The Pennsylvania State University

The Graduate School

Graduate Program in Acoustics

SENSITIVITY OF A COMPUTATIONAL VERSION OF THE KIRCHHOFF INTEGRAL THEOREM TO SURFACE DISCRETIZATION

A Thesis in

Acoustics

by

John Anderson Mills III

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

August 1997

John Anderson Mills III

We approve the thesis of John Anderson Mills III,

Date of Signature

Donald E. Thompson
Senior Research Associate
Thesis Advisor

Victor W. Sparrow
Associate Professor of Acoustics

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

Jiri Tichy
Professor of Acoustics
Chair of the Graduate Program in Acoustics

ABSTRACT

The Kirchhoff Integral Theorem is a procedure used to calculate far-field quantities from known values on a surface surrounding an acoustic source distribution. Choosing far-field pressure as the calculated quantity allows generation of pressure fields from complex sources which have no analytical expression for far-field pressure. The discretization of the Kirchhoff surface causes error in the calculation, but this error can be controlled to produce results of the desired accuracy. Grid point spacing, grid layer separation, time sampling, and grid size and shape are elements of the discretization shown to affect the error in the calculations. Grid layer separation distance is the strictest requirement which must be fulfilled to achieve accurate results.

ACKNOWLEDGMENTS

Many people helped in the creation of this thesis. Paul Cannan, Jessica Moore, Tad Rollow, and Brian Tuttle deserve special thanks for their technical suggestions and editing expertise during the process of making the thesis a physical reality. I would like to thank Cat Bradley and Julia Davenport for their visionary and kinetic support during the time I spent on this research. Dan Chen's artistic input is also appreciated.

I would like to thank Dr. Don Thompson for acting as my advisor throughout the completion of this work. The other members of my committee, Dr. Victor Sparrow and Dr. Philip Morris, also recognized for their time and effort in making this thesis an accurate and useful document.

My parents, Mr.and Mrs. John and Marie Mills, showed me support during this research with their unending patience and encouragement. I would like to also pay respect to my grandparents, Mr. and Mrs. Anderson and Jacqueline Mills, for the love and support they provided during the portion of their lives that I was fortunate enough to share. I will miss them dearly.

Discipline is the key.

Chapter 1.

INTRODUCTION

The Kirchhoff Integral Theorem is a mathematical formula for calculating a far-field quantity from an arbitrary wave source. The theorem can provide far-field acoustic pressure if pressure values are known on a surface surrounding the wave source. Because this "Kirchhoff" surface can be translated into the computational domain, numerical techniques fueled by modern computing resources have renewed interest in this century old theorem. This thesis is primarily concerned with acoustic pressure calculations, and therefore the Kirchhoff Integral Theorem is used for its ability to calculate pressure deviation about the ambient pressure. In this thesis, "pressure" always refers to the acoustic pressure.

The purpose of this thesis is to examine how sensitive the far-field calculations of a computational version of the Kirchhoff Integral Theorem are to variations in the discretization of the Kirchhoff surface. Parameters of the Kirchhoff surface include size, shape, point spacing, layer separation distance. Time sampling is also studied and is included as a parameter of discretization though it is not specifically a parameter of the Kirchhoff surface.

One of the significant uses of a computational version of Kirchhoff's formula is to determine far-field pressure for sources which have no closed-form expressions for pressure. An example of a complicated source with no analytical expression for far-field pressure is a turbomachine. If computational fluid dynamics simulations can provide pressure values on a surface surrounding a turbomachine, the Kirchhoff method can calculate far-field acoustic pressure from those pressure values. This example provides the motivation for the computational Kirchhoff formulations in this thesis.

The discretization of the Kirchhoff surface into a computational grid does introduce error, however. Effectively, the computational grid samples pressure waves which change in both time and space. This thesis suggests how the construction of computational grids is related to the error in the calculations of the Kirchhoff method.

1.1. Background

The following is a brief set of publications discussing the Kirchhoff Integral Theorem. Gustav Robert Kirchhoff (1824–1887) first presented his integral theorem in a paper entitled "Toward a Theory of Light Rays"[4] in 1882. A frequently cited modern derivation which uses the method in theory about the diffraction of light rays and other electromagnetic problems is presented by Stratton.[13] According to Pierce,[11] Kirchhoff's theorem also facilitates the discussion of sound radiation, which is also shown by the many references in "Review: The Use of Kirchhoff's Method in Computational Acoustics" presented by Lyrintzis.[6] This thesis corroborates accuracy studies of a computational implementation of the Kirchhoff Integral Theorem conducted by Meadows and Atkins[7]. Although their work explores some of the same discretization effects, both their study and this thesis include unique information.

1.2. The Kirchhoff Integral Theorem

In discussing the sensitivity of a computational version of the Kirchhoff Integral Theorem the starting point is the theorem itself. It states that for a nondeformable, closed surface, S, surrounding an arbitrary acoustic source distribution and enclosing any nonlinear effects, a quantity, Φ, satisfying the linear wave equation,

$$\frac{1}{c^2}\frac{\partial^2 \Phi}{\partial t^2} - \nabla^2 \Phi = 0\,, \tag{1.1}$$

in the exterior of S can be known at a far-field observer point, $\mathbf{x} = (x, y, z)$, if the values for Φ, $\frac{\partial \Phi}{\partial t}$ (the partial derivative with respect to time), and $\frac{\partial \Phi}{\partial \vec{n}}$ (the partial derivative with respect to the normal vector $\vec{n}$) are known on the surface. The speed of sound in the medium is represented by c. The far-field is defined as far enough away from the source so that $kR \gg 1$, where k is the wavenumber and is equal to $\frac{2\pi}{\lambda}$, λ is the wavelength of the oscillation of the source, and R is the distance from the source. The wavenumber k is also defined as $\frac{\omega}{c}$, where ω is the angular frequency.

Because this thesis deals with acoustic calculations, pressure, p, is chosen as the quantity which satisfies the linear wave equation. Because the formulation uses the linear wave equation, nonlinear pressures are handled as if they are linear. This can cause significant error in calculated far-field pressure values when the Kirchhoff surface is not far enough away from the source to completely enclose nonlinear effects.

From Morino's[10] formulation and Pierce's[11] suggested final form, Lyrintzis[6] develops the classical Kirchhoff formula using the free space Green's function[11] as

$$p(\mathbf{x},t) = \frac{1}{4\pi}\int_S \left[\frac{p}{r^2}\frac{\partial r}{\partial \vec{n}} - \frac{1}{r}\frac{\partial p}{\partial \vec{n}} + \frac{1}{cr}\frac{\partial r}{\partial \vec{n}}\frac{\partial p}{\partial \tau}\right]_\tau dS\,, \tag{1.2}$$

where r is the distance between the observer location, $\mathbf{x}$, and the source location on the surface, $\mathbf{y} = (x', y', z')$, and the normal vector, $\vec{n}$, points out of the surface at $\mathbf{y}$. For easy reference, the terms of the integrand are referred to as the "Kirchhoff terms." Figure 1.1 shows the geometrical relationship of some of the quantities in the Kirchhoff formula.

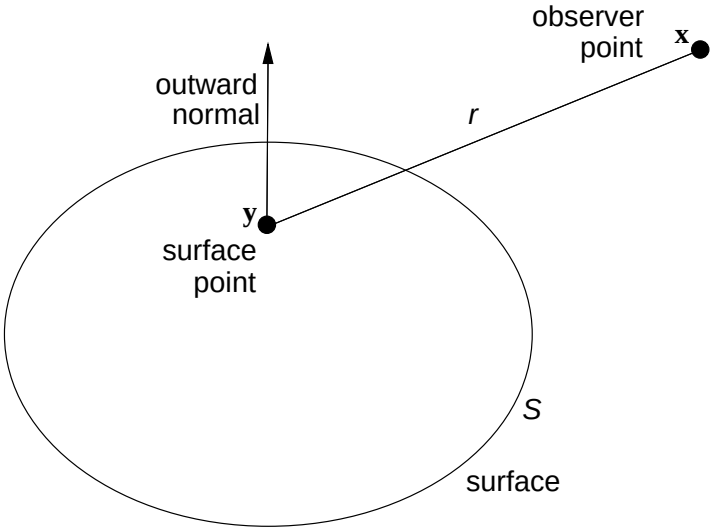


Figure 1.1: Illustration of the variables appearing in the Kirchhoff formula.

Because this formulation is based in the time domain, the phase of the pressure values at $\mathbf{y}$ represented at $\mathbf{x}$ must be corrected for the propagation time from $\mathbf{y}$ to $\mathbf{x}$. This is done by evaluating all values inside $[\,]_\tau$ at the retarded (emission) time, τ. Specifically, τ is given as

$$\tau = t - \frac{r}{c}\,. \tag{1.3}$$

The first term in the integrand of equation (1.2), $\frac{p}{r^2}\frac{\partial r}{\partial \vec{n}}$, has a $\frac{1}{r^2}$ dependence and for far-field calculations will not be significant. It remains in the computational formulations, however, because it does become significant as the $\mathbf{x}$ approaches the Kirchhoff surface.

1.3. Derivation

The following is Pierce's[11] derivation of the Kirchhoff-Helmholtz Integral Theorem with a few changes in notation to fit the formulation of the Kirchhoff Integral Theorem given in the last section. The Helmholtz Integral Theorem is derived in the frequency domain and then moved back into the time domain to form the Kirchhoff Integral Theorem.

Starting with the vector identity

$$G\left(\nabla^2 + k^2\right)\hat{p} - \hat{p}\left(\nabla^2 + k^2\right)G = \nabla \cdot \left(G\nabla\hat{p} - \hat{p}\nabla G\right) , \tag{1.4}$$

where G is any function of position, and integrating both sides over a volume, V, consisting of all points outside S that are within some large sphere of radius R centered at the origin, the identity becomes

$$-\int\int\int \hat{p}\left(\nabla^2 + k^2\right)G\, dV = -\int\int_S \left(G\nabla\hat{p} - \hat{p}\nabla G\right)\cdot\vec{n}\, dS + I_R\,, \tag{1.5}$$

where

$$I_R = R^2\int_0^{2\pi}\int_0^{\pi}\left(G\frac{\partial\hat{p}}{\partial R} - \hat{p}\frac{\partial G}{\partial R}\right)\sin\theta\, d\theta\, d\phi\,. \tag{1.6}$$

This results occurs because the contribution from the first term of equation (1.4) is zero because $(\nabla^2 + k^2)\hat{p} = 0$ within V and Gauss' theorem transforms the volume integration over the right side into a surface integration. I_R is the surface integration over the outer sphere.

Stipulating that G is a Green's function which satisfies the inhomogeneous Helmholtz equation,

$$\left(\nabla^2 + k^2\right)G_k\left(\mathbf{x}|\mathbf{y}\right) = -4\pi\delta\left(\mathbf{x} - \mathbf{y}\right) , \tag{1.7}$$

he left side of equation (1.5) becomes $4\pi\hat{p}(\mathbf{x})$ because of the integral property of the Dirac delta function. If the Green's function goes to zero at least as fast as $\frac{1}{R}$, I_R vanishes in the limit of large R. I_R is identically zero for any sphere containing the surface and the point $\mathbf{x_0}$ because the remaining terms in equation (1.5) are independent of the choice for R. For

$\mathbf{x_0}$ exterior to S, equation (1.5) reduces to

$$\hat{p}(\mathbf{x}) = -\frac{1}{4\pi}\int\int (G\nabla\hat{p} - \hat{p}\nabla G)\cdot\vec{n}\, dS \tag{1.8}$$

where the integration extends only over the surface.

Choosing the free space Green's function, $\frac{e^{ikR}}{R}$, using $\nabla\hat{p}\cdot\vec{n} = i\omega\rho\hat{v}_n$, and the following relationship,

$$\nabla G = \frac{\mathbf{y}-\mathbf{x}}{R^3}(ikR-1)\, e^{ikR}, \tag{1.9}$$

the transient version of equation (1.8) can be found with a few variable changes. Prescribing that $i\omega \to -\frac{\partial}{\partial t}$, $R \to r$, and a factor e^{ikR} multiplying $e^{-i\omega t}$ is equivalent to shifting $t \to \tau$, the following equation is similar to equation (1.2),

$$p(\mathbf{x},t) = \frac{\rho}{4\pi}\int\int \frac{1}{r}\frac{\partial v_n(\mathbf{y},\tau)}{\partial t}dS + \frac{1}{4\pi c}\int\int \vec{e}_r\cdot\vec{n}\left(\frac{\partial}{\partial t}+\frac{c}{r}\right)\frac{p(\mathbf{y},\tau)}{r}dS. \tag{1.10}$$

Recognizing that $\vec{e}_r\cdot\vec{n} = \frac{\partial r}{\partial n}$ and that $-\rho\frac{\partial v_n}{\partial t} = \nabla\hat{p}\cdot\vec{n} = \frac{\partial p}{\partial\vec{n}}$, equation (1.2) is found.

1.4. Thesis Overview

Chapter 2 describes the two computational formulations of the Kirchhoff Integral Theorem used in this thesis. One formulation mixes analytical and numerical techniques, while the other calculates all of the Kirchhoff terms numerically. Some limitations on the shape of the Kirchhoff surface are imposed by the formulations, and these limitations are also discussed.

Chapter 3 discusses validation of the formulations with simple, classical cases (dipoles and quadrupoles).[11] The ability of the formulations to calculate accurate far-field pressure and to separate sources operating at different frequencies is shown.

Chapter 4 discusses the discretization used in a computational version of the Kirchhoff Integral Theorem. The grid point spacing, grid layer separation distance, time sampling, and grid size and shape all affect the error in the far-field calculations. This error is explored by varying these parameters.

Chapter 5 discusses the findings of this thesis and how they might be used. In addition, suggestions for further research are given.

Chapter 2.

COMPUTATIONAL FORMULATION

In order to use the Kirchhoff Integral Theorem computationally, all of the terms of the theorem must be shifted from the mathematical to the computational domain. Specifying that the computational formulation is three dimensional, a computational grid represents the Kirchhoff surface. The values of p, $\frac{\partial p}{\partial t}$, and $\frac{\partial p}{\partial \vec{n}}$, normally known on the Kirchhoff surface, must be determined analytically from an acoustic source equation or numerically from p values on the grid. Another term, $\frac{\partial r}{\partial \vec{n}}$, must also be ascertained by either analytical or numerical techniques. Numerical schemes handle the integration in the Kirchhoff formula.

Three methods are used to calculate the far-field pressure from the validation sources. The first method uses analytical expressions for far-field pressure and is only used for error calculation by providing an analytical value for comparison to numerical methods. Two computational approaches are used to calculate far-field pressure using the Kirchhoff method: one which mixes analytical and numerical techniques and another which is fully numerical. The analytical-numerical formulation uses analytical expressions for the terms of the Kirchhoff formula and numerical integration techniques to arrive at far-field pressures. The analytical-numerical formulation is limited to a single source and single grid shape and is therefore used only for accurate validation of the fully numerical formulation and to study the effects of changing radius for a spherical grid. The fully numerical formulation calculates the equation terms and integrates the Kirchhoff equation numerically. The fully numerical formulation is the method which would be used to calculate far-field pressures from acoustic source distributions with no known far-field pressure expressions.

In both formulations, the far-field pressure, p, is found for a single time, t. This pressure can be found for many instants in a period, and by calculating far-field pressure at time samples that are closely spaced, the Kirchhoff method can approximate a continuous pressure signal. From this pressure signal, a discrete time Fourier algorithm calculates the magnitude of a Fourier component for a single frequency at a single point in the far-field. The number of samples in a period necessary for accurate calculations is discussed in section 4.1.4. By calculating this Fourier component at closely spaced far-field points, the method can determine far-field pressure directivities based on single frequencies. Directivity for this

thesis is a polar plot of the Fourier component magnitude of pressure at a single frequency versus the angle at which the pressure oscillation is observed. The angles for the directivity plots are 1° apart, providing enough points in the directivity plane to show the detail of the plots.

A set of Fortran programs listed in the Appendix actually calculates the far-field acoustic pressures using the techniques described in this chapter. The numerical method described in section 2.3 follows the steps of grid generation, pressure definition, and then Kirchhoff calculation. The programs `Grid.f`, `Press.f`, and `Kirch3.f` provide these steps for the cases shown in chapters 3 and 4. To calculate the far-field pressures from a computational fluid dynamics simulation, the simulation program would have to provide a computational grid and the pressures on it. The analytical method described in section 2.2 follows the same steps, but they are all contained in one program, `K3A.f`, so it is possible to study the effects of varying the radius of a spherical Kirchhoff grid.

2.1. Computational Grid Formulation

A computational grid is a set of points that lie on the surface of a conceptual three-dimensional object and is often referred to by the shape that its points describe, e.g., a cylindrical, spherical, or conical grid. Several example computational grids are shown in figure 2.1. The grid represents the Kirchhoff surface for the computational formulations described in this chapter, and therefore determines the spatial locations where the values of the Kirchhoff terms are calculated.

The $\frac{\partial}{\partial \vec{n}}$ terms of the Kirchhoff equation require special treatment at sharp grid edges, because normal vectors are undefined along edges. Rounding the corners of the computational grids avoids edges and the problems associated with finding the normal vector along them.

