NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

UNIFORM AND MULTI-GRID MODELING OF ACOUSTIC WAVE PROPAGATION WITH CELLULAR AUTOMATON TECHNIQUES

by

Jermaine A. Bailey

March 2013

Thesis Advisor: Young W. Kwon
Second Reader: Jarema M. Didoszak

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 2013	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE
UNIFORM AND MULTI-GRID MODELING OF ACOUSTIC WAVE PROPAGATION WITH CELLULAR AUTOMATON TECHNIQUES

5. FUNDING NUMBERS

6. AUTHOR(S) Jermaine A. Bailey

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943–5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release;distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words)

The Cellular Automaton (CA) method is an alternative computational technique used in understanding the behavioral response of dynamic systems. It allows great flexibility in the application of various types of boundary conditions. As such, this method is used in developing an alternative propagation model for ocean acoustics. The modeling scheme creates a profile of propagation losses versus range, in an acoustic medium. Moreover, the chosen complex boundary conditions, which are not easily modeled by ordinary numerical techniques, are shown to perform with remarkable ease with CA methods.

Accordingly, like any other modeling method, the computational time increases when a refined solution is desired. As such, an alternative multi-grid modeling scheme is shown to increase the performance time of the CA method significantly. This improvement is dependent upon the total number of global grid points inside the multi-grid domain. The end result show a multi-grid, with fewer nodal points, producing accurate results that replicate a uniform grid, which utilizes a larger quantity of nodal points.

14. SUBJECT TERMS Cellular Automaton, Uniform Grid, Multi-Grid, Local Grid, Underwater Acoustic, Wave Propagation			**15. NUMBER OF PAGES** 121 **16. PRICE CODE**
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540–01–280–5500

Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release;distribution is unlimited

UNIFORM AND MULTI-GRID MODELING OF ACOUSTIC WAVE PROPAGATION WITH CELLULAR AUTOMATON TECHNIQUES

Jermaine A. Bailey
Lieutenant, United States Navy
B.S., Louisiana State University, 2004
M.S., Florida Agricultural & Mechanical University, 2012

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 2013

Author: Jermaine A. Bailey

Approved by: Young W. Kwon
Thesis Advisor

Jarema M. Didoszak
Second Reader

Knox T. Millsaps
Chair, Department of Mechanical and Aerospace Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The Cellular Automaton (CA) method is an alternative computational technique used in understanding the behavioral response of dynamic systems. It allows great flexibility in the application of various types of boundary conditions. As such, this method is used in developing an alternative propagation model for ocean acoustics. The modeling scheme creates a profile of propagation losses versus range, in an acoustic medium. Moreover, the chosen complex boundary conditions, which are not easily modeled by ordinary numerical techniques, are shown to perform with remarkable ease with CA methods.

Accordingly, like any other modeling method, the computational time increases when a refined solution is desired. As such, an alternative multi-grid modeling scheme is shown to increase the performance time of the CA method significantly. This improvement is dependent upon the total number of global grid points inside the multi-grid domain. The end result shows a multi-grid, with fewer nodal points, producing accurate results that replicate a uniform grid, which utilizes a larger quantity of nodal points.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to dedicate this thesis to my wonderful wife, who has been my GREATEST supporter in completing this manuscript! I am truly thankful to my Lord and Savior for allowing me to meet her. I know I would not have been able to complete graduate school if it had not been for her continued encouragement during my time at Naval Postgraduate School.

With the utmost sincerity and gratitude I would also like to thank my advisor, Dr. Young Kwon for his endless supply of patience and vast knowledge, in helping me to complete this research assignment. The many months, of spending countless hours in his office for guidance and assistance, made timely completion of this topic feasible. I do not know of any other professor at NPS who makes himself as readily available, as he consistently does, in order to ensure the success of his students.

Lastly, I would like to extend a great deal of appreciation to Jarema Didoszak for his time and effort in helping to make this work of an acceptable, professional quality.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MOTIVATION AND OBJECTIVE

Sound Navigation and Ranging (SONAR) systems are having an increasing role in the implementation and design of future naval systems. SONAR was first used during World War I, as a ranging tool, to detect the presence of submarines [1]. Moreover, Z. Jiang and A. Quazi have researched the possible use of optical and electromagnetic sources as alternative underwater ranging techniques [2], [3]. However, electromagnetic waves can only travel short distances due to high attenuation and absorption effects that range up to 45 (dB), per kilometer, in water [2]. As a reference, acoustic wave absorption is typically in the range of 3 times lower than electromagnetic waves [3]. In addition, the use of optical sources, underwater, is limiting because scattering and absorption effects reduce their usefulness as an underwater ranging source [4].

As such, SONAR systems, on naval sea going vessels, have undergone vast improvements over the last century. Improvements in computer technology have contributed greatly in allowing these onboard sensors to be utilized more efficiently. In such a manner, the prior has given a submarine captain more options when embarking on voyages that may be navigationally challenging. Traversing in ocean areas with a high concentration of ships requires precise ship handling in interpreting the travel path and location of ships in the area. Nevertheless, the precise SONAR interpretation of real ships, vice miscellaneous background noise in the ocean environment, is the key to safe passage of a submarine. Driving factors influencing SONAR improvements have alluded to the fact that primary naval missions have shifted from open ocean to littoral waters (water depth <200 (ft) to shore) where diesel submarines and mines are the threat of concern.

Nevertheless, before naval operators are qualified to interpret the visual meaning of a SONAR display (see Figure 1), training in a simulation environment is a gateway to understanding the onboard system response [5]. There are many prepackaged models which form the basis of these simulation systems (see Figure 2) [6]. In addition, Meteorological and Oceanographic prediction systems utilize such models to predict the performance of naval acoustic sensors in advance of naval deployments to hostile coastal areas. Nonetheless, the focus of this research concentrates on developing an alternative propagation model, using Cellular Automaton (CA) methods. The complexity of the ocean environment, in particular, the structure of the ocean bottom, limits current models from behaving in an ideal manner [7]. In turn, this leads to an array of assumptions and approximations, which result in many models being intermingled with others to form hybrid models. In general, restrictions on boundary conditions, in current propagation models, are by far one of the key challenges that must be addressed. As such, having an effective modeling tool to account for these complex restraints are a great asset in understanding wave propagation. In sum, these limitations are addressed as a CA model is devised to reduce the complexity of these limitations.

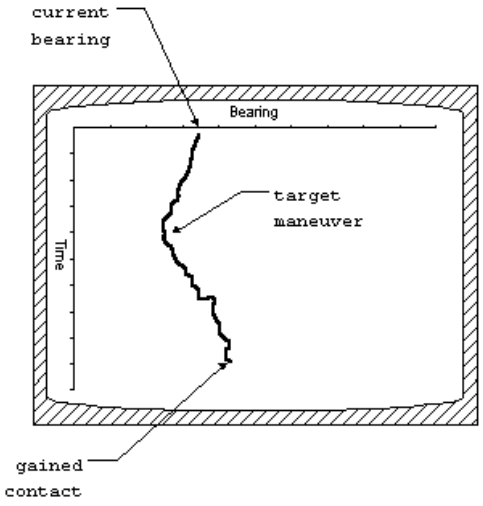


Figure 1. Sample of an onboard ship's sonar display. From [5]

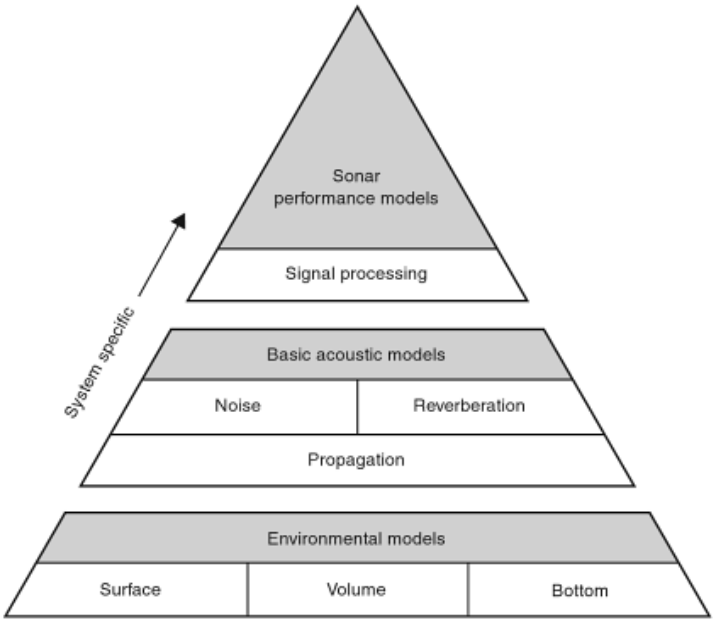


Figure 2. Generalized relationships among environmental models, basic acoustic modes, and sonar performance models. From [6]

B. CELLULAR AUTOMATON (CA) FUNDAMENTAL CONCEPTS

The theory behind the Cellular Automaton technique traces its roots back to the 1940s. The inventors, Stanislaw Ulam and John Von Neumann, at the time, were interested in conceptualizing the cellular method with crystal growth and the self-reproducing behavior of biological organisms [8]. The early developments in applying this concept later became prototype models for other complex systems in chemistry and biology. Likewise, a complex system is defined as a system that must be understood not just in terms of a set of components out of which it is constructed but the topology of the interconnections and interactions among those components. The study of fluid mechanics, dynamics, and crystal growth in materials are all notional examples of complex systems in mechanical engineering. However, the vast use of the cellular automaton technique has historically been confined to the area of applied mathematics, computer science, and traffic flow modeling [9], [10], [11].

Nevertheless, the strength of the CA technique is its straightforwardness and how it emulates the performance of a complex system with the application of a simple rule. Traditional ways in solving the complex nature of wave propagation have alluded to solving complex partial differential equations. As such, time evolvement is analyzed with the help of nonlinear partial differential equations. As a result, solutions to these types of problems are very complicated and tedious to solve. Moreover, incorrect results are produced due to rounding errors or wrong choice in the selection of initial conditions [12]. On the other hand, the cellular automaton provides an unconventional approach to study the behavior of the wave propagation. In essence, the CA method is by nature very simplistic and easier to solve in lieu of applying partial differential equations. Furthermore, the dynamics of this approach does not lead to instabilities when new interactions are added to the domain.

The discrete nature of the CA technique is summarized as follows. It defines a domain on which every point in the domain is governed by a set of rules. Each point in the domain assumes finite values once the rule is applied to that particular location. As time progresses, in discrete time iterations, the finite value at each individual point changes synchronously according to a rule that is being implemented at that particular time step. The nature of the rules being applied to each location is dependent upon the finite values contained by its associated neighbors. Figure 3 shows a sample depiction of the associated neighbors which affect the location of interest (the blue center) in 2D. This particular model, known as the Von Neumann model, depicts four neighbors as having an influence in affecting the value of the center location. If this is a 1D scenario, we only consider the immediate neighbors to the right and left, as shown in Ref [13]. Likewise for the 3D case, 6 neighbors are required (North, South, East, West, Front, and Back). While there are other modeling schemes which utilize more neighbors to affect the value of the center location, the Von Neumann scheme is discussed [14].

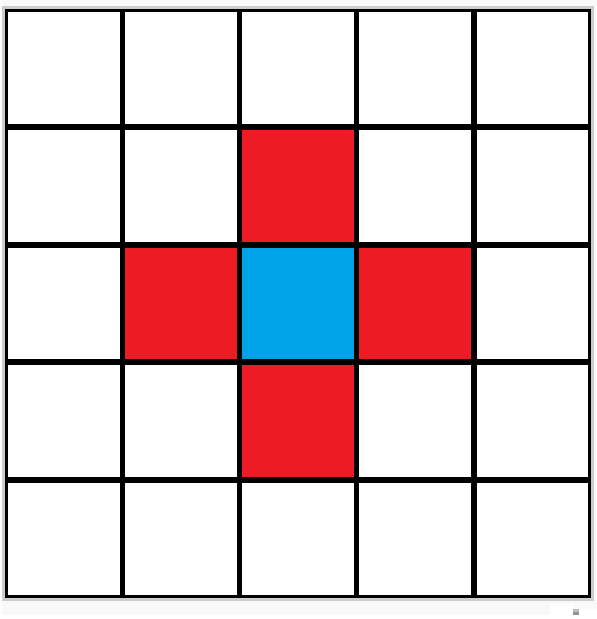


Figure 3. Von Neumann neighborhood domain consist of 4 cells surrounding the center cell

Overall, the beauty behind the Cellular Automaton technique is that it can produce a rich spectrum of complex patterns from the application of easy to apply rules. As such, the nature of these patterns captures the essence behind the behavioral response of complex systems.

C. BASIS OF METHODOLOGY FOR THE REMAINING CHAPTERS

The most recent research on the application of the CA method, in mechanical engineering, has traced its involvement in being coupled with other computational techniques, such as the Finite Element and Lattice Boltzmann methods [15]. In conjunction with the prior, complex domains for wave propagation are easily solvable as each method overcomes the weakness of the other, resulting in increases in computational efficiency. Moreover, advancements in applying this method to mechanical systems, such as spring mass systems, vibrational systems, and crack propagation scenarios, are introduced and studied to include forces into the CA domain. While there are few

drawbacks in modeling the prior schemes, one of the major limitations that has been overcome dealt with correlating the discrete nature of the CA scheme into the time domain [16]. This in turn opens the door for modeling more complex problems which may take on multiple dimensions.

The remaining chapters proceed in the following manner and build upon developments from prior research. Chapter II introduces a MATLAB coded model to replicate wave propagation with the Cellular Automaton technique. The model is explained and derived so that application in the remaining chapters is easily understood. In Chapter III, the model is implemented as a propagation model for underwater acoustics. The modeling scheme creates a generic profile of propagation losses, in terms of decibels (dB), versus range in an acoustic medium. In addition, visualization of wave propagation in a cellular domain, with complex boundaries, is animated to demonstrate their effects. The essences of all restraints, on each boundary, are not easily modeled by ordinary numerical techniques but easily modeled with the Cellular Automaton method. Chapter IV develops a multi-grid meshing technique to increase the computational efficiency in modeling a Cellular Automaton domain, while chapter V concludes with discussions and recommendations on how to improve the usefulness of this method.

II. WAVE MODELING WITH CELLULAR AUTOMATON IN MATLAB

A. INTRODUCTION

Matrix Laboratory (MATLAB) is an interactive software package used heavily in the engineering and science field. It provides an environment, with hundreds of built-in functions, to produce solutions to numerical and analytical problems. The beauty of this software is that it allows users to tailor a programming package for visual display. As such, this programming environment is utilized to customize functions in order to emulate propagation of pressure waves with the Cellular Automaton technique.

The remaining sections in this chapter elaborate on specifics of how each function is designed to work. In addition, information on the essentials of what information is inputted and outputted is explained. Lastly, important milestones within each function are depicted. To allow data to be easily readable within the pages of this thesis, a 2D Cellular Automaton space in a 15X15 domain is depicted. The code inherently takes into account a 3D scenario, thus, elaboration in that domain is limited for ease in understanding the model code.

B. DEFINE THE CA SPACE DOMAIN

Preliminary steps in creating a 2D domain, for the Cellular Automaton space, call for defining domain parameters and the point location for plot viewing (see Figure 4). The variables VN, HN, and ZN determine the lengths of the vertical, horizontal, and z-dimensions of the domain. The input source, into the domain, takes on a diameter specified by the user and stored in the variable *R_Nodes*, which for illustrative purposes are assumed to be 5. The variables *HS*, *VS*, and *ZS* define the horizontal, vertical, and z-direction shifts away from the middle of the domain. So, if the user likes to move the simulated input source to a different location, changing those particular values creates this effect. Negative

values correlate going to the left for HS and up for *VS,* while a negative value for *ZS* correlates to coming out of the plane.

```
1
2 -    VN = 15;        HN =15;       ZN =1;
3 -    R_Nodes=input(['\nPlease set the radial nodal distance',...
4                                                    ' from the source: ']);
5 -    VS =-2;         HS =0;        ZS = 0;
6 -    Plots=input(['\nPlease specify the point locations, inside the CA\n',...
7          ' space domain, where you like to see how a point is changing\n',...
8          ' with respect to time.  If you would like to see more than one\n',...
9          ' point please enter the information in matrix format.\n\n -- Ex)',...
10         ' [Row_1, Col_1, Z_1; Row_2, Col_2, Z_2;...] i.e \n----->']);
```

Figure 4. Important variable declaration in defining the dimensions and plot view location in the Cellular Automaton domain

After defining the dimensions of the domain, the next phase of the model calls for specifying the location in the domain where changes over time can be reviewed, while the CA domain is undergoing permutation. The model prompts the user to specify the coordinates *(in [Row, Col, Z] format)* of each point location. The variable *Plots* take the form of an array, if only one point is requested for viewing. If the user likes to analyze multiple points, entering the points in matrix format allows for such. Row 1 contains information on point 1; row 2 contains information on point 2, and so forth. For the purpose of demonstrating the technique, only one point has been selected with the following coordinates: [3, 5, 1]. Overall, remaining values of variables defined in Figure 4 are used in developing preliminary information to analyze a CA scheme.

C. INITIALIZE THE SOURCE DIAMETER VALUES

The prior section lays the ground work for creating the cellular automaton domain. In this section, information that has been provided by the user in the previous section is utilized, in order to insert the associated space in MATLAB. Functions are called upon to refine the initial values of each cell location prior to

cell permutation, which are discussed in the next section. The specifics of those customize functions are described below:

1. Circle Sink Source Function

The *CircleSinkSource* function is the first step in initializing the domain in 2D/3D. The specifics of its syntax are shown in Figure 5. Essentially, this function receives information on the number of nodes and any associated shifts in the source term from the middle of the domain. The first three lines create an array vector, with integer values, spanning from half of the number of nodes in each direction, rounded to the next higher number, to the complement of each associated value. MATLAB built-in *meshgrid* function creates a grid of values as shown in Table 1. Line 8 in this function executes the equation of a circle to formulate a grid space where the value of each grid space assumes a value that is the square of its radial distance from the center. Also note the effects of the variable *VS* from Figure 4. This allows for the radial distribution of values, in the domain, to shift its focal center up two units (see Table 2).

```
11
12 -    CAS = CirSinkSource(VN, HN, ZN, VS, HS, ZS);
```

```
1      function [CAS] = CirSinkSource (VN, HN, ZN, HS, VS, ZS)
2
3 -    x=-floor(VN/2):floor(VN/2);
4 -    y=-floor(HN/2):floor(HN/2);
5 -    z=-floor(ZN/2):floor(ZN/2);
6
7 -    [X, Y, Z] = meshgrid(x,y,z);
8 -    CAS=(X-HS).^2 +(Y-VS).^2 + (Z-ZS).^2;
9
10 -   end
```

Figure 5. Initial step in creating the diameter of the domain, in matrix format, in MATLAB

X =

-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7

Y =

-7	-7	-7	-7	-7	-7	-7	-7	-7	-7	-7	-7	-7	-7	-7
-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6
-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5
-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3
-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7	7	7	7	7	7

Table 1. This is a grid of X & Y values following the execution of the meshgrid command in the Circle Sink Source function.

CAS =

74	61	50	41	34	29	26	25	26	29	34	41	50	61	74
65	52	41	32	25	20	17	16	17	20	25	32	41	52	65
58	45	34	25	18	13	10	9	10	13	18	25	34	45	58
53	40	29	20	13	8	5	4	5	8	13	20	29	40	53
50	37	26	17	10	5	2	1	2	5	10	17	26	37	50
49	36	25	16	9	4	1	0	1	4	9	16	25	36	49
50	37	26	17	10	5	2	1	2	5	10	17	26	37	50
53	40	29	20	13	8	5	4	5	8	13	20	29	40	53
58	45	34	25	18	13	10	9	10	13	18	25	34	45	58
65	52	41	32	25	20	17	16	17	20	25	32	41	52	65
74	61	50	41	34	29	26	25	26	29	34	41	50	61	74
85	72	61	52	45	40	37	36	37	40	45	52	61	72	85
98	85	74	65	58	53	50	49	50	53	58	65	74	85	98
113	100	89	80	73	68	65	64	65	68	73	80	89	100	113
130	117	106	97	90	85	82	81	82	85	90	97	106	117	130

Table 2. The final defined nodal values of the Cellular Automaton Space (CAS) after the execution of the Circle Sink Source Function

2. Retrieving the Acoustic Source Indices Function

The *AcousticIndices* function returns the indices of the radial input source specified in Figure 4, line 3 (see Figure 6). Essentially, the radius of the source is centered on the zero point in Table 2 and spans in directions specified by the *R_Nodes* variable. The variable *R_Nodes* is to be 5. As such, execution of line 3 results in lines 9 and 10 being executed. The variable *CAS_Diameter* assumes the form shown in Table 3. In essence, all 1's define the location of the radial source in domain. As a result, line 16 captures and stores those particular index locations. On the other hand, line 17 creates a series of 1's to correlate to the nodes that are outside of the desire radius (see Table 4). This in turn makes it easy to capture the location of indices external to the source. Lastly, if a logical OR is applied to Table 3 and Table 4, as shown in line 19, this allows for gathering the indices of the nodes internal to the source diameter. In later sections of the code, the locations of nodes internal to the source are known because they must remain zero in order to simulate an acoustic source with the CA technique. This function concludes by passing all indices external to the source, along the source, and internal to the source back to the main script for future processing (see lines 16, 18, 20, 22, & 23).

```
13
14 -    [Indices,OC,OR,OZ] = AcousticIndices(CAS,R_Nodes);
```

```
 1      function [Indices,OC,OR,OZ]=AcousticIndices(CAS,R_Nodes)
 2
 3 -    R_Nodes=ceil(R_Nodes/2)-1;
 4 -    if R_Nodes<=-1
 5 -        error(['Please set radial nodal distance from the source to be be',...
 6              ' equal to 1 or greater'])
 7 -    elseif R_Nodes == 0
 8 -        CAS_Diameter = CAS==1;
 9 -    elseif R_Nodes==1
10 -        CAS_Diameter = CAS==1 | CAS==2;
11 -    else
12 -        CAS_Diameter = CAS==(R_Nodes.^2)   | CAS==(R_Nodes.^2 +1) | ...
13                                                  CAS==(R_Nodes.^2 -1);
14 -    end
15
16 -    [I.DRow, I.DCol, I.DZ]=ind2sub(size(CAS_Diameter),find(CAS_Diameter==1));
17 -    CAS_OuterValues = CAS>=(R_Nodes).^2+2;
18 -        [OR,OC,OZ] = ind2sub(size(CAS_OuterValues),find(CAS_OuterValues==1));
19 -    CAS_InnerZeros=CAS_OuterValues|CAS_Diameter;
20 -        [RowZero, ColumnZero, ZZero]=ind2sub(size(CAS_InnerZeros),...
21                                                  find(CAS_InnerZeros==0));
22 -    I.RowZero=RowZero;          I.ColZero=ColumnZero;       I.ZZero=ZZero;
23 -    Indices=I;
24
25 -    end
```

Figure 6. Storing the indices of nodes external to, internal to, and along the source diameter

CAS_Diameter =

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 3. All 1's correlate to the CAS a source diameter. All zeros external to the circle of 1's correlate to nodes external to the acoustic source.

CAS_OuterValues =

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	0	0	0	1	1	1	1	1	1
1	1	1	1	1	0	0	0	0	0	1	1	1	1	1
1	1	1	1	1	0	0	0	0	0	1	1	1	1	1
1	1	1	1	1	0	0	0	0	0	1	1	1	1	1
1	1	1	1	1	1	0	0	0	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 4. All ones correlate to nodes external to the acoustic diameter source.

3. Time Parameter Calculations

The nearest node can have a non-zero value after the third iteration [16]. As such, Equation (1) derives from that scheme and illustrates the amount of time passed after each iteration, or application of each rule in the CA scheme.

The variables dx, dy, and dz correlate to the distance between two nodes in the vertical, horizontal, and 3rd dimension. The function *CA_Time_Conversion* executes Equation (1) and returns the associated time increment, whenever the rule is applied (see Figure 7). For simplicity, each node is equally spaced with a distance of dx=dy=dz.

$$\triangle time = \frac{\sqrt{dx^2 + dy^2 + dz^2}}{3(speed\ of\ sound)} \tag{1}$$

```
16      speed_sound=50;
17      dx=1;
18      dt = CA_Time_Conversion (speed_sound,CAS,dx);
```

```
1       function [dt] = CA_Time_Conversion (SS,CAS,dx)
2
3 -     [L1, L2, L3]=size(CAS);
4 -        if (L1>1 && L2>1 && L3>1)
5 -             ND = 3;
6 -        else
7 -             ND = 2;
8 -        end
9 -        dt = dx/(sqrt(ND)*SS);
10 -    end
```

Figure 7. Defining spacing between nodes and the speed of sound to be utilized in the wave equation

4. Circle Value Function

The *CircleValFunc* function assumes the input source has a periodic waveform (see Figure 8). As such, the user inputs the desire frequency of oscillations, the associated peak, and a decay constant if the user desires for the acoustic source to decay. After the conclusion of this function, all values, which correlate to the input acoustic source pressure, are returned into the variable *Value.* The nature of the discrete pressure values return is a function of *dt* and the number of iterations. For this example, the assumption is the following: *Iterations* = 75, *freq* = 5 Hz, *Decay* = 1 and *Peak* = 10.

```
19
20 -    iterations=75;
21 -    [Value,time] = CircleValFunc(dt,iterations);
```

```
1      function[Values,time] = CircleValFunc(dt,iterations)
2
3 -    time = 0:dt:dt*(iterations-1);
4 -    freq = input(['\nPlease enter the frequcency of the sinusoidal source',...
5                    ' (in Hertz).\n If you desire a constant frequency enter 0: ']);
6 -    Decay = input('\nPlease enter the decay constant: ');
7 -    Peak = input('\nPlease enter the peak value of the input source: ');
8
9 -    Values(1:length(time))= Peak.*cos(2*pi.*freq.*time).*exp(-Decay.*time);
10
11 -   end
```

Figure 8. Defining the initial radial value of the source

5. Acoustic Radius Function

The acoustic pressure values have been determined for all iterations. The CA domain with the initial pressure is utilized (see Figure 9). This particular function serves to reinforce the diameter values to follow the pressure values store in the variable *Values*. This is a key function because every node in the domain changes once a rule is applied. This function essentially resets the nodal diameter after the application of a nodal rule. This is made possible because the indices of the source diameter are stored in memory as shown in Figure 6. Lastly, the *AdditionalConstraints* function resets all internal nodes to 0 and produces the CA domain that is shown in Table 5.

CAS =

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	10	10	10	0	0	0	0	0	0
0	0	0	0	0	10	0	0	0	10	0	0	0	0	0
0	0	0	0	0	10	0	0	0	10	0	0	0	0	0
0	0	0	0	0	10	0	0	0	10	0	0	0	0	0
0	0	0	0	0	0	10	10	10	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5. The initial CA space prior to the application of any rule that correlates to a peak input acoustic pressure of 10.

```
22
23 -   CAS =  AcousticRadius(CAS,Indices,Value,1);
```

```
function [CAS] =  AcousticRadius(CAS,Indices,Value,index)

    D=Indices;
    for i=1:length(D(1).DRow)
      CAS(D(1).DRow(i),D(1).DCol(i),D(1).DZ(i)) = Value(index);
    end

end
```

```
24 -   CAS = AdditionalConstraints(CAS,Indices);
```

```
function [CAS] = AdditionalConstraints(CAS,Indices)

I=Indices;
    for i=1:length(I(1).RowZero)
        CAS(I(1).RowZero(i),I(1).ColZero(i),I(1).ZZero(i))=0;
    end

end
```

```
25 -   for i=1:length(OC)
26 -       CAS(OR(i), OC(i), OZ(i))=0;
27 -   end
```

Figure 9. Applying the radial source value to the Cellular Automaton domain (line 23) and setting all internal (line 24) and external (lines 25–27) nodes equal to 0.

