))

 !--- This controls the amount of clustering in the grid ---!
 print*,"clustering factor = ",clustering_factor

 !--- This is defined for the Tanh function. See thesis for explanation ---!
 y_gridscale = clustering_factor*pi
 x_gridscale = clustering_factor*pi

 delx_gridgen = x_gridscale/(array_size_x-1)   ! Grid spacing in x direction
 dely_gridgen = y_gridscale/(array_size_y-1)   ! Grid spacing in y direction
 print*,"delta y grid gen= ", dely_gridgen
 print*,"delta x grid gen= ", delx_gridgen

 do j=1,array_size_y
 do i=1, array_size_x
    xgen(i,j) = ((i-1) * delx_gridgen)   ! Generate "uniform grid" in the
                                         ! x and y directions
    ygen(i,j) = ((j-1) * dely_gridgen)   ! which will be transformed into
                                         ! physical clustered grid
 end do
 end do

do j=1,array_size_y
   do i=1,array_size_x
      x(i,j) = xgen(i,j)*(tube_length/x_gridscale)
                                ! uniformly spaced in the x direction
      y(i,j) = Tanh(ygen(i,j))  ! clustering the grid in the y direction
                                ! towards the wall
   end do
end do

ymax = maxval(y)   ! ymax is actually Tanh(clustering_factor*pi)
print *,"ymax at cf ",clustering_factor, "= ",ymax

y = (y/ymax)*tube_height  ! The generated clustered grid must be scaled
                          ! so that the maximum value is one. It is then
                          ! scaled by the tube_height to get the right
                          ! channel height.
print *,"ymax after scaling= ",maxval(y)

 !---------------------------------------------------------------------!
 ! ksi and eta are in the computational domain (x and y directions  !
 ! respectively). They are uniformly spaced.                         !
 !---------------------------------------------------------------------!

 ksi = 0.0   ! initiliaze
 eta = 0.0   ! initialize
```

```
   ksi = x/tube_length   ! uniform grid, with max value of grid in x
                         ! direction being 1

   !--- For y = h/ymax * Tanh( eta * cf * pi). Wall at y =array_size_y ---!
   eta =  (1.0/(clustering_factor*pi))*0.5*log( (1.+(y*ymax/tube_height))/&
                                               (1.-(y*ymax/tube_height)) )

   delta_ksi = 0.0   ! computational domain spacing in x direction
   delta_eta = 0.0   ! computational domain spacing in y direction

   delta_ksi = ksi(2,1) - ksi(1,1)
   delta_eta = eta(1,2) - eta(1,1)

   print*,"Delta ksi = ",delta_ksi
   print*,"Delta eta = ",delta_eta

   !-----------------------------------------------------------!
   ! Calculate how many points are inside the boundary layer  !
   !-----------------------------------------------------------!
   nopoint_inside_bl = 0 !initialize
   acboundlayerthick_limit = tube_height - acboundlayerthick
                         ! wall at y = array_size_y

   !---- This is for wall at y = array_size_y ---!
   do i=1, array_size_y
      if ( y(1,array_size_y+1-i) > acboundlayerthick_limit) then
         nopoint_inside_bl = nopoint_inside_bl +1
      end if
   end do

   print *, "Bottom limit of ac boundary layer = ", acboundlayerthick_limit
   print *, "No of pts inside the boundary (including end points) = ", &
            nopoint_inside_bl

   nopoint_inside_bl = nopoint_inside_bl-1   ! discount the boundary point
   print *, "No of pts inside boundary layer (discounting boundary pt)= ",
             nopoint_inside_bl

!-------------------------------------------------------------!

   !--- Deallocate the temporary storage ---!
   Deallocate(xgen)
   Deallocate(ygen)

RETURN
END SUBROUTINE MAKE_GRID
```

C.2.2 Subroutine to calculate the metrics of transformation

```
SUBROUTINE DEFINE_METRICS

   USE PHYSICAL_PARAMETERS
   USE COMPUTATIONAL_PARAMETERS

IMPLICIT NONE

   Integer i, j

!-------------------------------------------!
! Define the metrics ksix, ksiy, etay, etayy  !
!-------------------------------------------!

   ksix = 0.0
   etay = 0.0
   etayy =0.0

   oneon2deltaksi = 1.0 / (2.0*delta_ksi)
   oneon2deltaeta = 1.0 / (2.0*delta_eta)

   !--- ksix is derived from taking the derivative of ---!
   !--- ksi with respect to x ---!
   ksix = 1./tube_length

   !--- wall is at y=array_size_y ---!
   !--- y = h/ymax * Tanh( eta * cf * pi) ----!

   ! --- etay is derived from taking the derivative of eta ---!
   ! --- with respect to y ---!
   etay = (1.0/(clustering_factor*pi))*((ymax/tube_height) / &
         ( 1.0 - ((ymax/tube_height)**2)*(y**2) ))

   ! --- derived from taking the 2nd derivative of eta ---!
   ! --- with respect to y ---!
   etayy = (1.0/(clustering_factor*pi))*((2.0*((ymax/tube_height)**3)*y) / &
           (( 1.0 - ((ymax/tube_height)**2)*(y**2) )**2))

RETURN
END SUBROUTINE DEFINE_METRICS
```

C.2.3 Subroutine to calculate analytical streaming velocity

```
SUBROUTINE CALCULATE_VEL_ANALYTIC

   USE PHYSICAL_PARAMETERS
   USE COMPUTATIONAL_PARAMETERS
```

```
   USE WRITING_TO_FILES

IMPLICIT NONE

   Double precision constG, distfromwall

   Double precision, dimension(array_size_x,array_size_y) :: bigU, bigW

   Integer i, j


   !--- Initializing ---!
   constG=0.0
   bigU=0.0
   bigW=0.0

   constG=(u_0**2)/(8.0*betha*c_0)


   ! --- Rayleigh's solution for acoustic streaming --- !
   ! --- velocity between parallel plate --- !
   do j=1, array_size_y
      do i=1,array_size_x

         distfromwall = y(i,array_size_y) - y(i,j)
                        ! distance of the grid point from the wall

         !--- analytical acoustic streaming velocity from Nyborg's chapter ---!
         bigU(i,j) = -((u_0**2)/(8.*c_0)) * Sin(2.*k*x(i,j)) * &
                     ( -(Exp(-2.*betha*distfromwall) + &
                        (2.*Exp(-betha*distfromwall)*&
                          cos(betha*distfromwall)) + &
                     (6.*Exp(-betha*distfromwall)*sin(betha*distfromwall))) + &
                       3. - (18.*(distfromwall/(2.*tube_height))*&
                        (1.-(distfromwall/(2.*tube_height)))) )

      end do
   end do

  print*,"Max analytical velocity from Rayleigh is = ", maxval(bigU),&
         "m/s at", maxloc(bigU)