For the formulations used in this thesis, computational grids are expected to be axisymmetric, because axisymmetry allows analytical and accurate calculation of the areas of small patches of the surface for surface integration. The calculated areas are equal circumferential divisions of a cone cut parallel to its base. Figure 2.2 shows this surface, called a frustum, which has two radii, one which may be zero, and a height, h, which is the distance between the plane of the cone's base and the plane of the cut. This area of one patch is

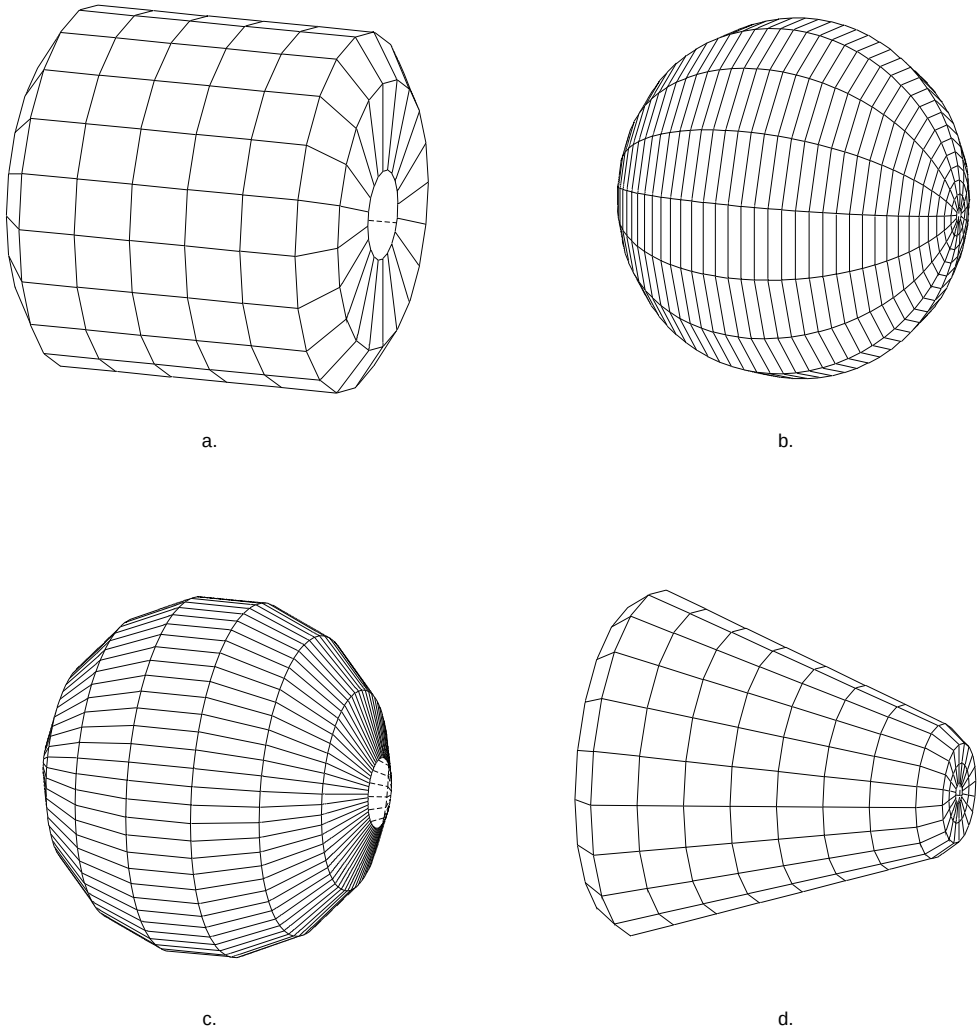


Figure 2.1: Several computational grids (Kirchhoff surfaces).

calculated as

$$dS = \frac{2\pi R_f h}{\texttt{ncirc}}, \tag{2.1}$$

where R_f is the average of the two radii, and `ncirc` is a program variable representing the number of circumferential points in the grid. This formula is derived from a cone mensuration formula given in the *CRC Standard Math Tables*.[1]

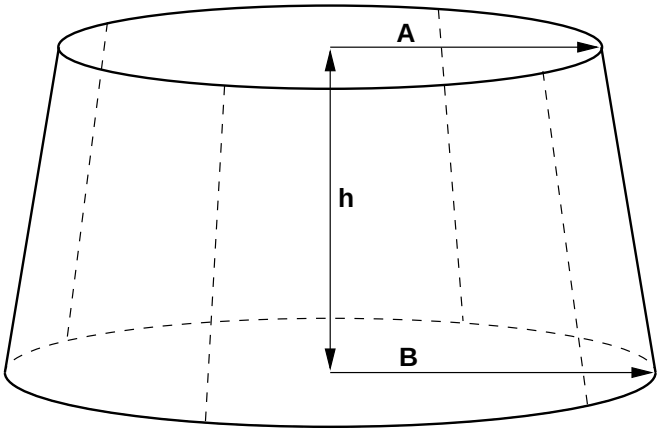


Figure 2.2: Area patches for surface integration ($R_f = \frac{A+B}{2}$, $\texttt{ncirc} = 4$).

Because the numerical $\frac{\partial}{\partial \vec{n}}$ term calculations depend on a second-order forward difference scheme which is discussed in section 2.3, the fully numerical formulation requires the computational grid to have exactly three layers, each with the same number of grid points. Corresponding points on different layers must form lines normal to the represented Kirchhoff surface. Computational fluid dynamics simulations generate enough geometry information that the requirement that corresponding points lie along normal lines is not difficult to achieve.

2.2. Analytical-Numerical Formulation

In order to solve the Kirchhoff Integral Theorem, the analytical-numerical formulation uses a mixture of analytical expressions for the terms of the Kirchhoff integrand and numerical integration techniques. This formulation starts with the equation for a lateral quadrupole with constituent monopole sources placed on the x and y axes which is given as

$$p = \frac{A}{R}\cos(2\theta)\, e^{i(kR-\omega t)}, \tag{2.2}$$

where A is the quadrupole strength, and R is the distance from the source. This is found by rotating the lateral quadrupole given by Pierce[11] 45°, specifying that θ in his geometry is 90°, and using the identity that $\sin 2\alpha = 2\sin\alpha\cos\alpha$. In equation (2.2), θ is the angle from the x axis towards the y axis in the x-y axis, which is given as ϕ in Pierce's quadrupole geometry.

$\frac{\partial p}{\partial \vec{n}}$ is found by taking the derivative of p with respect to the normal vector, which is shown in equation (2.3),

$$\frac{\partial p}{\partial \vec{n}} = \frac{A}{R}\cos(2\theta)\, e^{i(kR-\omega t)}\left(\frac{ikR-1}{R^2}\right). \tag{2.3}$$

The normal vector must lie along a radial line ($\vec{n} = \vec{R}$) which specifies that this grid is spherical. The derivative with respect to time is given by

$$\frac{\partial p}{\partial t} = \frac{-iA\omega}{R}\cos(2\theta)\, e^{i(kR-\omega t)}, \tag{2.4}$$

and the $\frac{\partial r}{\partial \vec{n}}$ factor in the Kirchhoff terms is given by

$$\frac{\partial r}{\partial \vec{n}} = \frac{\vec{r}\cdot\vec{n}}{\|\vec{r}\|\|\vec{n}\|}, \qquad \text{where } r = \|\vec{r}\| \tag{2.5}$$

and $\vec{r}$ is a vector pointing from $\mathbf{y}$ to $\mathbf{x}$. These equations allow a computationally quick and mathematically simple way to determine the Kirchhoff terms on the grid. Once the computational term values are calculated, the analytical-numerical method uses numerical techniques to integrate over the grid and compute the far-field pressure.

Using analytical expressions, however, limits both the computational grid shape and the source type. Choices for each must be made when determining the equations for the Kirchhoff terms. Equation (2.3) depends on a spherical grid where $\vec{n} = \vec{R}$. Equations (2.2), (2.3), and (2.4) exhibit spherical spreading with a quadrupole directivity, $\cos(2\theta)$, assuming

$kd \ll 1$, where d is the separation distance of the quadrupole's constituent monopole sources. Modifying the directivity of the source by changing the dependence on θ is simple, but to alter the shape of the grid, significant changes would have to be made in equation (2.3).

Although using analytical expressions for Kirchhoff terms imposes the above limitations, the analytical expressions allow validation of the fully numerical method and study of the effect of changing spherical grid radius. Because of the relationship between the analytical-numerical formulation and the fully numerical formulation, the difference between the two is error caused by the numerical schemes for calculating the Kirchhoff terms. Specifying a range of radii for a spherical grid shows how the size and resolution of the grid affect the far-field pressure calculations.

2.3. Fully Numerical Formulation

Because the fully numerical method computes the terms of the Kirchhoff equation numerically from the pressure history on the computational grid, no prior knowledge of the grid or the pressure generating mechanisms is required, although those mechanisms will probably be extensive simulations of acoustic sources. The grid and corresponding pressures could even be acquired from elaborate measurements of turbomachinery. The benefit of this formulation is its versatility.

The partial derivatives with respect to time, $\frac{\partial}{\partial t}$, and the normal vector, $\frac{\partial}{\partial \vec{n}}$, have complicated formulations. The time derivative is found by a centered difference scheme,[5]

$$\frac{\partial p}{\partial t} = \frac{p_{t+\Delta t} - p_{t-\Delta t}}{2\Delta t}, \tag{2.6}$$

where $p_{t+\Delta t}$ and $p_{t-\Delta t}$ are, respectively, the pressures at the time sample after and before the time sample at which the partial derivative is calculated. This numerical formulation requires the pressure history to be exactly one period, so $\frac{\partial p}{\partial t}$ at the first time sample can be calculated using p at the last time sample, and vice versa.

The numerical method calculates the $\frac{\partial p}{\partial \vec{n}}$ and $\frac{\partial r}{\partial \vec{n}}$ terms of the Kirchhoff equation using a second-order forward difference scheme based on values at three points. The three points are corresponding points in the three-layered computational grid. Because the grid is expected to be generated by a computational fluid dynamics simulation prior to the use of the Kirchhoff Integral Theorem, the grid layer which is closest to the acoustic source

distribution is chosen as the single layer which actually lies on the Kirchhoff surface. This choice is made because the grid points in directions orthogonal to the surface normal will more closely space than in the other two layers. Chapter 4 shows that large point spacing can be a source of error. Because of the choice of the first layer as the Kirchhoff surface, the forward difference scheme becomes necessary to find the $\frac{\partial}{\partial \vec{n}}$ terms.

The following is a general formulation to find the spatial derivative for a function, $f(x)$, and can be applied to both p and r. As shown in figure 2.3, three points, x_1, x_2, and x_3, located along a line normal to a surface, give the values $f(x_1)$, $f(x_2)$, and $f(x_3)$ for $f(x)$. x_1 is Δx_1 from x_2, x_2 is Δx_2 from x_3, and let $\Delta x_t = \Delta x_1 + \Delta x_2$. Taking the Taylor series[8] of $f(x_2)$ and $f(x_3)$,

$$f(x_2) = f(x_1) + \Delta x_1 f'(x_1) + \frac{\Delta x_1^2}{2} f''(x_1) + \cdots \tag{2.7}$$

$$f(x_3) = f(x_1) + \Delta x_t f'(x_1) + \frac{\Delta x_t^2}{2} f''(x_1) + \cdots, \tag{2.8}$$

multiplying equation (2.7) by Δx_t^2 and equation (2.8) by Δx_1^2, subtracting the second product from the first, and disregarding terms with Δx^4 or smaller values,

$$\frac{\partial f(x)}{\partial x} = \frac{\left(-2\Delta x_1 \Delta x_2 - \Delta x_2^2\right) f(x_1) + \left(\Delta x_1 + \Delta x_2\right)^2 f(x_2) - \Delta x_1^2 f(x_3)}{\Delta x_1 \Delta x_2 \left(\Delta x_1 + \Delta x_2\right)} \tag{2.9}$$

where $f'(x) = \frac{\partial f(x)}{\partial x}$ and $f''(x) = \frac{\partial^2 f(x)}{\partial x^2}$. Because x_1, x_2, and x_3 are along a normal, $\frac{\partial x}{\partial \vec{n}} = 1$, and if $f(x) = p(\mathbf{x}, t)$, then

$$\frac{\partial p}{\partial \vec{n}} = \frac{\partial f(x)}{\partial x} \frac{\partial x}{\partial \vec{n}} = \frac{\partial f(x)}{\partial x}. \tag{2.10}$$

The $\frac{\partial r}{\partial \vec{n}}$ term can be found in a similar way. This formulation handles grids with unevenly spaced layers, i.e., $\Delta x_1 \neq \Delta x_2$.

2.4. The Fourier Algorithm

The magnitude of the Fourier component at a single angle is found from the pressure history calculated at that angle by a discrete time Fourier transform. The transform computes the Fourier component for a discrete frequency from the sampled pressure history. The formula for this is given by

$$F_c = \sum_{m=1}^{n_t} p(m) e^{i\omega_c t} \Delta t, \tag{2.11}$$

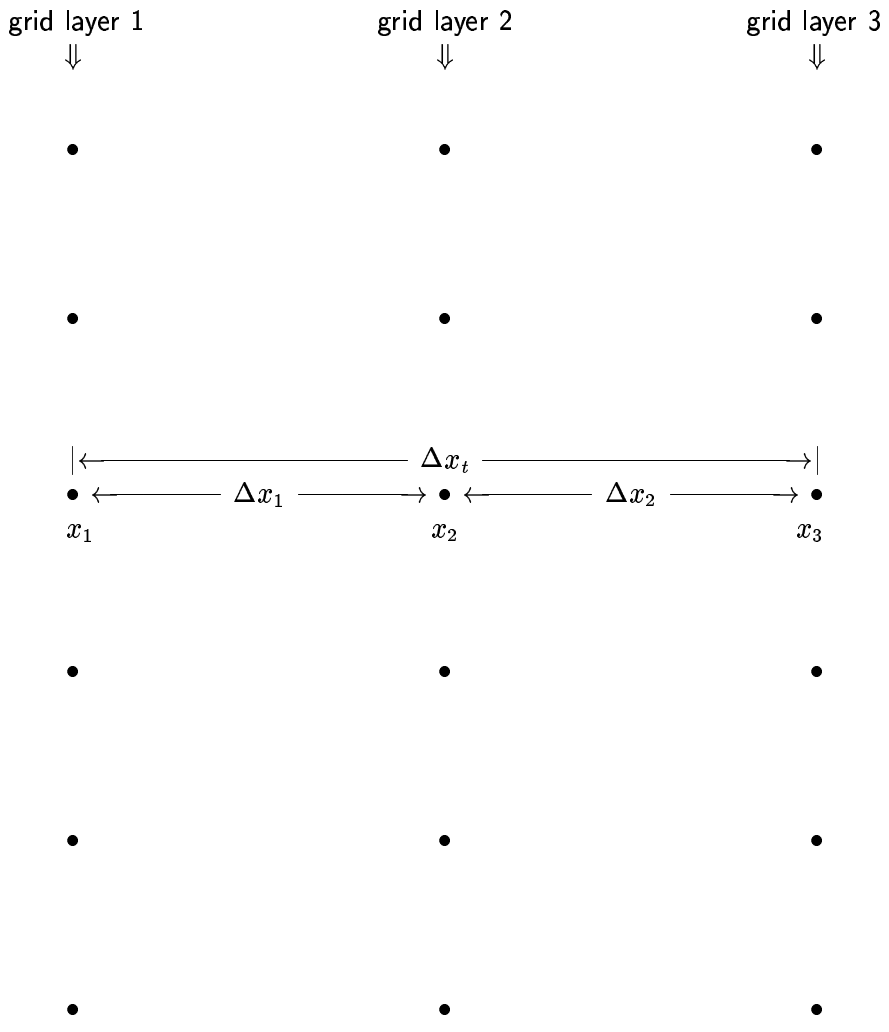


Figure 2.3: Three corresponding points in different grid layers.

where F_c is the Fourier component, n_t is the number of samples in one period of oscillation, $p(m)$ the sampled pressure history, and Δt is the size of the steps in time. ω_c represents the discrete frequency of interest, so only a component and not a function is found.

2.5. Summary

Computational grids represent Kirchhoff surfaces and provide locations for a computational version of the Kirchhoff method to calculate the terms of the Kirchhoff equation either analytically or numerically. Sharp edges on the surface require special attention when moving to the computational grid to avoid ambiguity in the direction of normal vectors. Both formulations expect an axisymmetric grid, and the fully numerical formulation also requires that the grid have three layers with corresponding points on increasing layers placed along lines normal to the surface represented, which should not be a difficult requirement of the computational fluid dynamics.

Analytical expressions for the Kirchhoff terms predetermine the computational grid and the source. This allows validation of the fully numerical method and study of changing the radius of a spherical computational grid. The fully numerical formulation is more versatile, however, since it finds the Kirchhoff terms numerically. $\frac{\partial p}{\partial t}$ is found through a centered difference scheme and $\frac{\partial p}{\partial \vec{n}}$ is found through a second-order forward difference scheme. The benefit of using the forward difference scheme to find $\frac{\partial p}{\partial \vec{n}}$ is that only pressure terms need to be known at the grid points. These terms are calculated at the first grid layer which lies on the Kirchhoff surface. This surface surrounds the acoustic source distribution and any nonlinear effects the distribution generates.

The directivities found in subsequent chapters are plots of the magnitude of a single Fourier component at each of the directivity angles. This Fourier component is calculated for a single frequency, ω_c, by a discrete time Fourier transform.

Chapter 3.

VALIDATION

Simple, classical sources, like the dipole and the quadrupole, have known far-field directivities and are therefore used as validation cases. They can be used to establish trust in the formulations shown in chapter 2. Keeping an eye toward turbomachinery, a ring source is used to check the fully numerical method. A ring source is a circular ring of monopoles vibrating at the same frequency with a particular phase relation and has been shown to simulate the fields generated by rotating sources.[12] Because the complexity of the ring source prevents the use of simple analytical equations to determine values of the Kirchhoff terms, the analytical-numerical method is not validated with the ring source.

To provide easy comparison, the directivity plane, shown in figure 3.1, is the x-y plane for all directivity plots, all source strengths are set to provide 1 Pa of pressure at the distance of the observer location, and all sources are centered at the origin. A quadrupole source with constituent monopole sources on the x and y axes is chosen for most of the validation cases because, even though it is not axisymmetric with respect to the axis of radial symmetry of the grid, it can point out problems in the grid that might be missed with a simpler source.

The analytical-numerical formulation is tested using a quadrupole and a spherical grid and the pressure calculated is compared to that found by analytical means from a quadrupole source. The fully numerical formulation is tested using the same source and grid and is compared to pressures found analytically from the same quadrupole source. The difference between the two formulations is also shown to show the difference between calculating the Kirchhoff terms analytically and numerically. The fully numerical method is also validated with different combinations of quadrupoles and ring sources, and spherical and cylindrical grids.

For this and subsequent chapters, the following notation will be used for brevity in figures: d_l is the grid layer separation distance, l_s is the length of the side of a cylindrical grid, n_{ax} is the number of axial points in the grid, n_{cr} is the number of circumferential points in the grid, n_t is the number of samples in one oscillation of the source, R is the distance from the source to the far-field observer point, r_s is the radius of the spherical grid, and r_c is the radius of the cylindrical grid.

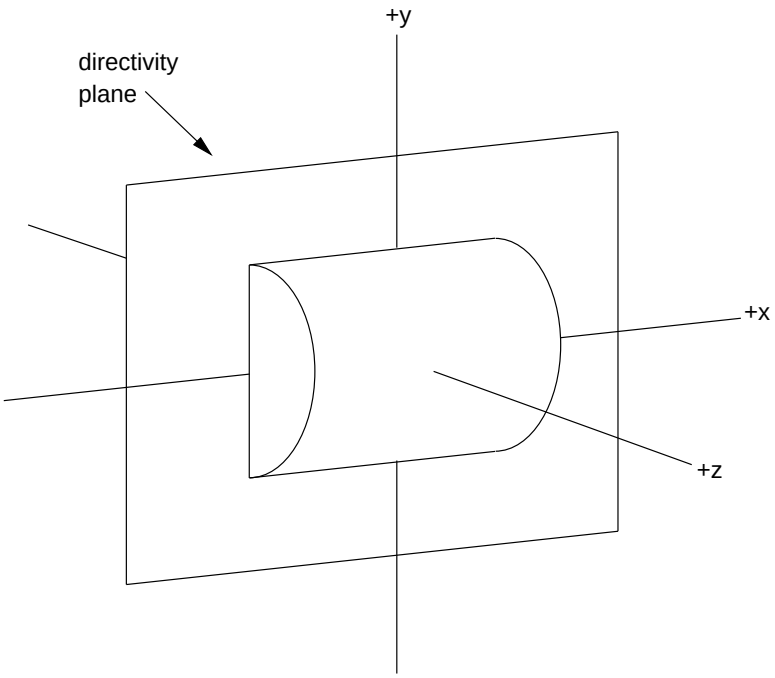


Figure 3.1: The directivity plane with a cylindrical Kirchhoff surface.