D. PERMUTATION OF THE CELLULAR AUTOMATON DOMAIN

Rules have been applied to each location in the Cellular Automaton domain. The functions below execute the following relation in Equations (2) and (3) [15], [16]. Equation (2) is applied when the domain is in 2D and Equation (3) is applied to 3D. The variables listed in those equations are determined as follows: N = Northern Node, S = Southern Node, W = Western Node, E = Eastern Node, F = Frontal Node in the 3rd dimension, and B = Backwards Node of the 3rd dimension. The conclusion of a calculation results in that particular domain location being updated.

$$NodeValue\left(t+\triangle t\right)=\frac{N\left(t\right)+S\left(t\right)+E\left(t\right)+W(t)-NodeValue(t-\triangle t)}{2} \quad (2)$$

$$NodeValue\left(t+\triangle t\right)=\frac{N\left(t\right)+S\left(t\right)+E\left(t\right)+W\left(t\right)+F\left(t\right)+B\left(t\right)-3NodeValue\left(t-\triangle t\right)}{3} \quad (3)$$

Rule 1:	Apply Equation (2) (or if 3D Equation (3)) to all odd number cells
Rule 2:	Apply Equation (2) (or if 3D Equation (3)) to all even number cells

Table 6. Associated rules to execute wave propagation in a cellular automaton domain

As such, the associated permutation rule being applied to this domain is defined in Table 6. The odd number cells are referenced as white-nodes, while the even cell nodes are black-nodes. Nodes along the boundary of the domain are given certain restraints to fully define the permutation in the domain. Therefore, Rule 1 and 2 do not apply along those locations. Application of each rule correlates to an iteration value define in Figure 8. Moreover, the application of a rule is also associated with a change in time, which is defined by Equation (1). The implementation of such rules and boundary constraints in MATLAB are illustrated.

1. Permute the Cellular Automaton Domain Function

Information that has been passed to the *PermuteCASpace* function (see Figure 10) derives in section II.B and II.C. Initial inputs for this sub function ask the user to specify the associated boundary conditions along each boundary of the domain. Once the user confirms the boundary constraints, all initial values that pass by the *CAS* variable are stored in the 4D matrix *Positions* (see Figure 11) prior to the first permutation of white nodes. The 4th dimension correlates to the iteration being executed. So as a recap, the variable *Positions* contain data from CA domain, which currently has the state shown in Table 5.

```
28
29 -    DomainData=PermuteCASpace(CAS,Indices,iterations,Value);
```

```
1      function [Position]= PermuteCASpace(CAS,Indices,iterations,Value)
2
3 -    OPTIONS = [' -Please enter a number from  the options below to specify',...
4          ' the \n constraint:\n  (1) Fix\n  (2) Reflective \n  (3) ',...
5          ' Free/Constant Flux',...
6          ' Conditions\n  (4) Non-Reflective\n  (5) Refractive....If 5 is',...
7          ' selected please enter an array in \n the following format',...
8          ' [5, A Number between 0 and 1]...The 2nd value\n will correlate to',...
9          ' the percentage reflected do to the wave arriving\n at an incident',...
10         ' angle  \n '];
11 -   OPTIONS2 = ' (6) Misc. Hill \n  ';
12 -   constraint1 = input(['LEFT BOUNDARY\n',   OPTIONS]);
13 -   constraint2 = input(['RIGHT BOUNDARY\n',  OPTIONS]);
14 -   constraint3 = input(['TOP BOUNDARY\n',    OPTIONS,OPTIONS2]);
15 -   constraint4 = input(['BOTTOM BOUNDARY\n', OPTIONS,OPTIONS2]);
```

Figure 10. The section of model will specify the constraints along the domain boundary…(some code omitted for brevity, continued in Figure 11)

```
21
22 -    Position(:,:,:,1:iterations)= zeros(L1,L2,L3,iterations);
23 -    s=1;
24 -    while 1
25 -        Position(:,:,:,s) = CAS;
26 -        CAS = CA_WhiteInnerNodes(CAS);
```

Figure 11. Data from the CA domain store for future processes before permuating the white inner node…(some code omitted for brevity continued in Figure 14)

a. Permute White Inner Nodes

The beginning section of the *CA_WhiteInnerNodes* function (see Figure 12) creates Equation (2) as an inline function (see line 5). Since the case is a 2D scenario, it passes the test presented in line 4. Otherwise, it skips to the omitted else statement not shown in Figure 12 for the 3D case. Nevertheless, the domain is reshaped into an array vector for ease in understanding the associated indexing relationship; column 1 is the first set of entries followed by column 2 entries, etc. (see line 6). The key parameter in this indexing scheme is the variable *i*. The model traverses the CA space column by column, starting from row 1 (see line 8 and 9). Rule 1 applies to odd cells. Therefore, line 11 determines if the variable *i,* which is initially 1 and correlates to row =1 and column =1, is at an odd or even cell. However, prior to reaching this inner if-statement, the variable must past the test of not being on the boundary (see line 10). If the variable *row* equals 1, the conditional if-statement fails. As a result, for the dimension of the space specified in this example, the variable *i* equals 17 before access to the MATLAB function *rem* is allowed. Since *rem(17,2) =1*, line 10 is true and allows for execution of the code within that loop. Once nested inside the loop, the variable *CAS_Modified* updates the associated index, in the array vector which correlates to the original CA domain (see lines 15 and 6). This process continues until all odd cells in the domain have been processed. The

end result is the *CAS_Modified* array vector being transformed into the same original shape of the CA domain (see line 21 and 22); Figure 12 depicts the prior.

```
1      function CA_Space_White = CA_WhiteInnerNodes(CAS)
2
3 -    [Length1, Length2, Length3]=size(CAS);
4 -    if (Length1==1 || Length2==1 || Length3==1)
5 -        New_C = @(W, E, S, N, Old_C) (1/2) * (W +E +S + N -2*Old_C);
6 -        CAS_Modified(1,:)=reshape(CAS,1,Length1*Length2);
7 -    i=1;
8 -      for column=1:(Length2)
9 -        for row=1:(Length1)
10 -            if (((row>1)&&(column>1)) && ((row<Length1)&&(column<Length2)))
11 -               if rem(i,2)
12 -                   W=CAS(row,column-1);   E=CAS(row,column+1);
13 -                   N=CAS(row-1,column);   S=CAS(row+1,column);
14 -                   old_C=CAS(row,column);
15 -                   CAS_Modified(i)=New_C(W, E, S, N, old_C);
16 -               end
17 -            end
18 -          i=i+1;
19 -        end
20 -      end
21 -   CAS_Modified2(1,1:Length1*Length2)=CAS_Modified(1,1:Length1*Length2);
22 -   CA_Space_White= reshape(CAS_Modified2, Length1, Length2);
```

Figure 12. Permutation of the White Inner Nodes …(some code omitted for brevity)

CAS =

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	5	0	5	0	0	0	0	0	0
0	0	0	0	0	10	10	0	10	10	0	0	0	0	0
0	0	0	0	5	10	10	0	10	10	5	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	5	10	10	0	10	10	5	0	0	0	0
0	0	0	0	0	10	10	0	10	10	0	0	0	0	0
0	0	0	0	0	0	5	0	5	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7. The status of the Cellular Automaton domain after applying Rule 1

b. Cellular Automaton Boundary conditions Function

The concept of how to apply boundary conditions using the CA technique is developed for one dimension in reference [17]. It is suggested that boundary conditions be generated by adding virtual cells along the boundary (see Figure 13). Recall in Figure 10, the user specifies the boundary conditions for each boundary. For simplicity, the assumption is that the user selects fixed boundary conditions for all boundaries. Figure 14 illustrates the left boundary (see line 4 and 5), but the same concept is applied to all boundaries. If the user wants to specify a boundary with a shape that is not flat, the indices for that boundary need to pass to this function.

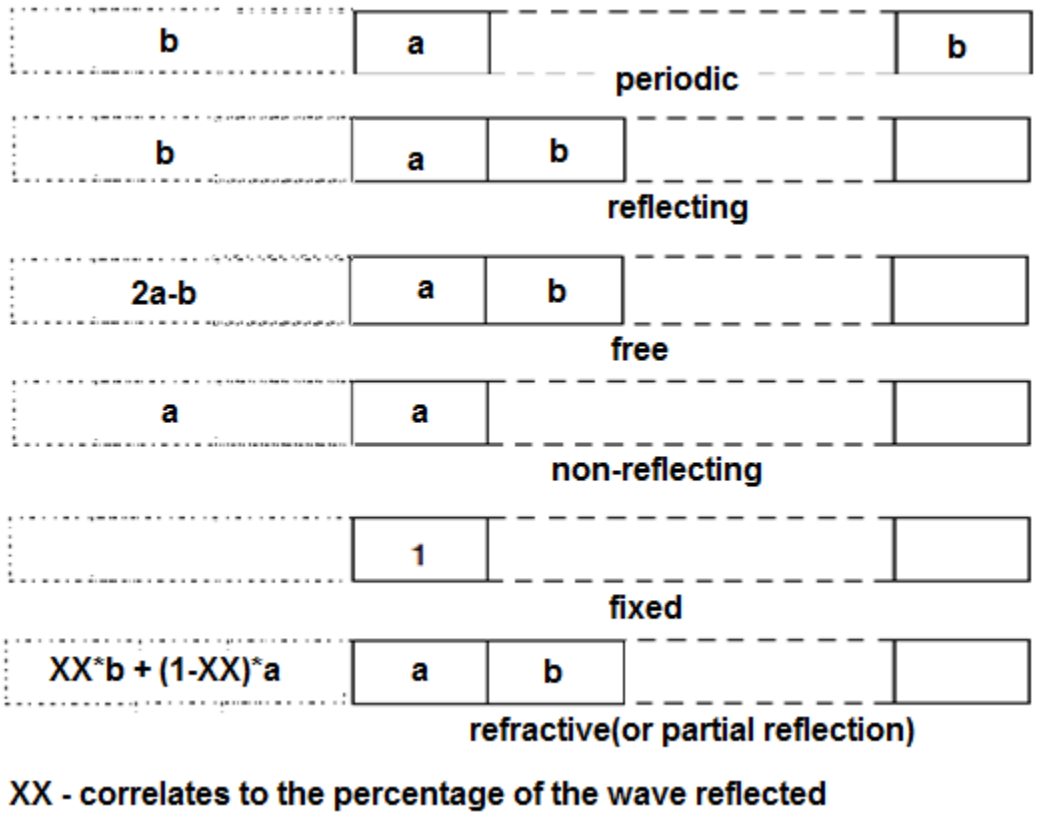


Figure 13. One dimension case of applying boundary conditions with the CA technique. After [17]

Lines 12–14 of Figure 14 address wave propagation that undergoes diffraction and reflection. For example, if a wave is simulated, arriving at an incident angle where only 25% of the wave front is reflected, the user enters an array in specifying the *constraint1* variable (see line 27 of Figure 14). The correct entry, for runtime selection of *constraint1* in Figure 10, is [5, 0.25]. The creation of lines 12–14 (see Figure 14) is a result of combing contributions from reflective and non-reflective boundary conditions in Figure 13. Overall, the selection of the appropriate boundary conditions is straight forward and calls for the user to specify the *side* and *constraint* variable in the Figure 10 model.

```
27 -     CAS = CA_BoundaryConstraints(CAS, Indices, 'left',   constraint1);
```

```
 1     function [CAS] = CA_BoundaryConstraints(CAS, Indices, side, constraint)
 2
 3 -   if  strcmp(side, 'left')
 4 -        if constraint(1) == 1          % (Fixed Conditions)
 5 -           CAS(:,1,:) = 0;
 6 -        elseif constraint(1) == 2      % (Reflective Conditions)
 7 -           CAS(:,1,:) = CAS(:,3,:);
 8 -        elseif constraint(1) == 3      % (Free/Constant Flux Conditions)
 9 -            CAS(:,1,:) = 2.*CAS(:,2,:) -  CAS(:,3,:);
10 -        elseif constraint(1) == 4      % (Non-Reflective Conditions)
11 -           CAS(:,1,:) = CAS(:,2,:);
12 -        elseif constraint(1) == 5      % (Refractive Conditions ...Flat Bottom)
13 -           CAS(:,1,:) =  CAS(:,3,:).*(constraint(2)) + ...
14                                                CAS(:,2,:).*(1-constraint(2));
15 -        end
```

Figure 14. Application of selected boundary conditions across the domain…(some code omitted for brevity)

c. Additional Constraints and Acoustic Source Radius Reinforced

At this point in the model, the additional constraints and reinforcement are applied to values along the acoustic source radius. For the scenario presented thus far, the nodes internal to and along the source diameter are the only nodes updated. As a side note, if an arbitrary constraint is placed in the domain, this section in the model is the appropriate time for its inclusion. Nevertheless, after calling the *AcousticRadius* function (see Figure 15), Table 8 reflects the CA domain status. In addition, Table 9 shows the status of the CA domain after applying the *AdditionalConstraints* function. Recall the value of the variable *s* is still 1 in line 36 of the *PermuteCASpace* function (see Figure 15). Moreover, the test of comparing the *s* variable to the *iterations* variable is applied. If the variable *s* exceeds the number of iterations, the model exits the *PermuteCASpace* function and returns to the main script. On the other hand, not exiting results in the permutation being stored in the variable *Positions.*

```
CAS = AcousticRadius(CAS,Indices,Value,s);
CAS = AdditionalConstraints(CAS,Indices);

s=s+1;
if s > iterations
   return
end
Position(:,:,:,s) =CAS;
```

Figure 15. Application of additional constraints and upddating the acoustic source radius values …(some code omitted for brevity)

CAS =

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	5	0	5	0	0	0	0	0	0
0	0	0	0	0	10	10	10	10	10	0	0	0	0	0
0	0	0	0	5	10	10	0	10	10	5	0	0	0	0
0	0	0	0	0	10	0	0	0	10	0	0	0	0	0
0	0	0	0	5	10	10	0	10	10	5	0	0	0	0
0	0	0	0	0	10	10	10	10	10	0	0	0	0	0
0	0	0	0	0	0	5	0	5	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8. The status of the Cellular Automaton domain after re-enforcing the source radial values

CAS =

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	5	0	5	0	0	0	0	0	0
0	0	0	0	0	10	10	10	10	10	0	0	0	0	0
0	0	0	0	5	10	0	0	0	10	5	0	0	0	0
0	0	0	0	0	10	0	0	0	10	0	0	0	0	0
0	0	0	0	5	10	0	0	0	10	5	0	0	0	0
0	0	0	0	0	10	10	10	10	10	0	0	0	0	0
0	0	0	0	0	0	5	0	5	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 9. The resultant Cellular Automaton domain after completing the first iteration with permutation of the white nodes (or odd cells)

d. Permute Black Inner Nodes

The code for executing the permutation of the black (even) nodes is the same as for the white nodes with the following exception. Line11 of Figure 12 is replaced with a conditional test of ~rem(i,2). Therefore, line 11 first evaluates the inner statements (lines 12–15) when the variable *i* equals 18. The results of applying all boundary conditions and additional constraints lead to Table 10. The results of Table 10 are stored in the variable *Positions* when variable *s,* in line 39–42 of Figure 15, is less than the number of iterations.

CAS =

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	2.5000	0	2.5000	0	0	0	0	0	0
0	0	0	0	0	7.5000	5.0000	10.0000	5.0000	7.5000	0	0	0	0	0
0	0	0	0	7.5000	10.0000	8.9024	8.9024	8.9024	10.0000	7.5000	0	0	0	0
0	0	0	2.5000	5.0000	8.9024	0	0	0	8.9024	5.0000	2.5000	0	0	0
0	0	0	0	10.0000	8.9024	0	0	0	8.9024	10.0000	0	0	0	0
0	0	0	2.5000	5.0000	8.9024	0	0	0	8.9024	5.0000	2.5000	0	0	0
0	0	0	0	7.5000	10.0000	8.9024	8.9024	8.9024	10.0000	7.5000	0	0	0	0
0	0	0	0	0	7.5000	5.0000	10.0000	5.0000	7.5000	0	0	0	0	0
0	0	0	0	0	0	2.5000	0	2.5000	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10. The resultant Cellular Automaton domain after completing the second iteration with permutation of the black nodes (or even cells)

2. Visual Representation of Cellular Automaton Permutation

The results of applying the Cellular Automaton technique to the domain modeled in this chapter are illustrated in Figure 17 and Figure 18. The elapse time, in the title portion of the graph, is based on Equation (1). The specific plot location in Figure 4 shows how this particular point changes over time; Figure 17 depicts this point. In conclusion, this chapter develops a model script to illustrate how to utilize the Cellular Automaton technique in MATLAB. Application of this devised technique is applied in the next chapter to an ocean environment. In addition, this scripted technique illustrates a concept of multi-grid propagation.

The Appendix contains the full version of this model code in its entirety and also provides an animation function to visually see the wave propagation over time.

```
31 -    CA_DomainPointPlots(Plots,DomainData,time)
32 -    CA_3D_and_VolumePlot(DomainData,dt)
```

Figure 16. Main ccript commands to produce the 3D and point location plots shown in Figure 17 and Figure 18

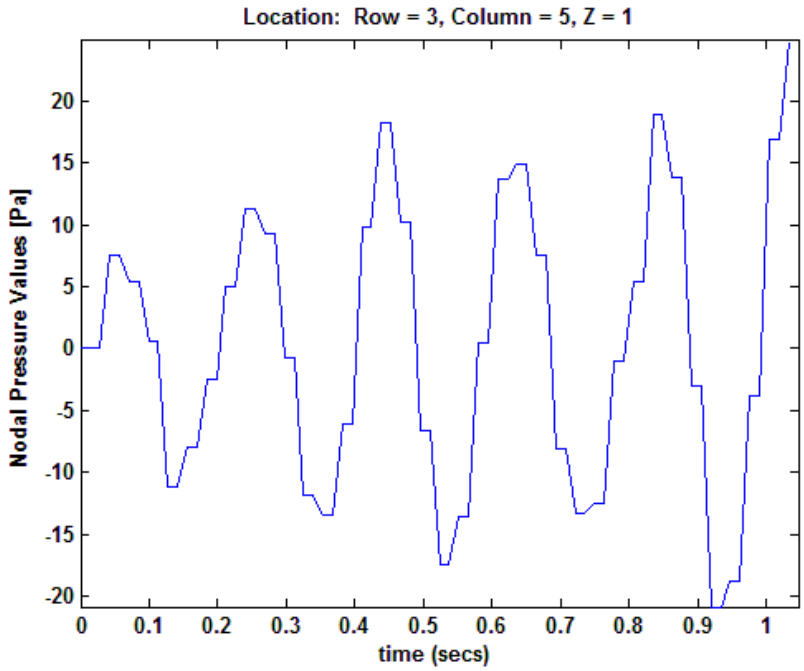


Figure 17. Evaluation of how location [3,5,1] changes, in a 15X15 domain, with fixed boundary conditions. A total of 75 iterations were applied to black and white nodes

```
DomainData(:,:,1,75) =

         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0
         0    5.1674    7.8194   16.2490   11.1406    9.7854    5.0707    6.9159    5.0707    9.7854   11.1406   16.2490    7.8194    5.1674         0
         0    6.3209   20.4571   20.8265   24.8390   13.8637   10.1893    4.9060   10.1893   13.8637   24.8390   20.8265   20.4571    6.3209         0
         0   14.7724   23.4281   34.6906   22.8609   14.1687    1.8728    1.8728    1.8728   14.1687   22.8609   34.6906   23.4281   14.7724         0
         0   15.0906   31.4564   29.2902   24.3691    1.8728         0         0         0    1.8728   24.3691   29.2902   31.4564   15.0906         0
         0   18.5097   25.9279   33.9835   20.4044    1.8728         0         0         0    1.8728   20.4044   33.9835   25.9279   18.5097         0
         0   13.8510   29.0696   24.9176   17.7218    1.8728         0         0         0    1.8728   17.7218   24.9176   29.0696   13.8510         0
         0   12.2998   17.8122   23.9213   11.3816    4.9543    1.8728    1.8728    1.8728    4.9543   11.3816   23.9213   17.8122   12.2998         0
         0    5.7487   16.0410   10.4133    9.0026    0.2185   -6.2185   -1.7772   -6.2185    0.2185    9.0026   10.4133   16.0410    5.7487         0
         0    4.7068    3.3351    7.9933   -0.1307   -7.0435   -9.4031  -16.4992   -9.4031   -7.0435   -0.1307    7.9933    3.3351    4.7068         0
         0   -0.8000    1.6088   -1.6828   -8.0220  -10.2846  -20.6962  -14.4773  -20.6962  -10.2846   -8.0220   -1.6828    1.6088   -0.8000         0
         0    1.2442    2.0402   -5.3628   -5.5654  -16.4307  -14.0316  -25.6459  -14.0316  -16.4307   -5.5654   -5.3628    2.0402    1.2442         0
         0    1.1079   -0.7316   -0.9665  -12.2839   -9.1452  -19.7431  -10.9488  -19.7431   -9.1452  -12.2839   -0.9665   -0.7316    1.1079         0
         0   -1.5318    0.1325   -4.0506   -1.6145   -8.6730   -6.4820  -11.2900   -6.4820   -8.6730   -1.6145   -4.0506    0.1325   -1.5318         0
         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0
```

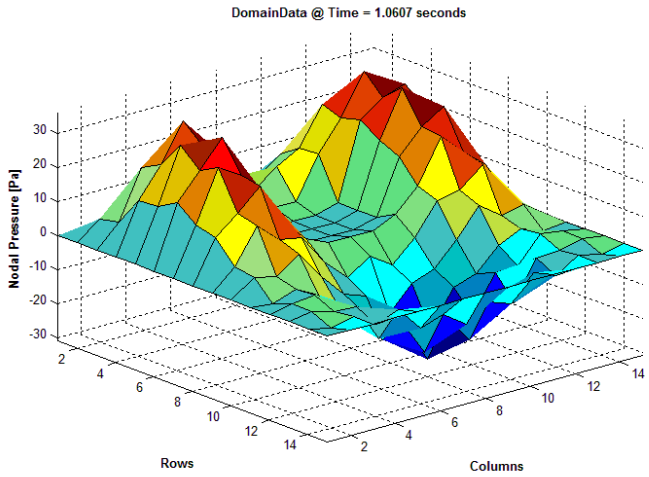


Figure 18. Cellular Automaton 2D (15X15) domain values after a total of 75 iterations on the black and white nodes.

THIS PAGE INTENTIONALLY LEFT BLANK

III. APPLICATION TOWARDS UNDERWATER ACOUSTICS

A. INTRODUCTION

An acoustic model of wave propagation enlarges our understanding of sound travel in water. The basis of these propagation models falls into one of the following five categories which are based on different variance of the wave Equation [6]:

1) Ray Theory
2) Normal Mode
3) Multipath Expansion
4) Fast Field (Wave Number Integration)
5) Parabolic Equation

Current selections of propagation models are limited in their application due to constraints presented in the domain [6]. Many propagation models circumvent this problem by establishing a hybrid modeling scheme [18]. The prior modeling approach links several different models into one, generating solutions in an *if-elseif* relational sense, to solve complex problems with various domain features. Nevertheless, the remaining portions of this chapter illustrate transmission losses due to wave propagation as a function of frequency and domain constraints. Examination of how the wave propagates under various ranges of frequency, different boundary conditions, and terrain shapes are discussed with visual depiction of 2D and 3D propagation.

B. TRANSMISSION LOSS DEVELOPMENT

Sound propagation in water undergoes a reduction in intensity as it travels between points. Spreading and attenuation are mechanisms used to quantify the reduction in sound as it travels over a distance. By definition, propagation losses are calculated by Equation (4) where the subscript "*1"* refers to the intensity at one meter and "*r"* references an arbitrary length greater than one meter [19].

For this point, the domain is homogenous and the speed of sound is constant throughout. As result, Equation (4) reduces to the form shown in Equation (6).

$$Propagation\,Losses\ (PL) = 10\log\left(\frac{Intensity_1}{Intensity_r}\right) \qquad (dB) \qquad (4)$$

where:

$$Intensity = \frac{(Pressure\ Amplitude)^2}{density_{of\ seawater} * speed\ of\ sound_{in\ sewater}} \qquad (5)$$

FIGURES	CODE LINE	VARIABLE	NEW VALUE
Figure 4	Line 2	VN	101
	Line 2	HN	101
	Line 3	R_Nodes	4
	Line 5	VS	0
	Line 6	Plots	[35, 65, 1]
Figure 7	Line 16	speed_sound	1500
Figure 8	Line 20	Iterations	300
	Line 4	freq	800
	Line 6	Decay	0
Figure 10	Line 12-15	constraint{1-4}	4

Table 11. Revised model modfication in simulating propogation losses

$$Propagation\,Losses\ (PL) = 20\log\left(\frac{Pressure\ Amplitude_1}{Pressure\ Amplitude_r}\right) \qquad (dB) \qquad (6)$$

As a way of illustrating propagation losses, a larger domain is established with revised parameters from Figure 4, Figure 7, Figure 8, and Figure 10, as shown in Table 11. The remaining features of the model update and perform in a manner as described in Chapter II. Since the source peak value is known, this value is implemented for the pressure amplitude at one meter in Equation (6). An additional script needed to reproduce the propagation losses based on peak

amplitude values is located in the Appendix. The particular script is called *TransmissionLossPeak* and produces results in both 2D and 3D.

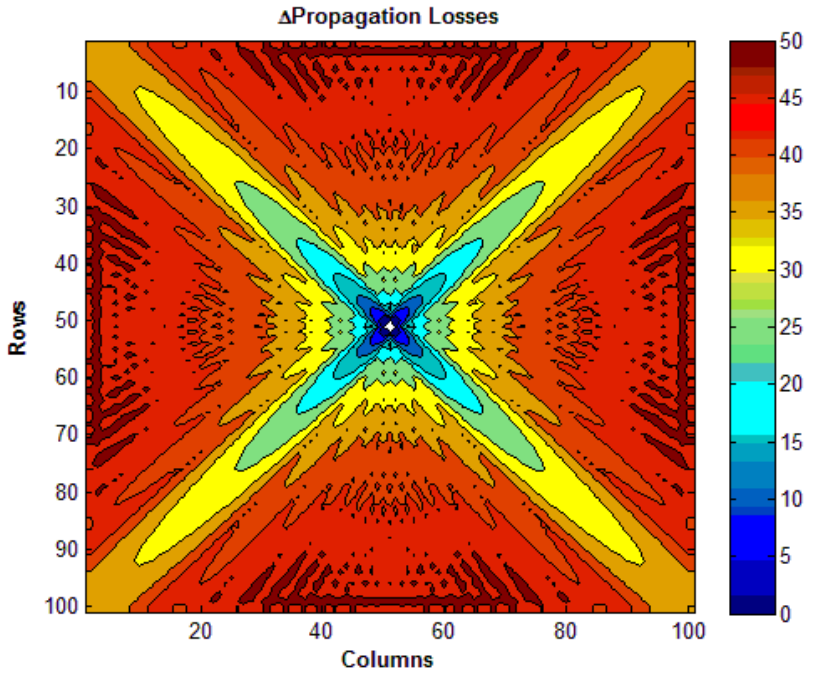


Figure 19. Propagation losses based off peak amplitude pressure values. Non-Reflecting boundary conditions were applied to all sides

When the revised model is executed based on changes made in Table 11, in addition to calling the *TransmissionLossPeak* function, depiction of propagation losses are displayed in Figure 19. The losses in this table are based off Equation (6). Figure 20 extracts a point, with the following coordinates [35, 65, 1], from Figure 19 and illustrates how the pressure changes as a function of time. From looking at Figure 20, the peak value of nodal pressures changes significantly around time 0.015 seconds. As a result, calculating propagation losses solely on peak values do not give an accurate depiction of the predominant values seen at that point throughout all the iterations. In order to correct this issue, take the root mean square (RMS) pressure from all iterations. Calculation of the RMS pressure at point [35, 65, 1] gives a value of 0.2830 [Pa].