   !--- Boundary Condition for acoustic streaming ---!
   do i=1,array_size_x
      bigU(i,1) = (4.0*bigU(i,1+1) - bigU(i,1+2))/3.0
                  ! on-axis BC using forward diff 2nd order
   end do

   do j=1,array_size_y
      bigU(1,j)=0.0              ! on S1
      bigU(array_size_x,j)=0.0   ! on S3
```

```
   end do

   do i=1,array_size_x
      bigU(i,array_size_y)=0.0     ! on S2
   end do

   !--- Save the acoustic streaming velocity data in TECPLOT format ---!
   myfile='u2analy3D.dat'
   call tecout(bigU, x, y, array_size_x, array_size_y, myfile)

   !--- Saving the cross section data for quick check of the results ---!
   open(unit=65,file='u2analy.dat',form='FORMATTED')
   do j=1,array_size_y
      write(65,*) y(6,j), bigU(6,j), bigU(16,j)
   end do
   close(65)

   !--- Saving the on-axis data for quick check of the results ---!
   open(unit=65,file='u2analy_alongx.dat',form='FORMATTED')
   do i=1,array_size_x
      write(65,*) x(i,array_size_y), bigU(i,1)
   end do
   close(65)

RETURN
END SUBROUTINE CALCULATE_VEL_ANALYTIC

!---------------------------------------------------------------------!

SUBROUTINE READ_LINE_RAYLEIGH(nooftime)

!*********************************************************************!
! This program read vertical mesh lines                               !
!*********************************************************************!
! CHANGE THE DIMENSION TO MATCH THE GRID SIZE BEFORE USING!!!!!!!   !
!*********************************************************************!

   USE ARRAYSIZEINFO
   USE COMPUTATIONAL_PARAMETERS

IMPLICIT NONE

Real time, delta_time

Integer i,j,k
Integer nooftime
Integer offset

             ! ***** REMEMBER TO CHANGE THE COLUMN SIZE TO "# OF TIME +1"
             ! ***** IF READING MORE OR LESS THAN 5 FILES,
             ! ***** I.E. MORE THAN 5 TIME DATA !!!!
```

```
Double precision, dimension (array_size_y+1,nooftime+1) :: Ux_towrite, &
                                                           Uy_towrite
             ! 5 columns time data for array_size_y rows, 1 column for
             ! y-coordinate values array_size_y+1 -> 1 additional
             ! row for time-value

Double precision, dimension (array_size_y+1,nooftime+1) :: Ux_towrite13

Double precision, dimension (array_size_x*array_size_y+3,3) :: Uxfromfile, &
                                                               Uyfromfile

Double precision, dimension (array_size_x,array_size_y) :: Uxread, Uyread

character*30 myfile, file_ext
character*30 input_line

i = 0
j = 0
k = 0
time_print = 0

Ux_towrite13 = 0.0

   open (unit=23,file='Urayl_atx13.dat',form='FORMATTED')
   write(23,*) 'TITLE="data"'
   write(23,*) 'variables="time","U"'


do k=1,nooftime
        time_print=time_print+1

        if (time_print < 10) then
           write(file_ext,'(i1,a)') time_print,'.dat'
        else if (10 <= time_print .AND. time_print < 100) then
           write(file_ext,'(i2,a)') time_print,'.dat'
        else if (100 <= time_print .AND. time_print < 1000) then
           write(file_ext,'(i3,a)') time_print,'.dat'
        end if

          myfile ='Ux_'//file_ext
          myfile='Ux_100dB'//file_ext
          myfile = 'Urayl.dat'
          myfile='Uy_'//file_ext
          print*,"myfile = ", myfile

        delta_time = 1
        time = time_print * delta_time

!-------------------------------------------------------------!
```

```
! This is to get the data at x=1 for the whole y's
!-------------------------------------------------------------!
       Uxfromfile = 0.0
       Uyfromfile = 0.0

       open (unit=70, file=myfile, form='FORMATTED')

       do i = 1, (array_size_x*array_size_y)+3
          if (i<=3) then           ! because the first 3 rows are
                                   !tecplot header
             read(70,*) input_line ! just read but don't save
          else if (i > 3) then
             read(70,*) Uxfromfile(i,:)
          end if
       end do

       close(70)

      do j = 1, array_size_y
         do i = 1, array_size_x
            offset = (j-1)*array_size_x
            Uxread(i,j)=Uxfromfile(i+offset+3,3) ! +3 is for the
                                                 ! tecplot header
            Uyread(i,j)=Uyfromfile(i+offset+3,3)

            x(i,j) = Uxfromfile(i+offset+3,1)
            y(i,j) = Uxfromfile(i+offset+3,2)

         end do
      end do

         Ux_towrite13(1,k+1) = time

      do j=2,array_size_y+1
            Ux_towrite13(j,1) = y(1,j-1)
      end do

      do j = 2, array_size_y+1    ! because the first row is for
                                  ! x-coordinate values
         Ux_towrite13(j,k+1) = Uxread(13,j-1)
                   ! at x13 (antinode). Use with CAUTION.
                   ! Change it according to the number of
                   ! grid point in the x direction!!!
      end do
end do

do j = 1, array_size_y+1
      write(23,*) Ux_towrite13(j,:)
end do

close(60)
```

```
RETURN
END SUBROUTINE READ_LINE_RAYLEIGH
```

C.2.4 Subroutine to calculate analytical streaming velocity

```
SUBROUTINE CALCULATE_VEL_ANALYTIC

   USE PHYSICAL_PARAMETERS
   USE COMPUTATIONAL_PARAMETERS
   USE WRITING_TO_FILES

IMPLICIT NONE

   Double precision constG, distfromwall

   Double precision, dimension(array_size_x,array_size_y) :: bigU, bigW

   Integer i, j


   !--- Initializing ---!
   constG=0.0
   bigU=0.0
   bigW=0.0

   constG=(u_0**2)/(8.0*betha*c_0)


   ! --- Rayleigh's solution for acoustic streaming --- !
   ! --- velocity between parallel plate --- !
   do j=1, array_size_y
      do i=1,array_size_x

         distfromwall = y(i,array_size_y) - y(i,j)
                        ! distance of the grid point from the wall

         !--- analytical acoustic streaming velocity from Nyborg's chapter ---!
         bigU(i,j) = -((u_0**2)/(8.*c_0)) * Sin(2.*k*x(i,j)) * &
                     ( -(Exp(-2.*betha*distfromwall) + &
                        (2.*Exp(-betha*distfromwall)*&
                          cos(betha*distfromwall)) + &
                     (6.*Exp(-betha*distfromwall)*sin(betha*distfromwall))) + &
                       3. - (18.*(distfromwall/(2.*tube_height))*&
                        (1.-(distfromwall/(2.*tube_height)))) )

      end do
   end do
```

```
  print*,"Max analytical velocity from Rayleigh is = ", maxval(bigU),&
         "m/s at", maxloc(bigU)