3.1. Far-Field Pressure

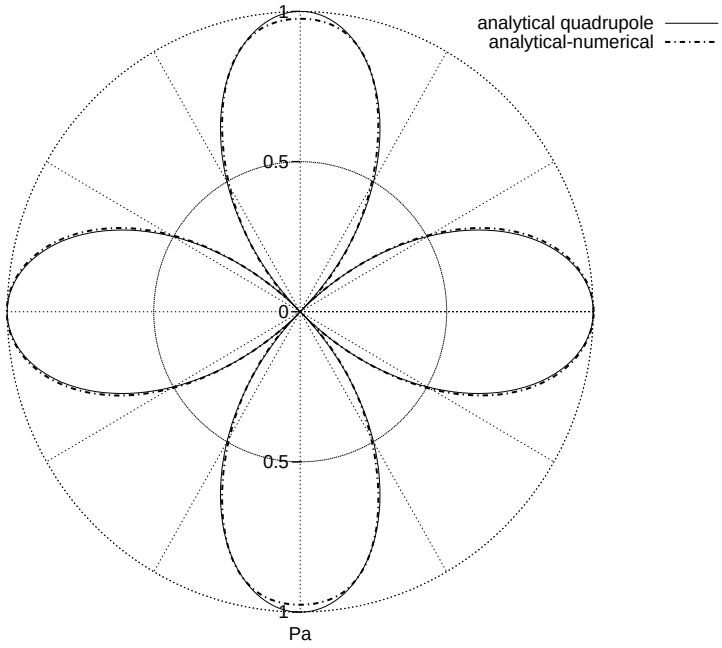
Figure 3.2.a shows the analytical-numerical method compared to the far-field directivity of a lateral quadrupole in the x-y plane with constituent monopoles on the x and y axes. Equation (3.1) gives the far-field pressure of a quadrupole with sources that are close together compared to wavelength, λ,

$$p = \frac{A}{R}\cos(2\theta)\, e^{i(kR-\omega t)}, \tag{3.1}$$

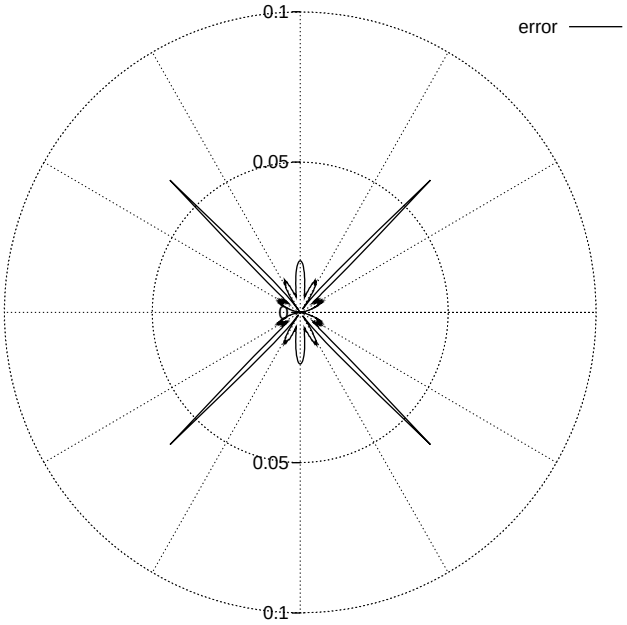
where A is the quadrupole strength, and R is the distance from the quadrupole source. θ is the angle from the x axis toward the y axis in the x-y plane. As stated in section 2.2, the analytical-numerical formulation depends on a spherical computational grid.

Figure 3.2.b is the error between the analytical-numerical method and the field found by analytical means from a quadrupole source. Error is defined in this thesis as

$$\frac{|p_f - p_a|}{\max p_a}, \tag{3.2}$$



a. Directivities of the analytical-numerical formulation and an analytical quadrupole.



b. Error between the directivities shown in a.

Figure 3.2: Quadrupole source validation of the analytical-numerical formulation with a spherical grid ($n_{ax} = 30$, $n_{cr} = 40$, $n_t = 60$, $R = 10.3\lambda$, $r_s = 2\lambda$).

where p_f is the Fourier component of the pressure of the formulation, p_a is the Fourier component of the pressure of the field found by analytical means, and $\max p_a$ is the pressure value of the analytical quadrupole at an antinode. This definition for error shows the difference between pressure found from a quadrupole by analytical means and from a formulation with respect to a value that is not dependent on directivity angle. The maximum scale of this plot is set to 0.1 Pa which is 10% of the analytical quadrupole's far-field pressure at an antinode. All error plots in this thesis share this same scale for easy comparison with other error plots. The consistent errors seen at 45°, 135°, 225°, and 315° are at the nodes of the quadrupole.

In figure 3.3, the fully numerical formulation shows error levels similar to the analytical-numerical method for a spherical grid around a quadrupole, but the errors occur at slightly different directivity angles. A comparison of the two methods is shown in figure 3.4, where the plot points out the difference between calculating the terms of the Kirchhoff equation analytically and numerically.

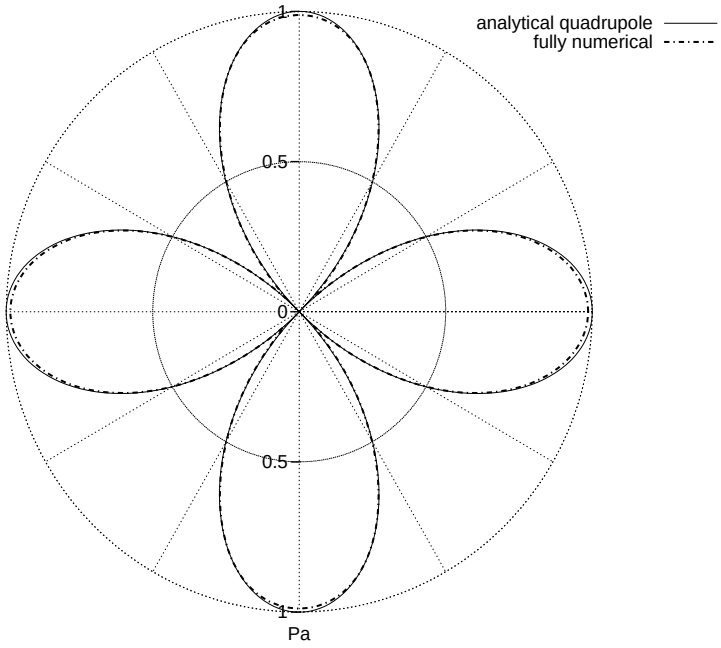
A cylindrical grid is used for the calculations shown in figure 3.5. At directivity angles near 0° and 180°, the error levels are larger than those found using a spherical grid, and the error seen at the quadrupole nodes is spread over more directivity angles. The reason for the change in the placement of error is not known.

Figure 3.6 shows the far-field directivity of a ring source calculated by the fully numerical method using a spherical grid. Note the small error compared to the other combinations of sources, formulations, and grid shapes.

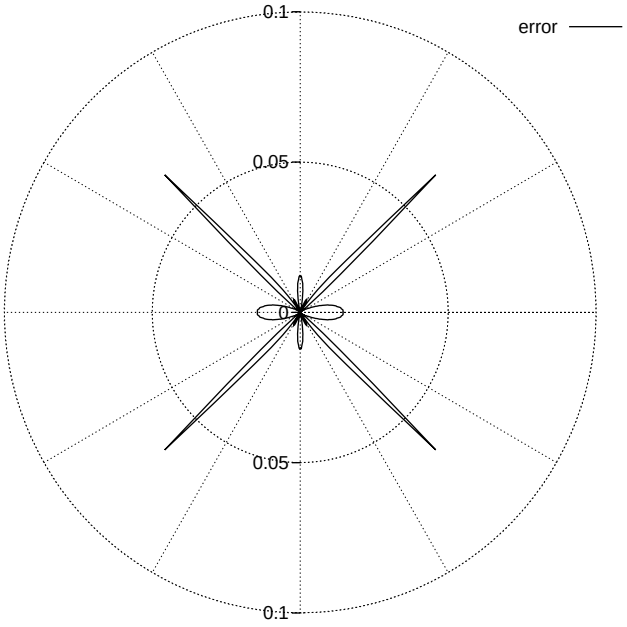
Figure 3.7, which uses a cylindrical grid to validate the ring source, contains significantly more error than the other validation cases. This error is located at angles where the source value is low, which may make the error negligible for turbomachinery applications. The far-field pressure calculations for the cylindrical grid are less accurate for some sources than for others.

3.2. Frequency Separation

Figure 3.8.a shows that the Fourier algorithm is effective in separating pressure directivities due to two sources vibrating at different frequencies. A lateral quadrupole source operating at a frequency of 686 Hz is in the same space as a dipole source with constituent

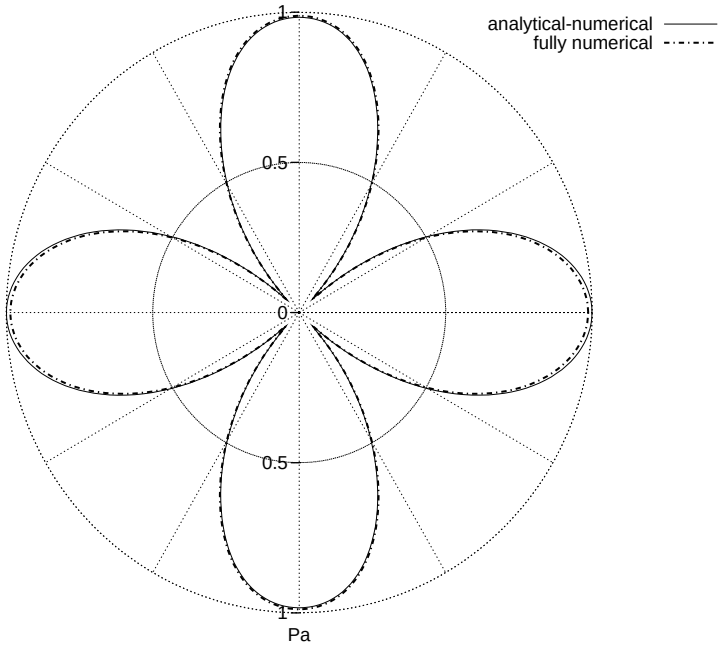


a. Directivities of the fully numerical formulation and an analytical quadrupole.

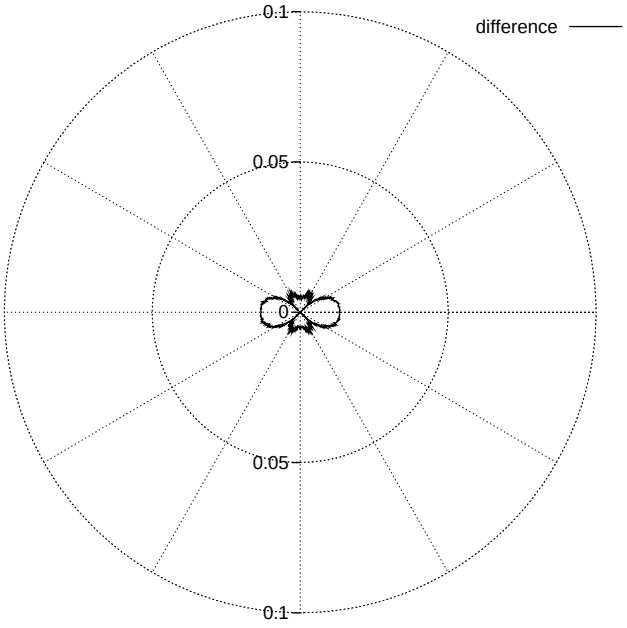


b. Error between the directivities shown in a.

Figure 3.3: Quadrupole validation of the fully numerical formulation with a spherical grid ($d_l = 0.01\lambda$, $n_{ax} = 30$, $n_{cr} = 40$, $n_t = 60$, $R = 10.3\lambda$, $r_s = 2\lambda$).

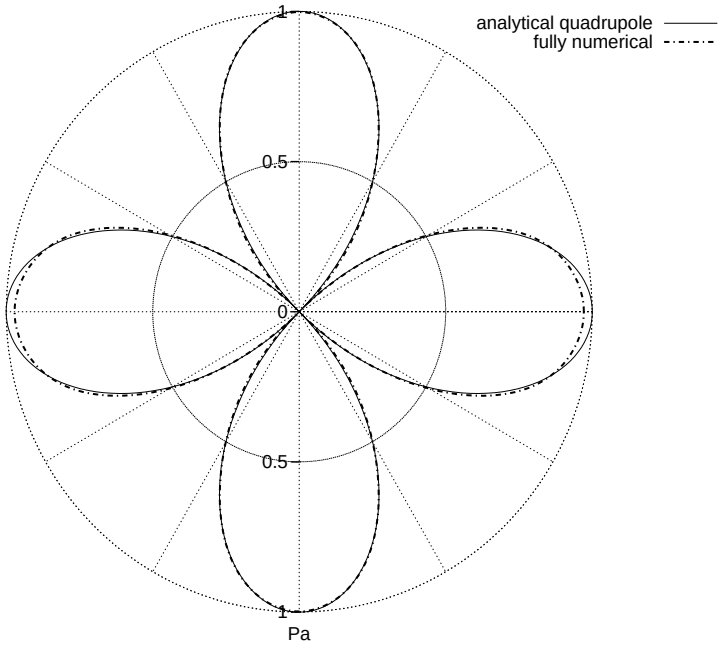


a. Directivities of the two formulations.

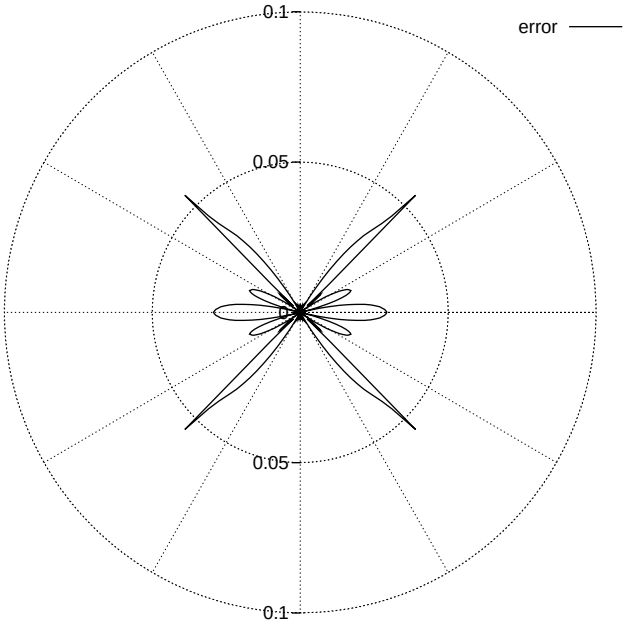


b. Difference between the two formulations.

Figure 3.4: Comparison of the analytical-numerical and fully numerical formulations using a quadrupole source and a spherical grid ($d_l = 0.01\lambda$ [for the fully numerical method only], $n_{ax} = 30$, $n_{cr} = 40$, $n_t = 60$, $R = 10.3\lambda$, $r_s = 2\lambda$).

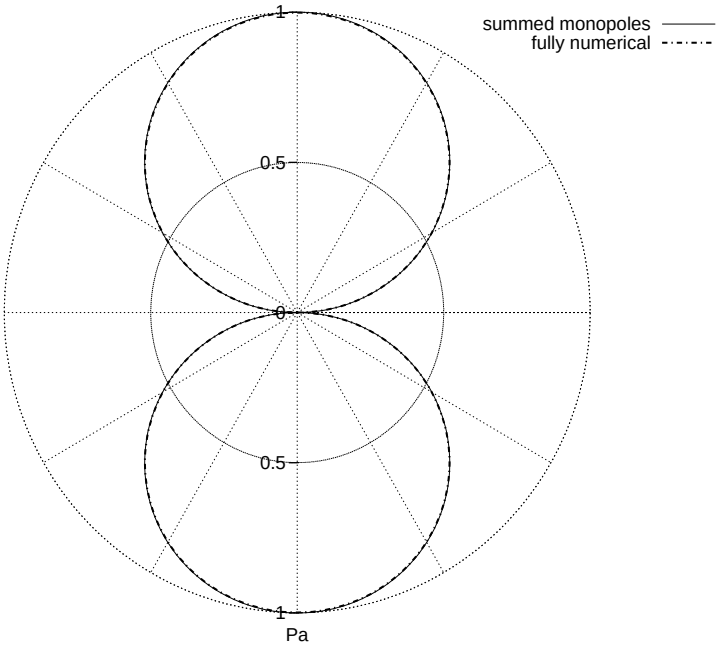


a. Directivities of the fully numerical formulation and an analytical quadrupole.

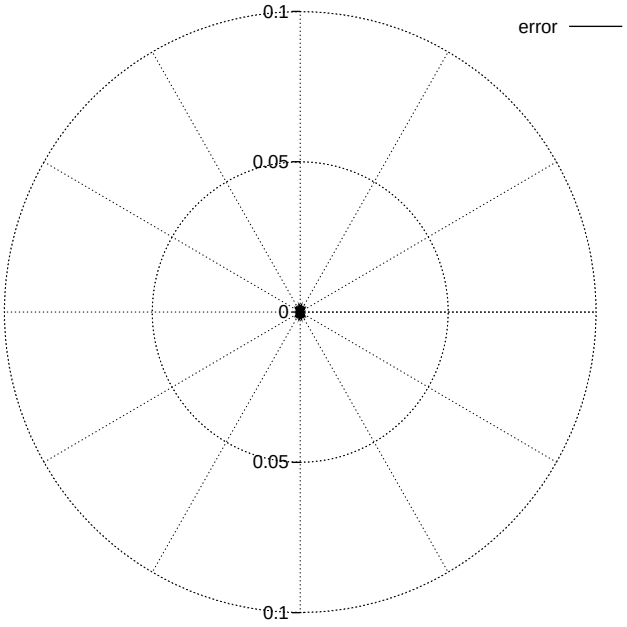


b. Error between the directivities shown in a.