This has a big effect in calculating propagation losses seeing that a peak of 2.709 [Pa] is originally observed in Figure 20. As a result of making this change, Figure 21 depicts how propagation losses are affected for every location in the domain. In turn, modification to Equation (6) results in the RMS values being used instead of peak values in the calculation of propagation losses (see Equation (7)).

$$Propagation\,Losses\ (PL) = 20\log\left(\frac{RMS\ Pressure\,of\ Source}{RMS\ Pressure_r}\right) \qquad (dB) \tag{7}$$

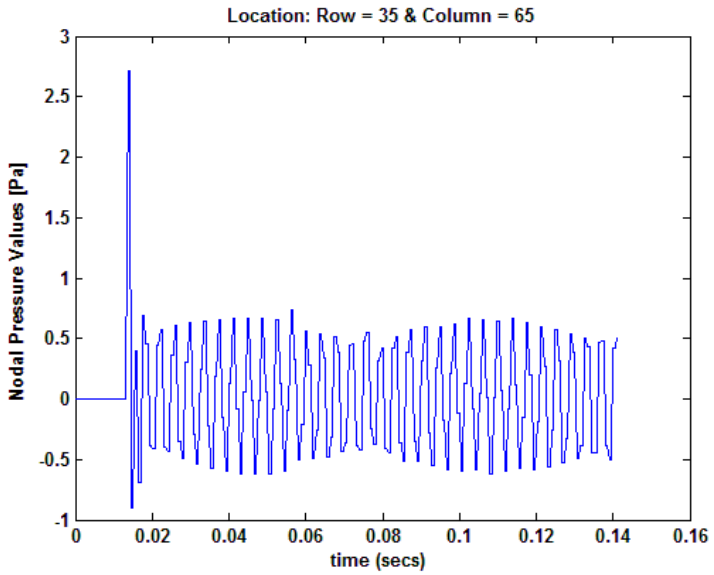


Figure 20. Depiction of nodal pressure changes inside a 101x101 domain at 800 (Hz).

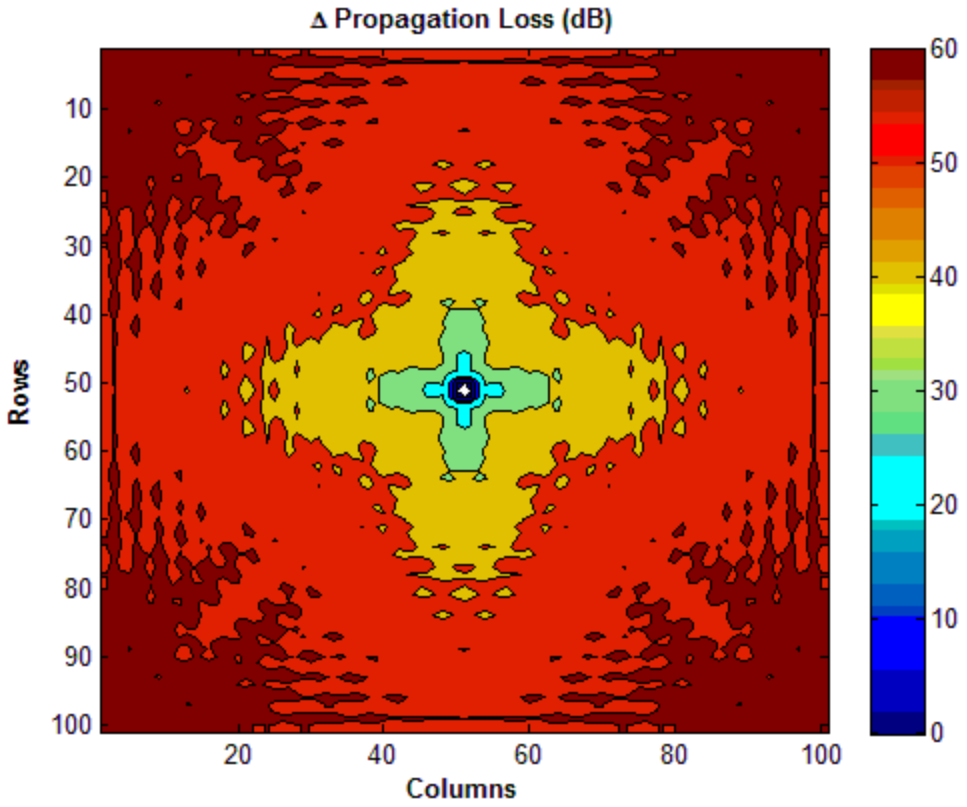


Figure 21. Propagation losses based off RMS pressure values at 800 (Hz) with non-reflecting boundary conditions on all sides

C. VARIOUS MISCELLANEOUS BOUNDARY CONSTRAINTS

The beauty behind the CA method is the ease in which various boundary conditions can be added to the domain. The intent of this section illustrates wave propagation in a domain not easily modeled by ordinary modeling methods. Parameters that define the domain in this section are based on Table 12. The below sections simulate a 2D and 3D case of a deep ocean environment. Based on variable *VN* and *VS* from Table 12, the acoustic source is now at a depth of 10 meters below the water line near the left boundary of the domain. Note that since the speed of sound is in meters/second, the resultant distance between nodes is in meters. In addition, line 17 of Table 12 equals 10; therefore, each nodal spacing correlates to 10 meters. Lastly, the *Plots* variable, from

Figure 4, is used in such a way to simulate operations as an acoustic receiver. Details of its location are described in the below subsections.

FIGURES	CODE LINE	VARIABLE	NEW VALUE
Figure 4	Line 2	VN	101
	Line 2	HN	101
	Line 3	R_Nodes	1
	Line 5	VS	-49
	Line 5	HS	-47
Figure 7	Line 16	speed_sound	1500
	Line 17	dx	10
Figure 8	Line 20	Iterations	500
	Line 4	freq	25
	Line 6	Decay	0

Table 12. Revised model paremeters to simulate operations under various boundary and domain restraints

1. Wave Propagation across a Flat Bottom Ocean Floor

For the first set of simulations there is no restriction on the left and right boundaries. This in turn requires non-reflecting boundary conditions on the left and right wall. Setting the *constraint1* and *constraint2* variable in Figure 10 to “4” establishes these conditions. The ocean floor behaves as a perfect reflector and the air to waterline boundary exhibits a free boundary. As a result, the *constraint3* variable is set to “3” and *constraint4* variable is set to “2” in Figure 10.

The *Plots* variable for this scenario are set at row=65 and column=65. This correlates to a receiver depth of 630 meters below the waterline. Figure 22 shows the results of how the pressure varies at the receiver as a function of time. As you can see, the effects of modeling the floor as a perfect reflector becomes apparent, at this particular point, around 1.0559 seconds (see Figure 23). The initial wave front peaks are approximately 1.2 [Pa]. As time progresses, the wave front, from modeling a perfect bottom reflector, changes the peaks to approximately 2.1 [Pa] around time 1.0559 seconds at our receiver location. In viewing Figure 23, the distortion propagates its way throughout the entire domain. Since the acoustic source is still active in the domain, interference patterns develop and result in a more chaotic profile after 2.2156 seconds.

Nevertheless, Figure 24 provides an effective measure of propagation losses over a range of reflection from the bottom boundary. In addition, Figure 25 illustrates propagation losses, at row 47, from the far left to right end of the domain for various ranges of bottom reflection.

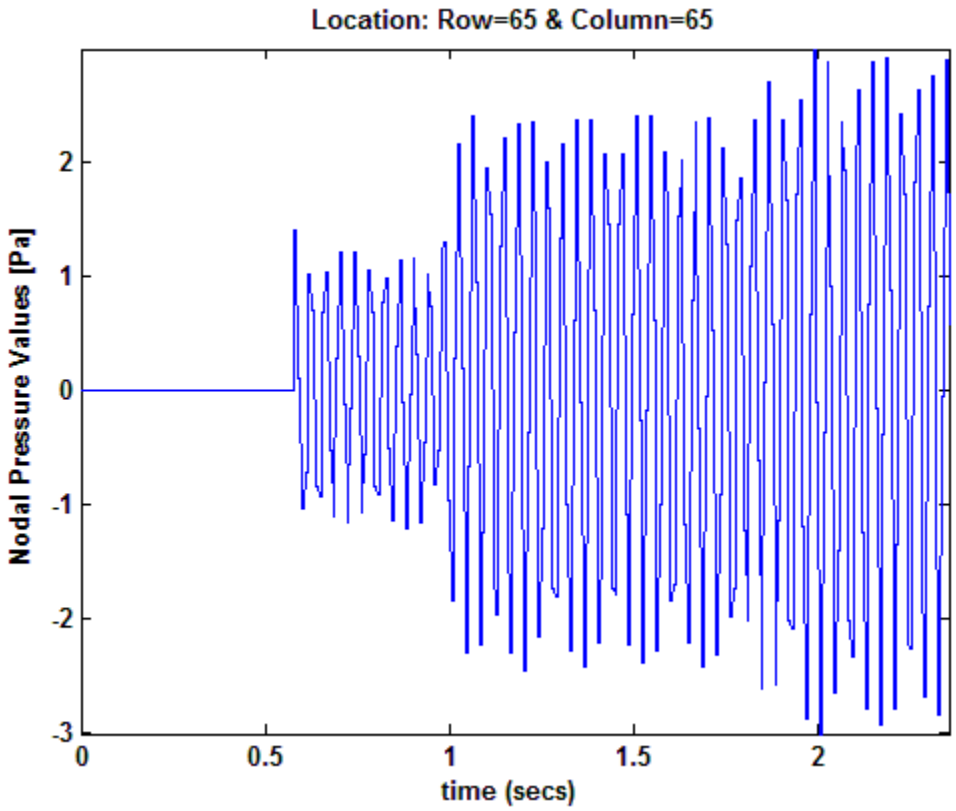


Figure 22. Reciever reception of changes in acoustic pressure from a 25 (Hz) source with a flat ocean bottom with 100% reflection

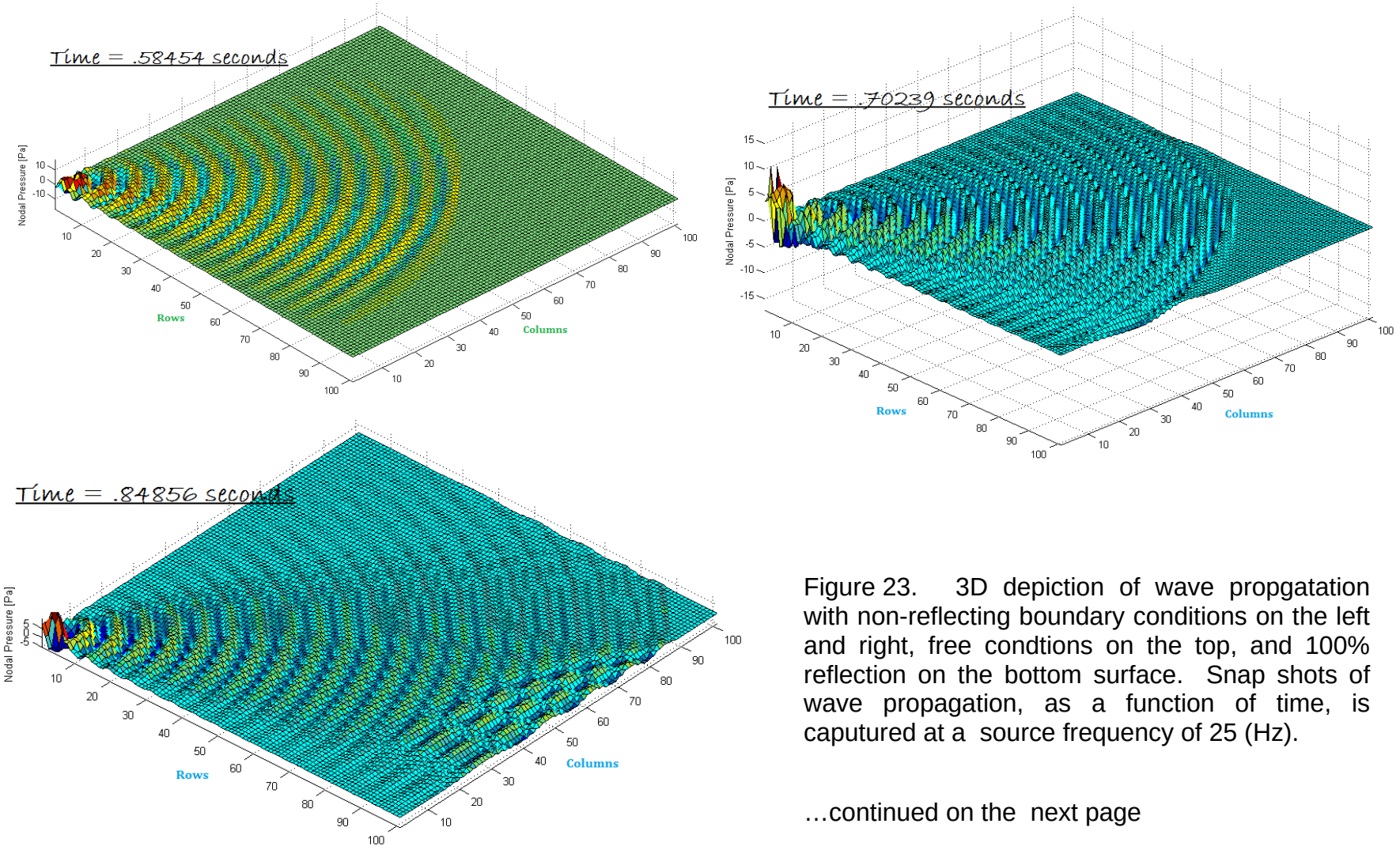
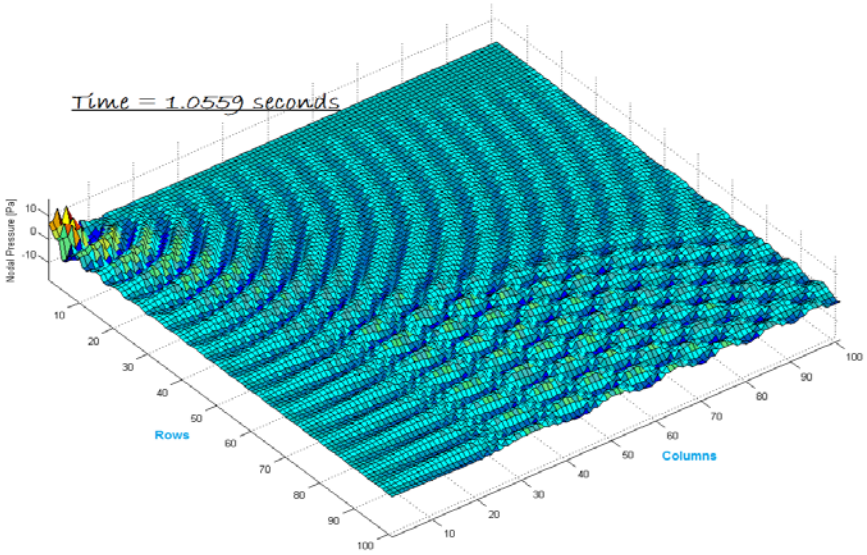
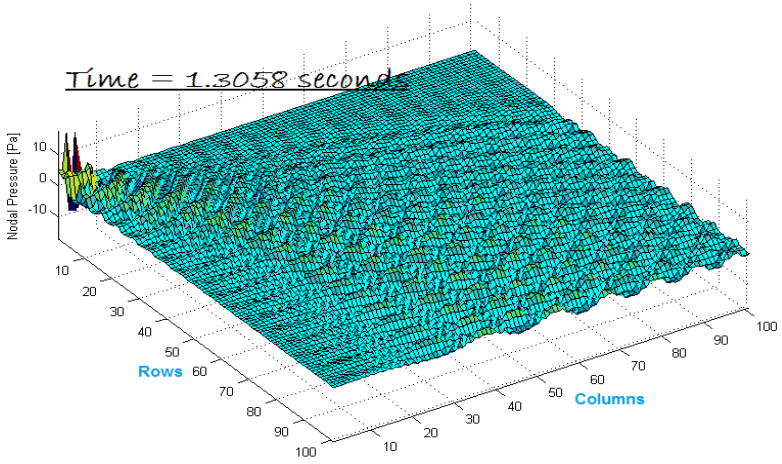


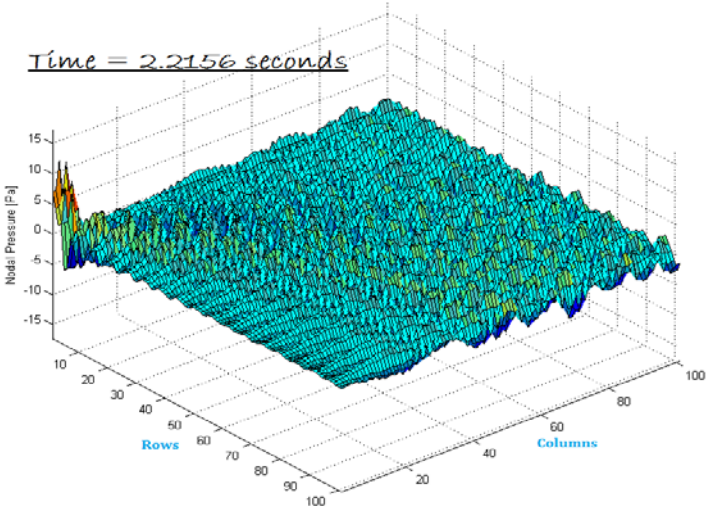
Figure 23. 3D depiction of wave propgatation with non-reflecting boundary conditions on the left and right, free condtions on the top, and 100% reflection on the bottom surface. Snap shots of wave propagation, as a function of time, is caputured at a source frequency of 25 (Hz).

…continued on the next page

Figure 23 …continued from the previous page







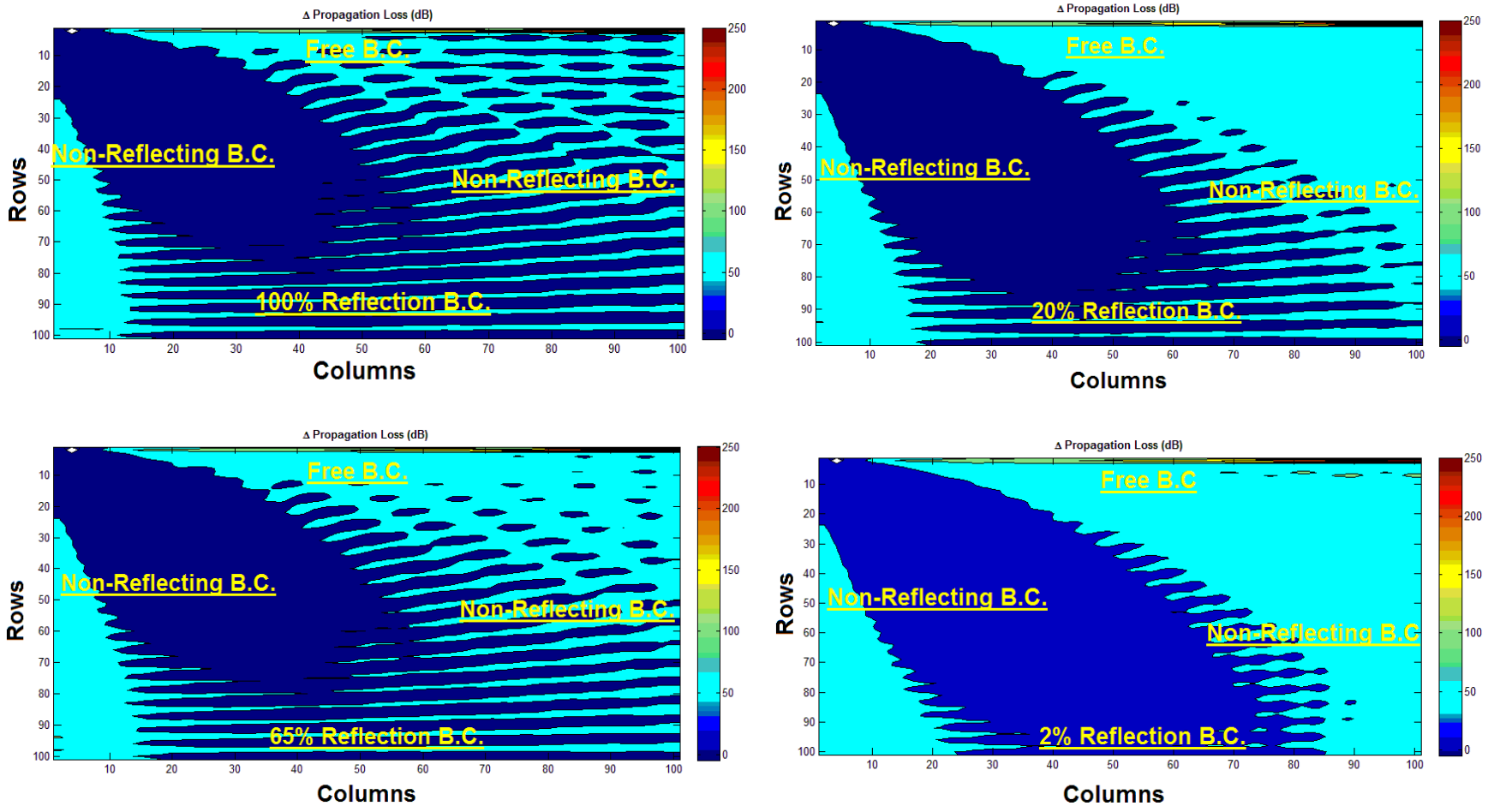


Figure 24. Propagation losses of 101x101 (i.e., 980 (meters) x 980 (meters)) domain at a frequency of 25 (Hz) with various ranges of refection from the bottom boundary.

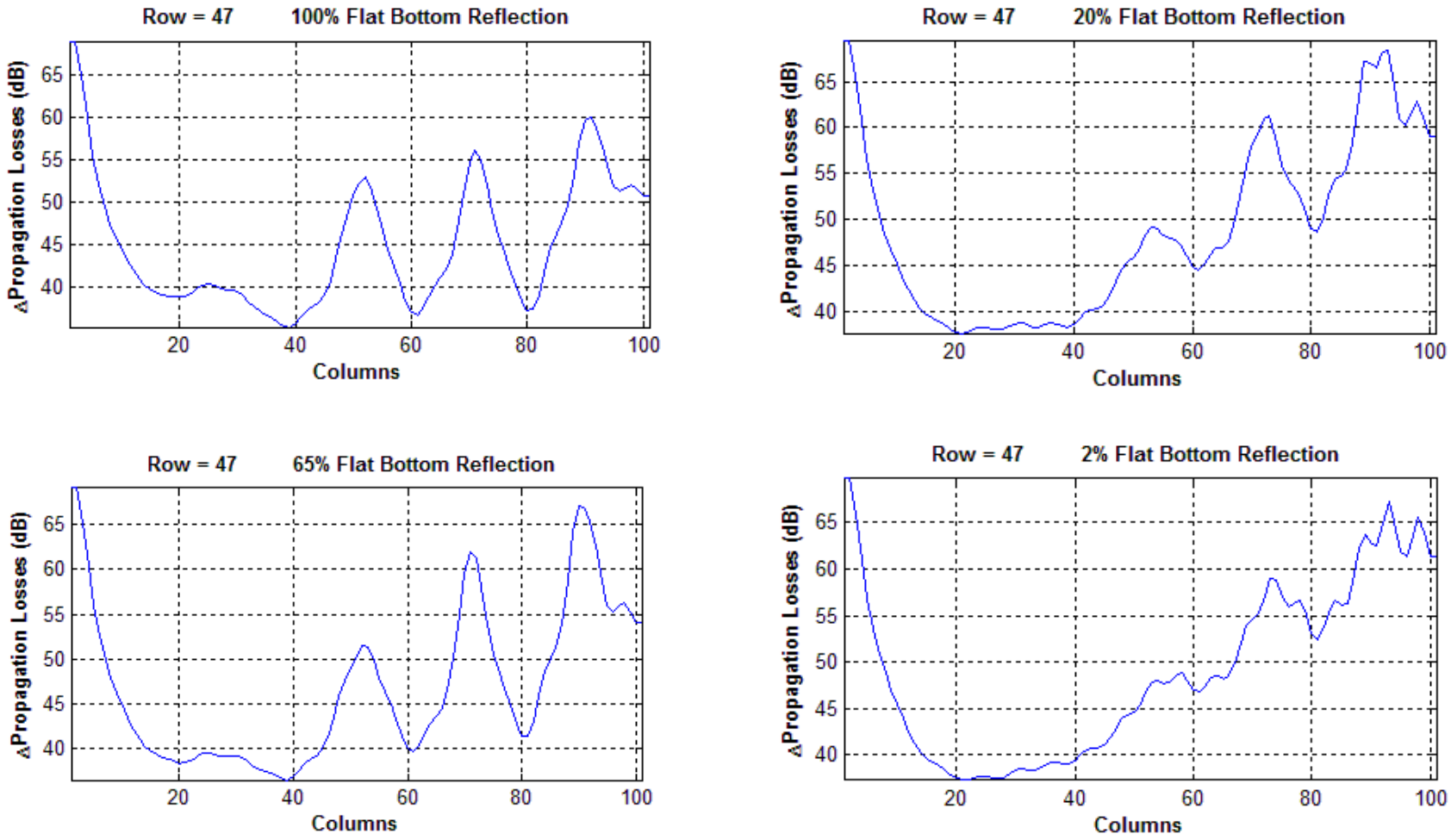


Figure 25. Propagation losses for a 101x101 domain (i.e., 980 (meters) x 980 (meters)) at a frequency of 25 (Hz) along Row 47 for various ranges of bottom reflection

2. Wave Propagation over a Curved Hill

The intent of this section is to illustrate the nature of wave propagation with an ocean floor that exhibits a curved shaped. To enforce boundary conditions along a non-flat surface requires the indices of that boundary. The function *CurvedHillIndices* is created to perform this task (see Figure 26). Table 13 shows a status, inside this function, upon reaching line 7. The size of the variable *CAS* is 11x11 for this particular case. Line 8 locates all values greater than 25 and assigns logic "1," to those particular locations, inside the variable *ZZ*. The execution of line 9–10 results in approximately 75% of the upper half of the matrix *BluePrint* being removed and rotated 180^{o} to yield the variable *Hill* at the conclusion of line 11 (see Table 14). Since the *Hill* variable has one extra column than the sample 11x11 CAS domain, the extra column is deleted by lines 13–16. Continuing with lines 18–19 append additional ones to the variable *Hill* such that the size of the variable *Hill* matches the size of the variable *CAS* (see Table 15). As result, all '0' correlate to the hill while the '1's' match the remaining parts of the domain. Line 22 of Figure 26 captures the indexes for all points that make up the curved hill. It captures the case for 3D, if necessary, because line 21 replicates this newly created boundary into the 3rd dimension if required. Lines 24–37 concludes this function by capturing all the "0" that are adjacent to the inner portions of the domain. The end result of this function is that all the row/column/z indices and indexes are stored in the structure variable *Indices*. The appropriate place to call this function is after the end of line 15 of Figure 10. Amendments to Figure 14 are required to ensure correct operation of the model. The Appendix of this thesis contains all the necessary revisions to allow refractive, reflective, non-reflective, and fix constraints along the curved hill.