   !--- Boundary Condition for acoustic streaming ---!
   do i=1,array_size_x
      bigU(i,1) = (4.0*bigU(i,1+1) - bigU(i,1+2))/3.0
                  ! on-axis BC using forward diff 2nd order
   end do

   do j=1,array_size_y
      bigU(1,j)=0.0              ! on S1
      bigU(array_size_x,j)=0.0   ! on S3
   end do

   do i=1,array_size_x
      bigU(i,array_size_y)=0.0     ! on S2
   end do

   !--- Save the acoustic streaming velocity data in TECPLOT format ---!
   myfile='u2analy3D.dat'
   call tecout(bigU, x, y, array_size_x, array_size_y, myfile)

   !--- Saving the cross section data for quick check of the results ---!
   open(unit=65,file='u2analy.dat',form='FORMATTED')
   do j=1,array_size_y
      write(65,*) y(6,j), bigU(6,j), bigU(16,j)
   end do
   close(65)

   !--- Saving the on-axis data for quick check of the results ---!
   open(unit=65,file='u2analy_alongx.dat',form='FORMATTED')
   do i=1,array_size_x
      write(65,*) x(i,array_size_y), bigU(i,1)
   end do
   close(65)

RETURN
END SUBROUTINE CALCULATE_VEL_ANALYTIC

!---------------------------------------------------------------------!

SUBROUTINE READ_LINE_RAYLEIGH(nooftime)

!*********************************************************************!
! This program read vertical mesh lines                               !
!*********************************************************************!
! CHANGE THE DIMENSION TO MATCH THE GRID SIZE BEFORE USING!!!!!!!   !
!*********************************************************************!

   USE ARRAYSIZEINFO
   USE COMPUTATIONAL_PARAMETERS
```

```
IMPLICIT NONE

Real time, delta_time

Integer i,j,k
Integer nooftime
Integer offset

            ! ***** REMEMBER TO CHANGE THE COLUMN SIZE TO "# OF TIME +1"
            ! ***** IF READING MORE OR LESS THAN 5 FILES,
            ! ***** I.E. MORE THAN 5 TIME DATA !!!!


Double precision, dimension (array_size_y+1,nooftime+1) :: Ux_towrite, &
                                                           Uy_towrite
            ! 5 columns time data for array_size_y rows, 1 column for
            ! y-coordinate values array_size_y+1 -> 1 additional
            ! row for time-value

Double precision, dimension (array_size_y+1,nooftime+1) :: Ux_towrite13

Double precision, dimension (array_size_x*array_size_y+3,3) :: Uxfromfile, &
                                                               Uyfromfile

Double precision, dimension (array_size_x,array_size_y) :: Uxread, Uyread

character*30 myfile, file_ext
character*30 input_line

i = 0
j = 0
k = 0
time_print = 0

Ux_towrite13 = 0.0

   open (unit=23,file='Urayl_atx13.dat',form='FORMATTED')
   write(23,*) 'TITLE="data"'
   write(23,*) 'variables="time","U"'


do k=1,nooftime
        time_print=time_print+1

        if (time_print < 10) then
           write(file_ext,'(i1,a)') time_print,'.dat'
        else if (10 <= time_print .AND. time_print < 100) then
           write(file_ext,'(i2,a)') time_print,'.dat'
        else if (100 <= time_print .AND. time_print < 1000) then
           write(file_ext,'(i3,a)') time_print,'.dat'
```

```
        end if

          myfile ='Ux_'//file_ext
          myfile='Ux_100dB'//file_ext
          myfile = 'Urayl.dat'
          myfile='Uy_'//file_ext
          print*,"myfile = ", myfile

        delta_time = 1
        time = time_print * delta_time

!-------------------------------------------------------------!
! This is to get the data at x=1 for the whole y's
!-------------------------------------------------------------!
       Uxfromfile = 0.0
       Uyfromfile = 0.0

       open (unit=70, file=myfile, form='FORMATTED')

       do i = 1, (array_size_x*array_size_y)+3
          if (i<=3) then           ! because the first 3 rows are
                                   !tecplot header
             read(70,*) input_line ! just read but don't save
          else if (i > 3) then
             read(70,*) Uxfromfile(i,:)
          end if
       end do

       close(70)

      do j = 1, array_size_y
         do i = 1, array_size_x
            offset = (j-1)*array_size_x
            Uxread(i,j)=Uxfromfile(i+offset+3,3) ! +3 is for the
                                                 ! tecplot header
            Uyread(i,j)=Uyfromfile(i+offset+3,3)

            x(i,j) = Uxfromfile(i+offset+3,1)
            y(i,j) = Uxfromfile(i+offset+3,2)

         end do
      end do

         Ux_towrite13(1,k+1) = time

      do j=2,array_size_y+1
            Ux_towrite13(j,1) = y(1,j-1)
      end do

      do j = 2, array_size_y+1    ! because the first row is for
                                  ! x-coordinate values
```

```
         Ux_towrite13(j,k+1) = Uxread(13,j-1)
                    ! at x13 (antinode). Use with CAUTION.
                    ! Change it according to the number of
                    ! grid point in the x direction!!!
      end do
end do

do j = 1, array_size_y+1
      write(23,*) Ux_towrite13(j,:)
end do

close(60)

RETURN
END SUBROUTINE READ_LINE_RAYLEIGH
```

C.2.5 Subroutine to calculate dummy variables

```
SUBROUTINE CALCULATE_SPEEDUP_CONSTANTS

   USE PHYSICAL_PARAMETERS
   USE COMPUTATIONAL_PARAMETERS

IMPLICIT NONE

   Integer i, j


!-------------------------------------------------------!
! Define some variables to minimize divison operation   !
!-------------------------------------------------------!

   !--- For Vorticity Transport Equation ---!
   N = ((mu/rho_0)*t_0) / (l**2)
                ! mu is the shear viscosity

   ND = Fmax*(t_0**2)/l
   print*, "ND=",ND

   lonhsqr = (l/h)**2

   delksisqr = delta_ksi**2
   deletasqr = delta_eta**2

   NDonynorm = 0.0
   oneonynorm = 0.0
   ksix2dksi2 = 0.0
   lonhsqronynorm = 0.0
   etay2ondeta2 = 0.0
```

```
   ksixon2dksi = 0.0
   etayon2deta = 0.0
   etayyon2deta = 0.0
   etayonynorm2deta = 0.0

   !--- These are array opertion ---!
   oneonynorm(1:array_size_x,2:array_size_y) = &
                1.0/ynorm(1:array_size_x,2:array_size_y)
                ! j index start at 2 because ynorm=0.0 at j=1.
                ! This way I avoid dividing by zero

   ! Equate the  axis value (i.e. j=1) to be equal to the value at j=2
   ! to avoid dividing by 0
   oneonynorm(1:array_size_x,1) = 1.0/ynorm(1:array_size_x,2)