Figure 3.5: Quadrupole validation of the fully numerical formulation with a cylindrical grid ($d_l = 0.01\lambda$, $l_s = 4\lambda$, $n_{ax} = 32$, $n_{cr} = 40$, $n_t = 60$, $R = 10.3\lambda$, $r_c = 2\lambda$).

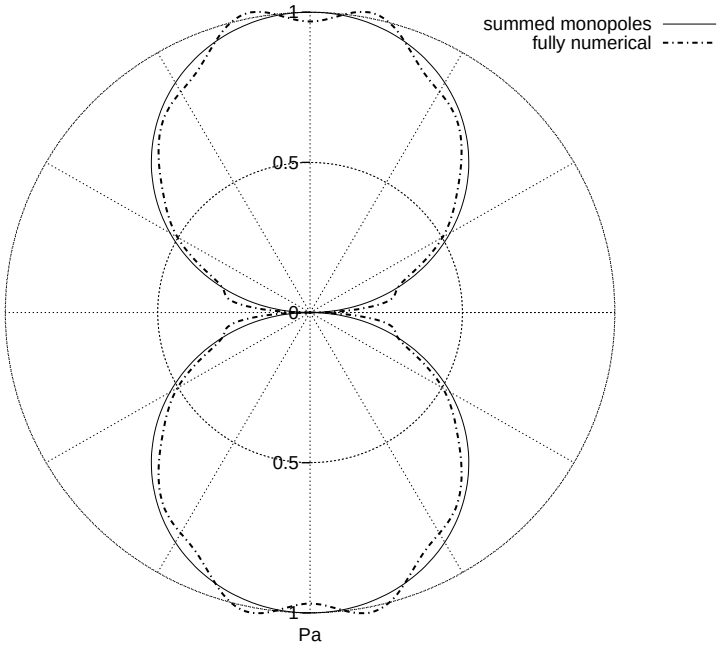


a. Directivities of the fully numerical formulation and the summed monopoles.

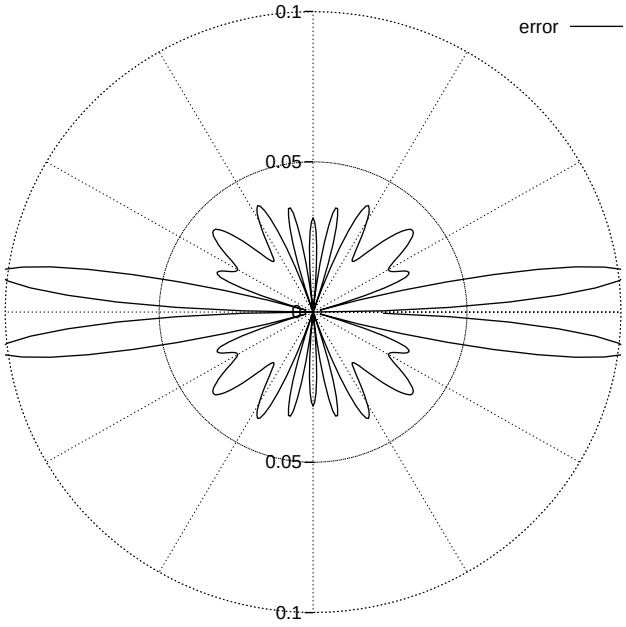


b. Error between the directivities shown in a.

Figure 3.6: Ring source validation of the fully numerical formulation with a spherical radius grid ($d_l = 0.01\lambda$, $n_{ax} = 32$, $n_{cr} = 40$, $n_t = 60$, $R = 10.3\lambda$, $r_s = 2\lambda$). The ring source contains 32 monopoles in a ring with a radius of 0.125λ around the x axis. The phase of each monopole source increases by $\frac{\pi}{16}$ radians around the ring.



a. Directivities of the fully numerical formulation and the summed monopoles.



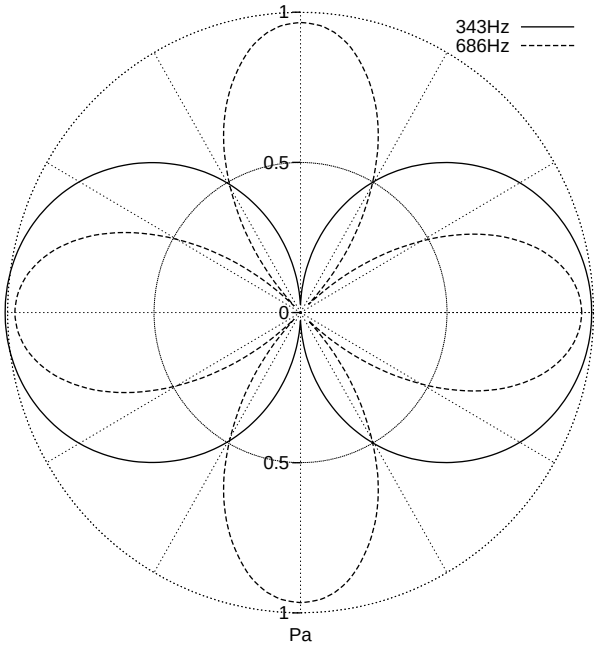
b. Error between the directivities shown in a.

Figure 3.7: Ring source validation of the fully numerical formulation with a cylindrical grid ($d_l = 0.01\lambda$, $l_s = 4\lambda$, $n_{ax} = 32$, $n_{cr} = 40$, $n_t = 60$, $R = 10.3\lambda$, $r_c = 2\lambda$). The ring source contains 32 monopoles in a ring with a radius of 0.125λ around the x axis. The phase of each monopole source increases by $\frac{\pi}{16}$ radians around the ring.

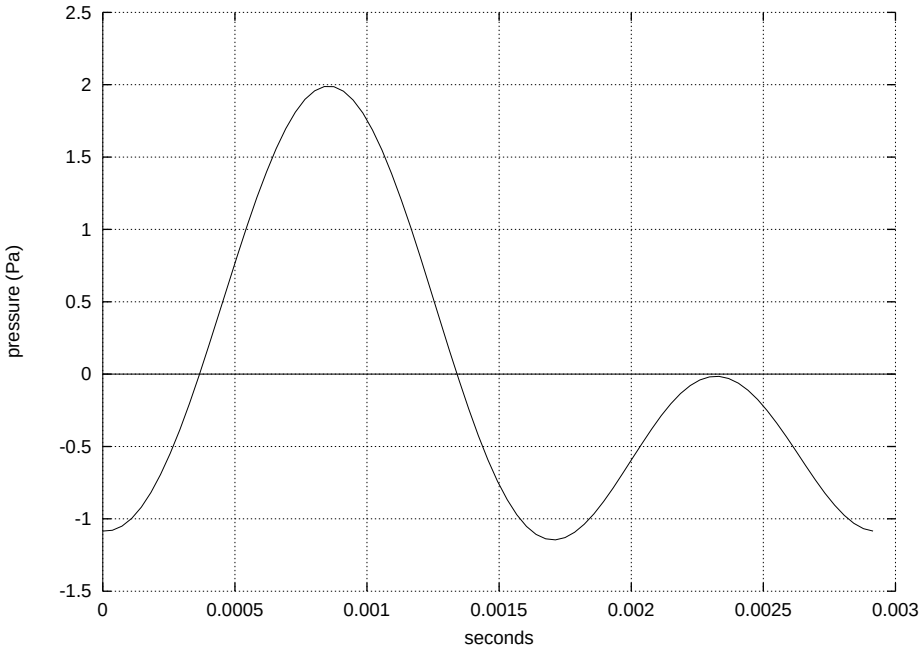
monopole sources located on the x axis operating at a frequency of 343 Hz. Because the shortest wavelength of interest is 0.5 m for this case, the number of axial and circumferential grid points are doubled. Section 4.1 discusses grid point spacing issues. The far-field pressure along the $+x$ axis is shown in figure 3.8.b. The Fourier algorithm is able to separate the two source directivities from the total pressure signal at each directivity angle and produce the pressure directivities shown in figure 3.8.a.

3.3. Summary

The validation cases in this chapter indicate that the computational Kirchhoff formulations shown in chapter 2 can calculate far-field pressures, but the grid has an impact on the error in the calculations. Chapter 4 examines how the grid point spacing, size, and shape and time sampling all affect the error in the far-field pressure.



a. Directivities of the two sources at different frequencies.



b. Time history of the pressure at $\mathbf{x}$ on the x axis.

Figure 3.8: Validation of frequency separation using a dipole source with a frequency of 343 Hz and a quadrupole source with a frequency of 686 Hz with a spherical grid ($d_l = 0.01\lambda$, $n_{ax} = 60$, $n_{cr} = 80$, $n_t = 60$, $R = 10.3\lambda$, $r_s = 2\lambda$).

Chapter 4.

DISCRETIZATION AND THE KIRCHHOFF GRID

Without a viable grid, a computational formulation of the Kirchhoff method cannot accurately calculate far-field pressure. This chapter discusses the issues involved in creating a usable grid and discretizing the grid and pressure histories.

One of the characteristics of a grid is its point spacing. For a given grid size, the distance between two consecutive grid points is determined by the total number of points on the grid. For the grid generation algorithms used in the formulations described in chapter 2, the number of grid points is determined by two program variables, `naxis` and `ncirc`, which, respectively, specify the number of axial and circumferential grid points. The grid point spacing of a computational grid can have a significant effect on the far-field pressure calculations, especially if the grid is too sparse.

In the fully numerical formulation, the separation distance between grid layers also plays a key role in accurate far-field calculations. It is important to specify that the layer distance is very small compared to a wavelength, because the forward difference scheme used for the $\frac{\partial p}{\partial \vec{n}}$ and $\frac{\partial r}{\partial \vec{n}}$ factors depends on small values of Δx, as shown in section 2.3. The actual values for grid layer separation are discussed in section 4.1.3.

The Fourier algorithm cannot accurately determine the contribution of a particular frequency to the far-field pressure without enough time samples in a period of pressure oscillation. The accuracy of the directivity of the far-field pressure depends on accurate frequency contribution values. Section 4.2 discusses the number of time samples necessary to generate accurate far-field pressures using the discrete time Fourier algorithm.

The size and shape of the grid also affect the pressure calculations. According to the Kirchhoff Integral Theorem, the Kirchhoff surface must be large enough to enclose any nonlinearities of the source, and cannot extend out beyond the far-field observer point. These restrictions are upheld by the computational formulation, and are shown in section 4.3. The shape of the grid can have an effect on the far-field pressure calculations, which is shown in section 4.4.

All distances in this chapter are specified in terms of wavelength, λ, because all spacings of the grid and the far-field distance depend on λ. If the wavelength of the source is changed

by changing either the frequency or the speed of sound, then all distances must be adjusted accordingly.

4.1. Grid Point Spacing

Grid point spacing refers to the distance between the points which comprise a computational grid and how those points are arranged. The axial grid point spacing is the distance between points a grid contains in the direction of the axis of radial symmetry, the x axis in all of the cases shown in this thesis. The circumferential grid point spacing specifies the distance between points around the axis. On a globe, axial lines correspond to longitude lines, circumferential lines correspond to latitude lines, and the axis of symmetry runs through the North and South Poles.

The distance between grid points, determined by the grid array dimensions, must be below a certain threshold for the Kirchhoff method to work effectively, but the dependence of the far-field pressure on the axial grid point spacing is different from the dependence on the circumferential grid point spacing. Each grid point spacing controls different aspects of the density of grid points. Figure 2.1.b shows a spherical grid with more axial than circumferential points and figure 2.1.c shows a spherical grid with more circumferential points than axial. The axial and circumferential grid point spacings are therefore explored separately.

When error is caused by the distance between points in the axial and circumferential directions at the same time, the effects are not directly additive. The total error is significantly less than the two errors simply added together. Because the density of points changes over the surface of a grid, it is difficult to specify how the effects from the two different directions interact.

The shape of the grid can affect its dependence on point spacing, therefore plots are generated for both a spherical and cylindrical grid. Because the analytical-numerical formulation is tied to a spherical grid, the fully numerical formulation is used to calculate the far-field directivities for the two differently shaped grids. This keeps the error between different grid shapes specific to the point spacing effects.

4.1.1. Axial Grid Point Spacing

4.1.1.1. Spherical Grid

The maximum axial distance between grid points, d_{ax}, in a spherical grid is given by

$$d_{ax} = \frac{\pi r_s}{\texttt{naxis}}, \tag{4.1}$$

where r_s is the maximum radius of the spherical grid and `naxis` is a program variable that represents the number of axial grid points. `naxis` is the first number given in the grid array dimensions in this thesis. A 17x24 grid therefore indicates a grid with 17 axial points (`naxis` = 17), and 24 circumferential points.

Figure 4.1 shows directivities for successively larger axial grid spacings, along with the error between each directivity and the pressure field generated by an analytical quadrupole. The lowest axial grid array dimension shown is the highest axial grid array dimension which causes a maximum local error of greater than 10% of the antinode pressure of the analytical quadrupole. A low axial grid array dimension refers to a small number of axial points, and a high axial grid array dimension refers to a grid with many axial points. Local error is defined as the difference between the far-field calculation and the pressure generated by an analytical quadrupole multiplied by the value of the analytical quadrupole each divided by the pressure of the quadrupole at an antinode. Specifically, local error is given by

$$\frac{|p_f - p_a|}{\max p_a} \frac{p_a}{\max p_a}. \tag{4.2}$$

The seven directivity plots which lead up to the largest axial grid spacing are also shown.

Local error is used to examine the difference between the pressure generated by a formulation and the pressure generated by a quadrupole found by analytical means with the difference scaled by the pressure from the quadrupole at the specific directivity angle of calculation. The benefit of local error is that the error seen at the antinodes is given more weight than the error at the nodes. Local error therefore helps in presenting the error generated by varying grid parameters, because the constant error seen at the nodes of the quadrupole becomes negligible. dB local error is also shown in several plots and is 20 times the log of the difference between the pressure found from a formulation and analytically from a quadrupole scaled by the pressure value of the quadrupole at the directivity angle

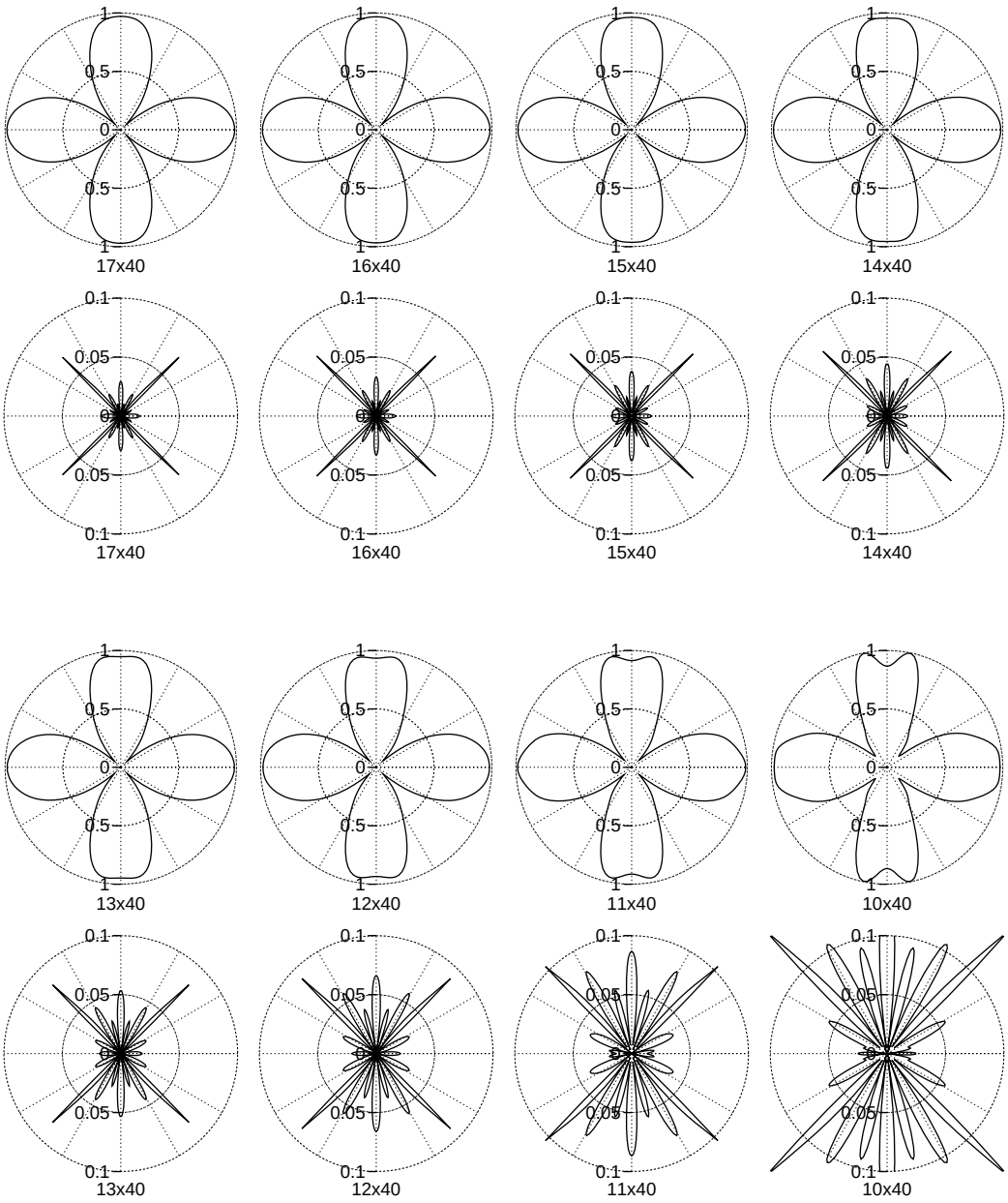


Figure 4.1: Directivity and error plots of successively larger axial grid point spacing for a 343 Hz quadrupole using a spherical grid ($d_l = 0.01\lambda$, $n_t = 60$, $R = 10.3\lambda$, $r_s = 2\lambda$). Each directivity and error plot is labeled with its corresponding grid array dimensions.

of calculation relative to the maximum pressure value of the quadrupole. Specifically, dB local error is given as

$$20 \log \left(\frac{|p_f - p_a|}{\max p_a} \frac{p_a}{\max p_a} \right) . \tag{4.3}$$

Figure 4.2 shows the directivities of the same grids as figure 4.1, but the error plots are of local error. Figure 4.3 shows the maximum local error of the same grids as in figures 4.1 and 4.2 versus the axial grid array dimensions. The data shown in figure 4.3 is discrete, and the line drawn between discrete data points is only there as a visual aid. The 10x40 grid is the first grid which exceeds a local error of 10%, so according to table 4.1 the maximum axial point spacing for the grid must be less than $\sim 0.57\lambda$ to ensure a local error of less than 10%. Table 4.1 is generated from equation (4.1) by specifying the number of axial points for the spherical grids used in figures 4.1–4.3.