ZZ =

50	41	34	29	26	25	26	29	34	41	50	61
41	32	25	20	17	16	17	20	25	32	41	52
34	25	18	13	10	9	10	13	18	25	34	45
29	20	13	8	5	4	5	8	13	20	29	40
26	17	10	5	2	1	2	5	10	17	26	37
25	16	9	4	1	0	1	4	9	16	25	36
26	17	10	5	2	1	2	5	10	17	26	37
29	20	13	8	5	4	5	8	13	20	29	40
34	25	18	13	10	9	10	13	18	25	34	45
41	32	25	20	17	16	17	20	25	32	41	52
50	41	34	29	26	25	26	29	34	41	50	61
61	52	45	40	37	36	37	40	45	52	61	72

Table 13. The status of the *CurvedHillIndices* function after reaching line 7

Hill =

1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	0	1	1	1	1	1	1
1	1	0	0	0	0	0	0	0	1	1	1
1	0	0	0	0	0	0	0	0	0	1	1

Table 14. The status of the *CurvedHillIndices* function after reaching line 12

```
function [Indices] = CurvedHillIndices(CAS,Indices)

    [L1, L2, L3]=size(CAS);
    x=ceil(-L2/2):ceil(L2/2);   y=x;
    [X, Y] =meshgrid(x,y);
    ZZ = X.^2 +Y.^2;

    BluePrint= ZZ>(L2/2-.5).^2;
    BluePrint(1:(ceil(L1/2)-1),:)=[];
    BluePrint(1:(ceil(L1/5)),:)=[];
    Hill=flipud(BluePrint);

    [C1, C2, C3] = size(Hill);
    if L2 ~= C2
        Hill(:,end)=[];
    end

    Add2 = L1-C1;   Add3 = ones(Add2,L2,C3);
    Hill = [Add3; Hill];

    Hill = repmat(Hill,[1, 1,L3]);
    Indices.All_Hill_Indexes = find(Hill==0);
    index=1;
    for k=1:L3
        for j=1:L2
            K = find(Hill(:,j,k)==0);
            if ~ isempty(K)
                Indices.Hill_Row_Borderline(index)=K(1);
                Indices.Hill_Col_Borderline(index)=j;
                Indices.Hill_Z_Borderline(index)=k;
                index=index+1;
            end
        end
    end

Indices.HillBorderIndexes=sub2ind(size(Hill),Indices.Hill_Row_Borderline,..
                    Indices.Hill_Col_Borderline,Indices.Hill_Z_Borderline);
end
```

Figure 26. *CurrvedHillIndices* fuction used to created a curved hill in the bottom of any 2D or 3D domain

```
Hill =

     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     0     1     1     1     1     1
     1     1     0     0     0     0     0     0     0     1     1
     1     0     0     0     0     0     0     0     0     0     1
```

Table 15. The status of *CurvedHillIndices* function after the conlusion of line 19

The simulated domain is the same as that in the previous section. The only exception is that reflective boundary conditions are applied to the curved hill. The results of making changes in the bottom boundary result in significant changes in the acoustic pressure profile. Changes in the receiver pressure begin increasing around .59 second, after the initial front of the wave begins reflecting back from the curved hill (see Figure 27). This is a result of reflective waves reinforcing the propagation of the incoming source. As time progress, there is attenuation in the magnitude, as a result of the reflective phenomenon occurring out of phase with the incoming source, while returning at the same incident angle. This is more apparent when examining column 65 from Figure 25 and Figure 28, as the loss propagation changes by a factor of nearly 12 (dB) at the receiver location. Nevertheless, wave propagation is behaving in the manner expected.

When comparing Figure 27 to Figure 23, at time 1.0559 seconds, the addition of the curved boundary has allowed for a reflected wave to essentially propagate throughout the entire domain in a faster time frame. This essentially shows an accurate depiction of wave propagation when the curvature of the domain is changed, and the model is accurately accounting for the constraints along the boundary. Nevertheless, the addition of the curved boundary causes

significant changes in propagation losses along the lower left and right corners of the domain (see Figure 27 and Figure 29). The decrease in magnitude can be attributed to the fact that the perfect reflection off the curved hill is increasing the acoustic pressure. The reflection is at an oblique angle to the source so destructive interference does not occur in a manner as shown on the top left and right portions of the hill, which are feeling more of the effects from the free boundary constraints being applied to the top portions of the hill.

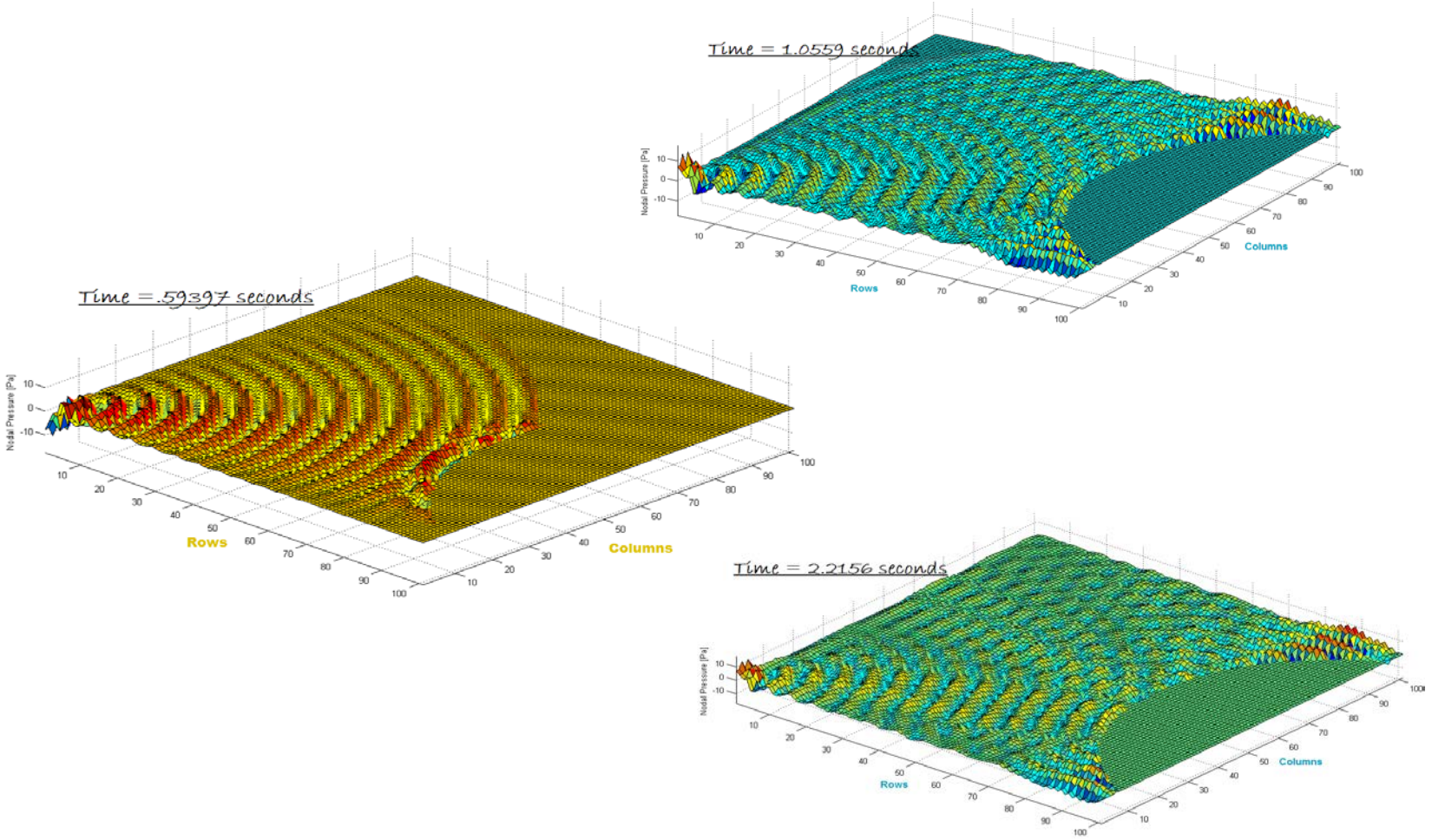


Figure 27. Snap shot at three instances in time of wave propagation in a 2D domain with a curved floor bottom boundary

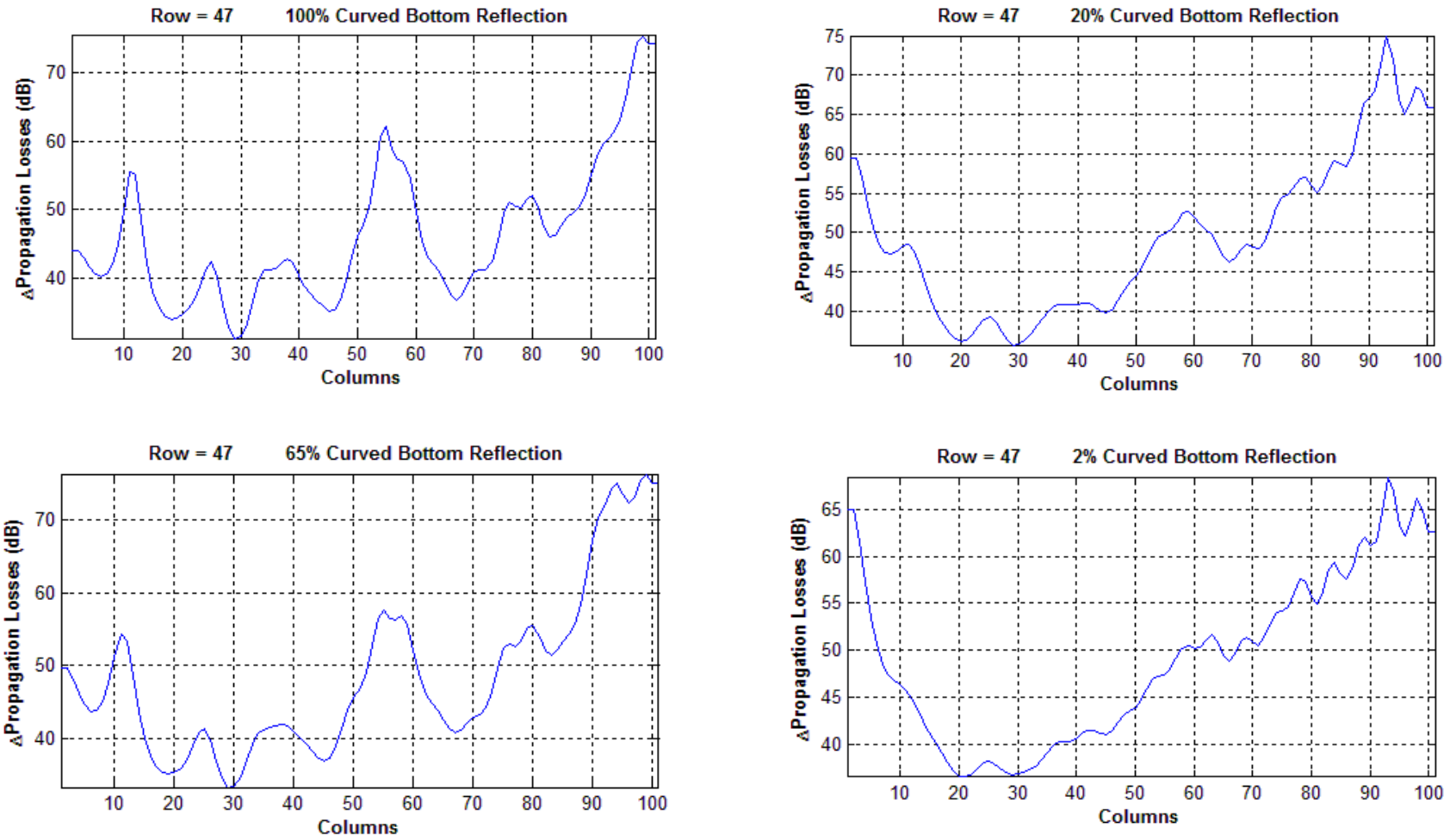


Figure 28. Propagation losses for a 101x101 domain (i.e., 980 (meters) x 980 (meters)), with a curved ocean floor, from 25 (Hz) acoustic source for various ranges of bottom reflection

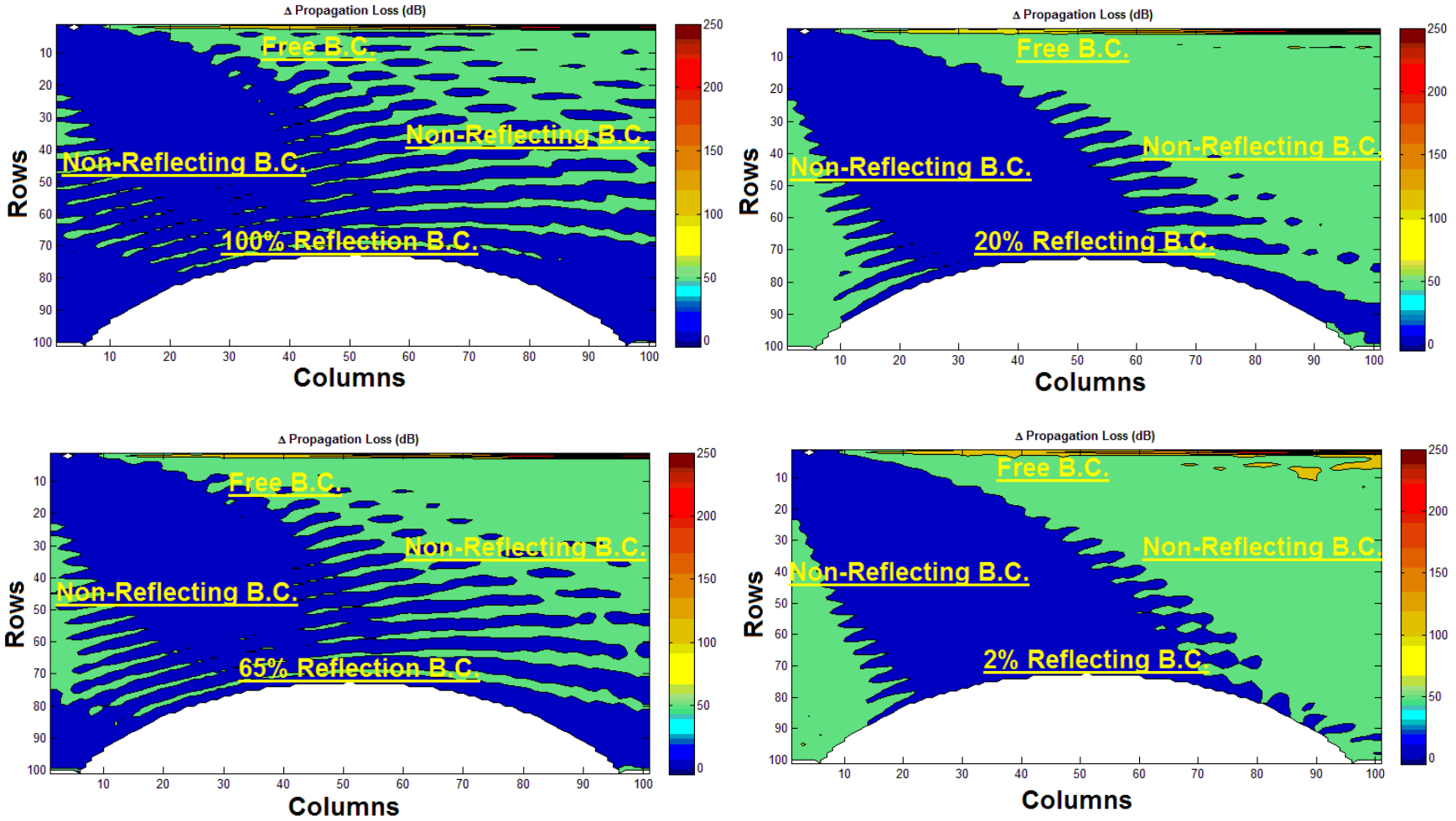


Figure 29. Propagation losses for a 101x101 domain (i.e., 980 (meters) x 980 (meters)), with a curved ocean floor, from 25 (Hz) acoustic source for various ranges of bottom reflection

3. Wave Propagation over a Sloping Bottom

The scenario in this section depicts modeling wave propagation across a sloping bottom. The bases of this domain formulation are from the wedge shape domain depiction established by the Acoustic Society of America (ASA) [20]. Here a similar size domain emulates the 3D shape depicted in Figure 30. The first part of this section discusses propagation in an open ocean environment, that is, a domain with infinite lateral limits. The later part of this scenario models wave propagation in a confined waterway channel. The script to develop the domain is shown in Figure 31. The development of this particular script is similar to the script used to create the curved hill. The biggest differences are in lines 3–6.

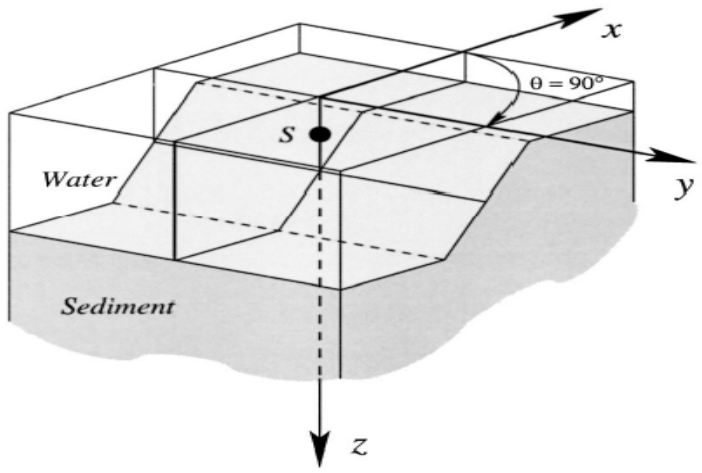


Figure 30. Geometry of a 3-D ~truncated wedge shaped waveguide. The acoustic source S is located above the sloping bottom. From [20]

```
function [Indices] = WedgeHillIndices(CAS,Indices)

    [L1, L2,L3]=size(CAS);
    L2RDownHill=tril(ones(L1,L2),40);
    Remove=floor(L1/2);
    L2RDownHill(1:Remove,:)=0;

    L2RDownHill = repmat(L2RDownHill,[1, 1,L3]);
    Indices.All_Hill_Indexes = find(L2RDownHill==1);
    index=1;
    for k=1:L3
        for j=1:L2
            K = find(L2RDownHill(:,j,k)==1);
            if ~ isempty(K)
                Indices.Hill_Row_Borderline(index)=K(1);
                Indices.Hill_Col_Borderline(index)=j;
                Indices.Hill_Z_Borderline(index)=k;
                index=index+1;
            end
        end
    end

Indices.HillBorderIndexes=sub2ind(size(L2RDownHill),...
        Indices.Hill_Row_Borderline, Indices.Hill_Col_Borderline,...
                                                Indices.Hill_Z_Borderline);
end
```

Figure 31. Script used to create the sloping wedge domain

a. Infinite Wide Ocean Scenario

Non-Reflecting boundary conditions are applied to the front, back, left, and right sides of the domain to simulate infinite lateral limits. The top boundary, once again, is simulated with free boundary conditions, while the bottom boundary exhibits a range of partially reflections. In essence, this allows for modeling losses, due to contact with the sediment floor. Moreover, Table 16 shows revisions to important parameters to produce the results of this scenario. A view of propagation losses in this particular 3D domain is shown in Figure 32. A model of this sort can be very useful in predicting propagation losses across the continental shelf. Nevertheless, the depiction of column plane 45 is depicted in addition to losses at a depth of 290 (m) (see Figure 33).

FIGURES	CODE LINE	VARIABLE	NEW VALUE
Figure 4	Line 2	VN, HN, & ZN	61
	Line 3	R_Nodes	5
	Line 5	VS	-11
	Line 5	HS	-11
	Line 5	ZS	0
Figure 7	Line 16	speed_sound	1500
	Line 17	dx	10
Figure 8	Line 20	Iterations	300
	Line 4	freq	110

Table 16. Revised model paremeters to simulate operations under various boundary and domain restraints

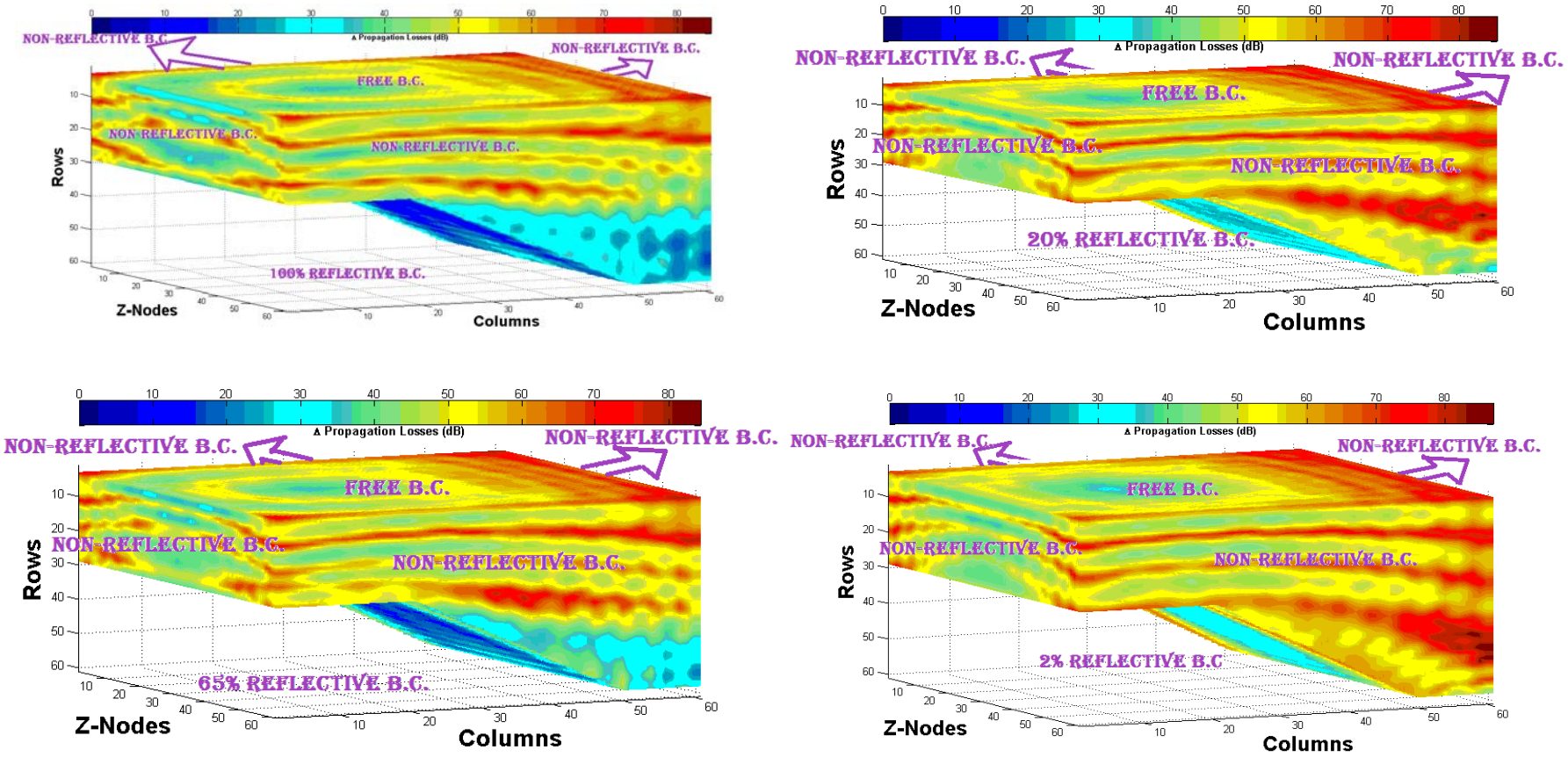
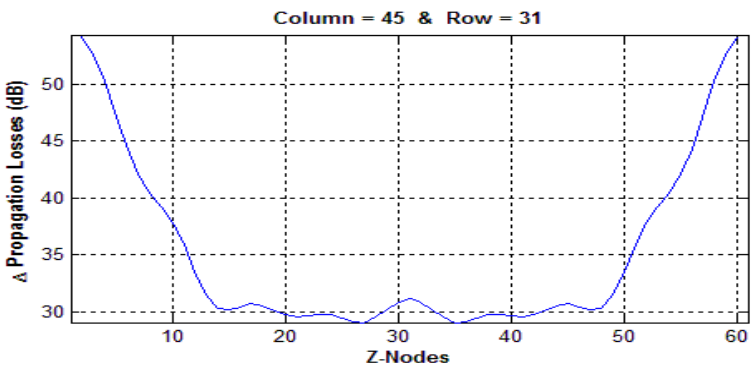
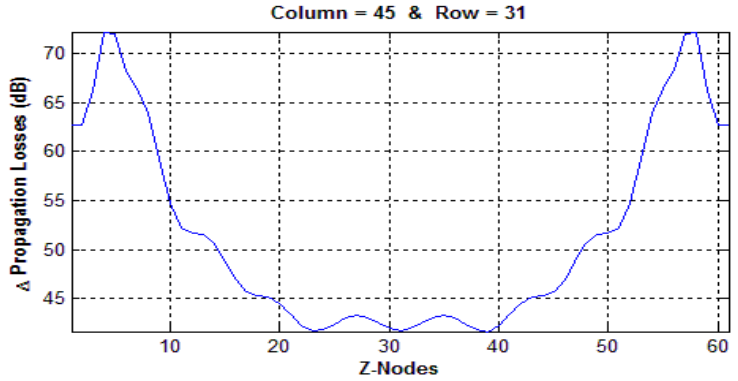


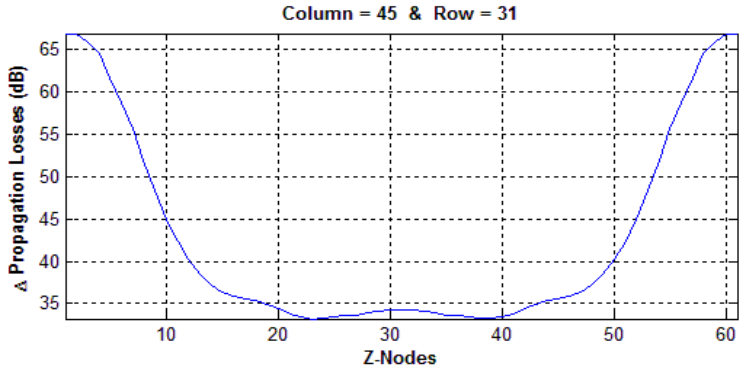
Figure 32. Propagation losses of a wedge shape bottom domain at 110 (Hz) for various ranges of bottom reflection. The front and rear boundary will exhibit non-reflecting boundary conditions.



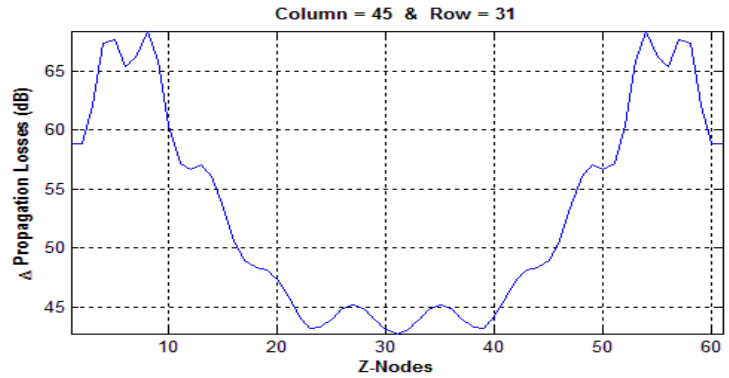
(a)



(c)



(b)



(d)

Figure 33. Propagation losses of a wedge shape bottom domain along column 45 and row 31 from Figure 32 at 110 (Hz) for: (a) 100% (b) 65% (c) 20% (d) 2% bottom reflection. The front and rear boundary exhibited non-reflecting boundary conditions.

b. Confined Water Channel Scenario

In order to simulate wave propagation in a confined water channel, the front and back wall of the domain must exhibit reflective boundary constraints. In turn, updates to variable *constraint5* and *contraint6* which come after line 15 in Figure 10 (not shown but updated in the Appendix) are required. The floor of the domain, on the other hand, is maintained at 65% partial reflection. Lastly, the water to air interface is maintained with free constraints in addition to retaining essential parameters to perform the simulation from Table 16.

Simulation results, of propagation losses, produce the profiles in Figure 34 and Figure 35. Accordingly, the effects of reflection are really noticeable along the sides of the domain at higher percentages of reflection. Consequently, development of this modeling scheme causes changes in the pressure profile and emulates changes in the loss profile. Significant changes can be, on the order of 16 (dB), very significant when comparing part (a) and (d) from Figure 35. Nevertheless, refinement of this domain can be adapted to meet the needs of a modeling scenario that may require these restraints.