   NDonynorm(1:array_size_x,2:array_size_y) = &
                 ND/ynorm(1:array_size_x,2:array_size_y)
   NDonynorm(1:array_size_x,1) = ND/ynorm(1:array_size_x,2)

   ksix2dksi2 = (ksix**2)/delksisqr

   lonhsqronynorm(1:array_size_x,1:array_size_y) = &
             lonhsqr*oneonynorm(1:array_size_x,1:array_size_y)
             ! j starts at 1 because oneonynorm has been initialized
             ! above and it has value at j=1

   etay2ondeta2 = (etay**2)/deletasqr
   ksixon2dksi = ksix/(2.0*delta_ksi)
   etayon2deta = etay/(2.0*delta_eta)
   etayyon2deta = etayy/(2.0*delta_eta)

   etayonynorm2deta(1:array_size_x,1:array_size_y) = &
             etay(1:array_size_x,1:array_size_y)*&
             oneonynorm(1:array_size_x,1:array_size_y)/(2*delta_eta)
             ! j starts at 1 because oneonynorm has been initialized
             ! above and it has value at j=1


   !-- For Stream function / Poisson eqn --!
   dummy_const = 0.0
   oneondummyconst = 0.0

   dummy_const = -2.0*( ksix2dksi2 + lonhsqr*etay2ondeta2 )
   oneondummyconst = 1.0 / dummy_const

   print*,"Finish calculating misc speed-up variables"

RETURN
END SUBROUTINE CALCULATE_SPEEDUP_CONSTANTS
```

C.2.6 Subroutine to calculate the driving force

```
SUBROUTINE CALCULATE_FORCE_RAYLEIGH

   USE PHYSICAL_PARAMETERS
   USE COMPUTATIONAL_PARAMETERS

IMPLICIT NONE

   Double precision distfromwall, SPL

   Integer i, j

   Character*30 myfile


   !--- Initialize ----!
   Fx = 0.0
   Fy = 0.0
   distfromwall = 0.0

   !--- This is from Nyborg's chapter with wall at y=array_size_ny ---!
   do j=1,array_size_y
      do i=1,array_size_x
          distfromwall = y(i,array_size_y) - y(i,j)

          Fx(i,j) = ( k * (u_0**2) * sin(2.*k*x(i,j)) ) + &
                    ( 0.25 * k * (u_0**2) * sin(2.*k*x(i,j)) * &
                      ((-3.*exp(-betha*distfromwall)* &
                        cos(betha*distfromwall)) + &
                      (exp(-betha*distfromwall) * &
                       sin(betha*distfromwall)) + &
                      (exp(-2.*betha*distfromwall))) )

      end do
   end do
   !---------------------------------------------------------------------!

   Fx=-Fx*rho_0

   !--- Save data in TECPLOT format ---!
   myfile='Fx3D.dat'
   call tecout(Fx, x, y, array_size_x, array_size_y, myfile)

   !--- Save cross sectional data for quick viewing of results ---!
   open(unit=65,file='Fx_analy_along_y.dat',form='FORMATTED')
   do j=1,array_size_y
      write(65,*) y(5,j), Fx(5,j)
   end do
   close(65)
```

```
   !--- Save axial data for quick viewing of results ---!
   open(unit=65,file='Fx_analy_along_x.dat',form='FORMATTED')
   do i=1,array_size_x
      write(65,*) x(i,1), Fx(i,1), Fx(i,array_size_y)
   end do
   close(65)

   Fxnorm=Fx

RETURN
END SUBROUTINE CALCULATE_FORCE_RAYLEIGH
```

C.2.7 Subroutine to setup the coefficients of the vorticity

```
!============================================================!
! This subroutine calculates the coefficents of the voriticity !
! in the computational domain. Notice that it involves         !
! the METRICS of transformation                                !
!============================================================!
SUBROUTINE OMEGA_COEFF_SETUP

   USE PHYSICAL_PARAMETERS
   USE COMPUTATIONAL_PARAMETERS
   USE COEF_FOR_MATRIX
   USE WRITING_TO_FILES

IMPLICIT NONE

   Double precision distfromwall

   Integer i, j

   a_coef_omega = 0.0
   b_coef_omega = 0.0
   c_coef_omega = 0.0
   d_coef_omega = 0.0
   e_coef_omega = 0.0
   forcing_function = 0.0


   do j=2,array_size_y-1
      do i=2,array_size_x-1
      distfromwall = y(i,array_size_y) - y(i,j)
         a_coef_omega(i,j) = ksix2dksi2(i,j) - &
                             ( (rho_0/mu)*Ux(i,j)*ksixon2dksi(i,j) )

         b_coef_omega(i,j) = ksix2dksi2(i,j) + &
                             ( (rho_0/mu)*Ux(i,j)*ksixon2dksi(i,j) )

         c_coef_omega(i,j) = etayyon2deta(i,j) + etay2ondeta2(i,j) + &
```

```
                              ((rho_0/mu)*Uy(i,j)*etayon2deta(i,j))

         d_coef_omega(i,j) = -etayyon2deta(i,j) + etay2ondeta2(i,j) - &
                              ((rho_0/mu)*Uy(i,j)*etayon2deta(i,j))

         e_coef_omega(i,j) = -(2.*ksix2dksi2(i,j)) - (2.*etay2ondeta2(i,j)) - &
                              (rho_0/mu)*&
                              ( (ksixon2dksi(i,j)*(Ux(i+1,j)-Ux(i-1,j))) + &
                                (etayon2deta(i,j)*(Uy(i,j+1)-Uy(i,j-1))) )
      end do
   end do


   do j=1,array_size_y
      do i=1,array_size_x
         distfromwall = y(i,array_size_y) - y(i,j)

         forcing_function(i,j) = -0.5*betha*(rho_0/mu) * k * (u_0**2) * &
                                  Sin(2.*k*x(i,j))* &
                 ((2.*Exp(-betha*distfromwall)*Cos(betha*distfromwall))+&
                   (Exp(-betha*distfromwall)*Sin(betha*distfromwall))-&
                   (Exp(-2.*betha*distfromwall)))
      end do
   end do

RETURN
END SUBROUTINE OMEGA_COEFF_SETUP
```

C.2.8 Subroutine to setup the stiffness matrix

```
!=====================================================================!
! This subroutimne puts the coefficient of vorticity (or the stream   !
! function - depending on the input) into the stiffness matrix        !
!=====================================================================!
SUBROUTINE INNERPOINTS_MATRIX_SETUP(a_coef, b_coef, c_coef, d_coef, &
                                    e_coef, forcefunc, nx, ny)

   USE STIFFNESS_MATRIXandRHS_VECTOR

IMPLICIT NONE

   Double precision, dimension(nx, ny) :: a_coef, b_coef, c_coef, d_coef, &
                                          e_coef
   Double precision, dimension (nx, ny) :: forcefunc