Table 4.1: Axial grid array dimensions and maximum axial grid spacing for a spherical grid.

naxis	d_{ax}
10	0.63λ
11	0.57λ
12	0.53λ
13	0.48λ
14	0.45λ
15	0.41λ
16	0.39λ
17	0.37λ

4.1.1.2. Cylindrical Grid

Discussion of a cylindrical grid is simplified by naming the circular end-caps in terms of their computational fluid dynamics characteristics. One of the end-caps is called the inlet

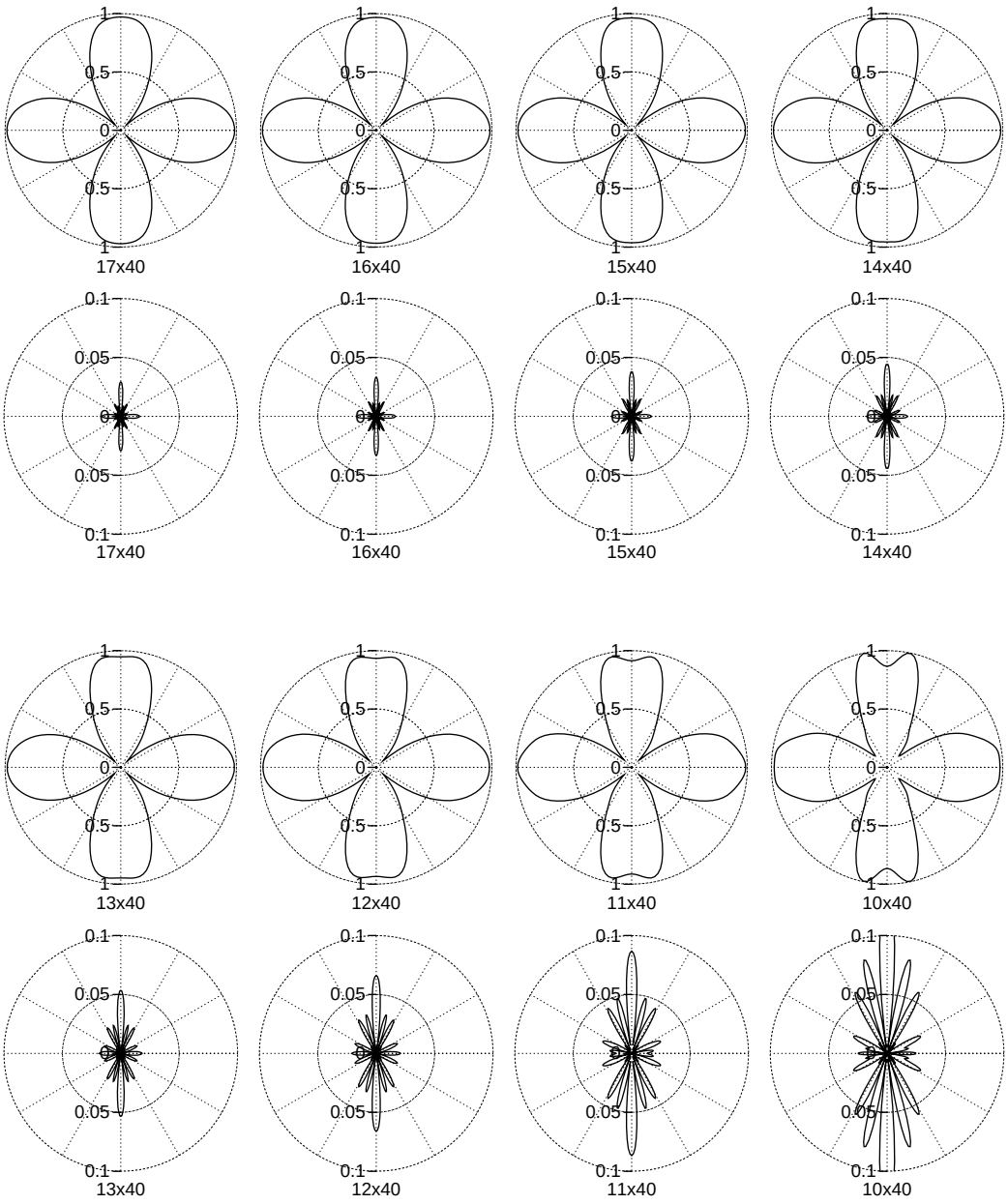


Figure 4.2: Directivity and local error plots of successively less dense axial grid resolutions for a 343 Hz quadrupole using a spherical grid ($d_l = 0.01\lambda$, $n_t = 60$, $R = 10.3\lambda$, $r_s = 2\lambda$).

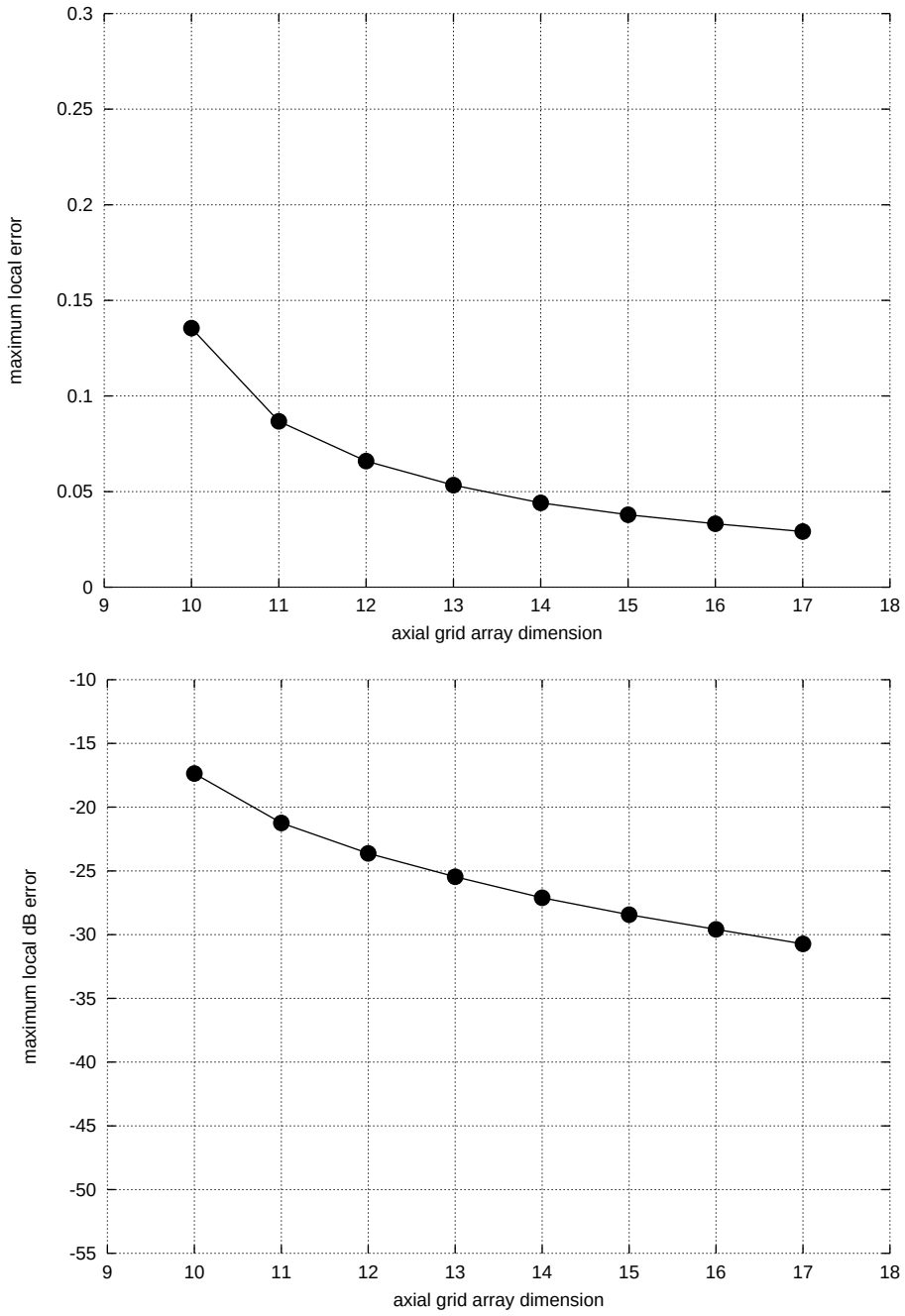


Figure 4.3: Maximum local error for successively larger axial grid point spacing for a 343 Hz quadrupole using a spherical grid ($d_l = 0.01\lambda$, $n_t = 60$, $R = 10.3\lambda$, $r_s = 2\lambda$).

end-cap, and the other is called the outlet end-cap. Because there is no flow in these computations, this is purely a matter of nomenclature.

The formula for the maximum axial grid point spacing, d_{ax}, depends on where the d_{ax} is located. If d_{ax} is on the side of the cylinder, then d_{ax} is given as

$$d_{ax} = \frac{l_s}{\texttt{naxis} - \texttt{nptinlrad} - \texttt{nptotlrad}}, \tag{4.4}$$

where l_s is the length of the cylinder's side, `nptinlrad` is a program variable representing the number of axial points contained in the inlet end-cap, and `nptotlrad` is another program variable representing the number of axial points in the outlet end-cap. If the d_{ax} is found on the inlet end-cap, then d_{ax} is given as

$$d_{ax} = \frac{r_{inl}}{\texttt{nptinlrad}}, \tag{4.5}$$

where r_{inl} is the radius of the inlet end-cap. If d_{ax} is found on the outlet end-cap, then d_{ax} is given as

$$d_{ax} = \frac{r_{otl}}{\texttt{nptotlrad}}, \tag{4.6}$$

where r_{otl} is the radius of the outlet end-cap. For a cylindrical grid, r_{inl} and r_{otl} are the same, but for a conical grid they are not.

Figure 4.4 shows directivity plots produced from cylindrical grids with successively larger axial grid resolutions. This figure also shows the error between the pressure field calculated by the fully numerical formulation and the pressure field found from an analytical quadrupole. Figure 4.5 shows the cylindrical grid pressure directivities with local error plots, and figure 4.6 shows the maximum local error versus axial grid array dimension. From the local error plots, the grid that exceeds a local error of 10% is the 21x40 grid. Because the maximum axial spacing is forced to be along the side of the cylinder, table 4.2 gives the axial grid spacing between the two most distant, consecutive axial points using equation (4.4). An axial grid spacing of no more than $\sim 0.40\lambda$ is necessary to have less than 10% maximum local error for a cylindrical grid.

The requirement that axial grid points be no more than $\sim 0.40\lambda$ apart for a maximum local error of less than 10% is stricter than for a spherical grid, which requires the spacing to be no more than $\sim 0.57\lambda$ for the same maximum local error. One possible reason for this is that the maximum axial spacing causes a larger portion of a cylindrical grid to be sparsely covered by points than the maximum axial spacing does on a spherical grid.

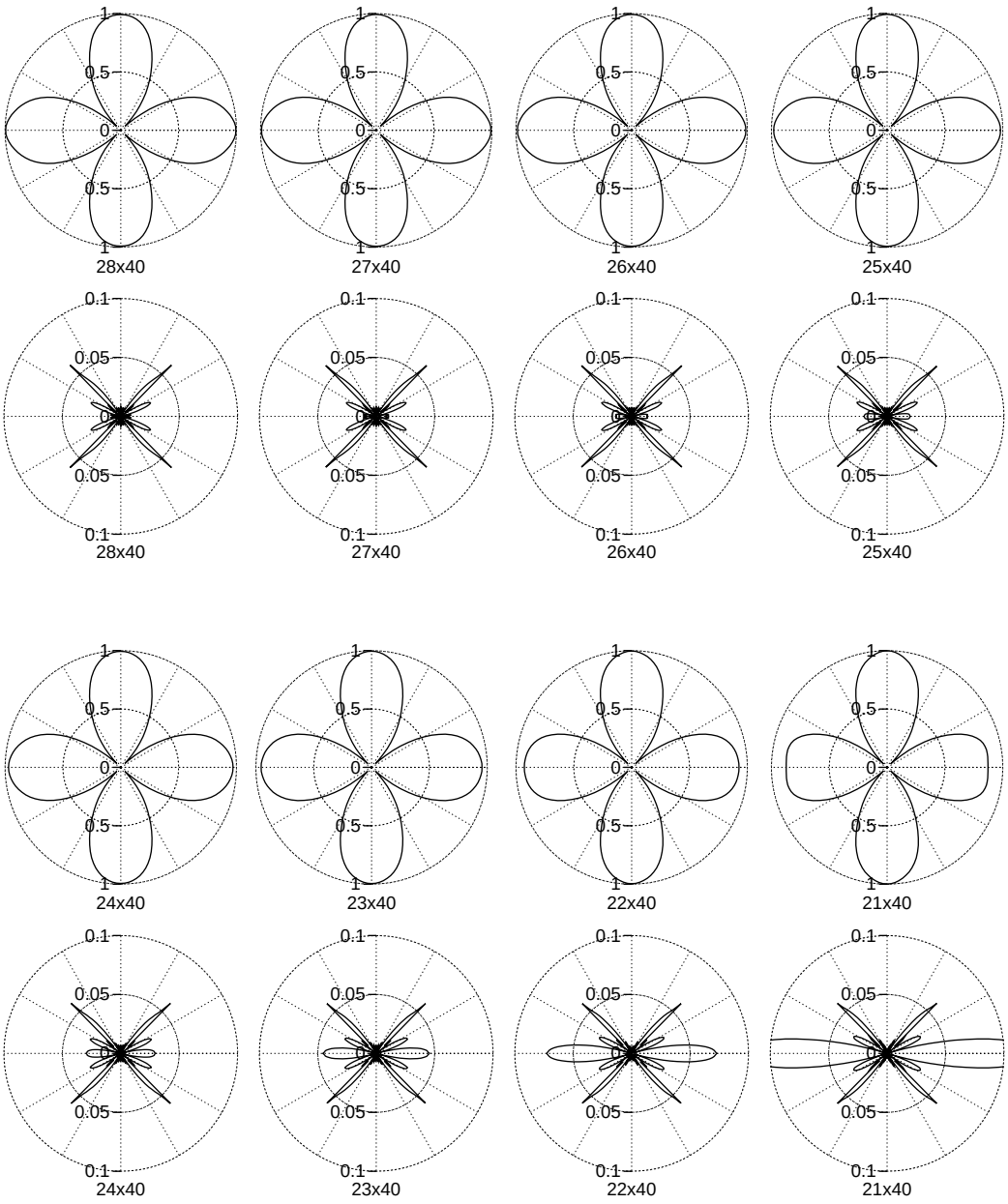


Figure 4.4: Directivity and error plots of successively larger axial grid point spacing for a 343 Hz quadrupole using a cylindrical grid ($d_l = 0.01\lambda$, $l_s = 4\lambda$, $n_t = 60$, $R = 10.3\lambda$, $r_c = 2\lambda$).

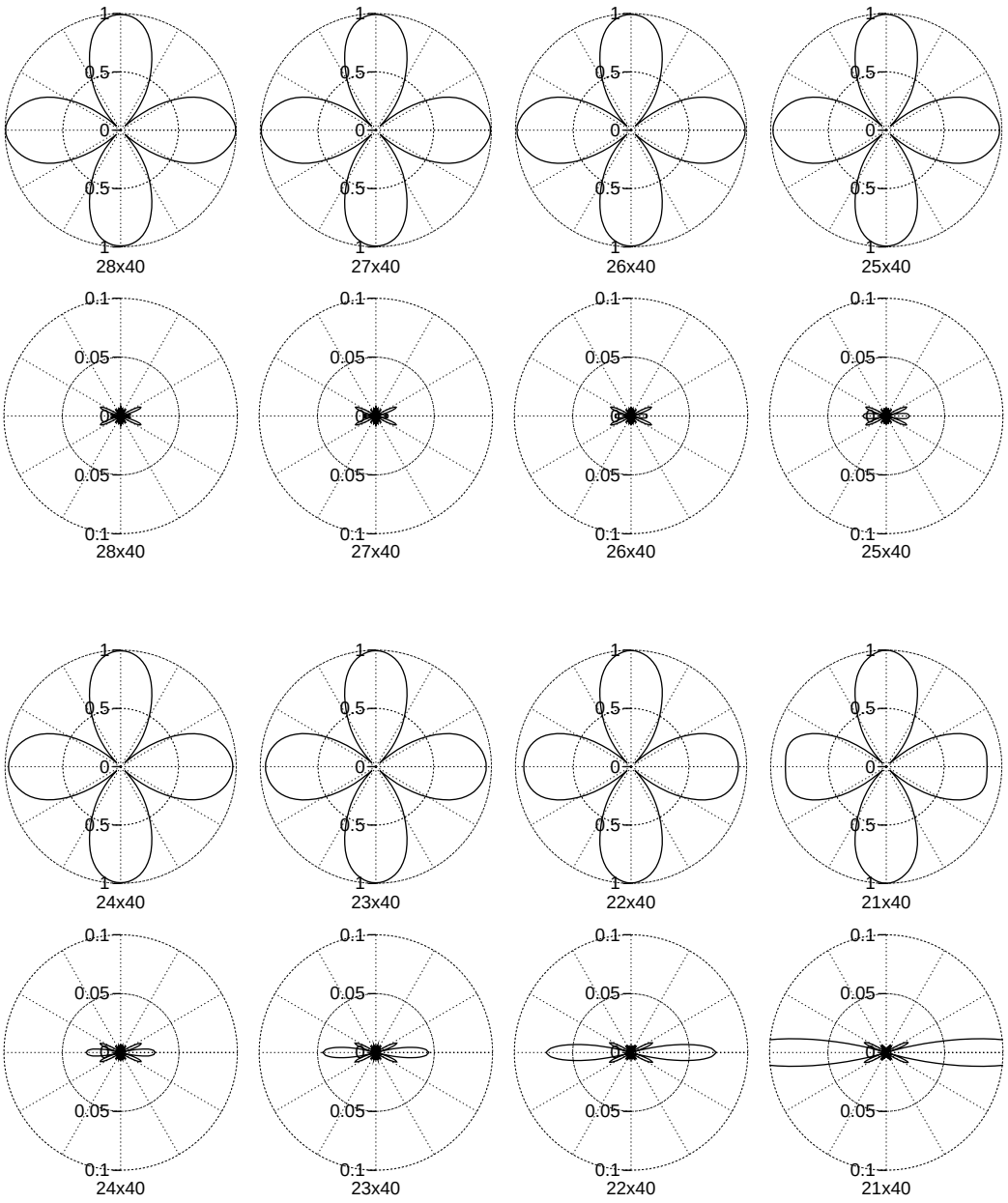


Figure 4.5: Directivity and local error plots of successively larger axial grid point spacing for a 343 Hz quadrupole using a cylindrical grid ($d_l = 0.01\lambda$, $l_s = 4\lambda$, $n_t = 60$, $R = 10.3\lambda$, $r_c = 2\lambda$).