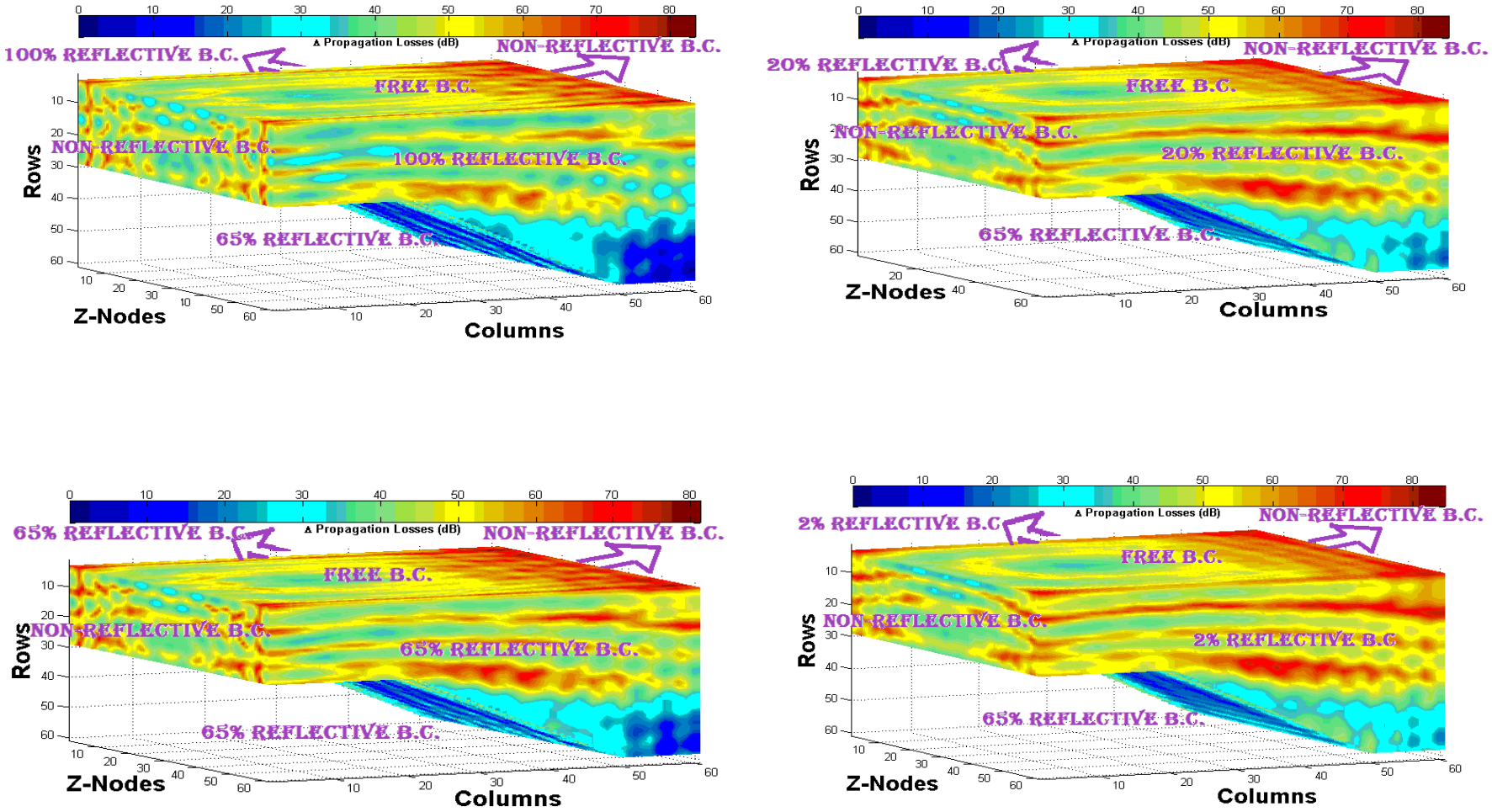


Figure 34. Propagation losses of a wedge shape bottom domain at 110 (Hz) for various ranges of front and real deflection. The bottom boundary will reflect 65% of the incoming wave.

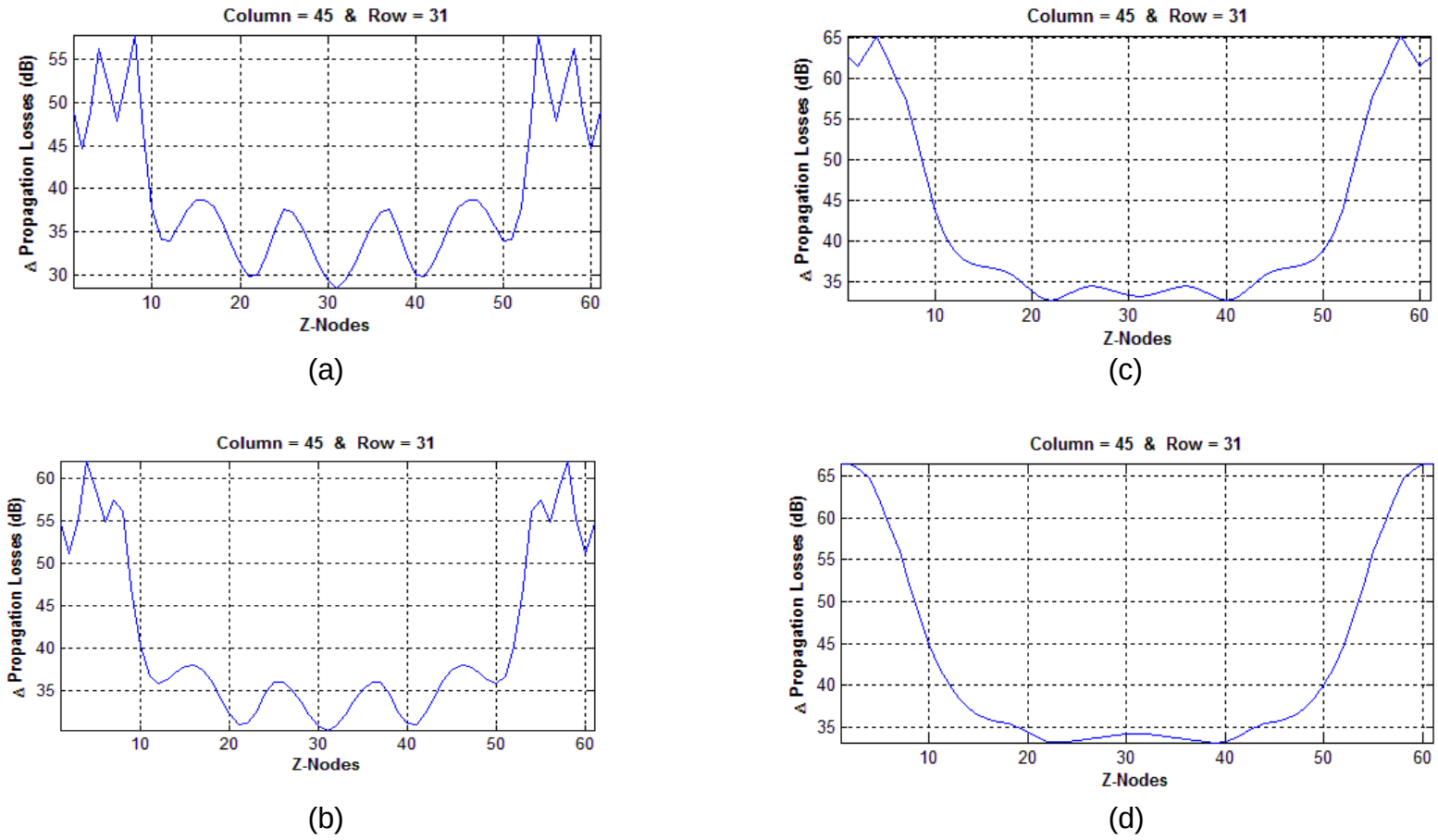


Figure 35. Propagation losses of a wedge shape bottom domain along column 45 and row 31 from Figure 34 at 110 (Hz) for: (a) 100% (b) 65% (c) 20% (d) 2% front and rear reflection. The bottom boundary exhibited 65% partial reflective boundary conditions.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. CELLULAR AUTOMATON MULTI-GRID SOLUTION MODELING

A. INTRODUCTION

The breath of what has been discussed relates to numerically solving a complex boundary value problem with the application of a simple rule. The intent of this chapter is to develop a refinement scheme in order to increase the accuracy of a solution in certain regions of the cellular domain. The problem that has not been addressed is what happens if the number of columns and rows are expanded to be large in magnitude, for example, 10^4 nodes in all three dimensions? In all likelihood, a user utilizing a standard personal computer is going to experience many problems and receive a simulation error due to not having sufficient memory. Ways to circumvent this issue require the user to perform the simulation on a super computer or in an iteratively save and repeat manner. Nevertheless, the size of the domain increases by a scalar factor, which is dependent on the number iterations utilized in executing the simulation run. This 4th dimension, which is the time component, severely limits memory allocation and essentially eliminates the possibility of solving this domain problem due to its size.

To circumvent issues with memory allocation, this chapter develops a refinement process to capture the essence of the pressure profile in a uniform grid with development of a localized grid. A uniform grid correlates to a domain with only global nodal points inside the CA domain (i.e., the domains discussed in chapter II and III). A multi grid domain refers to a domain which has global grid points with a local grid system internal to the dimensions of the global domain. In essence, the multi-grid domain has its local grid virtually spread throughout the remaining portions of the global grid. An analogy to this method, in computational mechanics, is similar to creating smaller and smaller mesh sizes, in a local region, by use of the finite element method (FEM) [21]. Functions that have been

created in Chapter II are used, but with slight modifications to make the whole model self-reliant in alternating between a multi-grid and uniform grid solutions.

B. MESH MODEL OVERVIEW

In order to illustrate the concept of this technique, examine the cellular automaton grid points in Figure 36. Here the multi-grid domain has 11x11 global grid points. Of interest is the local grid domain (the yellow section) and its associated red corners. The red corners and yellow highlighted regions in Figure 36 correlate to the red corners and yellow region of Figure 37. Establishing this correlation between the global and local grid is the same as establishing a uniform grid which has 81x81 grid points, as shown in Figure 38. In essence, a virtual refined mesh is spread throughout the non-highlighted sections in Figure 36. The inclusion of the gray areas account for necessary boundary regions to perform the CA technique that is discussed in Section II.D.1.b.

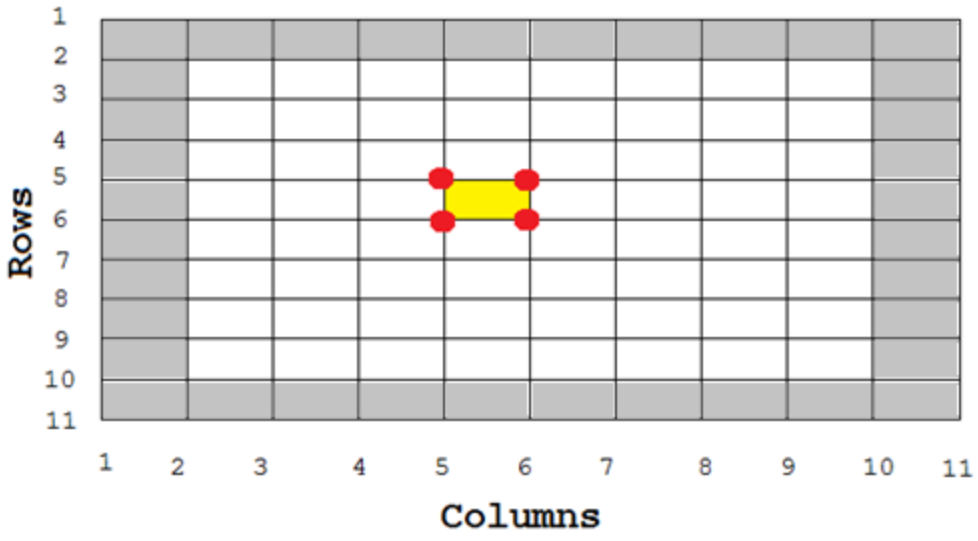


Figure 36. Cellular Automaton multi-grid domain to include the boundary nodes

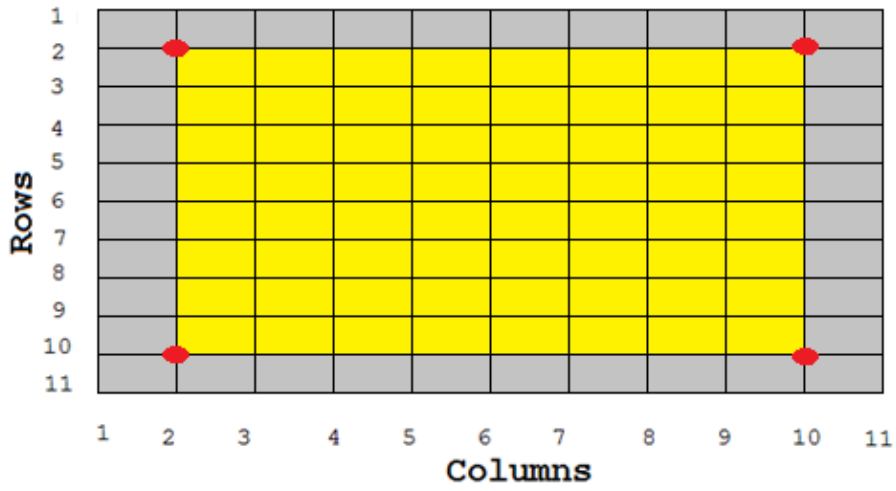


Figure 37. Local grid domain of a Cellular Automaton domain to include the associated boundary nodes

Up until this point all simulations are created without a mesh refinement. The remaining parts of this chapter illustrate the pressure observed at a node, inside the uniform grid, and its corresponding point inside a multi-grid. For example, analysis are gathered to observe the pressure changes at point (2,2) in Figure 36 to its corresponding point (2,2) from Figure 38. Throughout all simulations the acoustic source is placed in the local grid for all multi-grid scenarios. Several different boundary schemes are applied to the local grid of Figure 37 but it is determined that applying non-reflecting boundary conditions, as the interpolation mechanism between the local and global domain, produce the best modeling results. A devised formula, relating the dimensions of the uniform grid to the multi-grid is shown in Equations (8) and (9). The overall application into the development of the model algorithm is explained in the next section with a comparison of the multi-grid and uniform grid scheme for final validity of this method.

$$NSGG = GNP - 3 \tag{8}$$

where:

Number of Spacing in the Global Grid (NSGG) counts the number of dx dimensions in the global grid in the horizontal/vertical direction. For example, node (2,1) to (2,5) from Figure 36, correlates to three NSGG. Node (2,1) to (2,2) is not included because nodal point (2,1) is a boundary node.

Global Nodal Points (GNP) correlates to the number of horizontal/vertical global (assuming square dimensions) grid points inside the multi-grid system. The associate value includes the boundary nodes.

$$GPUG = (NSGG)(LNP - 4) + GNP \tag{9}$$

where:

Grid Points in the Uniform Grid (GPUG) correlates to the number of horizontal/vertical (assuming square dimensions) grid points inside the uniform grid. The associate value includes the boundary nodes.

Local Nodal Points (LNP) correlates to the number of horizontal/vertical (assuming square dimensions) local grid points inside the multi-grid. The associate value includes the boundary nodes.

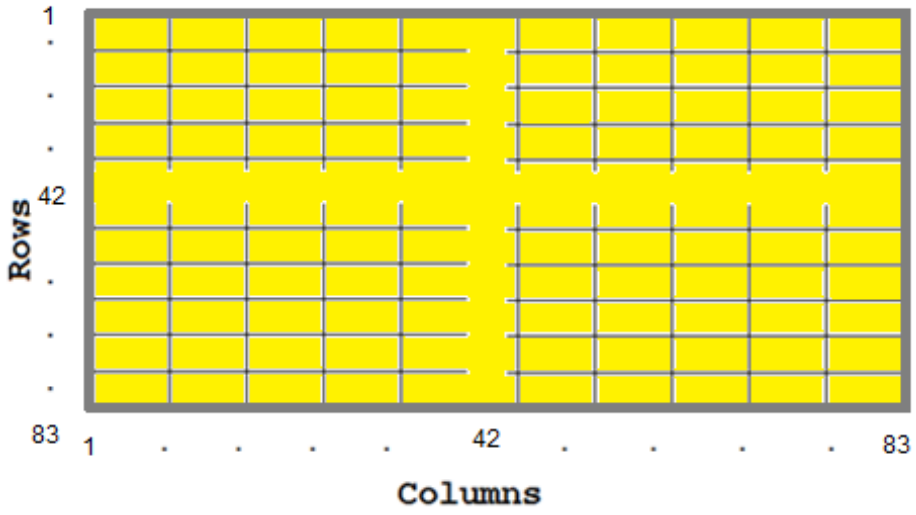


Figure 38. A uniform grid including the two additional boundary nodes

C. MULTI-GRID CA MODEL DEVELOPMENT

The beginning of the main script to execute this model begins with Figure 4 and requires no changes. Figure 39 is an amendment to the main script that is inserted after line 10 of Figure 4. Global variables are defined in line 12 and 13 serve as a gateway to determine if a multi-grid solution is desired. In order to refine the solution the global variable *ANS* is set to "1." As a result, when line 20 is reached, the user enters the upper left corner where the local grid is inserted inside the global grid. In addition, the last input of the array correlates to the horizontal and vertical lengths of the local grid. Figure 36 correlates to setting the *Refine* global variable on line 20 to [5, 5, 1, 11]. In line 32, the global variable *SubDomainPoints* generates nodal locations inside the local grid (see Figure 37) for plot viewing. The last global variable, *cas,* holds permutation information of the local grid (see Figure 37) while the *CAS* variable, that is defined in Figure 5, holds data on the global/uniform grid system (see Figure 36 and Figure 38). Although the user selects the size of the radial nodal distance in line 3 of Figure 4, this information, along with line 5 is reflected to the local grid, if a multi-grid solution is desired.

A comparable solution is generated by permeating the black and white nodes of the local grid in an amount equal to the length of the mesh spacing. Recall that the variable *dx,* from Figure 7, correlates to the distance between nodes. As a result, Figure 37 gives a mesh spacing between the far left and right global grid points of "*8*dx."* This, in turn, means that the local grid is iterated 8 times before the values stored along the corner grid points (see the red dots of Figure 37), are reflected to the global grid (see the red dots of Figure 36).

```
11
12 -    global ANS Refine SubDomainPoints cas
13 -    ANS = input(['\nWould you like to place the acoustic source in a ',...
14          'refined\n', ' location in a portion of the ',num2str(VN),...
15          'x',num2str(HN),'x',num2str(ZN),' CA space domain?  \n',...
16          ' Please select (1) for Yes or (2) for No.\n--->'] );
17 -    if ANS==1
18 -        Comment=[' Ensure Row is <=',num2str(VN),', Column is <= ',...
19              num2str(HN),', Z is <= ', num2str(ZN)];
20 -        Refine = input(['\nEnter the Row, Column, & Z-Cordinate, of the',...
21            ' upper left hand\n corner square grid cordinates where',...
22            ' the CA space domain will be refined\n with the associated',...
23            ' acoustic pressure source.\nLastly, please ',...
24            ' enter a positive integer, at the end of the\n array,',...
25            ' that will correlate to the dimensions of the refined square\n',...
26            ' space.',...
27            ' Your input will be an array as shown below as a example\n\n',...
28            '--Example: [Row, Column, Z, Positive Integer Value];\n\n',...
29           Comment,',\n and the Postitive Interger Value is greater than\n',...
30            ' the radial nodal diameter which is currently\n',...
31            ' set to ',num2str(R_Nodes),'.\n--->']);
32 -        SubDomainPoints=input(['\nPlease enter an array or matrix(for',...
33            ' multiple graphs\n in the subdomain) to specify the location(s)',...
34            ' within\n the Mesh. If the space is 2D then enter 1 in each',...
35            ' last column.\n Below is the max value not to exceed, for',...
36            ' each row,colum, and Z\n entries, base on your input',...
37            ' provided for Positive Integer Value\n above (3D Case):\n    ',...
38           num2str(Refine(end)'),...
39            '\nPlease enter the Refine Mesh location in the same format as was',...
40            '\n in selecting the domain cordinates for plotting:\n',...
41            '---Example: [Row_1, Column_1, Z_1; Row_2, Column_2, Z_2; etc...]',...
42            '\n--->']);
```

Figure 39. Script to determine as to whether or not a multi-grid or uniform grid solution is desired (remaining code omitted for brevitty)

Revisions are included inside the *PermuteCASpace* function. Figure 40 shows two additional functions that are created inside this function to permeate the local grid after line 25 in Figure 11. The *PermuteMeshCASpace* function executes rule 1, from Table 6, on the local grid, an amount equal to the mesh spacing which is calculated in line 18 of Figure 40. After that, data is returned to the 4D matrix *positioN* for later simulations. Moreover, the *UpdateRefineCASMesh* function updates the global grid of Figure 36. As an

additional note, the global grid data is passed to the *PermuteMeshCASpace* function but no permutation is performed on this grid. This is to ensure the global and local grids remain synchronized on the same time scale. Overall, execution of the remaining portions of the model performs in the same manner for black nodes with details of all amendments, which are depicted in Appendix B.

```
17      if ANS==1
18          Refine(end)=Refine(end)-3;
19      end
20

26      if ANS==1
27          [cas,positioN,Position]= PermuteMeshCASpace(positioN,cas,...
28                                                  Indices,Value,Position,CAS);
29          CAS= UpdateRefineCASMesh(positioN(:,:,:,index-1),CAS);
30      end
31      CAS = CA_WhiteInnerNodes(CAS);
```

Figure 40. Revison to the *PermuteCASpace* function for mesh refinement of the CA multi-grid (remaining code omitted for brevitty)

D. MODEL VALIDATION

Throughout experimentation to correlate the validity of this method, a dependence on the frequency and the level of refinement of the global and local grid contribute in establishing the accuracy of this scheme. The first trial run of data has uniform grid parameters (see Table 17). Comparison of an equivalent multi-grid scheme has its associated parameters in Table 18. Important relations to note in Table 17 are in regards to the setting of the *VS* and *HS* variable. In order to align the acoustic source of the multi-grid and uniform grid, the uniform grid center has to be shifted an amount equal to half the mesh size of the local grid (see Figure 36 for a visual interpretation). On the contrary, no shift is required in Table 18. Upon execution of both models, the results of both simulations for the zero input frequency case are shown in Figure 41. A direct comparison of the associated pressure rise, at near identical locations, shows similar profiles between the two methods. In particular the starting time for the

associated pressure rise is nearly identical. On the other hand, the multi grid method appears to approach the peak magnitude at a faster rate.

FIGURES	CODE LINE	VARIABLE	NEW VALUE
Figure 4	Line 2	VN, HN	401
		ZN	1
	Line 3	R_Nodes	1
	Line 5	VS, HS	-5
	Line 6	Plots	[2 2 1; 105 105 1; 206 1 1; 206 103 1]
Figure 7	Line 16	speed_sound	1500
	Line 17	dx	1
Figure 8	Line 20	Iterations	5000
	Line 4	freq	0

Table 17. Parameter values to simulate a uniform grid performance with non-reflecting boundary condtions along the boundary

FIGURES	CODE LINE	VARIABLE	NEW VALUE
Figure 4	Line 2	VN, HN	71
		ZN	1
	Line 3	R_Nodes	1
	Line 5	VS, HS	0
	Line 6	Plots	[2 2 1; 8 8 1; 14 14 1; 21 21 1; 36 1 1; 36 18 1]
Figure 7	Line 16	speed_sound	1500
	Line 17	dx	1
Figure 8	Line 20	Iterations	5000
	Line 4	freq	0
Figure 39	Line 13	ANS	1
	Line 20	Refine	[35 35 1 9]
	Line 32	SubDomainPoints	[2 3 1] * arbitrary selection inside the local grid

Table 18. Parameter values to simulate multi-grid performance with non-reflecting boundary condtions

Uniform Grid Nodal Points | **Multi-Grid Nodal Points**

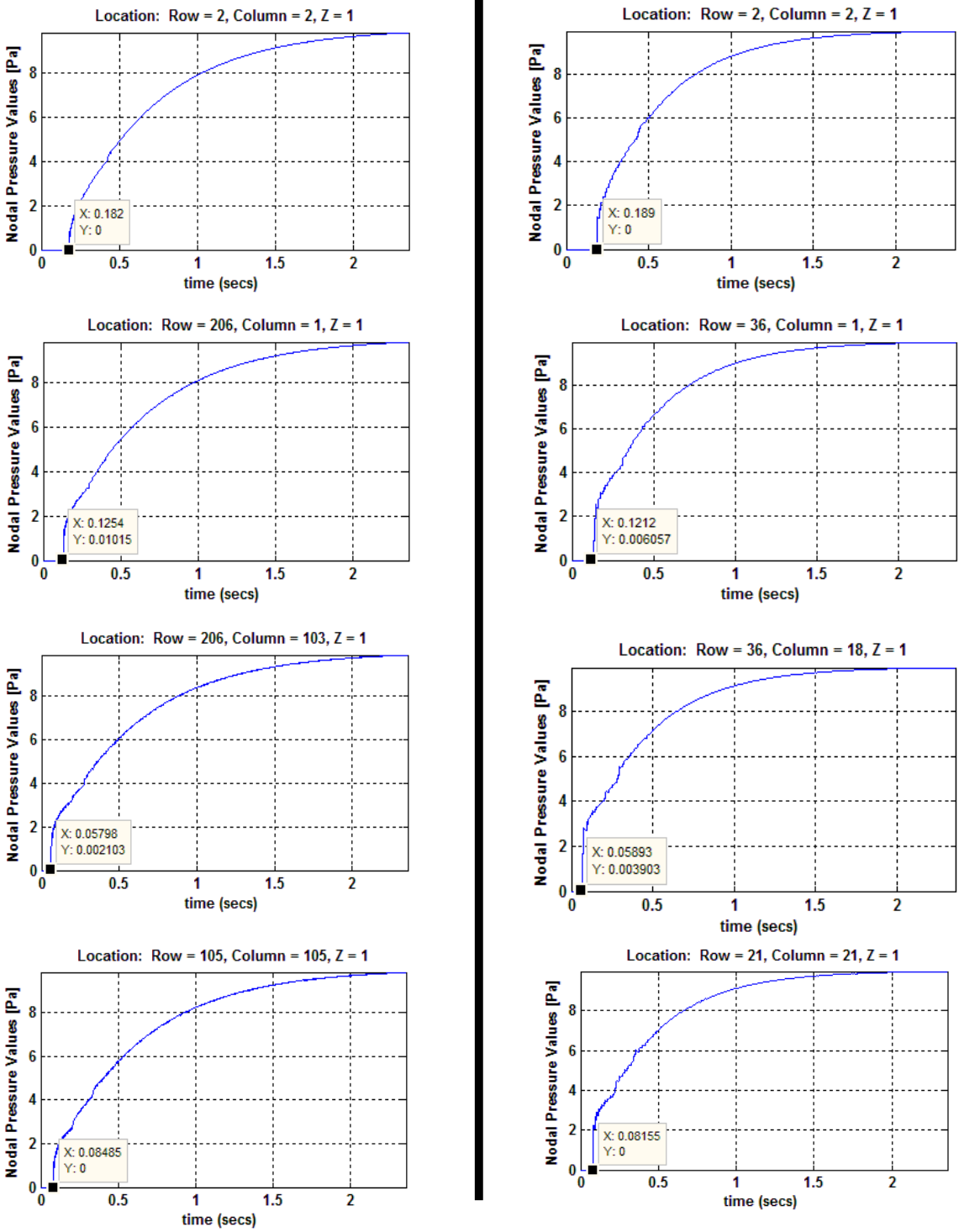


Figure 41. Comparison of equivalent nodal points, inside a uniform and multi-grid, from a 0 (Hz) acoustic source

The domain dimensions that are discussed in Table 17 and Table 18 have been reviewed again with changes being made to the acoustic input frequency (set to 10(Hz)) number of iterations (set to 2000), and the addition of [35 35 1; 70 70 1] to the *Plots* variable in Table 17. The correlations between the two domains show consistent higher peaks with the multi-grid propagation scheme (see Figure 42). On the contrary, there is consistency in the onset time for wave propagation. Nevertheless, the effects of the local and global grid mesh size contribute in preventing the multi-grid solution from matching the uniform grid solution. As such, the next set of simulation evaluates how changes in their associated values affect the overall performance.