   Integer i, j, nx, ny, rownum, m1, m2

   m1 = nx   ! subdiagonal bandwidth
   m2 = nx   ! superdiagonal bandwidth
```

```
   !-- initialize --!
   StiffMatrix = 0.0
   rhs_vector = 0.0

   ! --- For inner points ---!
   do j= 2, ny-1
      do i=2, nx-1
         rownum = (j-1)*nx + i

         StiffMatrix(rownum,m1+1) = e_coef(i,j)    ! diagonal

         StiffMatrix(rownum,m1+2) = a_coef(i,j)    ! 1st SUPERdiagonal

         StiffMatrix(rownum,m1) = b_coef(i,j)      ! 1st SUBdiagonal

         StiffMatrix(rownum,m1+1+nx) = c_coef(i,j) ! 2nd SUPERdiagonal

         StiffMatrix(rownum,1) = d_coef(i,j)       ! 1st SUBdiagonal

         rhs_vector(rownum) = forcefunc(i,j)       ! forcing function on
                                                   ! right hand side vector
      end do
   end do

RETURN
END SUBROUTINE INNERPOINTS_MATRIX_SETUP
```

C.2.9 Subroutine to apply the vorticity boundary condition

```
!================================!
! Vorticity Boundary Condition   !
!================================!
SUBROUTINE OMEGA_BOUNDARY_CONDITION

   USE PHYSICAL_PARAMETERS
   USE COMPUTATIONAL_PARAMETERS
   USE STIFFNESS_MATRIXandRHS_VECTOR

IMPLICIT NONE

   Double precision, dimension (array_size_x, array_size_y) :: dChidx,&
                                                               dChidy

   Double precision distfromwall

   Integer i, j, rownum, m1


   m1 = array_size_x
```

```
! -- S1 (left boundary) -- !
do j=1,array_size_y
   delta_ksi = x(2,j)-x(1,j)

   rownum = (j-1)*array_size_x + 1

   dChidx(1,j) = ksixon2dksi(1,j) * (-Chi(1+2,j)+4.*Chi(1+1,j)-&
                                      3.*Chi(1,j))

   rhs_vector(rownum) = -(3./(delta_x**2)) * ( Chi(1+1,j) - Chi(1,j) + &
                        (dChidx(1,j)*delta_x) ) - (Omega_old(1+1,j)/2.)

   StiffMatrix(rownum,m1+1) = 1.0   ! for Boundary Condition
end do

! -- S3 (right boundary) -- !
do j=1,array_size_y
   delta_eta = x(array_size_x,j)-x(array_size_x-1,j)
   rownum = j*array_size_x

   dChidx(array_size_x,j) = ksixon2dksi(array_size_x,j) * &
                            (3.*Chi(array_size_x,j)-&
                             4.*Chi(array_size_x-1,j)+&
                             Chi(array_size_x-2,j))

   rhs_vector(rownum) = -(3./(delta_x**2)) * &
                        ( Chi(array_size_x-1,j) - Chi(array_size_x,j) - &
                          (dChidx(array_size_x,j)*delta_x) ) - &
                          (Omega_old(array_size_x-1,j)/2.)

   StiffMatrix(rownum,m1+1) = 1.0   ! for Boundary Condition
end do

! -- S2 (top boundary) -- !
do i=2,array_size_x-1
   delta_eta = y(i,array_size_y) - y(i,array_size_y-1)
   rownum = (array_size_y-1)*array_size_x + i

   dChidy(i,array_size_y) = etayon2deta(i,array_size_y) * &
                            (3.*Chi(i,array_size_y)-&
                             4.*Chi(i,array_size_y-1)+&
                             Chi(i,array_size_y-2))

   rhs_vector(rownum) = -(3./(delta_y**2)) * &
                        ( Chi(i,array_size_y-1) - Chi(i,array_size_y) - &
                          (dChidy(i,array_size_y)*delta_y) ) - &
                          (Omega_old(i,array_size_y-1)/2.)

   StiffMatrix(rownum,m1+1) = 1.0   ! for Boundary Condition
end do
```

```
   ! -- Bottom boundary (axis) -- !
   do i=2,array_size_x-1
      delta_eta = y(i,array_size_y) - y(i,1)
      rownum = i

      dChidy(i,1) = etayon2deta(i,1) * (-Chi(i,1+2)+4.*Chi(i,1+1)-&
                                         3.*Chi(i,1) )

      rhs_vector(rownum) = 0.

      StiffMatrix(rownum,m1+1) = 1.0   ! for Boundary Condition
  end do

RETURN
END SUBROUTINE OMEGA_BOUNDARY_CONDITION
```

C.2.10 Subroutine to do banded matrix decomposition

```
!================================================================!
! Subroutine to decompose banded matrix. From Numerical Recipes.  !
!================================================================!
SUBROUTINE BANDEC(nx,ny)

   USE STIFFNESS_MATRIXandRHS_VECTOR

IMPLICIT NONE

Integer row, column, nx, ny, m1, m2, i, j, k, el, bandwidth, rownum
Double precision d, dum
Double precision, parameter :: tiny = 1.0e-20


!-------------------------------------------------------------------------!
! Given an N x N band diagonal matrix A with m1 subdiagonal rows and      !
! m2 superdiagonal rows, compactly stored inthe array a(1:N,1:m1+m2+1)     !
! as described in the comment for routine banmul, this routine construct  !
! an LU decomposition of a rowwise permutation of A. The upper triangular !
! matrix replaces a, while the lower triangular matrix is returned in     !
! al(1:N,1:m1). indx is an output vector of length N that records the row !
! permutation effected by the partial pivoting; d is output as +-1 depending!
! on whether the number of row interchanges was even or odd, respectively.!
! This routine is used in combination with banbks to solve band-diagonal  !
! sets of equations.                                                      !
!-------------------------------------------------------------------------!

indx=0

m1=nx
m2=nx
```

```
row = nx*ny
column = nx*ny
bandwidth = m1 + 1 + m2

el = m1

do i =1,m1                        ! rearrange the storage a bit
   do j = m1+2-i, bandwidth
      StiffMatrix(i,j-el) = StiffMatrix(i,j)
   end do

   el = el-1
   do j= bandwidth - el, bandwidth
      StiffMatrix(i,j) = 0.0
   end do
end do

d = 1.0
el = m1

do k=1,row          ! For each row ...

   dum = StiffMatrix(k,1)
   i = k
   if (el < row) el = el+1

   do j = k+1,el      ! Find the pivot element
      if (abs(StiffMatrix(j,1)) > abs(dum)) then
         dum = StiffMatrix(j,1)
         i=j
      end if
   end do

   indx(k) = i

   if (dum == 0.0) StiffMatrix(k,1) = tiny

   if (i /= k) then
      d = -d
      do j = 1,bandwidth
         dum = StiffMatrix(k,j)
         StiffMatrix(k,j) = StiffMatrix(i,j)
         StiffMatrix(i,j) = dum
      end do
   end if