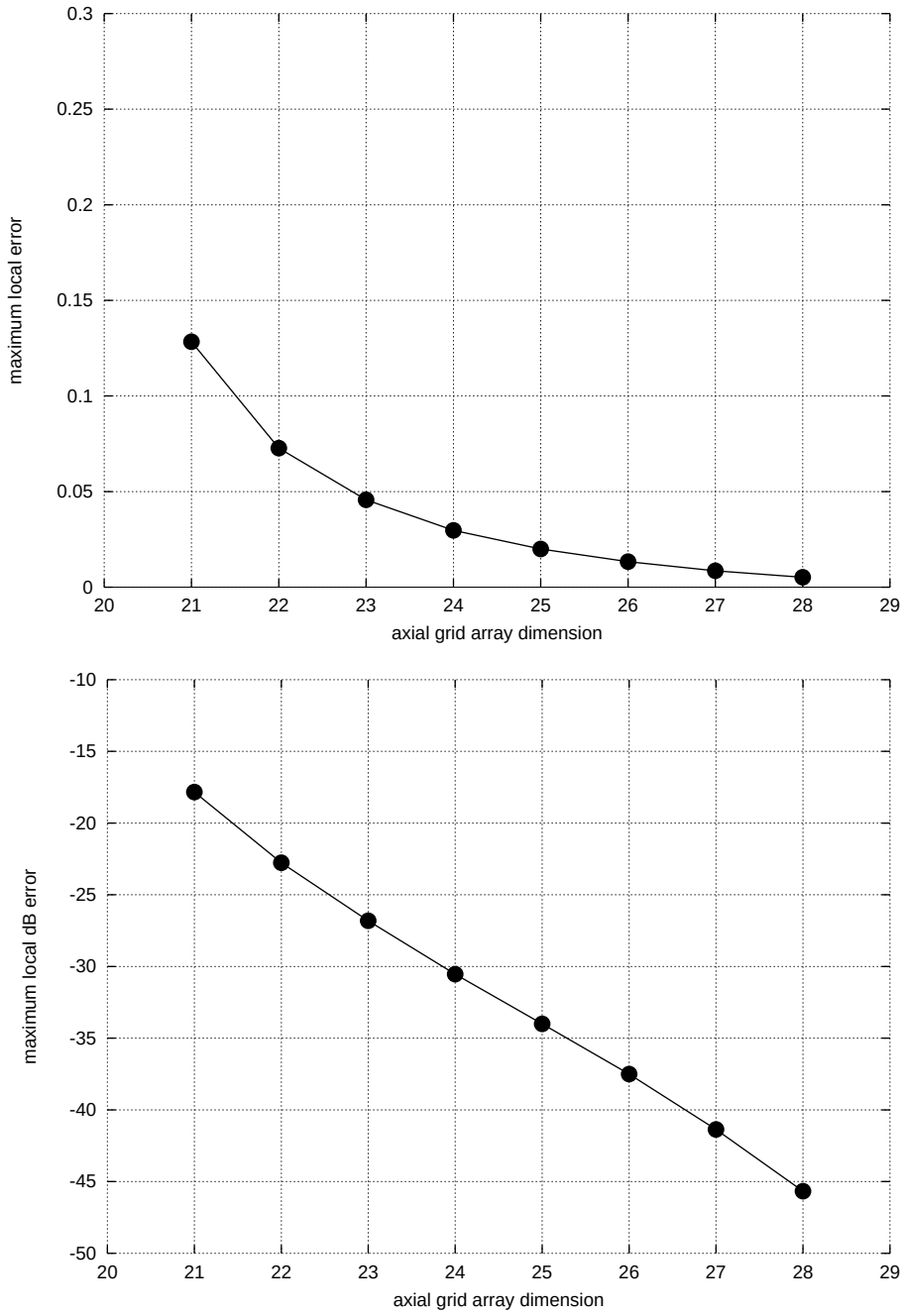


Figure 4.6: Maximum local error plots of successively larger axial grid point spacing for a 343 Hz quadrupole using a cylindrical grid ($d_l = 0.01\lambda$, $l_s = 4\lambda$, $n_t = 60$, $R = 10.3\lambda$, $r_c = 2\lambda$).

Table 4.2: Axial grid array dimensions and maximum axial grid point spacing for a cylindrical grid.

naxis	d_{ax}
21	0.44λ
22	0.40λ
23	0.36λ
24	0.33λ
25	0.31λ
26	0.29λ
27	0.27λ
28	0.25λ

4.1.2. Circumferential Grid Point Spacing

4.1.2.1. Spherical Grid

The maximum circumferential grid point distance, d_{cr}, for a spherical grid is given by

$$d_{cr} = \frac{2\pi r_x}{\texttt{ncirc}}, \tag{4.7}$$

where r_x is the radius of the grid for a constant x and `ncirc` is a program variable representing the number of circumferential points in the grid. `ncirc` is written second in the grid array dimensions in this thesis, so for a 20x22 grid, `ncirc` = 22.

Figure 4.7 shows directivities for successively larger circumferential grid point spacing on a spherical grid along with the error between each pressure field and the field generated by analytical means from a quadrupole. Figure 4.8 shows the same directivities with the local error. Figure 4.9 arranges the maximum local error to compare circumferential grid array dimensions. From the local error plots, the grid with the lowest circumferential grid array dimension which still has a local error of less than 10% is the 30x18 grid. From table 4.3, which is generated from equation (4.7), the maximum circumferential point spacing for this grid is $\sim 0.70\lambda$.

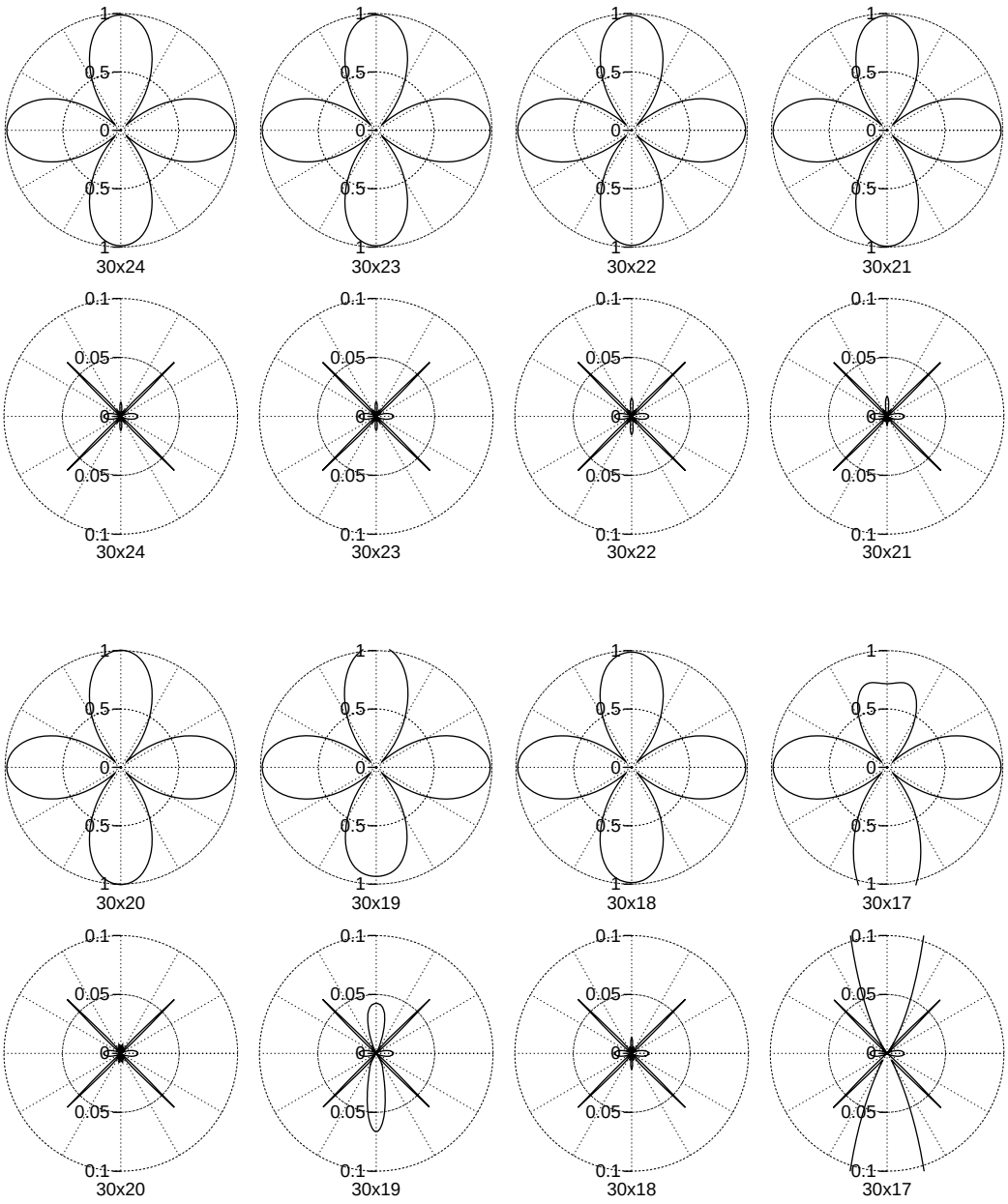


Figure 4.7: Directivity and error plots of successively larger circumferential grid point spacing for a 343 Hz quadrupole using a spherical grid ($d_l = 0.01\lambda$, $n_t = 60$, $R = 10.3\lambda$, $r_s = 2\lambda$).

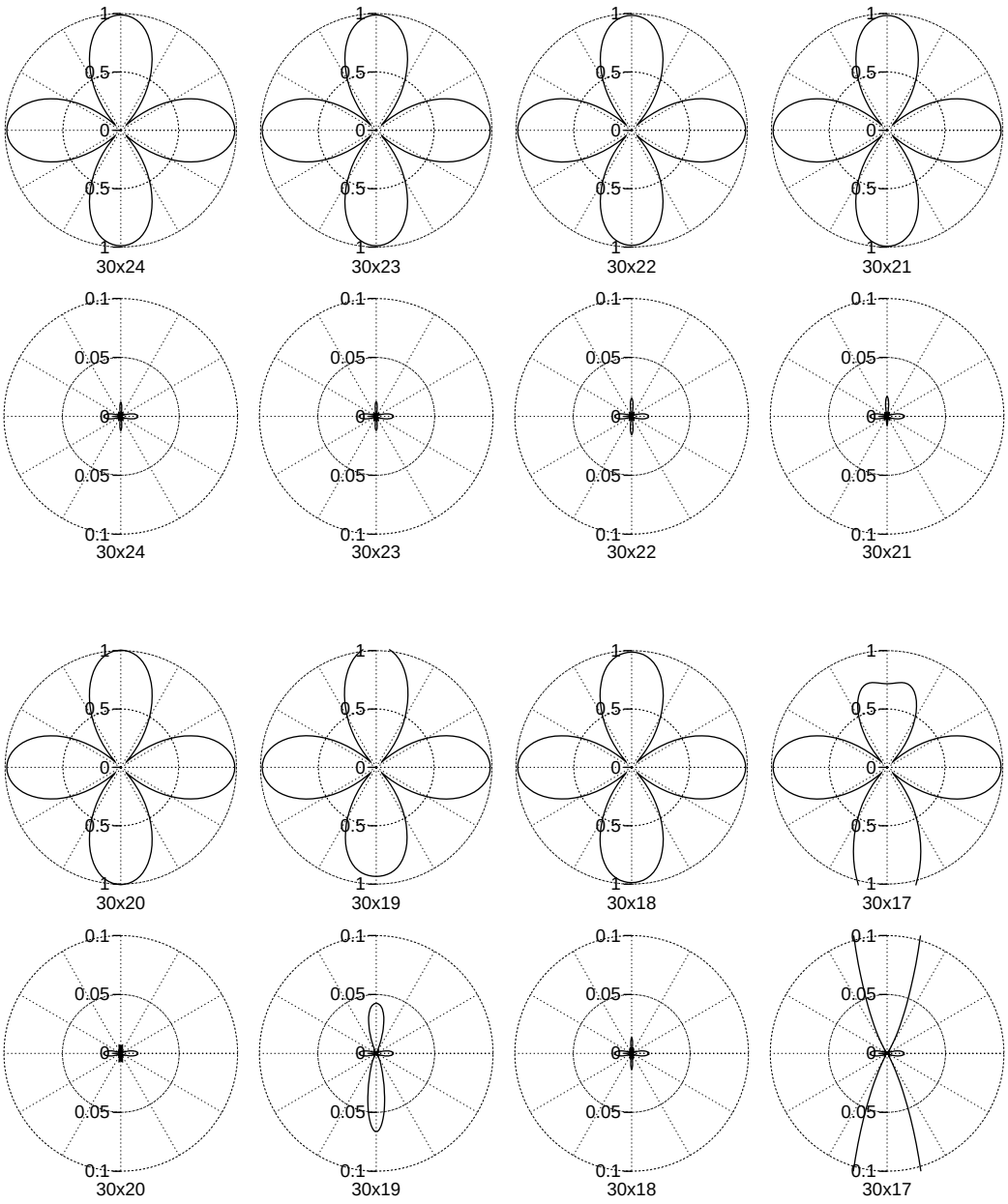


Figure 4.8: Directivity and local error plots of successively larger circumferential grid point spacing for a 343 Hz quadrupole using a spherical grid ($d_l = 0.01\lambda$, $n_t = 60$, $R = 10.3\lambda$, $r_s = 2\lambda$).

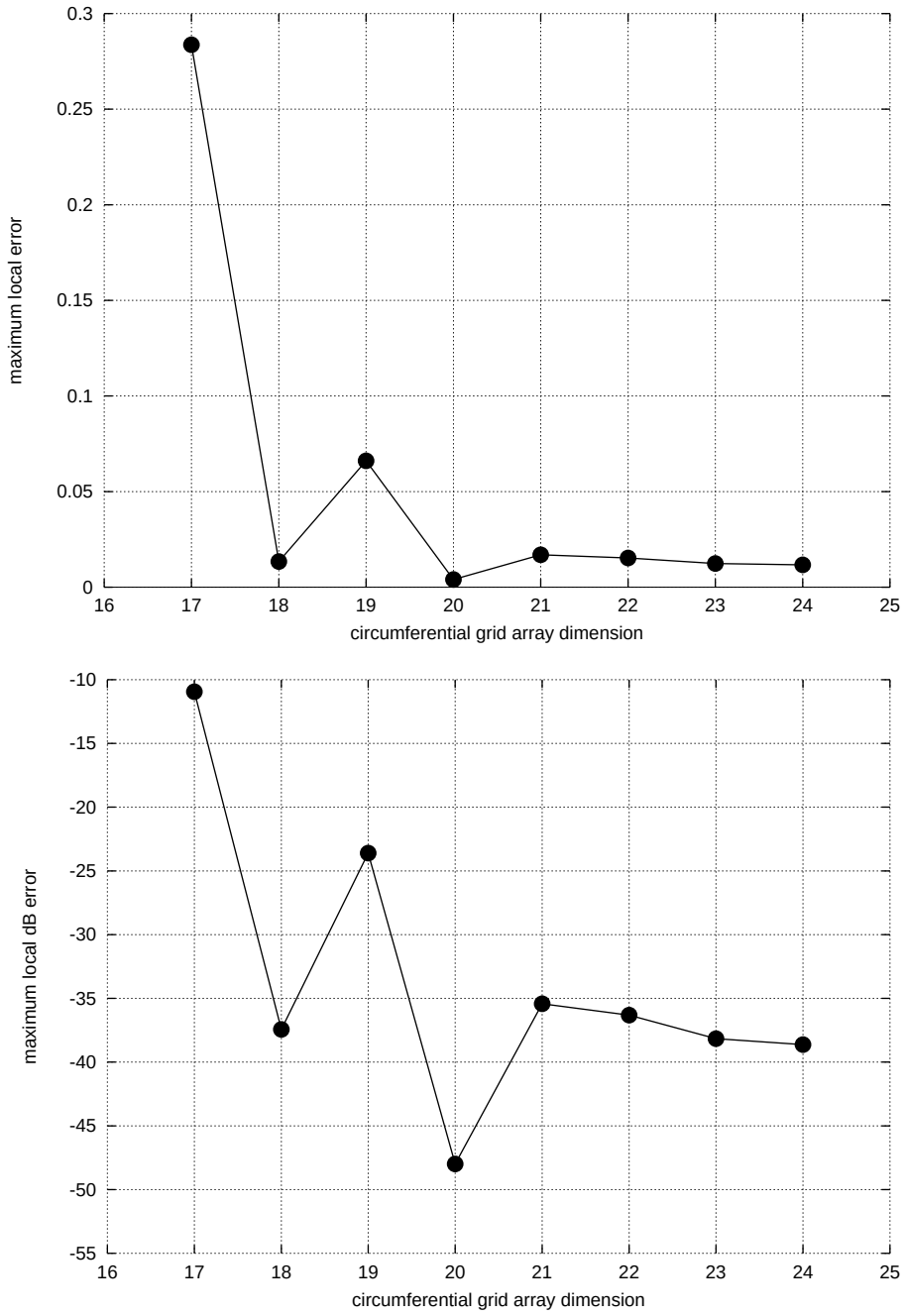


Figure 4.9: Maximum local error for successively larger circumferential grid point spacing for a 343 Hz quadrupole using a spherical grid ($d_l = 0.01\lambda$, $n_t = 60$, $R = 10.3\lambda$, $r_s = 2\lambda$).

Table 4.3: Circumferential grid array dimensions and maximum circumferential grid point spacing.

ncirc	d_{cr}
17	0.74λ
18	0.70λ
19	0.66λ
20	0.63λ
21	0.60λ
22	0.57λ
23	0.55λ
24	0.52λ

The odd circumferential grid array dimensions produce a large error when used with the quadrupole sources placed as they are because the quadrupoles are not axisymmetric with respect to the grid axis. When an odd number of circumferential grid points surrounds a quadrupole, a point is directly above the antinode at the $+y$ axis, but there is no corresponding grid point directly below the antinode at the $-y$ axis. This circumferential spacing of grid points for odd values of `ncirc` causes the extra sensitivity to the non-axisymmetric quadrupole.

4.1.2.2. Cylindrical Grid

Plots of directivity calculated on a cylindrical grid for successively larger circumferential grid spacing by the fully numerical method are shown in figure 4.10 along with plots of the error between the calculations and the field found from a quadrupole by analytical means. Figure 4.11 shows the local error associated with each grid array dimension and figure 4.12 arranges the maximum local error by circumferential grid array dimension.