The first sets of revisions are made to the global grid. The uniform grid in Table 17 has an actual horizontal/vertical length of NSGG*dx = (*401–3)*(1) = 398 (meters)*.[1] Likewise, the actual length of the local grid in Table 18 is (9–3)*dx= 6 (meters). In order to get more nodal points inside the global grid, to produce results that closely mirror the uniform grid, the value of the *dx* variable is decreased to 1/4. As a result, the local grid from Table 18 has a new length of (9–3)*dx=6*(1/4) = 1.5(meters). In order to have the same length of 398(meters) in the global grid, the horizontal (HN) and vertical (VN) nodes must increase to 398(meters)/1.5(meters) +3 = 268.33.[2] Since the CA method only works for odd integers, HN and VN are rounded to the nearest odd number (i.e., 269). Table 19 shows a summary of the revised changes to increase the number of global grid points. A comparison of the refined global grid in reference to the original uniform grid (see Table 17) is shown in Figure 43. The effects of refining the global grid produce results that closely match the original uniform grid with a

[1] This assumes the *speed_sound* variable has units of meters per second.

[2] Based onEquation (8).

computational time of 2 minutes and 52 seconds for the multi-grid in comparison to 7 minutes and 20 seconds for the uniform grid.[3]

[3]Computer Properties: An Intel(R) Xenon(R) CPU @3.2GHZ, 64bit operating system, with 12.0 GB of Installed Memory.

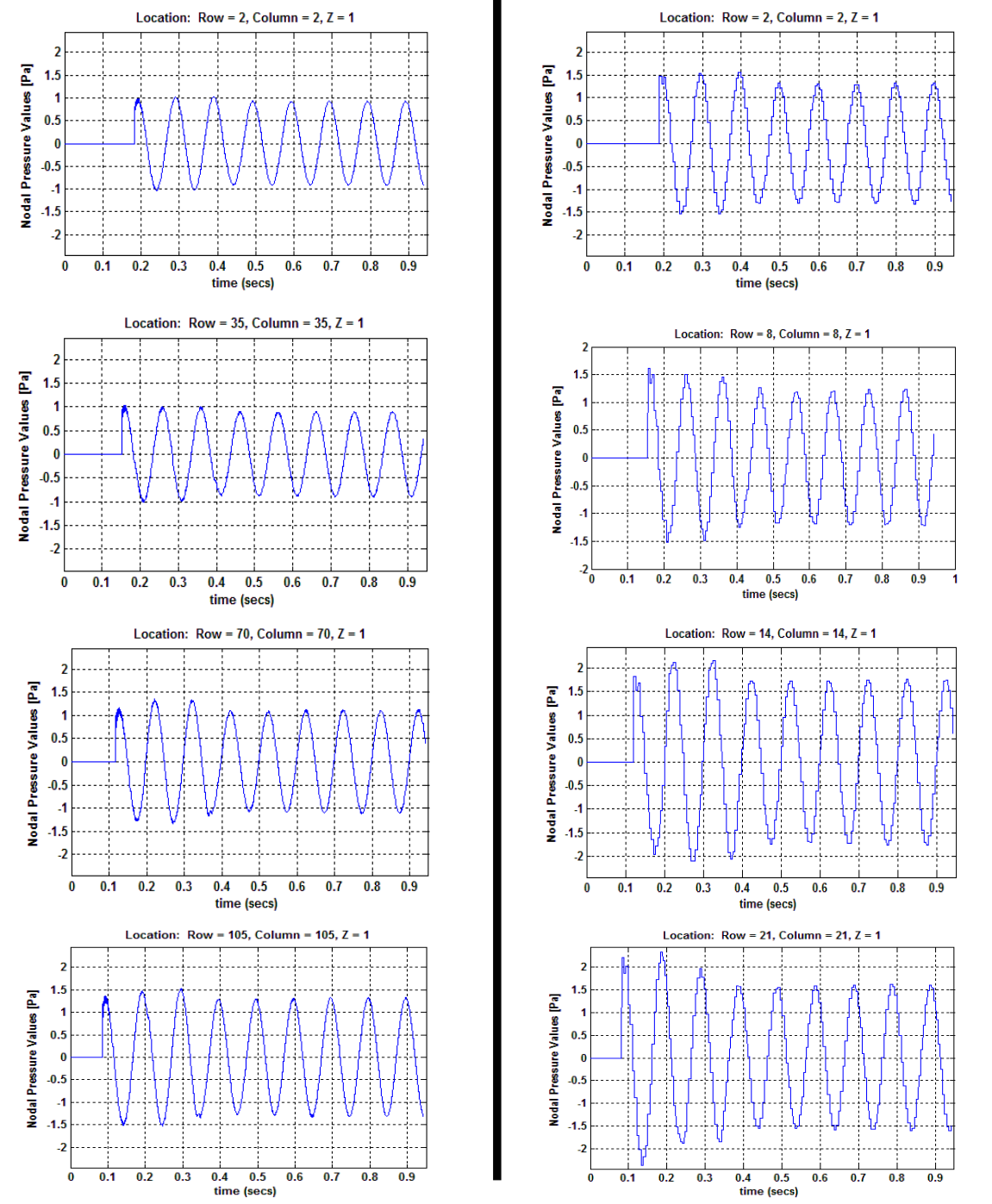


Figure 42. Comparison of a uniform and multi-grid, from a 10 (Hz) acoustic source, at near identical nodal distances from the source

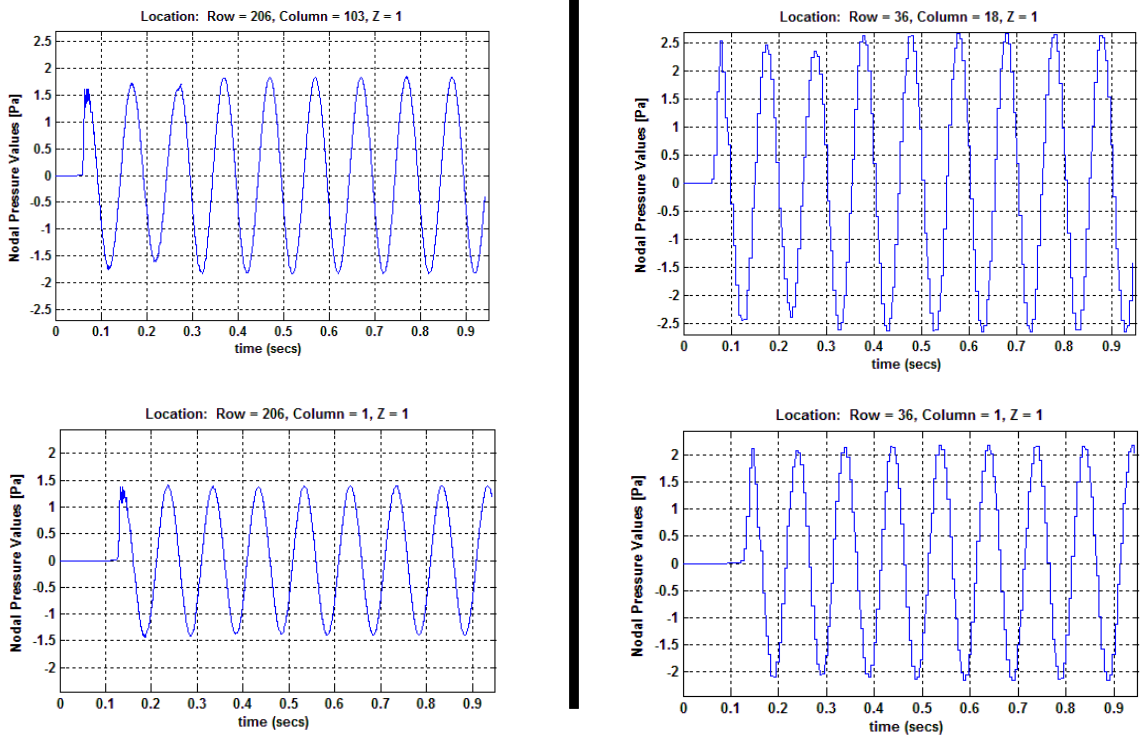


Figure 42. …continued from the previous page

FIGURES	CODE LINE	VARIABLE	NEW VALUE
Figure 4	Line 2	VN, HN	269
		ZN	1
	Line 3	R_Nodes	1
	Line 5	VS, HS	-5
	Line 6	Plots	[2 2 1; 28 28 1; 14 14 1; 21 21 1; 36 1 1; 36 18 1]
Figure 7	Line 16	speed_sound	1500
	Line 17	dx	¼
Figure 8	Line 20	Iterations	8000
	Line 4	freq	10
Figure 39	Line 13	ANS	1
	Line 20	Refine	[134 134 1 9]
	Line 32	SubDomainPoints	[2 3 1] * arbitrary selection inside the local grid

Table 19. Parameter values to refine the global grid from Table 18 to simulate multi-grid performance with non-reflecting boundary condtions

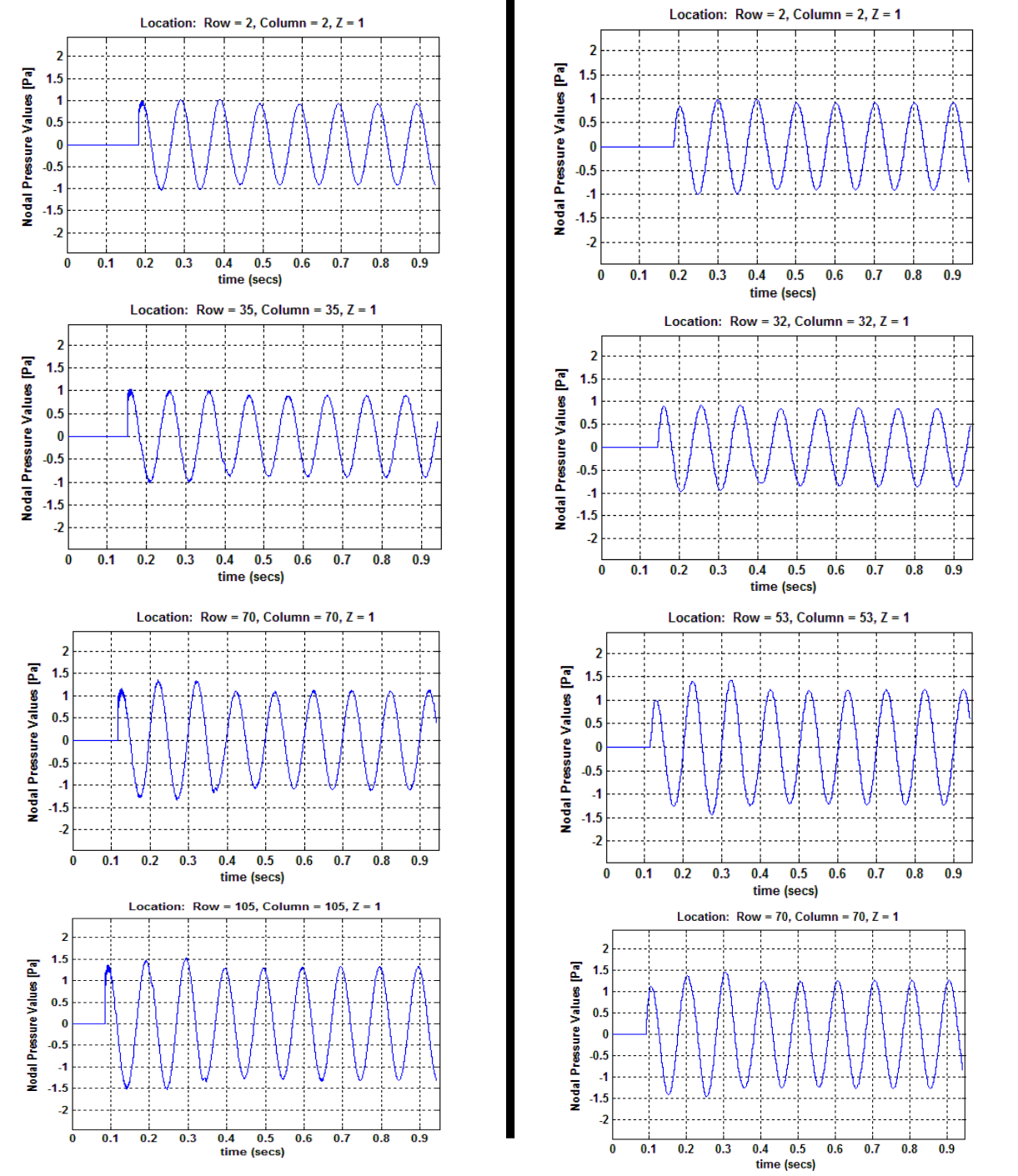


Figure 43. Comparison of a uniform and multi-grid, where the global grid points, of the multi-grid, were refined by a factor of 4 from Table 18. The nodal points from both domains are at approximate equal nodal distances from the 10 (Hz) acoustic source.

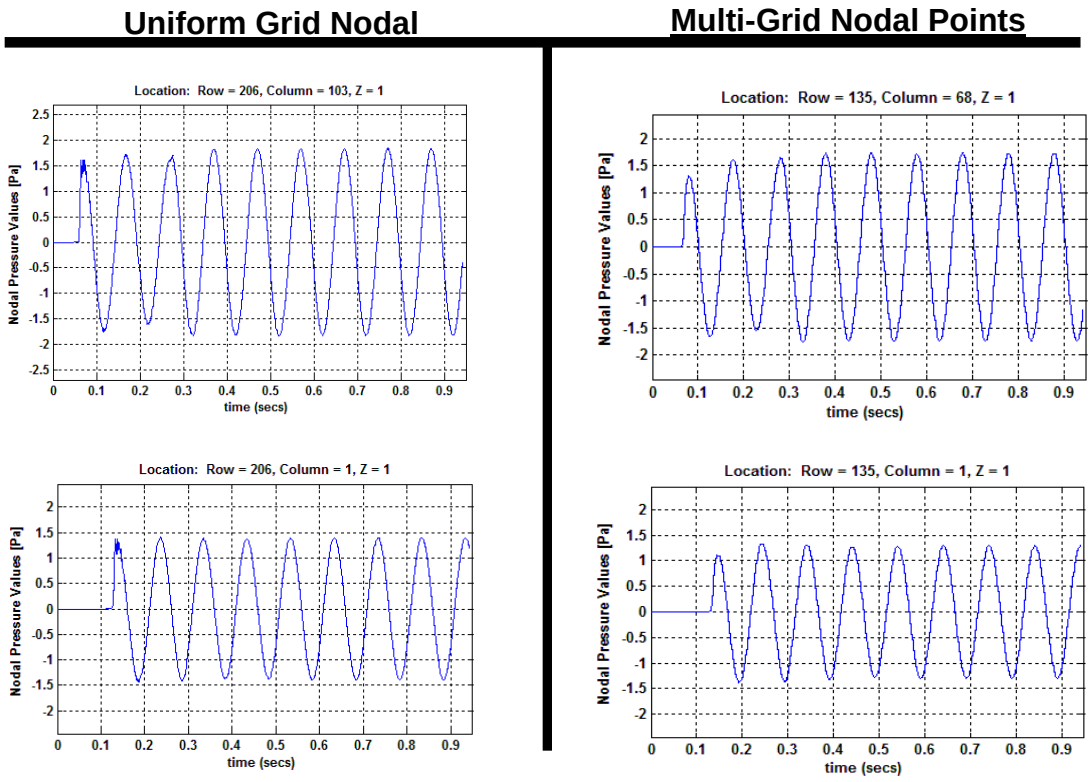


Figure 43…continued from the previous page

The next set of evaluations considers the effects of changing the mesh size inside the local grid. The local grid is increased by a factor of approximately 4. The reason for choosing this factor is for direct comparison with the global refinement process, which has been discussed. As a result, the closest odd number mesh size that is off by a minimum factor of 4, is from setting the *LNP* variable, in Equation (9), to 29. The variable *UNL* is the same as variable *VN* or *HN* from Table 17. Re-arrangement of Equation (9) to solve for GNP, gives 19.84, which is rounded to the nearest odd number (i.e., 19) as the new input assignment for variables *VN* and *HN*. A summary of all additional changes for this associated evaluation is shown in Table 20 with simulation results are shown

in Figure 44. The performance of the model shows that refining only the local grid does not produce more accurate results in the global domain by that associated method alone.

FIGURES	CODE LINE	VARIABLE	NEW VALUE
Figure 4	Line 2	VN, HN	19
		ZN	1
	Line 3	R_Nodes	1
	Line 5	VS, HS	0
	Line 6	Plots	[2 2 1; 3 3 1 ; 14 14 1; 21 21 1; 36 1 1; 36 18 1]
Figure 7	Line 16	speed_sound	1500
	Line 17	dx	1
Figure 8	Line 20	Iterations	2000
	Line 4	freq	10
Figure 39	Line 13	ANS	1
	Line 20	Refine	[9 9 1 29]
	Line 32	SubDomainPoints	[2 3 1] * arbitrary selection inside the local grid

Table 20. Parameter values to refine the local grid from Table 18 to simulate multi-grid performance with non-reflecting boundary condtions

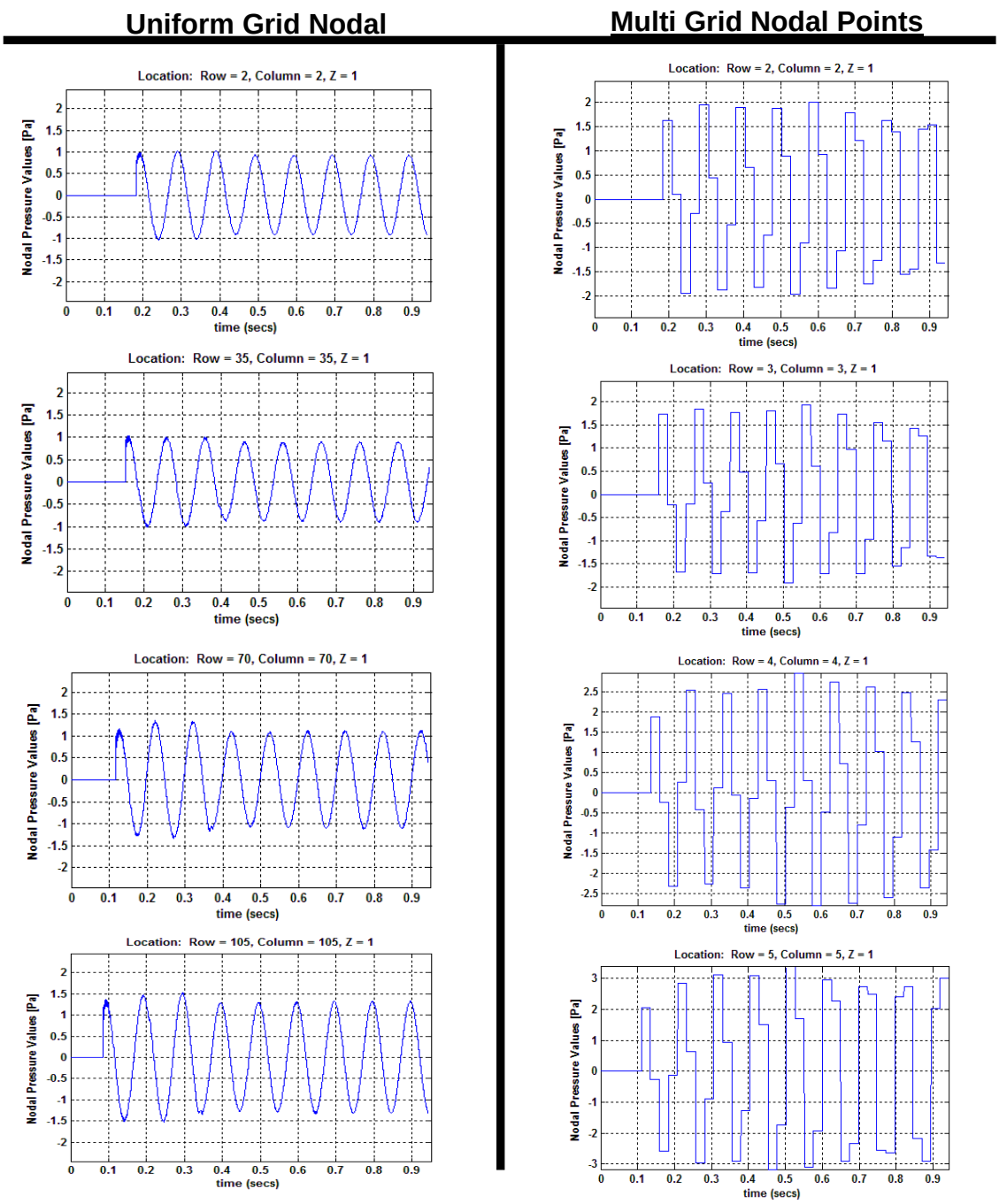


Figure 44. Comparison of a uniform and multi-grid, where the local grid points of the multi-grid were refined by a factor of 4 from Table 18. The nodal points from both domains are at approximate identical distances from the 10 (Hz) acoustic source.

Uniform Grid Nodal | **Multi-Grid Nodal Points**

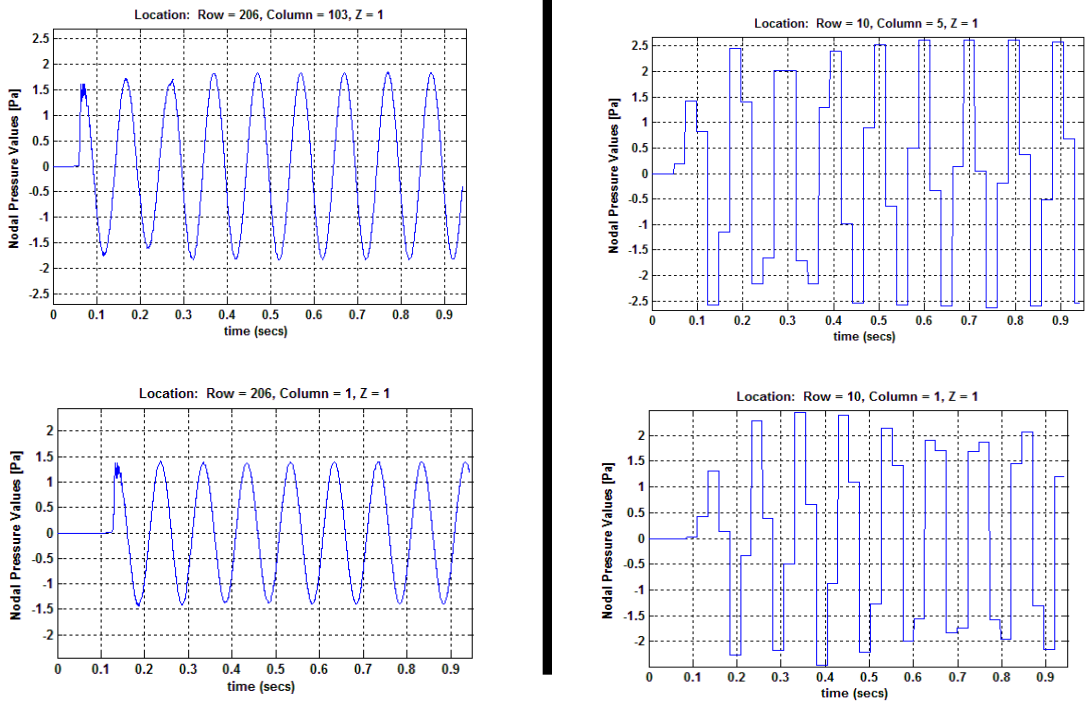


Figure 44…continued from the previous page

V. CONCLUSIONS AND RECOMMENDATIONS

The goal of this work has been to develop a computational model to accurately simulate acoustic propagation with Cellular Automaton methods. The nature of wave propagation in an ocean environment alludes to solving complex non-linear differential equations, which can be tedious and difficult to solve. Nevertheless, it has been shown that the application of a simple rule can capture the essence behind wave propagation. Moreover, a multi-grid refinement process has been developed to minimize the computational time in generating a solution. The associate improvement time is dependent upon the difference between the total numbers of grid points of the uniform grid and multi-grid schemes, respectively. Moreover, the ease with which different types of boundary conditions can be applied to any location inside the domain makes this method a viable modeling scheme for future naval applications.

On the contrary, the CA method does have limitations that require further research to improve its usefulness. In particular, all simulations have assumed a domain which exhibited homogeneous conditions. While, the prior serves as a good starting point, the nature of this type of domain does not accurately reflect the sound speed profile in an ocean environment. Moreover, other facets, such as salinity, water temperature, and water depth alter the travel velocity of sound in water. An amendment to this method must seek to resolve the aforementioned issue.

In addition, improvements to the multi-grid method can be explored. The methodology of this work only considers the placement of the acoustic source inside the local grid. Further research should investigate analyzing the accuracy of the results, after the acoustic source has been placed inside the global grid. Development of such a scheme can go a long ways in fully realizing the potential of this method for other applications, such as understanding the response in a localized area to underwater shock waves.

In summary, this work has been viewed as a starting point to understanding the manner in which we interpret and model acoustic propagation. The modeling methodology employed in this thesis uses a local rule on microscopic scale to devise a response on a macroscopic level as the summation of all microscopic activities. The prior approach is not a typical path sought by engineers to solve real world problems. However, this particular method shows great potential for application in other areas of mechanical engineering. The results of this paper have explored the usefulness in one area, but the framework of the method can be carried over to other areas for future research.