   !--- The elimination ---!
   do i = k+1,el      ! Do the elimination
      dum = StiffMatrix(i,1)/StiffMatrix(k,1)
      lowertriang(k,i-k) = dum      !lower triangular
      do j = 2, bandwidth
```

```
         StiffMatrix(i,j-1) = StiffMatrix(i,j) - dum*StiffMatrix(k,j)
      end do

      StiffMatrix(i,bandwidth) = 0.0
   end do

end do

RETURN
END SUBROUTINE BANDEC
```

C.2.11 Subroutine to do banded matrix backsubstitution

```
!=============================================!
! Subroutine that does the backsubstitution.  !
! From Numerical Recipes.                     !
!=============================================!
SUBROUTINE BANBKS(nx,ny)

   USE COMPUTATIONAL_PARAMETERS
   USE STIFFNESS_MATRIXandRHS_VECTOR
   USE WRITING_TO_FILES

IMPLICIT NONE

Integer row, nx, ny, m1, m2
Double precision, parameter :: tiny = 1.0e-20

Integer i, j, k, l, bandwidth, rownum

Double precision d, dum

!-----------------------------------------------------------------------------!
! Given the arrays a, al, and indx as returned form bandec, and given         !
! a right-hand side vector b(1:n), solves the band diagonal linear eqns       !
! Ax = b. The solution vector x overwrites b(1:n). The other input arrays     !
! are not modified, and can be left in place for succesive calls with         !
! different right-hand sides.                                                 !
!-----------------------------------------------------------------------------!
m1=nx
m2=nx
row = nx*ny
bandwidth = m1 + 1 + m2

l = m1

do k=1,row       ! Forward substitution, unscrambling the permutated
                 ! rows as we go
   i = indx(k)
```

```
   if(i /= k) then
      dum = rhs_vector(k)
      rhs_vector(k) = rhs_vector(i)
      rhs_vector(i) = dum
   end if

   if (l < row) l = l+1

   do i = k+1,l
      rhs_vector(i) = rhs_vector(i) - lowertriang(k,i-k)*rhs_vector(k)
   end do

end do

l = 1

do i=row,1,-1                          ! Backsubstitution
   dum = rhs_vector(i)
   do k = 2,l
      dum = dum - Stiffmatrix(i,k)*rhs_vector(k+i-1)
   end do

   rhs_vector(i) = dum / Stiffmatrix(i,1)

   if (l < bandwidth) l = l+1

end do


RETURN
END SUBROUTINE BANBKS
```

C.2.12 Subroutine to convert the result from vector to array format

```
!=========================================================================!
! This subroutine put the results that is in a vector format into an array  !
! format                                                                    !
!=========================================================================!
SUBROUTINE RESULTINVECTOR_TO_ARRAY(rhs_vector, originalarrayform, &
                                   nx, ny)

IMPLICIT NONE

   Integer i, j, nx, ny, rownum

   Double precision, dimension (nx,ny) :: originalarrayform
   Double precision, dimension (nx*ny) :: rhs_vector
```

```
   ! --- For inner points ---!
   do j= 1, ny
      do i=1, nx
         rownum = (j-1)*nx + i

         originalarrayform(i,j) = rhs_vector(rownum)
      end do
   end do

RETURN
END SUBROUTINE RESULTINVECTOR_TO_ARRAY
```

C.2.13 Subroutine to setup the coefficients of the stream function

```
!=====================================================================!
! This subroutine calculates the coefficents of the stream function   !
! in the computational domain. Notice that it involves the METRICS    !
! of transformation.                                                  !
!=====================================================================!
SUBROUTINE CHI_COEFF_SETUP

   USE PHYSICAL_PARAMETERS
   USE COMPUTATIONAL_PARAMETERS
   USE COEF_FOR_MATRIX

IMPLICIT NONE

   Integer i, j

   a_coef_chi = 0.0
   b_coef_chi = 0.0
   c_coef_chi = 0.0
   d_coef_chi = 0.0
   e_coef_chi = 0.0

   do j=2,array_size_y-1
      do i=2,array_size_x-1

         a_coef_chi(i,j) = ksix2dksi2(i,j)

         b_coef_chi(i,j) = ksix2dksi2(i,j)

         c_coef_chi(i,j) = ( etayyon2deta(i,j) + etay2ondeta2(i,j) )

         d_coef_chi(i,j) = ( -etayyon2deta(i,j) + etay2ondeta2(i,j) )

         e_coef_chi(i,j) = -(2.*ksix2dksi2(i,j)) - (2.*etay2ondeta2(i,j))
```

```
         forcing_function(i,j) = Omega_old(i,j)

      end do
   end do

RETURN
END SUBROUTINE CHI_COEFF_SETUP
```

C.2.14 Subroutine to apply the stream function boundary condition

```
!=====================================================================!
! Apply stream function boundary condition for the stiffness matrix   !
! and right hand side vector.                                         !
!=====================================================================!
SUBROUTINE CHI_BOUNDARY_CONDITION

   USE PHYSICAL_PARAMETERS
   USE STIFFNESS_MATRIXandRHS_VECTOR
   USE COMPUTATIONAL_PARAMETERS

IMPLICIT NONE

   Double precision distfromwall

   Integer i, j, rownum, m1

   !-- Note: Analytically, psi is 0 at the boundaries. --!
   !-- I double-checked it with Mathematica --!

   m1 = array_size_x

   ! -- S1 (left boundary) -- !
   do j=1,array_size_y
      rownum = (j-1)*array_size_x + 1

      StiffMatrix(rownum,m1+1) = 1.0

      rhs_vector(rownum) = 0.
   end do


   ! -- S3 (right boundary) -- !
   do j=1,array_size_y
      rownum = j*array_size_x

      StiffMatrix(rownum,m1+1) = 1.0

      rhs_vector(rownum) = 0.
   end do
```

```
   ! -- S2 (top boundary) -- !
   do i=2,array_size_x-1
      rownum = (array_size_y-1)*array_size_x + i

      StiffMatrix(rownum,m1+1) = 1.0

      rhs_vector(rownum) = 0.
   end do

   ! -- Axis (bottom boundary) -- !
   do i=2,array_size_x-1
      rownum = i

      StiffMatrix(rownum,m1+1) = 1.0

      rhs_vector(rownum) = 0.
   end do

RETURN
END SUBROUTINE CHI_BOUNDARY_CONDITION
```

C.2.15 Subroutine to calculate the numerical acoustic streaming velocity

```
!=======================================================!
! Calculate the numerical acoustic streaming velocity.  !
!=======================================================!
SUBROUTINE CALCULATE_VELOCITY