As with the spherical grid, the maximum circumferential grid distance is calculated by equation (4.7). The same formula is used because the maximum radius of the cylinder or the sphere seen in the y-z plane is the same. The same grid resolution as for the spherical grid, 30x17, is the first cylindrical grid to exceed the 10% local error limit that is used in this thesis. The 30x18 grid, which ensures a local error of less than 10% for the cylindrical

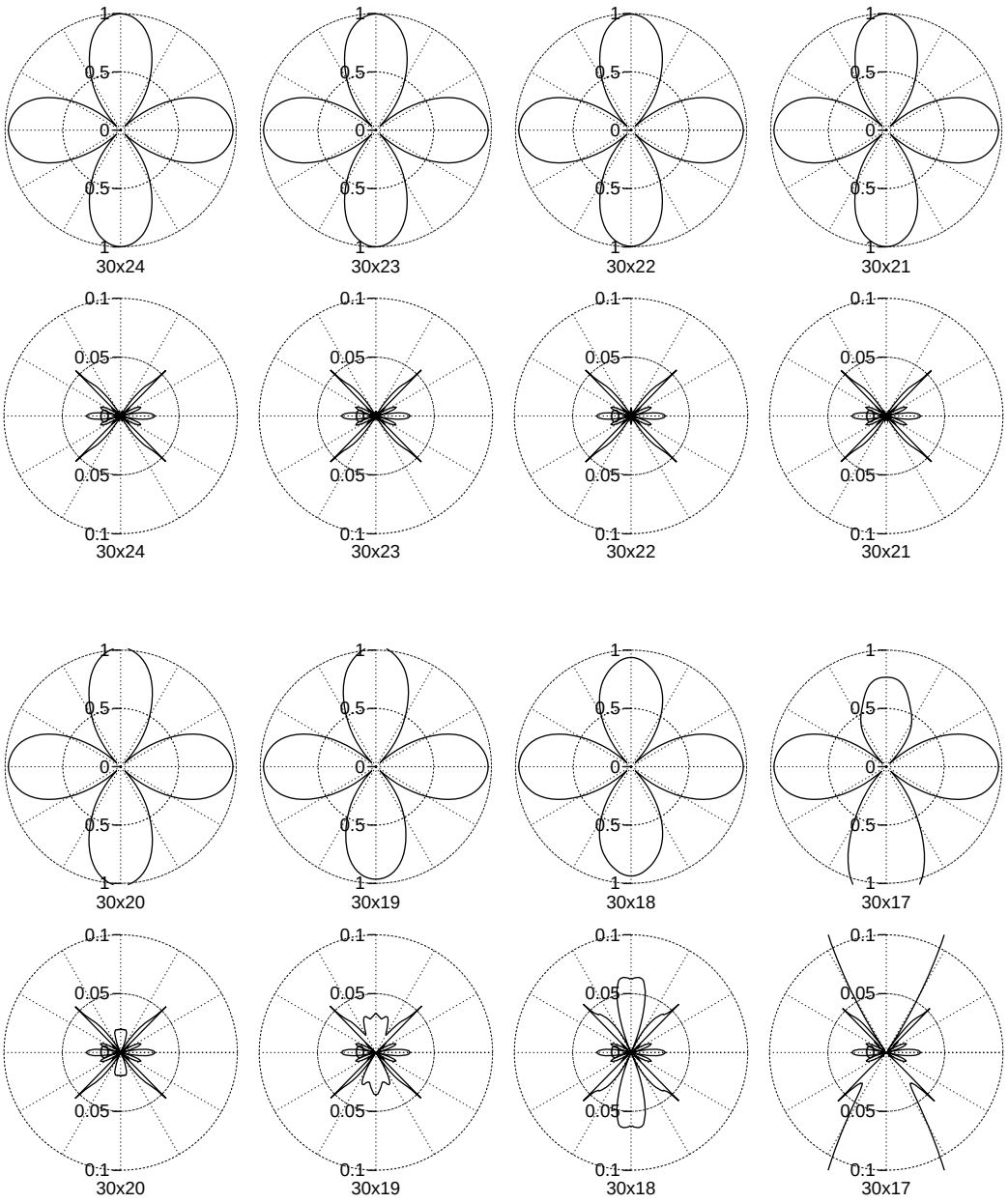


Figure 4.10: Directivity and error plots of successively larger circumferential grid point spacing for a 343 Hz quadrupole using a cylindrical grid ($d_l = 0.01\lambda$, $l_s = 4\lambda$, $n_t = 60$, $R = 10.3\lambda$, $r_c = 2\lambda$).

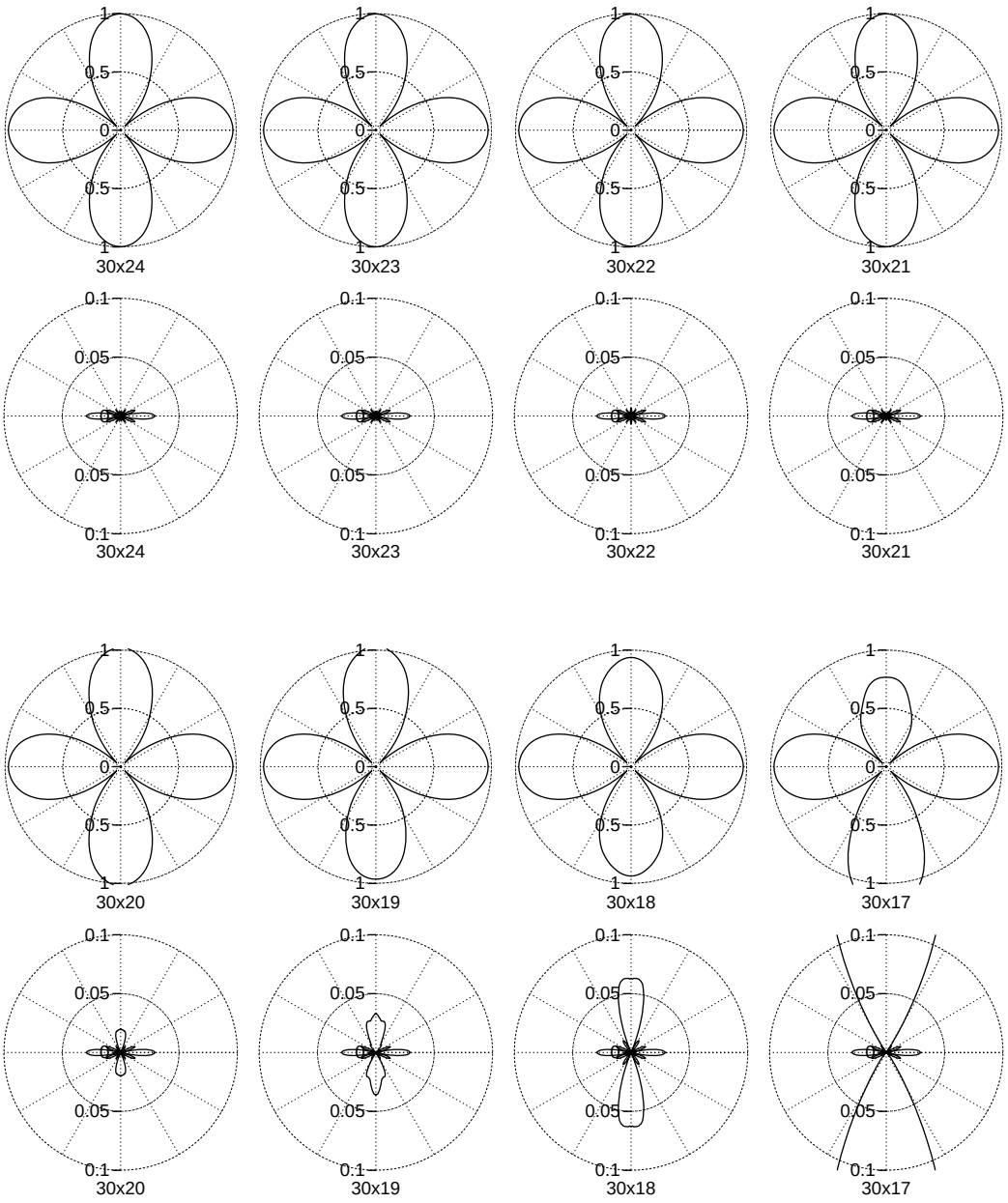


Figure 4.11: Directivity and local error plots of successively larger circumferential grid point spacing for a 343 Hz quadrupole using a cylindrical grid ($d_l = 0.01\lambda$, $l_s = 4\lambda$, $n_t = 60$, $R = 10.3\lambda$, $r_c = 2\lambda$).

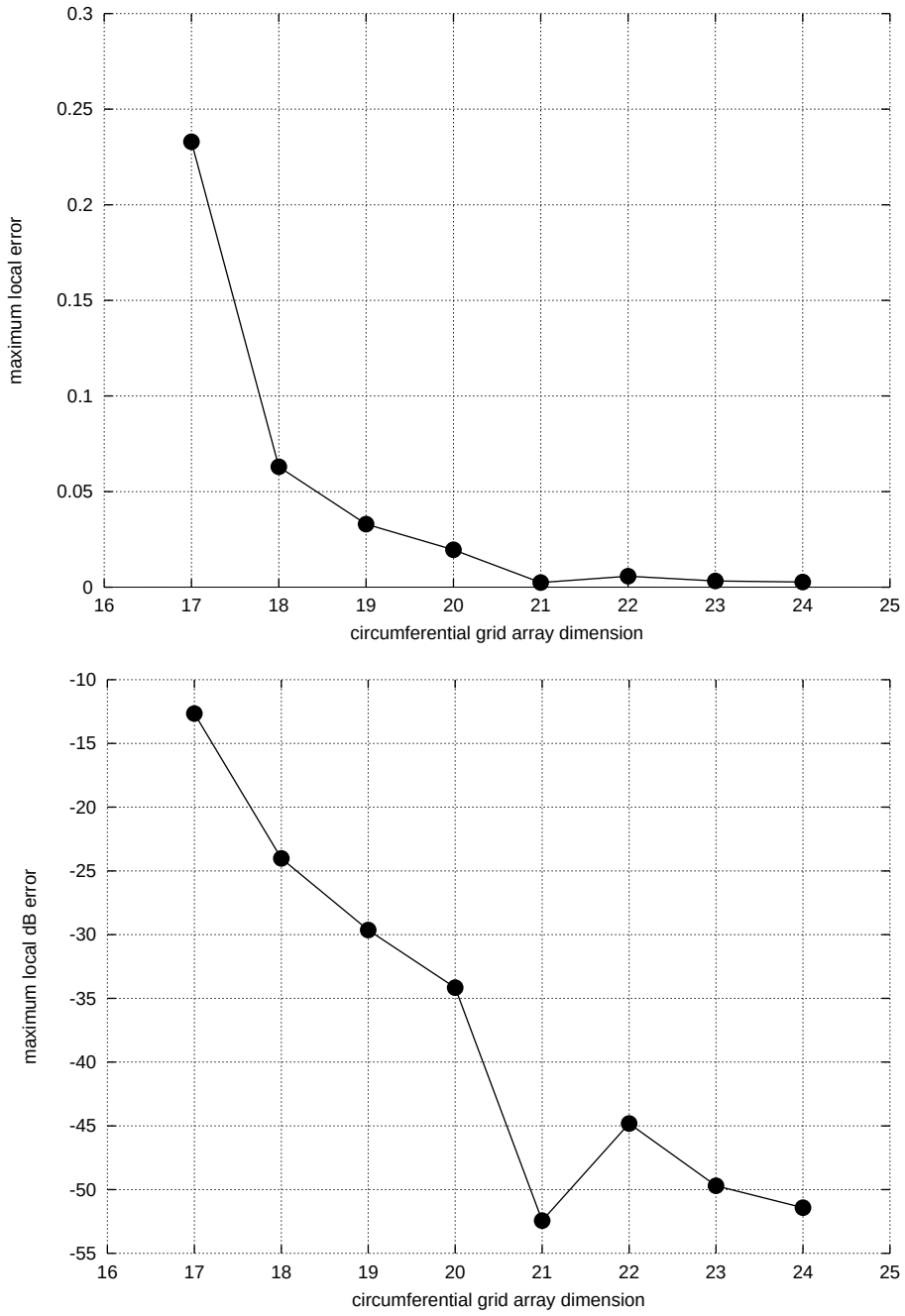


Figure 4.12: Maximum local error plots of successively larger circumferential grid point spacing for a 343 Hz quadrupole using a cylindrical grid ($d_l = 0.01\lambda$, $l_s = 4\lambda$, $n_t = 60$, $R = 10.3\lambda$, $r_c = 2\lambda$).

grid, has a maximum circumferential grid point spacing of $\sim 0.70\lambda$ from table 4.3 which was generated from equation (4.7).

4.1.3. Grid Layer Separation

The separation distance between the layers of a grid has an extreme effect on the pressure calculations. To examine this effect, figure 4.13 shows $\left.\frac{\partial f(x)}{\partial x}\right|_0$ as given by the forward difference scheme shown in equation (2.9) for $f(x) = \sin x$. $\left.\frac{\partial f(x)}{\partial x}\right|_0$ should be 1, which it is as $\Delta x \to 0$, however, as Δx increases even slightly, $\left.\frac{\partial f(x)}{\partial x}\right|_0$ varies considerably.

Figure 4.14 shows how the $\frac{\partial}{\partial \vec{n}}$ formulation in section 2.3 affects the pressure calculations as the separation distance between grid layers increases. The bottom plot is an enlargement of the box shown in the top plot. The effect shown is not surprising because each term in the Kirchhoff equation has a $\frac{\partial}{\partial \vec{n}}$ factor which is based on the $\frac{\partial f}{\partial x}$ formulation shown in equation (2.9).

4.2. Time Sampling

Figure 4.15 shows the far-field pressure around a quadrupole for a range of samples in a period of oscillation and the error between the pressure fields calculated by the fully numerical method and obtained by analytical means from a quadrupole. Figure 4.16 shows the same range of samples in a period and local error plots, and figure 4.17 is a graph of the maximum local error versus the number of samples in a period. Only 15 time samples are needed in one period to give a local error of less than 10% for a spherical grid.

In figure 4.18, The Fourier algorithm is tested by itself for sampling error. As the number of samples in a period of pressure oscillation increases, the acoustic pressure of a 343Hz quadrupole source is calculated more closely to its actual value of 1 Pa along the $+x$ axis. This trend agrees with the sampling error shown in figure 4.15.

4.3. Size of the Computational Grid

Assuming that the grid point spacing is appropriate, there are still limitations on the computational grid. It must not extend beyond the far-field observer point, $\mathbf{x}$ and must be large enough to allow accurate pressure calculations. These size issues are discussed in the

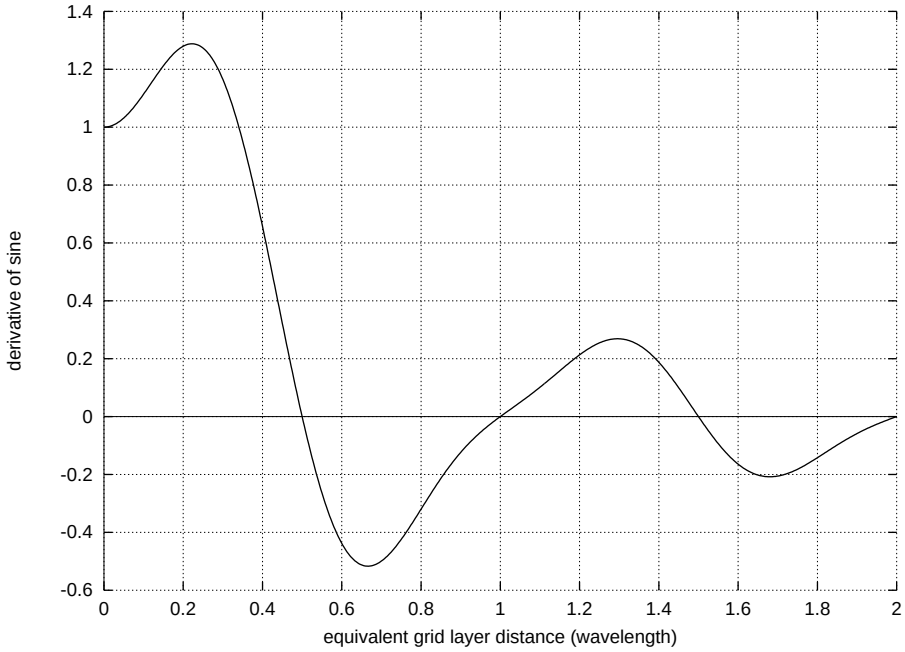


Figure 4.13: Effect of equivalent grid layer distance on the spatial derivative of sine, $\left.\frac{\partial \sin x}{\partial x}\right|_0$.

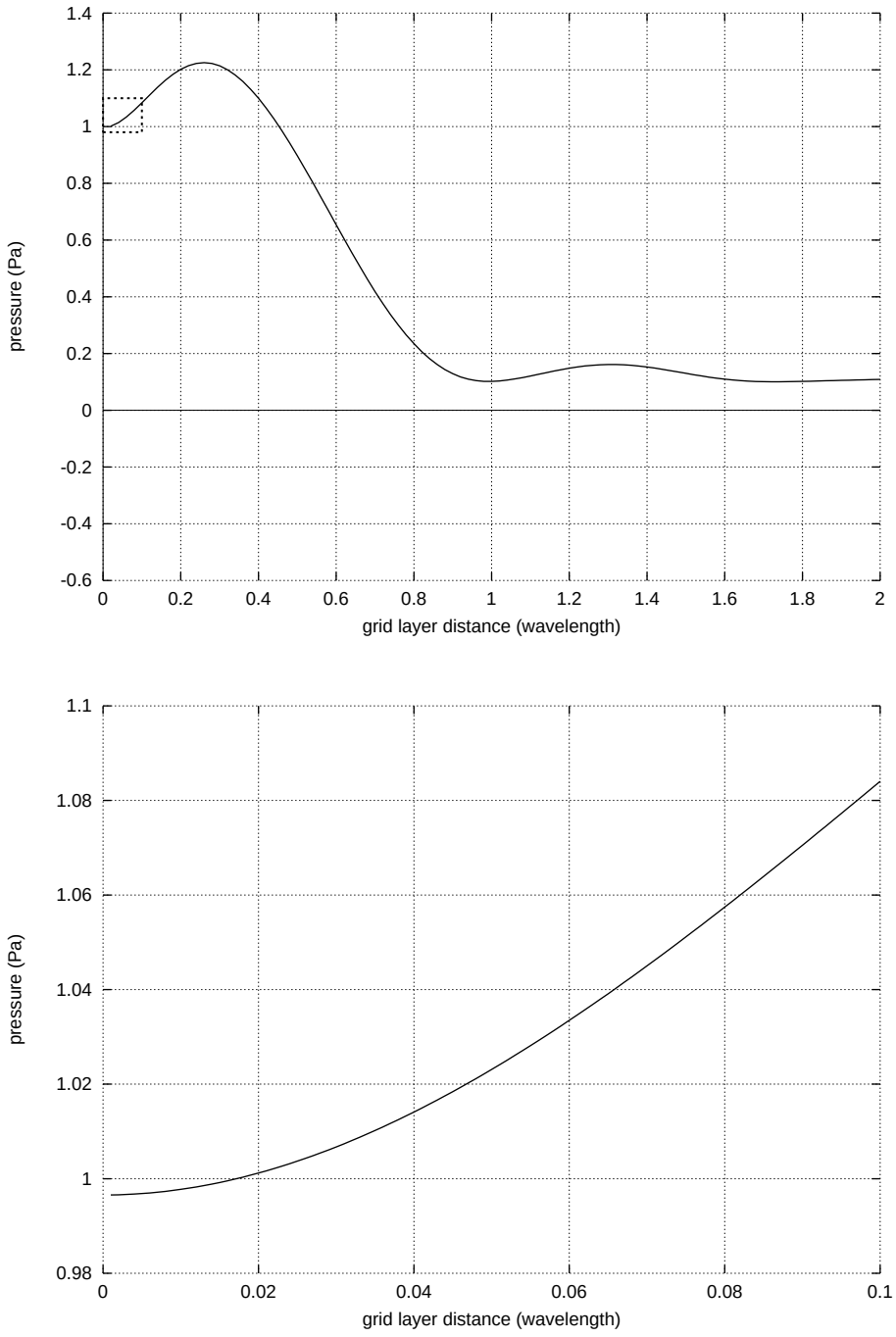


Figure 4.14: Effect of grid layer distance on far-field pressure from the fully numerical formulation. The bottom plot is an enlargement of the dotted box shown in the top plot.