APPENDIX A. MAIN FUNCTION

```
%% Main Function to execute the CA model
close all, clear all, clc
%% Step 1) Define The Space Domain and Associated Variables
%% NODAL Variables
    % 2D or 3D Dimensions (Choose Odd Number for VN, HN, & ZN)
    VN =61;        HN =61;       ZN =1;
    % Number of Nodes in vertical(VN), Horizontal(HN), and Z (ZN) direction
    VS = 0;         HS =0;          ZS = 0;
    % Shifts from the center of the domain
    % Negative values will correlate to going left for HS and up for VS.
    % A Negative Value for Z correlates to coming out of the plane
    speed_sound=1500;                    dx=1;
    % Number o1f times to update the CA Space
    iterations=400;
 %% Defining the Source Diameter, Distance from Middle, and Values
 % Source Diameter
 R_Nodes=input(['\nPlease set the radial nodal distance',...
        ' from the source: ']);
 % Circle Source Distance from the Middle of the Domain
%% Specify the Domain Plot Location(s)
 Plots=input(['\nPlease specify the point locations, inside the CA\n',...
     ' space domain, where you like to see how a point is changing\n',...
     ' with respect to time. If you would like to see more than one\n',...
     ' point please enter the information in matrix format.\n\n -- Ex)',...
     ' [Row_1, Col_1, Z_1; Row_2, Col_2, Z_2;...] i.e. \n----->']);
%% Refine Mesh Option & Specify SubDomain Mesh Plot Location
global ANS Refine SubDomainPoints cas
ANS = input(['\nWould you like to place the acoustic source in a refined\n',...
    ' location in a portion of the ',num2str(VN),...
    'x',num2str(HN),'x',num2str(ZN),' CA space domain?  \n',...
    ' Please select (1) for Yes or (2) for No.\n--->'] );
if ANS==1
    Comment=[' Ensure Row is <=',num2str(VN),', Column is <= ',...
        num2str(HN),', Z is <= ', num2str(ZN)];
    Refine = input(['\nEnter the Row, Column, & Z-Coordinate, of the',...
        ' upper left hand\n corner square grid coordinates where',...
        ' the CA space domain will be refined\n with the associated',...
        ' acoustic pressure source.\nLastly, please ',...
        ' enter a positive integer, at the end of the\n array,',...
        ' that will correlate to the dimensions of the refined square\n',...
        ' space.',...
        ' Your input will be an array as shown below as an example\n\n',...
        '--Example: [Row, Column, Z, Positive Integer Value];\n\n',...
        Comment,',\n and the Positive Integer Value is greater than\n',...
        ' the radial nodal diameter which is currently\n',...
        ' set to ',num2str(R_Nodes),'.\n--->']);

    SubDomainPoints=input(['\nPlease enter an array or matrix(for',...
        ' multiple graphs\n in the subdomain) to specify the location(s)',...
        ' within\n the Mesh. If the space is 2D then enter 1 in each',...
        ' last column.\n Below is the max value not to exceed, for',...
        ' each row, column, and Z\n entries, based on your input',...
        ' provided for Positive Integer Value\n above (3D Case):\n    ',...
        num2str(Refine(end)')),...
        '\nPlease enter the Refine Mesh location in the same format as was',...
        '\n in selecting the domain coordinates for plotting:\n',...
```

```
        '---Example: [Row_1, Column_1, Z_1; Row_2, Column_2, Z_2; etc...]',...
        '\n--->']);
    s=SubDomainPoints;
    if ((s(1)>Refine(end)) || (s(2)> Refine(end)) ||  s(3)>(Refine(end)))
        error(['Verify each row entry does not exceed ',...
            num2str(Refine(end)'),'. If 2D ensure the last ',...
            'columns does not exceed 1'])
    end
end

%% Step 2) Initialize the 2D or 3D Space Diameter(s) Source Value
if ANS==1
    % Global/Uniform Grid
    VN2=VN; HN2=HN; ZN2=ZN;       HS2=0; VS2=HS2;  ZS2=HS2;
    % Local Grid - each side will be the same length
    VN=Refine(end); HN=VN;
    if (VN2>1  && HN2>1  && ZN2>1)
        ZN=VN;
    else
        ZN=1;
    end
end
CAS = CirSinkSource(VN, HN, ZN, HS, VS, ZS);
% Radial Nodes of the Source
[Indices,OC,OR,OZ] = AcousticIndices(CAS,R_Nodes);
%%  Time domain parameters
dt = CA_Time_Conversion (speed_sound,CAS,dx);
[Value,time] = CircleValFunc(dt,iterations);
    % Values of the Source Diameter
CAS =  AcousticRadius(CAS,Indices,Value,1);
    % Every node internal to the Diameter is set equal to zero
CAS = AdditionalConstraints(CAS,Indices);
    % Every Node outside of Source Diameter is now set equal to zero
for i=1:length(OC)
    CAS(OR(i), OC(i), OZ(i))=0;
end
if ANS ==1
    cas=CAS;                % Local Grid (or SubDomainData)Domain Information
    CAS=zeros(VN2,HN2,ZN2); % Uniform/Global Grid (or DomainData) Information
end

%% Step 3) Update the CA Space by iterating the Black and White Nodes
[DomainData,SubDomainData]=PermuteCASpace(CAS,Indices,iterations,Value);

%% Step 4)  Plot Selected Points, Center Plane, and/or Volume Plot
CA_DomainPointPlots(Plots,DomainData,SubDomainPoints,SubDomainData,time)

%% Step 5 Transmission Loss and Animation Plots
type={'Local Grid Data Plot','Uniform/Global Grid Data Plot'};
CA_3D_and_VolumePlot (DomainData,dt, type{2},ZS)
    % Global/Uniform Grid
TranssmissionLoss_dB(DomainData,Value,type{2})
    % Global/Uniform Grid-Based Solely on RMS Values
%% Un Comment if interested in Transmission Losses Based on Peak Values
% TranssmissionLoss(DomainData, iterations,type{2})
    % Global/Uniform Grid-Based Solely on Peak Values
%% Un Comment if interested in viewing the Local Grid
% To avoid confusion comment the corresponding functions that correlate
% with DomainData under Step 5 to avoid confusion
if ANS==1
```

```
    % CA_3D_and_VolumePlot (SubDomainData,dt,type{1},ZS)
        % Local Grid
    % TranssmissionLoss_dB(SubDomainData,Value,type{1})
        % Local Grid-Based Solely on RMS Values
end
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. SUB FUNCTIONS

A. CIRCLE SINK SOURCE FUNCTION

```
function [CAS] = CirSinkSource (VN, HN, ZN, HS, VS, ZS)
%% Initializing the CA 2D/3D Space
%  VN =Vertical Nodes (or Rows)
%  HN = Horizontal Nodes (or Columns)
%  ZN = Nodes in the 3rd dimension
%  ZS = Offset from the center of the grid into the 3rd dimension
%  VS = Offset from the center of the grid in the vertical direction
%  HS = Offset from the center of the grid in the horizontal direction
x=-floor(VN/2):floor(VN/2);
y=-floor(HN/2):floor(HN/2);
z=-floor(ZN/2):floor(ZN/2);
[X, Y, Z] = meshgrid(x,y,z);
CAS=(X-HS).^2 +(Y-VS).^2 + (Z-ZS).^2;
end
```

B. ACOUSTIC INDICES FUNCTION

```
function [Indices,OC,OR,OZ]=AcousticIndices(CAS,R_Nodes)
% This function will return the Indices of
% Matrix Values that make up the circle of the source. In addition, the
%  indices of the internal nodes that are equal to zero are returned

% DRow - Rows of the matrix values along the diameter of the source
% DCol - Columns of the matrix values along the diameter of the source
% DZ - Z direction of the 3D matrix along the outer diameter of the source
% RowZero – Nodes internal to the acoustic source, Row Values, that equal zero
% ZZero – “                                        ,”  Z Values  ‘”             “
% ColZero - “                                      ,”  Column Values  ‘”        “
R_Nodes=ceil(R_Nodes/2)-1;
if R_Nodes<=-1
    % Diameter Value of < 1
    error([‘Please set radial nodal distance from the source to be’,...
        ‘ equal to 1 or greater’])
elseif R_Nodes == 0
    % Diameter Value of 1 or 2
    CAS_Diameter = CAS==1;
elseif R_Nodes==1
    % Diameter Value of 3 or 4
    CAS_Diameter = CAS==1 | CAS==2;
else
    % Diameter Value >4
    CAS_Diameter = CAS==(R_Nodes.^2)  | CAS==(R_Nodes.^2 +1) | ...
                                                CAS==(R_Nodes.^2 -1);
end
% Source Diameter Row, Column, and Z Indices
    [I.DRow, I.DCol, I.DZ]=ind2sub(size(CAS_Diameter),...
        find(CAS_Diameter==1));
% Nodes Outside of the Desired Radius
    CAS_OuterValues = CAS>=(R_Nodes).^2+2;
    [OR,OC,OZ] = ind2sub(size(CAS_OuterValues),find(CAS_OuterValues==1));
% Inner Diameter Zero Row and Columns
    CAS_InnerZeros=CAS_OuterValues|CAS_Diameter;
```

```
    [RowZero, ColumnZero, ZZero]=ind2sub(size(CAS_InnerZeros),...
                                                 find(CAS_InnerZeros==0));
    I.RowZero=RowZero;         I.ColZero=ColumnZero;      I.ZZero=ZZero;
    Indices=I;
end
```

C. TIME CONVERSION FUNCTION

```
function [dt] = CA_Time_Conversion (SS,CAS,dx)
% This functions returns the incremental changes in time (dt)
% SS = Speed of Sound in the Medium
% ND = Number of Dimensions
% dx = distance between nodes in the horizontal/vertical/Z planes
% CAS = Cellular Automaton Space Elements (or Grid Point Values)

[L1, L2, L3]=size(CAS);
   if (L1>1 && L2>1 && L3>1)
        ND = 3;
   else
        ND = 2;
   end
   dt = dx/(sqrt(ND)*SS);
end
```

D. CIRCLE VALUE FUNCTION

```
% Circle Source Values Function
% This function creates an array of all pressure values for the amount of
% iterations specified by the user
function[Values,time] = CircleValFunc(dt,iterations)
time = 0:dt:dt*(iterations-1);
freq = input(['\nPlease enter the frequency of the sinusoidal source',...
            ' (in Hertz).\n If you desire a constant frequency enter 0: ']);
Decay = input('\nPlease enter the decay constant: ');
global Peak
    % Defining the Peak varialbe as global as being usefull when the
    % TransmissionLoss function is called
Peak = input('\nPlease enter the peak value of the input source: ');
% Sinusoidal & Exponential Decay Function
Values(1:length(time))= Peak.*cos(2*pi.*freq.*time).*exp(-Decay.*time);
end
```

E. ACOUSTIC RADIUS FUNCTION

```
function [CAS] =  AcousticRadius(CAS,Indices,Value,index)
% This particular function will maintain the Diameter of the source at a
% value determined by the variable 'Value'.
% Value – Array of acoustic source values, along the source diameter, for all
%    iterations
% index – determines which iterations value to extract from the 'Values' array
% Indices - contains all of the diameter indices of the acoustic source
D=Indices;
if isstruct(D)
    for i=1:length(D(1).DRow)
      CAS(D(1).DRow(i),D(1).DCol(i),D(1).DZ(i)) = Value(index);
    end
end
end
```

F. ADDITIONAL CONSTRAINTS FUNCTION

```
function [CAS] = AdditionalConstraints(CAS,Indices)
%%
% Sets the nodes internal to the source diameter nodes equal to zero
I=Indices;
   for i=1:length(I(1).RowZero)
        CAS(I(1).RowZero(i),I(1).ColZero(i),I(1).ZZero(i))=0;
   end
%%
end
```

G. PERMUTE CELLULAR AUTOMATON DOMAIN

1. Permutation the Uniform or Global Grid Points

```
function [Position,positioN]= PermuteCASpace(CAS,Indices,iterations,Value)
% This function will permeate all black and white nodes of the uniform or
%   multi-grid
% Position - correlates to the global/uniform grid
% positioN - correlates to the local grid
global ANS cas index INDEX Refine
%% Define Boundary Constraints
OPTIONS = [' -Please enter a number from  the options below to specify the
\n',...
    '  constraint:\n  (1) Fix\n  (2) Reflective \n  (3) Free/Constant Flux',...
    ' Conditions\n  (4) Non-Reflective\n  (5) Refractive....If 5 is',...
    ' selected please enter an array in \n the following format',...
    ' [5, A Number between 0 and 1]...The 2nd value\n will correlate to',...
    ' the percentage reflected do to the wave arriving\n at an incident',...
    ' angle  \n '];
OPTIONS2 = [' (6) Curved Bottom Hill with Fixed Boundaries\n',...
            '  (7) Curved Bottom Hill with Reflective Boundaries\n',...
            '  (8) Curved Bottom Hill with Refractive Boundaries',...
            '....If 8 is\n selected please enter an array in ',...
            ' the following format\n [8, A Number between 0 and 1].',...
            ' The 2nd value will correlate to\n the percentage',...
            ' reflected do to the wave arriving at an incident\n',...
            ' angle.  \n '];
OPTIONS3 = ['  (9) Wedge Bottom Hill with Fixed Boundaries\n',...
            '  (10) Wedge Bottom Hill with Reflective Boundaries\n',...
            '  (11) Wedge Bottom Hill with Refractive Boundaries',...
            '....If 11 is\n selected please enter an array in ',...
            ' the following format\n [11, A Number between 0 and 1].',...
            ' The 2nd value will correlate to\n the percentage',...
            ' reflected do to the wave arriving at an incident\n',...
            ' angle.  \n '];

constraint1 = input(['LEFT BOUNDARY\n',   OPTIONS]);
constraint2 = input(['RIGHT BOUNDARY\n',  OPTIONS]);
constraint3 = input(['TOP BOUNDARY\n',    OPTIONS]);
constraint4 = input(['BOTTOM BOUNDARY\n', OPTIONS,OPTIONS2,OPTIONS3]);

[L1, L2, L3]=size(CAS);
if ~(L3==1 || L2==1 || L1==1)
    constraint5 = input(['FRONT BOUNDARY \n', OPTIONS]);
    constraint6 = input(['BACK BOUNDARY \n',  OPTIONS]);
end
```

```
if constraint4(1) == 6 || constraint4(1)==7 || constraint4(1)==8
    Indices = CurvedHillIndices(CAS,Indices);
end
if constraint4(1) == 9 || constraint4(1)==10 || constraint4(1)==11
    Indices = WedgeHillIndices(CAS,Indices);
end
           %%%% Pre-Allocating for speed %%%%
% Larger Domain
Position(:,:,:,1:iterations)= zeros(L1,L2,L3,iterations);
% Sub Domain
if ANS==1
    [L1,L2,L3]=size(cas);
    positioN(:,:,:,1:iterations)= zeros(L1,L2,L3,iterations);
    index=1;  INDEX=1;
else
    positioN=[];
end
%
s=1;
% Subtracting three to allow the number of permutation in the local grid
% to correlate to the mesh spacing
if ANS==1
   Refine(end)=Refine(end)-3;
end
fprintf(['Iterations Completed: ',num2str(s),'\n'])
%%             Apply Constraints & Permute the uniform/multi-grid
while s

if ANS == 1
    Position(:,:,:,index) = CAS;  % Saving Global grid data
    fprintf(['Iterations Completed: ',num2str(index),'\n'])
else
    Position(:,:,:,s) = CAS;      % Saving Uniform grid data
    fprintf(['Iterations Completed: ',num2str(s),'\n'])
end

       %% Permute the local grid
       if ANS==1
        [cas,positioN,Position]= PermuteMeshCASpace(positioN,cas,...
                                              Indices,Value,Position,CAS);
        %  Update the global grid  %
        CAS= UpdateRefineCASMesh(positioN(:,:,:,index-1),CAS);
       end

       %% Permute the global/uniform grid
       CAS = CA_WhiteInnerNodes(CAS);
      %%%%%%%%%%%% Apply the Constraints/Boundary Conditions %%%%%%%%%%%%%%
                           % Boundary Constraints
        CAS = CA_BoundaryConstraints(CAS, Indices, 'left',   constraint1);
        CAS = CA_BoundaryConstraints(CAS, Indices, 'right',  constraint2);
        CAS = CA_BoundaryConstraints(CAS, Indices, 'top',    constraint3);
        CAS = CA_BoundaryConstraints(CAS, Indices, 'bottom', constraint4);
        if L3>1
          CAS = CA_BoundaryConstraints(CAS, Indices, 'front', constraint5);
          CAS = CA_BoundaryConstraints(CAS, Indices, 'back',  constraint6);
        end

        if ANS==2
```

```
                % Update the Source Value Indices
    CAS = AcousticRadius(CAS,Indices,Value,s);
                %  Additional Constraints
    CAS = AdditionalConstraints(CAS,Indices);
 else
    CAS= UpdateRefineCASMesh(positioN(:,:,:,index-1),CAS);
 end

 if ANS ==2
    s=s+1;
    if s > iterations
          fprintf(['Iterations Completed: ',num2str(s),'\n'])
          return
    else
        Position(:,:,:,s)=CAS;
    end
 else
    if index > iterations
          fprintf(['Iterations Completed: ',num2str(index),'\n'])
          return
    else
        Position(:,:,:,index) =CAS;
    end
 end
%% Permute the local grid
if ANS==1
  [cas,positioN,Position]= PermuteMeshCASpace(positioN,cas,Indices,...
                                              Value,Position,CAS);
  %  Update the global grid
  CAS= UpdateRefineCASMesh(positioN(:,:,:,index-1),CAS);
end
%% Permute the global/uniform grid
CAS = CA_BlackInnerNodes(CAS);
%%%%%%%%%%%% Apply the Constraints/Boundary Conditions %%%%%%%%%%%%%%
                    % Boundary Constraints
 CAS = CA_BoundaryConstraints(CAS, Indices, 'left',   constraint1);
 CAS = CA_BoundaryConstraints(CAS, Indices, 'right',  constraint2);
 CAS = CA_BoundaryConstraints(CAS, Indices, 'top',    constraint3);
 CAS = CA_BoundaryConstraints(CAS, Indices, 'bottom', constraint4);
 if L3>1
    CAS = CA_BoundaryConstraints(CAS, Indices,'front', constraint5);
    CAS = CA_BoundaryConstraints(CAS, Indices, 'back', constraint6);
 end
 if ANS==2
                % Update the Source Value Indices
    CAS = AcousticRadius(CAS,Indices,Value,s);
                    %  Additional Constraints
    CAS = AdditionalConstraints(CAS,Indices);
 else
    CAS= UpdateRefineCASMesh(positioN(:,:,:,index-1),CAS);
 end

 if ANS ==2
    s=s+1;
    if s > iterations
          fprintf(['Iterations Completed: ',num2str(s),'\n'])
          return
    else
        Position(:,:,:,s) = CAS;
    end
```

```
        else
            if index > iterations
                  fprintf(['Iterations Completed: ',num2str(index),'\n'])
                  return
            else
                Position(:,:,:,index) =CAS;
            end
        end
end

end
```

2. Permute the Local Grid Points

```
function [cas,positioN,Position]= PermuteMeshCASpace(positioN,cas,...
                                                    Indices,Value,Position,CAS)
% This function will permeate the points that make local grid, which is a
%   subset of the multi-grid
global Refine index
%%                  Apply Constraints & Permute Space
[L1,L2,L3]=size(cas);
s=1;
constraint=4;
while s
positioN(:,:,:,index) = cas; % Saving local grid data
Position(:,:,:,index) = CAS; % Maintaining global grid in sync with local grid
       % Permute Odd Cells (or White Cells)
        cas = CA_WhiteInnerNodes(cas);
          %%%%%%%%%%%%%%%%%%%% Apply the Constraints %%%%%%%%%%%%%%%%%%%%%%%
                              % Boundary Constraints
       cas = CA_BoundaryConstraints(cas, [], 'left',constraint);
       cas = CA_BoundaryConstraints(cas, [], 'right',constraint);
        cas = CA_BoundaryConstraints(cas, [], 'top', constraint);
        cas = CA_BoundaryConstraints(cas, [], 'bottom',constraint);
        if ~(L1==1 || L2==1 || L3==1)
          cas = CA_BoundaryConstraints(cas, [], 'front',constraint);
          cas = CA_BoundaryConstraints(cas, [], 'back', constraint);
        end

                        % Update the Source Value Indices

        if index<=length(Value)
            cas = AcousticRadius(cas,Indices,Value,index);
                        %  Additional Constraints
            cas = AdditionalConstraints(cas,Indices) ;
        end

         s=s+1; index=index+1;
         if (s > (Refine(end)) || (index>length(Value)))
            return
        end

positioN(:,:,:,index) = cas;
Position(:,:,:,index) = CAS;

        % Permute Even Cells (or Black Cells)
        cas = CA_BlackInnerNodes(cas);
         %%%%%%%%%%%%%%%%%%%% Apply the Constraints %%%%%%%%%%%%%%%%%%%%%%%
```

```
                        % Boundary Constraints
        cas = CA_BoundaryConstraints(cas, [], 'left',constraint);
        cas = CA_BoundaryConstraints(cas, [], 'right',constraint);
        cas = CA_BoundaryConstraints(cas, [], 'top', constraint);
        cas = CA_BoundaryConstraints(cas, [], 'bottom',constraint);
        if ~(L1==1 || L2==1 || L3==1)
          cas = CA_BoundaryConstraints(cas, [], 'front',constraint);
          cas = CA_BoundaryConstraints(cas, [], 'back', constraint);
        end
                    % Update the Source Value Indices

        if index<=length(Value)
            cas = AcousticRadius(cas,Indices,Value,index);
                        %  Additional Constraints
            cas = AdditionalConstraints(cas,Indices);
        end
        s=s+1; index=index+1;
        if (s > (Refine(end)) || (index>length(Value)))
            return
        end
end

end
```

3. Permute the Black(Even) Nodes

```
function CA_Space_Blk = CA_BlackInnerNodes(CAS)
% CAS: is the CA Space
% The following function will return the permutated inner nodes, which are
% black (even), inside the Cellular Automaton Space.

% Even Cells (or Black Cells) 2D or 3D CASE Inner Nodes
[Length, Length2, Length3]=size(CAS);
%% 2D Case
if Length3==1
    New_C = @(W, E, S, N, Old_C) (1/2) * (W +E +S + N -2*Old_C);
    CAS_Modified(1,:)=reshape(CAS,1,Length*Length2);
    i=1;
    for column=1:(Length2)
        for row=1:(Length)
            if (((row>1)&&(column>1)) && ((row<Length)&&(column<Length2)))
                % Inner Nodes
                if ~rem(i,2)
                    % Black Nodes Only
                    W=CAS(row,column-1);   E=CAS(row,column+1);
                    N=CAS(row-1,column);   S=CAS(row+1,column);
                    old_C=CAS(row,column);
                    CAS_Modified(i)=New_C(W, E, S, N, old_C);
                end
            end
            i=i+1;
        end
    end
    % Delete the extra column created by i=1+1;
    CAS_Modified2(1,1:Length*Length2)=CAS_Modified(1,1:Length*Length2);
    CA_Space_Blk = reshape(CAS_Modified2, Length, Length2);
%% 3D CASE
elseif Length3>1
    New_C = @(W,E,S,N,F,B,C) (1/3) .* (W + E + S + N + F + B - 3.*C);
```

```
    CAS_Modified(1,:)=reshape(CAS,1,Length*Length2*Length3);
    i=1;
    for z=1:(Length3)
        for column=1:(Length2)
            for row=1:(Length)
            if (((row>1)&&(column>1)&&(z>1)) && ...
                   ((row<Length)&&(column<Length2)&&(z<Length3)))
                    % Inner Nodes
                    if ~rem(i,2)
                        % Black Nodes Only
                        W=CAS(row,column-1,z);   E=CAS(row,column+1,z);
                        N=CAS(row-1,column,z);   S=CAS(row+1,column,z);
                        F=CAS(row,column,z-1);   B=CAS(row,column,z+1);
                        C=CAS(row,column,z);
                        CAS_Modified(i)=New_C(W, E, S, N, F, B, C);
                    end
             end
                i=i+1;
            end
        end
    end
    LLL=Length*Length2*Length3;

    % Delete the extra column created by i=1+1;
    CAS_Modified2(1,1:LLL)=CAS_Modified(1,1:LLL);
    CA_Space_Blk = reshape(CAS_Modified2, Length, Length2,Length3);
%% NO CASE
else
    fprintf('Please resolve issues with the BlackInnerNodes Sub Function\n');
    return
end


end
```

4. Permute the White (Odd) Nodes

```
function CA_Space_White = CA_WhiteInnerNodes(CAS)
% The following function will return the permutated inner nodes that are
% white (odd) inside the Cellular Automaton Space.
% Odd Cells (or White Cells) 2D or 3D CASE Inner Nodes
[Length, Length2, Length3]=size(CAS);
%% 2D Case
if Length3==1
    New_C = @(W, E, S, N, Old_C) (1/2) * (W +E +S + N -2*Old_C);
    CAS_Modified(1,:)=reshape(CAS,1,Length*Length2);
i=1;
  for column=1:(Length2)
    for row=1:(Length)
         if (((row>1)&&(column>1)) && ((row<Length)&&(column<Length2)))
         % Inner Nodes
            if rem(i,2)
                % White Nodes Only
                W=CAS(row,column-1);   E=CAS(row,column+1);
                N=CAS(row-1,column);   S=CAS(row+1,column);
                old_C=CAS(row,column);
                CAS_Modified(i)=New_C(W, E, S, N, old_C);
            end
         end
```

```
       i=i+1;

    end
  end
  % Delete the extra column created by i=1+1;
    CAS_Modified2(1,1:Length*Length2)=CAS_Modified(1,1:Length*Length2);
    CA_Space_White= reshape(CAS_Modified2, Length, Length2);
%% 3D Case
elseif Length3>1
    New_C = @(W,E,S,N,F,B,C) (1/3) .* (W + E + S + N + F + B - 3.*C);
    CAS_Modified(1,:)=reshape(CAS,1,Length*Length2*Length3);
    i=1;
   for z=1:(Length3)
    for column=1:(Length2)
        for row=1:(Length)
             if (((row>1)&&(column>1)&&(z>1))&&...
                    ((row<Length)&&(column<Length2)&&(z<Length3)))
             % Inner Nodes
                if rem(i,2)
                    % White Nodes Only
                    W=CAS(row,column-1,z);   E=CAS(row,column+1,z);
                    N=CAS(row-1,column,z);   S=CAS(row+1,column,z);
                    F=CAS(row,column,z-1);   B=CAS(row,column,z+1);
                    C=CAS(row,column,z);
                    CAS_Modified(i)=New_C(W, E, S, N, F, B, C);
                end
            end
           i=i+1;

        end
    end
   end
    LLL=Length*Length2*Length3;
    % Delete the extra column created by i=1+1;
    CAS_Modified2(1,1:LLL)=CAS_Modified(1,1:LLL);
    CA_Space_White = reshape(CAS_Modified2, Length, Length2,Length3);
%% No CASE
else
    fprintf('Please resolve issues with the WhiteInnerNodes Sub Function\n');
    return
end

end
```

5. Application of the Boundary Conditions

```
function [CAS] = CA_BoundaryConstraints(CAS, Indices, side, constraint)

% The following function will enforce the boundary conditions of the Cellular
% Automaton Space (Uniform and Multi-Grid). The choice of options will
% determine which boundary conditions to utilize.