   USE PHYSICAL_PARAMETERS
   USE ARRAYSIZEINFO
   USE COMPUTATIONAL_PARAMETERS

IMPLICIT NONE

Integer i, j

!*******************************************************!
! Then we need to calculate Ux and Uy.                  !
!*******************************************************!

   do j=2,array_size_y-1
      do i=2,array_size_x-1
         Ux(i,j) = etayon2deta(i,j) * (Chi(i,j+1)-Chi(i,j-1))

         Uy(i,j) = -ksixon2dksi(i,j) * (Chi(i+1,j)-Chi(i-1,j))
      end do
   end do
```

```
!---------------------------------------------------------!
! BC for U: Ux & Uy =0 on S1, wall, S3                    !
! Uy=0 & dUy/dy=0 on axis                                 !
!---------------------------------------------------------!

   do j=1,array_size_y
      Uy(1,j)=0.0             ! on S1
      Uy(array_size_x,j)=0.0  ! on S3

      Ux(1,j)=0.0              ! on S1
      Ux(array_size_x,j)=0.0   ! on S3
   end do

   do i=1,array_size_x
      Uy(i,1)=0.0                 ! on axis (axis is at j=1)
      Uy(i,array_size_y)=0.0      ! at wall (wall is at j=array_size_y)

      Ux(i,1) = (4.*Ux(i,1+1)-Ux(i,1+2))/3.
                                  ! on axis (axis is at j=1)
      Ux(i,array_size_y)=0.0      ! at wall (wall is at j=array_size_y)
   end do

RETURN
END SUBROUTINE CALCULATE_VELOCITY
```

C.2.16 Subroutine to write data in TECPLOT format

```
!=========================================================================!
! This subroutine writes the data to a file in a format that can be read !
! by TECPLOT                                                              !
!=========================================================================!

SUBROUTINE TECOUT (functowrite, x, y, arraysize_x, arraysize_y, &
                   myfile)

IMPLICIT NONE

   character*30 myfile
   integer arraysize_x, arraysize_y
   integer i, j

   Double precision, dimension (arraysize_x,arraysize_y) :: functowrite, &
                                                            x, y

   open(unit=75, file=myfile, form='FORMATTED')

      write(75,*) 'TITLE="data"'
      write(75,*) 'variables="x","y","f"'
      write(75,*) 'ZONE I=',arraysize_x,'J=',arraysize_y,'f=point'
```

```
   do j=1,arraysize_y
      do i=1,arraysize_x
         write(75,*) x(i,j), y(i,j), functowrite(i,j)
      end do
   end do

   close(75)

RETURN
END SUBROUTINE TECOUT
```

C.2.17 Subroutine to write axial and tangential (or radial) data in TECPLOT format

```
!=======================================================================!
! Write AXIAL and TANGENTIAL (or radial) data in a format that can be   !
! read by TECPLOT                                                       !
!=======================================================================!
SUBROUTINE TECOUT_COMBO (Function_x, Function_y, x, y, arraysize_x, &
                         arraysize_y, myfile)

IMPLICIT NONE

   character*30 myfile

   Integer i,j
   Integer arraysize_x, arraysize_y

   Double precision, dimension (arraysize_x,arraysize_y) :: Function_x, &
                                                            Function_y, x, y


    open (unit=80, file=myfile, form='FORMATTED')
    write(80,*) ' TITLE=" Ur Uz"'
    write(80,*) 'variables="X","Y","Ux","Uy"'
    write(80,*) ' ZONE, i=', arraysize_x, ',j=', arraysize_y, ',F=POINT'

        do j=1,arraysize_y
           do i=1,arraysize_x
              write(80,*) x(i,j), y(i,j), Function_x(i,j), Function_y(i,j)
           end do
        end do
    close(80)

RETURN
END SUBROUTINE TECOUT_COMBO
```

C.2.18 Subroutine to write stream function and vorticity data for restarting the calculation

```
!=====================================================================!
! Write stream function and vorticity to a file so that when the      !
! process is interrupted, it can be started back up from the last saved   !
! step without staring over                                           !
!=====================================================================!
SUBROUTINE RESTART_FILE (Omega_old, Chi, myfile)

   USE ARRAYSIZEINFO

IMPLICIT NONE

   character*30 myfile
   integer i, j

   REAL(KIND = 8),DIMENSION(array_size_x,array_size_y), &
   INTENT(IN) :: Omega_old, Chi

   OPEN(75, FILE=myfile, STATUS='unknown', ACTION='write', FORM='unformatted')
   REWIND(75)

WRITE (75)  ((Omega_old(i,j), i=1,SIZE(Omega_old,1)), &
                                  j=1,SIZE(Omega_old,2)),&
((Chi(i,j), i=1,SIZE(Chi,1)), j=1,SIZE(Chi,2))
   CLOSE(75)

RETURN
END SUBROUTINE RESTART_FILE
```

Bibliography

[1] Murat K. Aktas and Bakhtier Farouk. Numerical simulation of acoustic streaming generated by finite-amplitude resonant oscillations in an enclosure. *J. Acoust. Soc. Am.*, 116, no.5:2822–2832, 2004.

[2] E. Andrade. On the circulations caused by the vibration of air in a tube. *Proc. Roy. Soc.*, 134:445–470, 1931.

[3] Helene Bailliet, Vitalyi Gusev, Richard Raspet, and Robert A. Hiller. Acoustic streaming in closed thermoacoustic devices. *J. Acoust. Soc. Am.*, 110, no. 4:1808–1821, 2001.

[4] Said Boluriaan and Philip J. Morris. Acoustic streaming: From rayleigh to today. *International Journal of Aeroacoustics*, 2, no. 3 and 4:255–292, 2003.

[5] P. E. Doak. *Proc. Roy. Soc.*, A226:7, 1954.

[6] C. Eckert. *Physical Review*, 73, no. 1:68–76, 1948.

[7] S. A. Elder. *J. Acoust. Soc. Am.*, 31:54, 1959.

[8] M. Faraday. On a peculiar class of acoustical figures. *Phil. Trans. Roy. Soc. London*, 121:299–340, 1831.

[9] F. E. Fox and K. F. Herzfeld. *Phys. Rev.*, 78:156, 1950.

[10] D. Gedeon. *DC Gas Flows In Stirling And Pulse-Tube Cryocoolers. In R. G. Ross, Editor, Cryocoolers 9*, pages 385–392. Plenum, New York, 1997.

[11] Mark F. Hamilton, Yurii A. Illinskii, and Evgenia A. Zabolotskaya. Acoustic streaming generated by standing waves in two-dimensional channels of arbitrary width. *J. Acoust. Soc. Am.*, 113, no. 1:153–160, 2003.

[12] Mark F. Hamilton, Yurii A. Illinskii, and Evgenia A. Zabolotskaya. Thermal effects on acoustic streaming in standing wave. *J. Acoust. Soc. Am.*, 114, no. 6:3092–3101, 2003.

[13] Klaus A. Hoffmann and Steve T. Chiang. *Computational Fluid Dynamics for Engineers Vol.1 and 2*. Engineering Education System, Austin, Texas, USA, 1989.

[14] J. Holtzmark, I. Johnsen, T. Sikkeland, and S. Skavlem. Boundary layer flow near a cylindrical obstacle in an oscillating, incompressible flow. *J. Acoust. Soc. Am.*, 26:26, 1954.

[15] U. Ingard and S. Labate. Acoustic circulation effects and the nonlinear impedance of orifices. *J. Acoust. Soc. Am.*, 32:1387–1395, 1960.

[16] F. J. Jackson. Sonically induced microstreaming near a plane boudnary. ii. acoustic streaming field. *J. Acoust. Soc. Am.*, 32:1387–1395, 1960.

[17] F. J. Jackson and W. L. Nyborg. *J. Acoust. Soc. Am.*, 30:614, 1958.

[18] I. Johnsen and S. Tjotta. *Acustica*, 7:7, 1957.

[19] Tomoo Kamakura, Kazuhisa Matsuda, Yoshiro Kumamoto, and Mack A. Breazeale. Acoustic streaming induced in focused gaussion beams. *J. Acoust. Soc. Am.*, 97, no. 5:2740–2746, 1995.

[20] J. Kolb and W. L. Nyborg. *J. Acoust. Soc. Am.*, 28:1237, 1956.

[21] L. N. Liebermann. The second viscosity of liquids. *Phys. Rev.*, 75:1415–1422, 1949.

[22] M. J. Lighthill. Acoustic streaming. *Journal of Sound and Vibration*, 61, No. 3:391–418, 1978.

[23] J. J. Markham. *Phys. Rev.*, 86:497–502, 1952.

[24] H. Medwin. *J. Acoust. Soc. Am.*, 26:332–341, 1954.

[25] A. Meissner. *Z. tech. Physik*, 7:585, 1926.

[26] H. Ninomiya and K. Onishi. *Flow Analysis Using a PC*, chapter 6, pages 109–129. Computational Mechanics Publications and CRC Press, Inc., 1991.

[27] W. L. Nyborg. *Acoustic Streaming*, chapter 11, pages 265–331. Academic Press, New York, 1965.

[28] J. E. Piercy and J. Lamb. *Proc. Roy. Soc.*, A226:43, 1954.

[29] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes In Fortran 77 The Art Of Scientific Computing*, chapter 2, pages 22–89. Press Syndicate of the University of Cambridge, 1992.

[30] W. P. Raney, J. C. Corelli, and P. J. Westervelt. Acoustical streaming in the vicinity of a cylinder. *J. Acoust. Soc. Am.*, 26:1006, 1954.

[31] N. Riley.

[32] N. Riley. Acoustic streaming. *Theoretical And Computational Fluid Dynamics*, 10:349–356, 1998.

[33] N. Rott. The influence of heat conduction on acoustic streaming. *A. Angew. Math. Phys.*, 25:417–421, 1974.

[34] O. V. Rudenko, S. I. Soluyan, and Robert T. Beyer. *Theoretical Foundations Of Nonlinear Acoustics*, pages 187–211. Consultants Bureau, New York, 1977.

[35] H. Schlichting. Berechnung ebener periodischer grenzschichtstromungen (calculation of plane periodic boundary layer streaming). *Physikalische Zeitschrift*, 33, No. 8:327–335, 1932.

[36] H. Schlichting. *Boundary Layer Theory.* McGraw-Hill, New York, 1955.

[37] V. K. Schuster and W. Matz. Uber stationare stromungen im kundtschen rohr (on stationary streaming in kundt tubes). *Akust. Zeitschrift*, 5:349–352, 1940.

[38] S. Skavlem and S. Tjotta. Steady rotational flow of an incompressible, viscous fluid enclosed between two coaxial cylinders. *J. Acoust. Soc. Am.*, 27:26–33, 1955.

[39] B. L. Smith and A. Glezer. The formation and evolution of synthetic jets. *Phys. Fluids*, 10:2281–2297, 1998.

[40] William Frederick Spotz. *Higher-Order Compact Finite Difference Schemes for Computational Mechanics.* PhD thesis, The University of Texas at Austin, 1995.

[41] J. W. Strutt and Baron Rayleigh. *The Theory Of Sound.* Dover, New York, 1945.

[42] G. W. Swift. *Thermoacoustics: A Unifying Perspective For Some Engines And Refrigerators*, chapter 7, pages 176–196. The Acoustical Society of America, 2002.

[43] Frank Kreith *et al.* *"Fluid Mechanics" Mechanical Engineering Handbook.* CRC Press LLC, Boca Raton, 1999.

[44] Michael W. Thompson. *Measurement Of Acoustic Streaming In A Standing Wave Using Laser Doppler Anemometry.* PhD thesis, The Pennsylvania State University, 2004.

[45] Michael W. Thompson and Anthony A. Atchley. Optical measurement of acoustic streaming in a standing wave with a temperature gradient. *J. Acoust. Soc. Am.*, 115, no.5, Pt.2:2532, 2004.

[46] C. Truesdell. *Proc. Roy. Soc.*, A226:59, 1954.

[47] J. P. Walker and C. H. Allen. *J. Acoust. Soc. Am.*, 22:680A, 1950.

[48] R. Waxler. Stationary velocity and pressure gradients in a thermoacoustic stack. *J. Acoust. Soc. Am.*, 109:2739–2750, 2001.

[49] P. J. Westervelt. The theory of steady rotational flow generated by a sound field. *J. Acoust. Soc. Am.*, 25:60–67, 1953.

[50] Steven Joseph Younghouse. Acoustic streaming at high reynolds numbers produced by focused sound beams with shocks in real fluids. Master's thesis, The University of Texas at Austin, 1998.

Vita

Debbie Sastrapradja

Education:

M.S. in Acoustics, December 1997, The Pennsylvania State University

B.S. in Electrical Engineering, May 1995, University of Hawaii at Manoa

Work Experience:

Teaching Assistant, Fundamental Acoustics Course for Distance Education, Graduate Program in Acoustics, The Pennsylvania State University, August 2003 - December 2003

Research Assistant, Graduate Program in Acoustics, The Pennsylvania State University, August 2000 - July 2003

Teaching Assistant, Digital Signal Processing Course for Distance Education, Graduate Program in Acoustics, The Pennsylvania State University, August 2001 - December 2001

Development Engineer, Caterpillar Incorporated, May 1999 - June 2000

Test Engineer, Delphi Automotive System, January 1998 - May 1999

Teaching Assistant, Digital Signal Processing Course for Distance Education, Graduate Program in Acoustics, The Pennsylvania State University, August 1996 - December 1996

Conference Presentation:

"Application of a stream-function numerical approach to Rayleigh streaming", 142nd Meeting: Acoustical Society of America, Fort Lauderdale, Florida, December 4, 2001