% The 'constraint' variable is a scalar or 1x2 array if '5,8,10 or 11' is
% selected as the associated boundary conditions

[L1, L2, L3]=size(CAS);
% Exist the function if the constraint variable is empty
```

```
if isnan(constraint)
    return
end
%%
if  strcmp(side, 'left')
      % The left wall
     if constraint(1) == 1           % (Fixed Conditions)
        CAS(:,1,:) = 0;
     elseif constraint(1) == 2       % (Reflective Conditions)
        CAS(:,1,:) = CAS(:,3,:);
     elseif constraint(1) == 3          % (Free/Constant Flux Conditions)
         CAS(:,1,:) = 2.*CAS(:,2,:) -  CAS(:,3,:);
     elseif constraint(1) == 4       % (Non-Reflective Conditions)
        CAS(:,1,:) = CAS(:,2,:);
     elseif constraint(1) == 5       % (Refractive Conditions ...Flat Bottom)
        CAS(:,1,:) =  CAS(:,3,:).*(constraint(2)) + ...
                                            CAS(:,2,:).*(1-constraint(2));
     end
%%
elseif strcmp(side, 'right')
    % The right of the 3D CA space
    if  constraint(1) == 1          %  (Fixed Conditions)
        CAS(:,L2,:) = 0;
    elseif constraint(1) == 2       %  (Reflective Conditions)
        CAS(:,L2,:)=CAS(:,L2-2,:);
    elseif constraint(1) == 3       %   (Free/Constant Flux Conditions)
        CAS(:,L2,:) = 2.*CAS(:,L2-1,:) - CAS(:,L2-2,:);
    elseif constraint(1) == 4       %  (Non-Reflective Conditions)
        CAS(:,L2,:)=CAS(:,L2-1,:);
    elseif constraint(1) == 5       % (Refractive Conditions ...Flat Bottom)
        CAS(:,L2,:) = CAS(:,L2-2,:).*constraint(2) + ...
                                          (1-constraint(2))*CAS(:,L2-1,:);
    end
%%
elseif strcmp(side, 'bottom')
    % The bottom of the 2D/3D CA space
    if constraint(1) == 1          % (Fixed Conditions)
        CAS(L1,:,:)=0;
    elseif constraint(1) == 2      % (Reflective Conditions)
        CAS(L1,:,:)=CAS(L1-2,:,:);
    elseif constraint(1) == 3      % (Free/Constant Flux Conditions)
        CAS(L1,:,:) = 2.*CAS(L1-1,:,:) - CAS(L1-2,:,:);
    elseif constraint(1) == 4      % (Non-Reflective Conditions)
        CAS(L1,:,:)=CAS(L1-1,:,:);
    elseif constraint(1) == 5      % (Refractive Conditions ...Flat Bottom)
        CAS(L1,:,:)= constraint(2).*CAS(L1-2,:,:)+ ...
                                        (1-constraint(2)).*CAS(L1-1,:,:);
    elseif constraint(1) == 6      % (Curved Hill Fixed Boundary)
            CAS(Indices.All_Hill_Indexes) = 0;
    elseif constraint(1) == 7      % (Curved Hill Reflective Boundary)
        IR=(Indices.Hill_Row_Borderline(:)-2)';
        IC=Indices.Hill_Col_Borderline;
        IZ=Indices.Hill_Z_Borderline;
        RR=sub2ind(size(CAS), IR,IC,IZ);
        CAS(Indices.All_Hill_Indexes) = 0;
        CAS(Indices.HillBorderIndexes)=CAS(RR);
    elseif constraint(1) ==8       % (Curved Hill Refractive Boundary)
        IR1=Indices.Hill_Row_Borderline(:)-1;
        IR2=Indices.Hill_Row_Borderline(:)-2;
        IC=Indices.Hill_Col_Borderline;
```

```
        IZ=Indices.Hill_Z_Borderline;
        RR1=sub2ind(size(CAS), IR1',IC,IZ);
        RR2=sub2ind(size(CAS), IR2',IC,IZ);
        CAS(Indices.All_Hill_Indexes) = 0;
        CAS(Indices.HillBorderIndexes)= constraint(2).*CAS(RR2)+ ...
                                        (1-constraint(2)).*CAS(RR1);
    elseif constraint(1) == 9      % (Wedge Hill Fixed Boundary)
        CAS(Indices.All_Hill_Indexes) = 0;
    elseif constraint(1) == 11     % (Wedge Hill Refractive Boundary)
        IR1=Indices.Hill_Row_Borderline(:)-1;
        IR2=Indices.Hill_Row_Borderline(:)-2;
        IC=Indices.Hill_Col_Borderline;
        IZ=Indices.Hill_Z_Borderline;
        RR1=sub2ind(size(CAS), IR1',IC,IZ);
        RR2=sub2ind(size(CAS), IR2',IC,IZ);
        CAS(Indices.All_Hill_Indexes) = 0;
        CAS(Indices.HillBorderIndexes)= constraint(2).*CAS(RR2)+ ...
                                        (1-constraint(2)).*CAS(RR1);
    elseif constraint(1) == 10    % (Wedge Hill Reflective Boundary)
        IR=Indices.Hill_Row_Borderline(:)-2;
        IC=Indices.Hill_Col_Borderline;
        IZ=Indices.Hill_Z_Borderline;
        RR=sub2ind(size(CAS), IR',IC,IZ);
        CAS(Indices.All_Hill_Indexes) = 0;
        CAS(Indices.HillBorderIndexes)=CAS(RR);
    end
%%
elseif strcmp(side, 'top')
    % The top of the 2D/3D CA space
    if constraint(1)==1       % (Fixed Conditions)
        CAS(1,:,:)=0;
    elseif constraint(1)==2   % (Reflective Conditions)
        CAS(1,:,:)=CAS(3,:,:);
    elseif constraint(1) == 3   % (Free/Constant Flux Conditions)
        CAS(1,:,:) = 2.*CAS(2,:,:) - CAS(3,:,:);
    elseif constraint(1)==4   % (Non-Reflective Conditions)
        CAS(1,:,:)= CAS(2,:,:);
    elseif constraint(1)==5   % (Refractive Conditions ...Flat Bottom)
        CAS(1,:,:) = constraint(2).*CAS(3,:,:) +...
                             (1-constraint(2)).*CAS(2,:,:);
    end
%%
elseif strcmp(side, 'front')
    % The front of the 3D CA space
    if L3>1
        if constraint(1) == 1       %(Fixed Boundary Conditions)
            CAS(:,:,1) = 0;
        elseif constraint(1) == 2   %(Reflective Conditions)
            CAS(:,:,1) = CAS(:,:,3);
        elseif constraint(1) == 3   %(Free/Constant Flux Conditions)
            CAS(:,:,1) = 2.*CAS(:,:,2) - CAS(:,:,3);
        elseif constraint(1) == 4   %(Non-Reflective Conditions)
            CAS(:,:,1) = CAS(:,:,2);
        elseif constraint(1) == 5   %(Refractive Conditions ...Flat Bottom)
            CAS(:,:,1) = constraint(2).* CAS(:,:,3) + ...
                             (1-constraint(2)).*CAS(:,:,2);
        end
    end
%%
elseif strcmp(side, 'back')
```

```
    % The back of the 3D CA space
    if L3 >1
        if constraint(1) == 1        %(Fixed Boundary Conditions)
            CAS(:,:,L3) = 0;
        elseif constraint(1) == 2    %(Reflective Conditions)
            CAS(:,:,L3) = CAS(:,:,L3-2);
        elseif constraint(1) == 3    %(Free/Constant Flux Conditions)
            CAS(:,:,L3) = 2.*CAS(:,:,L3-1) - CAS(:,:,L3-2);
        elseif constraint(1) == 4     %(Non-Reflective Conditions)
            CAS(:,:,L3) = CAS(:,:,L3-1);
        elseif constraint(1) == 5   %(Refractive Conditions ...Flat Bottom)
            CAS(:,:,L3) = constaint(2).* CAS(:,:,L3-2) + ...
                                        (1-constraint(2)).*CAS(:,:,L3-1);
        end
    end
end

end
```

a. Wedge Shape Domain Configuration

```
function [Indices] = WedgeHillIndices(CAS,Indices)
 % Returns the indices of a wedge shape bottom boundary

 %Create the wedge shape bottom boundary
    [L1, L2,L3]=size(CAS);
    L2RDownHill=tril(ones(L1,L2),-10);
    Remove=floor(L1/2);
    L2RDownHill(1:Remove,:)=0;

    L2RDownHill = repmat(L2RDownHill,[1, 1,L3]);
 % Save the indices along the boundary
  Indices.All_Hill_Indexes = find(L2RDownHill==1);
    index=1;
    for k=1:L3
        for j=1:L2
            K = find(L2RDownHill(:,j,k)==1);
            if ~ isempty(K)
                Indices.Hill_Row_Borderline(index)=K(1);
                Indices.Hill_Col_Borderline(index)=j;
                Indices.Hill_Z_Borderline(index)=k;
                index=index+1;
            end
        end
    end

Indices.HillBorderIndexes=sub2ind(size(L2RDownHill),...
        Indices.Hill_Row_Borderline, Indices.Hill_Col_Borderline,...
                                                Indices.Hill_Z_Borderline);
end
```

b. Bottom Circle Floor Domain

```
function [CAS, Indices] = BottomCurvedHill(CAS, Indices)
% Returns the indices and grid data of a circle shape bottom boundary

 %Create the circle shape bottom boundary
```

```
[L1 L2 L3]=size(CAS);

Space=ceil(L1/2);
UpHill=tril(ones(Space,Space),-1);
UnderHill=triu(2*ones(Space,Space),1);
LeftHill= UnderHill+UpHill;
RightHill=fliplr(LeftHill);

Hill=cat(2,LeftHill,RightHill);
Hill = flipud(Hill);

% ~40 percent of the plane space is taken up by the hill
[H1 H2 H3] = size(Hill);
HillTop=ceil(.5*H1);
Hill(end,:)=[];
% Creating the Curve Hill in 2D and 3D
%   As of now the ZERO's are the borderline of the hill
[C1 C2 C3] = size(Hill);
if L2 ~= C2
    Hill(:,end)=[];
    C2 =L2;
end

 Add2 = L1-C1;
 Add3 = ones(Add2,L2,C3);
 Add4 = [Add3; Hill];

 Hill = repmat(Add4,[1,1,L3]);
 % Gathering the indices of the Hill
HillIndex =ind2sub(size(Hill), find(Hill==0));      % Hill Border line
UnderHill = ind2sub(size(Hill), find(Hill==2));     % Under the Hill

CAS(UnderHill)=0;       % Enforcing the Hill Boundary in the CAS
CAS(HillIndex)=0;       % Enforcing the Hill Boundary in the CAS

Indices.UnderHill=UnderHill;
Indices.HillIndex=HillIndex;

end
```

6. Update the Global Grid Points

```
function[CAS]= UpdateRefineCASMesh(cas,CAS)
% Updating the global grid after the local grid has performed its permutation
% CAS correlates to data from the global grid
% cas correlates to data from the local grid

global Refine

R=Refine;
[L1, L2, L3]=size(cas);
% 2D Case
% Top Left Corner
  CAS(R(1),R(2),R(3)) = cas(1,1,1);
% Lower Left Corner
```

```
  CAS(R(1)+1,R(2),R(3)) = cas(end,end,1);
% Top Right Corner
  CAS(R(1),R(2)+1,R(3)) = cas(1,end,1);
% Lower Right Corner
  CAS(R(1)+1,R(2)+1,R(3)) = cas(end,end,1);

if ~(L1==1 || L2==1 || L3==1)
   % 3D
   % Top Left 3rd Dimension
   CAS(R(1),R(2),R(3)+1) =  cas(1,1,end);
   % Top Right 3rd Dimension
   CAS(R(1),R(2)+1,R(3)+1) = cas(1,end,end);
   % Lower Left Corner 3rd Dimension
   CAS(R(1)+1,R(2),R(3)+1) = cas(end,1,end);
   % Lower Right Corner 3rd Dimension
   CAS(R(1)+1,R(2)+1,R(3)+1) = cas(end,end,end);
end

end
```

H. PLOTTING UNIFORM, GLOBAL, AND LOCAL GRID POINTS ITERATION DATA

```
function[]=CA_DomainPointPlots(Plots,DomainData,SubDomainPoints,...
           SubDomainData,time)

% This function will plot the associated uniform or multi-grid data points as a
%  function of time.
% The 'Plots' matrix/array contains the associated points inside global or
% uniform grid where the plots are to be viewed
% The 'SubDomainPoints' matrix/array contains the associated points inside
% local grid where the plots are to be viewed
%%
global Refine ANS
disp('Beginning the function to plot each specified location point')
S=Plots;
%% Uniform/Global Grid Plots
NumOfPlots=size(Plots);                Iterations=size(DomainData);
for j = 1:NumOfPlots(1)
    for i = 1:Iterations(end)
        eval(['GRAPH',num2str(j),'(i)= DomainData(S(',num2str(j),',1),',...
            'S(',num2str(j),',2),S(',num2str(j),',3),',num2str(i),');'])
    end
        figure()
        eval(['plot(time,GRAPH',num2str(j),')'])
        title(['Location:  Row = ',num2str(Plots(j,1)),', Column = ',...
            num2str(Plots(j,2)),', Z =
',num2str(Plots(j,3))],'FontWeight','bold')
        set(gca, 'FontWeight','bold')
        xlabel('time (secs) ','FontWeight','bold'),
        ylabel('Nodal Pressure Values [Pa]','FontWeight','bold')
        grid on
end

%% Perform Local Grid if Asked
    if ANS==2, return, end
%% Local Grid Domain Plots
```

```
S=SubDomainPoints;
NumOfPlots=size(SubDomainPoints);
for j = 1:NumOfPlots(1)
    for i = 1:length(SubDomainData)
        eval(['GRAPH',num2str(j),'(i)= SubDomainData(S(',num2str(j),',1),',...
            'S(',num2str(j),',2),S(',num2str(j),',3),',num2str(i),');'])
    end
%% Determine the local grid decimal location in reference to the global grid
    [MeshL(1), MeshL(2), MeshL(3),MeshL(4)]= size(SubDomainData);
    if  S(j,1)==1
        r = 0;
    elseif ((MeshL(1)== S(j,1))  || (MeshL(1)== S(j,1)-1));
        r=1;
    else
        r = S(j,1)/MeshL(1);
    end
  %%%
    if S(j,2)==1
        c =0;
    elseif  ((MeshL(2)== S(j,2))  || (MeshL(2)== S(j,2)-1))
        c=1;
    else
        c =  S(j,2)/MeshL(2);
    end
  %%%
    if S(j,3)==1
        z = 0;
    elseif  (MeshL(3) == S(j,3)) && (S(j,3)~=1)
        z=1;
    else
        z = S(j,3)/MeshL(3);
    end
  %%% Plotting
    figure('Name',['The decimal point in Row, Col, & Z is in reference to',...
        ' the larger domain'])
    eval(['plot(time,GRAPH',num2str(j),')'])
    L=Refine;
    title(['Local Grid(or Mesh) Point:   ROW = ',num2str(L(1)+r),...
        ', COLUMN = ', num2str(L(2)+c),', Z = ',num2str(L(3)+z)])
        set(gca,'FontWeight','bold')
    xlabel('time (secs) ','FontWeight','bold'),
    ylabel('Nodal Pressure Value [Pa]','FontWeight','bold')
end

end
```

I. 3D VOLUME PLOT AND ANIMATION

```
function [] = CA_3D_and_VolumePlot (DomainData,dt,type,ZS)
% The function will provide an animation view of 2D and 3D grid point value
% changes as a function of time

%% Plotting of Selected Points
[L1, L2, L3, L4]=size(DomainData);

ZMAX=0; ZMIN=0;
```

```
for j=1:length(DomainData)
     zmax=max(max(max(DomainData(:,:,:,j))));
     zmin=min(min(min(DomainData(:,:,:,j))));
     if zmax>ZMAX
        ZMAX=zmax;
     end
     if zmin<ZMIN
        ZMIN=zmin;
     end
end
%% Animation of how the Values in the Plane Changes vs. Iterations
%% 2D Case
if (L3==1 || L2==1 || L1==1)
    figure('Name','2D Space')
    Handle=surf(DomainData(:,:,ceil(L3/2),1),'ZDataSource','ZZ');
    axis([1 L2, 1 L1, ZMIN, ZMAX]);
    for i=1:L4
        ZZ=DomainData(:,:,ceil(L3/2),i);
        refreshdata(Handle,'caller')
        title([type,' @ Time =',num2str(i*dt),' seconds'])
        xlabel('Columns','FontWeight','bold'), ylabel('Rows','FontWeight',...
            'bold')
        zlabel('Nodal Pressure [Pa]','FontWeight','bold'),
        drawnow, pause(.01),  set(gca,'YDir','reverse')

    end
%% 3D Case
else
    h=figure('Name',[' Time = ', num2str(0),' seconds']);
    h1=gca;
    for i=1:L4
        % 3D Perspective in Viewing Every 8th Plane
        [xi,yi,zi] = meshgrid(1:L1, 1:L3, 1:L2);
        vi = interp3(DomainData(:,:,:,i),xi,yi,zi);
        RT=slice(xi,yi,zi,vi,linspace(1,L1,7),ceil(L2/2)+ZS,...
             linspace(1,L3,5));
        shading interp
        axis tight, view(-38,66)
        rotate(RT,[1 0 0],90)
        set(gca,'ZDir','reverse','YDir','reverse')
        xlabel(h1,'Columns','FontWeight','bold'),
        zlabel(h1,'Rows','FontWeight','bold')
        ylabel(h1,'Z-Nodes','FontWeight','bold')
        set(h,'Name',[' Time = ', num2str(i*dt),' seconds']);
        colorbar('peer',h1,'location','NorthOutside');
        caxis(h1,[ZMIN, ZMAX]);
        drawnow, pause(.01),

    end
    %%
end

end
```

J. RMS TRANSMISSION LOSSES

```
function [] = TranssmissionLoss_dB(DomainData,Value,type)
% This function will provide a contour view of changes in propagation losses.
% The root mean square pressure at all nodal points is used in reference to the
% acoustic source peak pressure
%%
disp('Beginning function to calculate transmission losses across the domain')
[L1, L2, L3] = size(DomainData(:,:,:,1));
                          %%  Acoustic Intensity
                    %% 2D Contour View of Transmission Losses
INDEX=1;
RMS_Values = zeros(L1,L2,L3);
disp('....Calculating RMS Values')
for k=1:L3
    for j=1:L2
        for i=1:L1
            PressValues=DomainData(i,j,k,:);
            PP=reshape(PressValues,1,numel(PressValues));
            P_rms=rms(PP);
            RMS_Values(INDEX)=P_rms;
            INDEX=INDEX+1;
        end
    end
end

Loss_dB=20*log(rms(Value)./RMS_Values);
% Removing the Nodes Internal to the Source Diameter and setting
% equal to NaN because those values will be infinite once I take
% the log of them
Delete= Loss_dB==Inf;                   Loss_dB(Delete)=NaN;
%%
if (L3==1 || L2==1 || L1==1)
    figure()
    contourf(Loss_dB),              contourcbar
    title('\Delta Propagation Loss (dB)','FontWeight','bold')
    xlabel('Columns','FontWeight','bold'),
    ylabel('Rows','FontWeight','bold')
    set(gca,'YDir','reverse')
                    %%  1D range view  of Transmission Losses
R = input(['Please input the starting location for the receiver to det',...
       'ermine\n Propagation Losses (dB) vs. Range plot\n',...
      ' Please enter in the following format [Row, Col, Z]\n-->']);
   figure()
   plot(R(2):L2, Loss_dB(R(1),R(2):end))
   xlabel('Columns','FontWeight','Bold'),
   ylabel('\DeltaPropagation Losses (dB)','FontWeight','Bold')
   title(type)
   title([' Row = ', num2str(R(1))], 'FontWeight','Bold')
%%
else
                %%  3D Volume Plot of Transmission Losses
[xi,yi,zi] = meshgrid(1:L1, 1:L3, 1:L2);
vi = interp3(Loss_dB,xi,yi,zi);
RT= slice(vi,1:L1,1:L3,1:L2);  shading interp
title('\Delta Propagation Losses (dB)','FontWeight','Bold')
zlabel('Rows','FontWeight','Bold')
ylabel('Z-Nodes','FontWeight','Bold')
xlabel('Columns','FontWeight','Bold')
view(-25,14), axis tight
```

```
set(gca,'ZDir','reverse','YDir','reverse')
colorbar('location','northoutside');
rotate(RT,[1 0 0],90)
                    %%  Select a Plane to View
 R = input(['Please input the X-Plane for viewing',...
       '\n Propagation Losses (dB) vs. Range plot\n',...
       ' Please enter a number between 1 and ',num2str(L2),' :\n-->']);
   figure()
   Loss_dB2=reshape(Loss_dB(:,R,:),L1,L3);
   contourf(fliplr(Loss_dB2));
   contourcbar
   ylabel('Rows','FontWeight','Bold'),
   xlabel('Z-Nodes','FontWeight','Bold'), set(gca,'YDir','reverse')
   title(['Column = ', num2str(R),' \Delta Propagation Losses'],...
       'FontWeight','Bold')
   %%
   R1=input(['Select which row of Z-Nodes from Plane ', num2str(R),...
        '\n to view in a 1D plot. Before selecting a number between',...
        '\n 1 and ',num2str(L3),' please look at the prior Figure.\n-->']);
    figure()
    Loss_dBB2=fliplr(Loss_dB2);
    plot(Loss_dBB2(R1,:))
    xlabel('Z-Nodes','FontWeight','Bold')
    ylabel('\Delta Propagation Losses (dB)','FontWeight','Bold')
    title(['Column = ', num2str(R), '  &  Row = ', num2str(R1)],...
        'FontWeight','Bold')
end
%%
end
```

K. PEAK TRANSMISSION LOSSES

```
function [] = TranssmissionLoss(DomainData, iterations,type)
% This function will provide a contour view of changes in propagation losses.
% The max pressure at all nodal points is used in reference to the
% acoustic source peak pressure
%%
global Peak
[L1, L2, L3, L4] = size(DomainData(:,:,:,1));
                            %%  Acoustic Intenstiy
                        %% 2D Contour View of Transmission Losses
    MaxPress = MaxPressure(DomainData,iterations);
    % Reference Intensity along the Nodal Diameter
    Ref_Intensity = Peak;
%%
if (L3==1 || L2==1 || L1==1)
    Loss_dB=20*log(Ref_Intensity./MaxPress);
    % Removing the Nodes Internal to the Source Diameter and setting
    % equal to NaN because those values will be infinite once I take
    % the log of them
    Delete= Loss_dB==Inf;
    Loss_dB(Delete)=NaN;

    figure()
    contourf(flipud(Loss_dB))
    set(gca,'YDir','reverse')
    contourcbar
    title('\DeltaPropagation Losses','FontWeight','Bold')
```

```
    xlabel('Columns','FontWeight','Bold')
    ylabel('Rows','FontWeight','Bold')
                    %%  1D range view  of Transmission Losses
    R = input(['Please input the starting location for the receiver to det',...
      'ermine\n Transmission Losses (dB) vs. Range plot\n',...
      ' Please enter in the following format [Row, Col, Z]\n-->']);
    Intensity = MaxPress(R(1),R(2):end,R(3));
    %Intensity = MaxPress(10,1:end,1);
    Loss_dB=10*log(Ref_Intensity./Intensity);
    figure()
    plot(Loss_dB)
    xlabel('Nodal Distance')
    ylabel('Transmission Loss (dB)')
    title([type,'Row 1 of the 2D Contour Plot'])

else
    Loss_dB=10*log((Ref_Intensity.^2)./(MaxPress).^2);
    % Removing the Nodes Internal to the Source Diameter and setting
    % equal to NaN because those values will be infinite once I take
    % the log of them
    Delete= Loss_dB==Inf;
    Loss_dB(Delete)=NaN;
    %%              3D Volume Plot of Transmission Losses
    planes=[L1,1,ceil(L2*.125),ceil(L1*.25),ceil(L1*.375), ceil(L1*.5),...
                               ceil(L1*.625), ceil(L1*.75), ceil(L1*.875)];
    R = input(['Please input the Z-Plane for viewing',...
       '\n Transmission Losses (dB) vs. Range plot\n',...
       ' Please enter a number between 1 and ',num2str(L3),' :\n-->']);
   %%
    left = 120;     bottom = 50;     height = 700;    width=(4/3)*height;
    rect = [left, bottom, width, height];
    figure('OuterPosition',rect);
    slice(Loss_dB,[ceil(L2/2),1],planes,1)
    title([type,':  Transmission Losses (dB)'])
    xlabel({'Columns'; 'This Side is the Bottom Face'})
    ylabel({'Z-Nodes'; 'This Side is the Right Face'})
    zlabel({'Rows'; 'This Side is the Top Face'})
    axis([1 L2 1 L1 1 L3])
    view(54,80)
    colorbar('location','northoutside');

     %%  2D Contour view  of Transmission Losses from a selected Plane
    Intensity = MaxPress(:,:,R);
    Intensity=reshape(Intensity,[L1,L2,1]);
    Loss_dB=10*log((Ref_Intensity.^2)./(Intensity).^2);
    figure()
        Delete= Loss_dB==Inf;
        Loss_dB(Delete)=NaN;
    contourf(flipud(Loss_dB))
    contourcbar
    xlabel('Horizontal Nodes')
    ylabel('Vertical Nodes (dB)')
    title([type,':   Z = ', num2str(R),' Plane Transmission Losses'])
             %% 1D View Along the Selected Plane
    R1=input(['Select which row of Vertical Nodes from Plane ', num2str(R),...
        '\n to view in a 1D plot. Before selecting a number between',...
        '\n 1 and ',num2str(L1),' please look at Figure 1 and 2.\n-->']);
    figure()
    plot(Loss_dB(R1,:))
    xlabel('Horizontal Nodes')
```

```
    ylabel('Transmission Losses (dB)')
    title([type,':  Z-Plane = ', num2str(R), '  &  Row = ', num2str(R1)])
end
%%
end
```

L. MAX PRESSURE VALUE

```
function [MaxPress] = MaxPressure(DomainData,iterations)
% This function will find the max pressure seen at every node from all
% iterations performed on the domain.
%%
[L1,L2,L3,L4]=size(DomainData(:,:,:,1));
MaxValue = reshape(DomainData(:,:,:,1),[L1*L2*L3,1,]);
  for j=2:iterations
     Max = reshape(DomainData(:,:,:,j),[L1*L2*L3,1]);
     MaxValue = cat(3,MaxValue,Max);
  end

  for t=1:L1*L2*L3
  % Finding the max node pressure from all iterations
     MV(t)=max(MaxValue(t,1,:));
  end

  MaxPress=reshape(MV,[L1,L2,L3]);

%%
end
```

LIST OF REFERENCES

[1] "National Oceanic and Atmopheric Administration," United States Department of Commerce, 8 Nov 2012. [Online]. Available: http://oceanexplorer.noaa.gov/technology/tools/sonar/sonar.html. [Accessed 18 Nov 2012].

[2] Z. Jiang, "Underwater acoustic networks – issues and solutions," *International Journal of Intelligent Control and Systems,* vol. 13, no. 3, pp. 152–161, 2008.

[3] A. Quazi and W. Konrad, "Underwater acoustic communications," *IEEE Communication Magazine,* pp. 24–29, March 1982.

[4] J. Partan, J. Kurose and B. N. Levine, "A survey of practical issues in underwater networks, international conference on mobile computing and networking," *Proc. of the 1st ACM international workshop on Underwater networks,*, Los Angeles, 2006.

[5] "Introduction to naval weapons engineering," [Online]. Available: http://www.fas.org/man/dod-101/navy/docs/es310/asw_sys/asw_sys.htm. [Accessed 28 Feb 2013].

[6] P. C. Etter, *Underwater Acoustic Modeling*, New York: Spon Press, 2003.

[7] M. J. Buckingham, "Ocean-acoustic propagation models," *J. Acoustique,* vol. 43, no. 30, pp. 223–287, 1992.

[8] B. A., "Von Neumann's self-reproducing automata," *In Essays on Cellular Automata,* pp. 3–64, 1970.

[9] A. Ilachinski, "Cellular automata," in *Cellular Automata A Discrete Universe*, New Jersey, World Scientific Publishing Co. Pte. Ltd, 2001, p. 5.

[10] L. Villar and A. Souza, "Cellular automata models for general traffic conditions on a line," *Physica A,* vol. 211, pp. 84–92, 1994.

[11] A. Schadschneider and M. Schreckenberg, "Cellular automaton models and traffic flow," *Journal of Physics,* vol. 26, pp. 649–683, 1993.

[12] T. Toffoli, "Cellulular automata as an alternative to (rather than an approximation of) differential equations in modeling physics," *Physica D,* vol. 10, pp. 117–127, 1984.

[13] S. Hosoglu, “Cellular automata: an approach to wave propagation and fracture mechanics problems,” M.S. thesis, MAE, NPS, Monterey, CA 2006.

[14] E. W. Weisstein, “Cellular Automaton,” MathWorld--A Wolfram Web Resource, [Online]. Available: http://mathworld.wolfram.com/CellularAutomaton.html . [Accessed 15 Oct 2012].

[15] Y. Kwon and S. Hosoglu, “Application of lattice boltzmann method, finite element method, and cellular automata and their coupling to wave propagation problems,” *Elsevier Computers and Structures,* vol. 86, pp. 663–670, 2008.

[16] L. E. Craugh, “Coupled finite element and cellular automata methods for analysis of composite structures in an acoustic domain,” Ph.D. dissertation, MAE, NPS, Monterey, CA, 2012.

[17] B. Chopard and M. Droz, *Cellular Automata Modeling of Physical Systems*, Cambridge: Cambridge University Press, 1998.

[18] H. Schmidt and F. B. Jensen, “Computational ocean acoustics: advances in 3D ocean acoustic modeling,” *American Institute of Physics*, Beijing, China , 2012.

[19] A. D. White, *SONAR for Practising Engineers*, England: John Wiley & Sons, 2002.

[20] F. Sturm and J. A. Fawcett, “On the use of higher-order azimuthal schemes in 3-D modeling,” *Acoustical Society of America,* vol. 113, no. 6, pp. 3134–3145, 2003.

[21] Y. W. Kwon and B. Hyochoong, *The Finite Element Method using MATLAB*, New York: CRC Press, 1997.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Distinguished Professor Young W. Kwon
   Naval Postgraduate School
   Monterey, California

4. Research Assistant Professor Jarema M. Didoszak
   Naval Postgraduate School
   Monterey, California

5. Jermaine A. Bailey
   Supervisor of Ships
   Groton, Connecticut