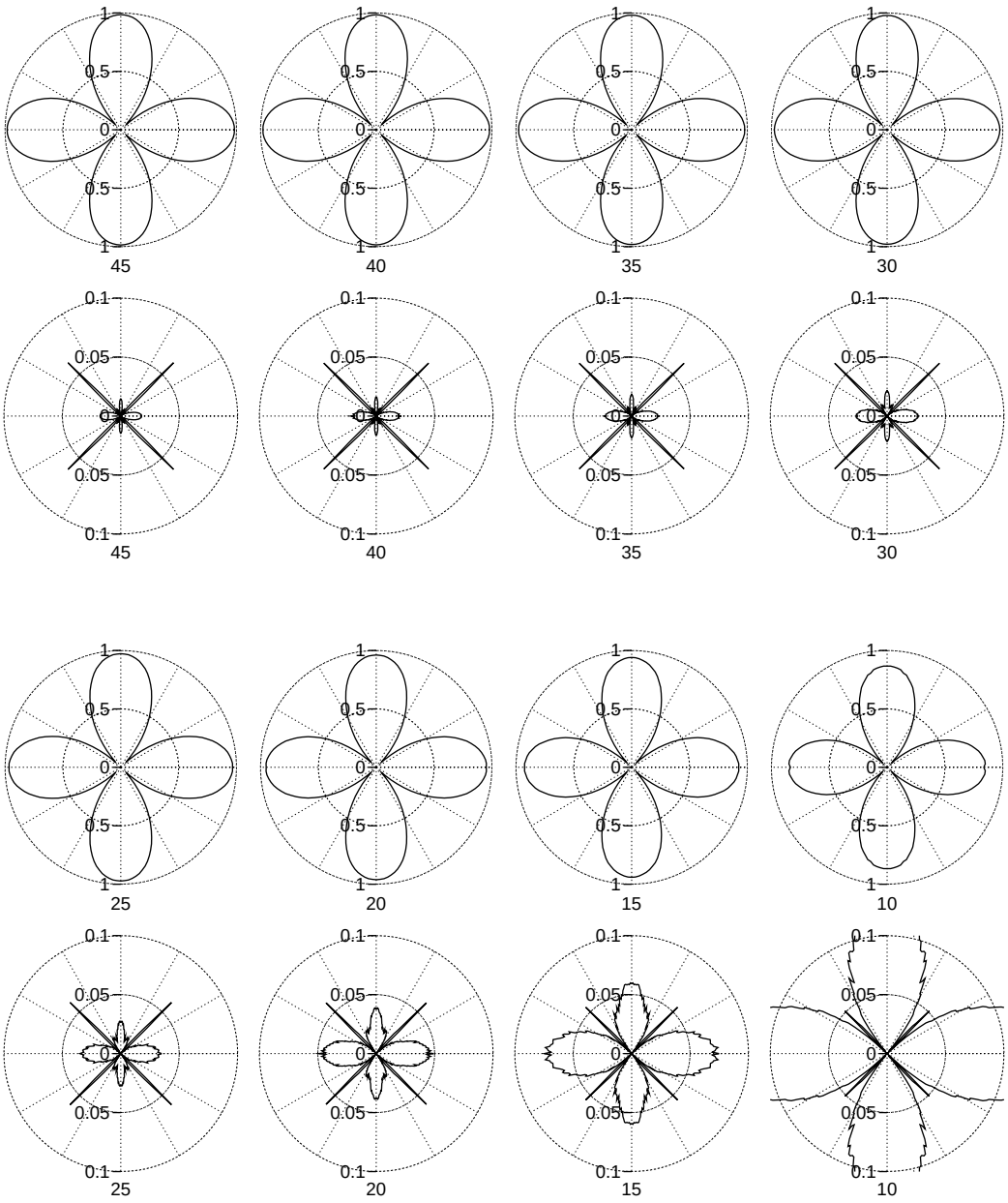


Figure 4.15: Directivity and error plots of successively fewer time samples in a period for a 343 Hz quadrupole using a spherical grid ($d_l = 0.01\lambda$, $n_{ax} = 30$, $n_{cr} = 40$, $R = 10.3\lambda$, $r_s = 2\lambda$). Each plot is labeled with its corresponding number of samples in a period.

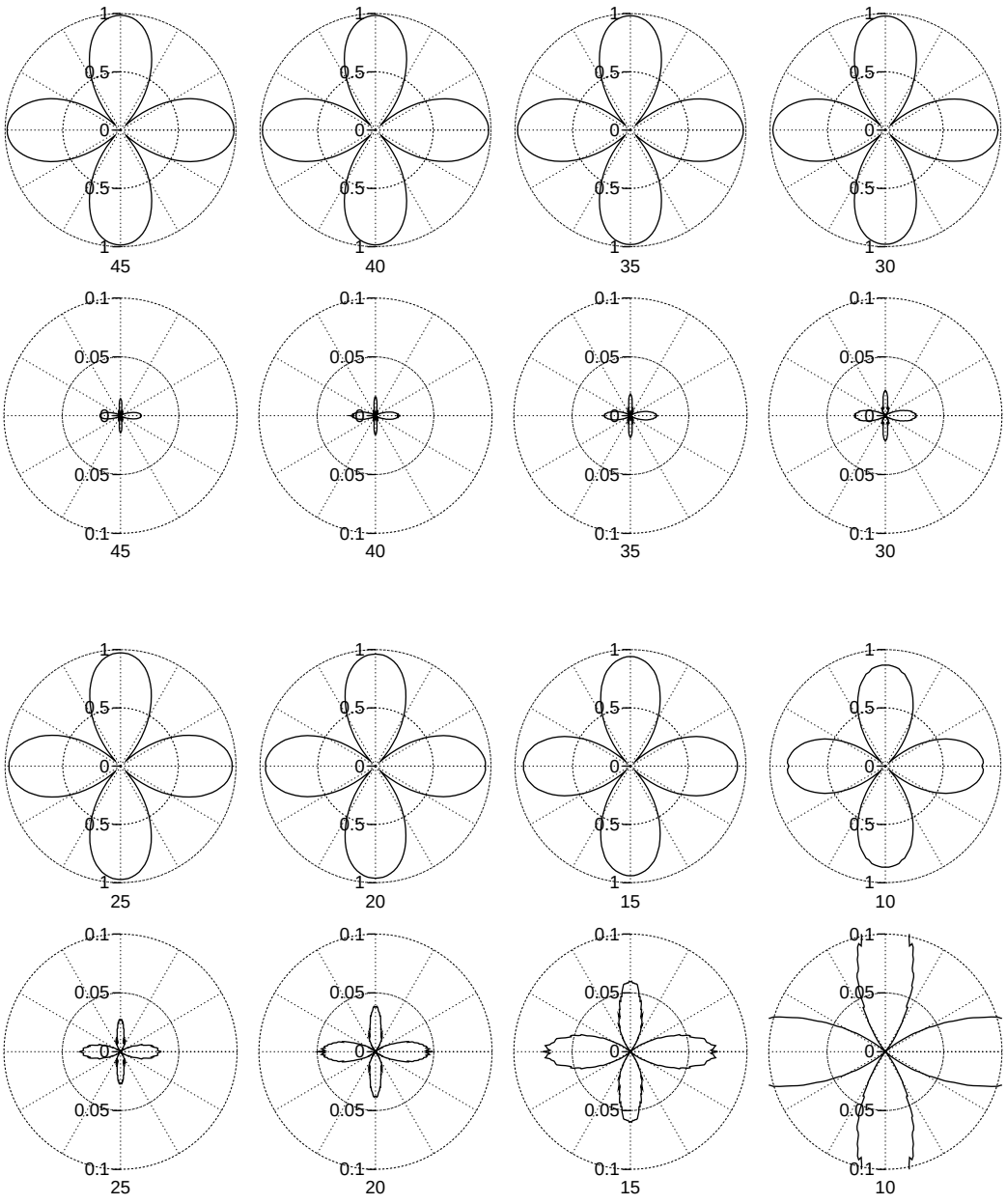


Figure 4.16: Directivity and local error plots of successively fewer time samples in a period for a 343 Hz quadrupole using a spherical grid ($d_l = 0.01\lambda$, $n_{ax} = 30$, $n_{cr} = 40$, $R = 10.3\lambda$, $r_s = 2\lambda$).

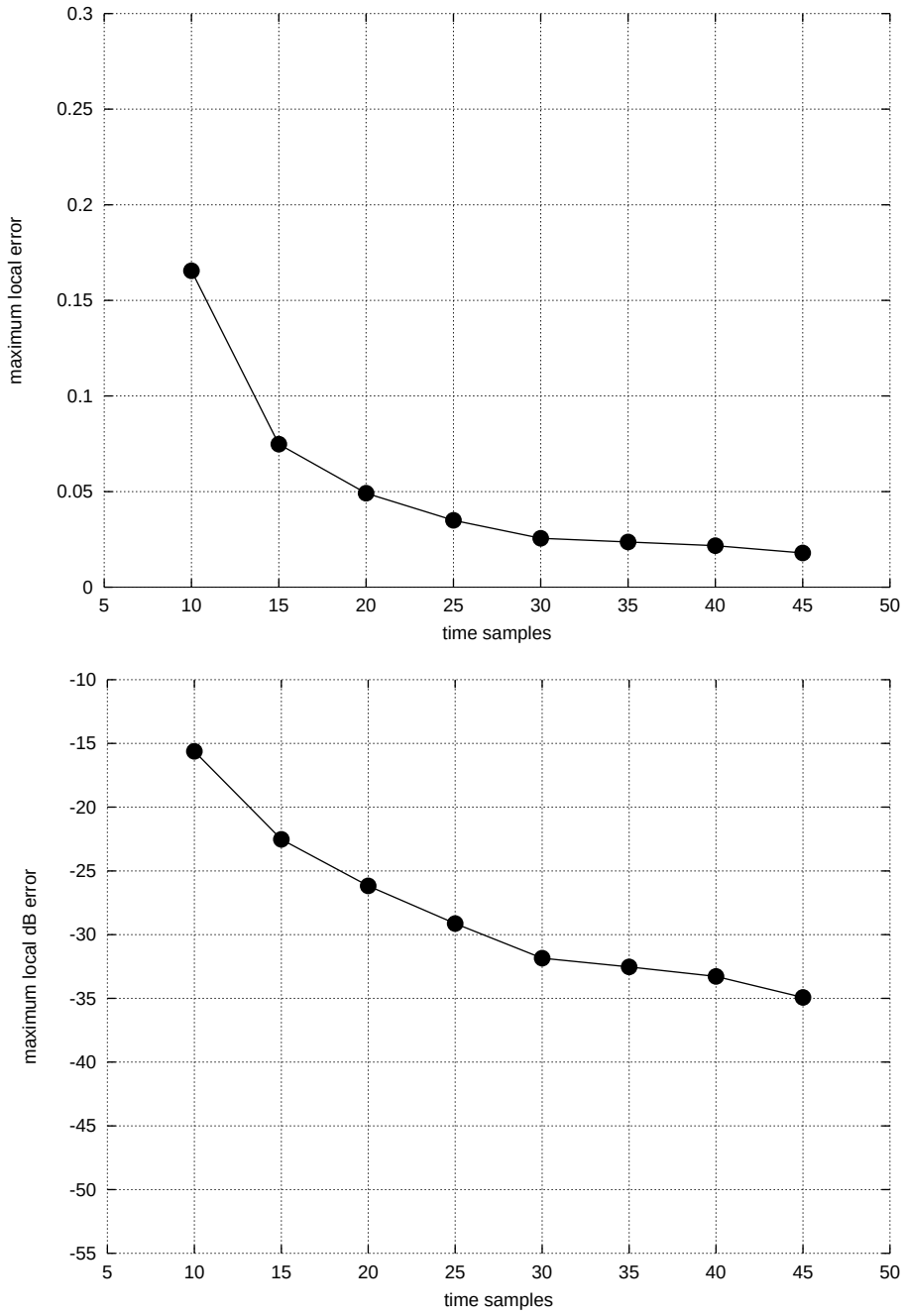


Figure 4.17: Maximum local error plots of successively fewer time samples in a period for a 343 Hz quadrupole using a spherical grid ($d_l = 0.01\lambda$, $n_{ax} = 30$, $n_{cr} = 40$, $R = 10.3\lambda$, $r_s = 2\lambda$).

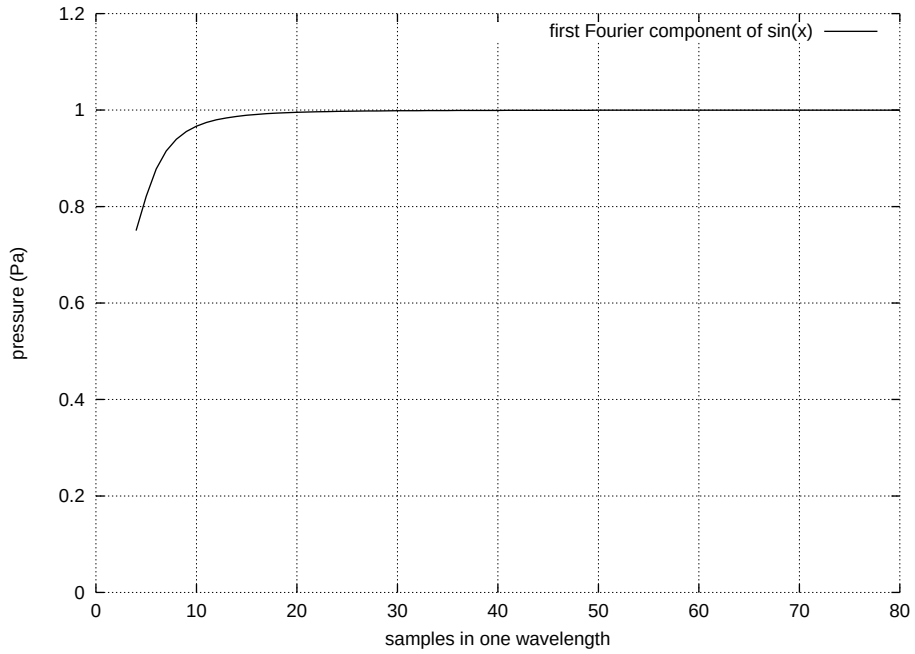


Figure 4.18: Output of the Fourier algorithm versus samples in one oscillation.

next two sections. The analytical-numerical formulation provides the far-field calculations for the grid size plots because grid radius is easy to vary with this formulation.

4.3.1. Maximum Grid Size

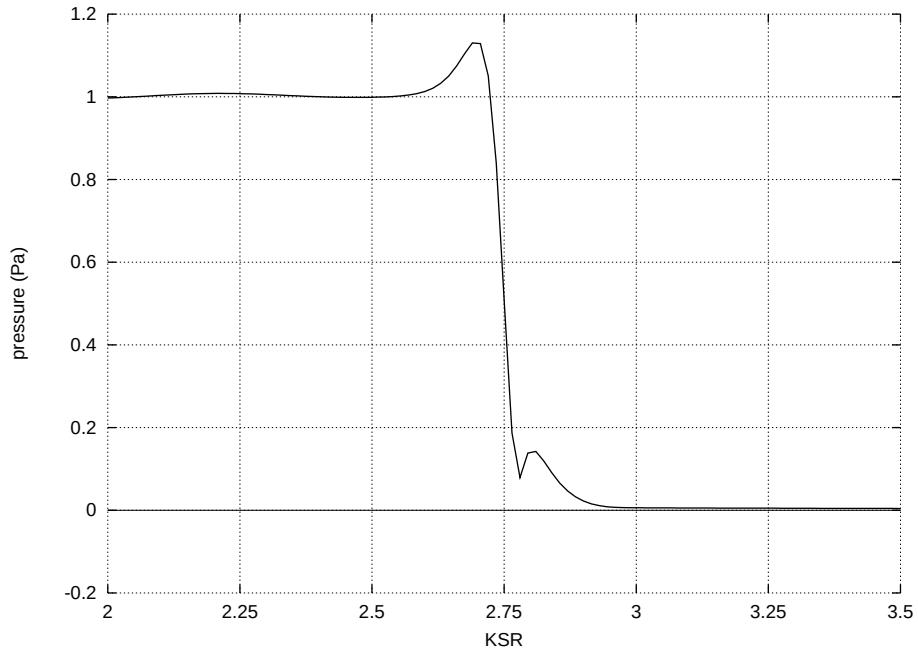
No part of the grid may extend beyond the far-field observer point, **x**. If it does, then the pressure is calculated to be zero. This follows from the Kirchhoff formulation described in section 1.2, because Φ is required to satisfy the wave equation, only outside of the Kirchhoff surface. Figure 4.19.a shows the results when the computational grid extends beyond the far-field distance. A 70x70 grid resolution meets the grid spacing requirements set forth in section 4.1 for a spherical grid with an $r_s = 3.5\lambda$.

4.3.2. Minimum Grid Size

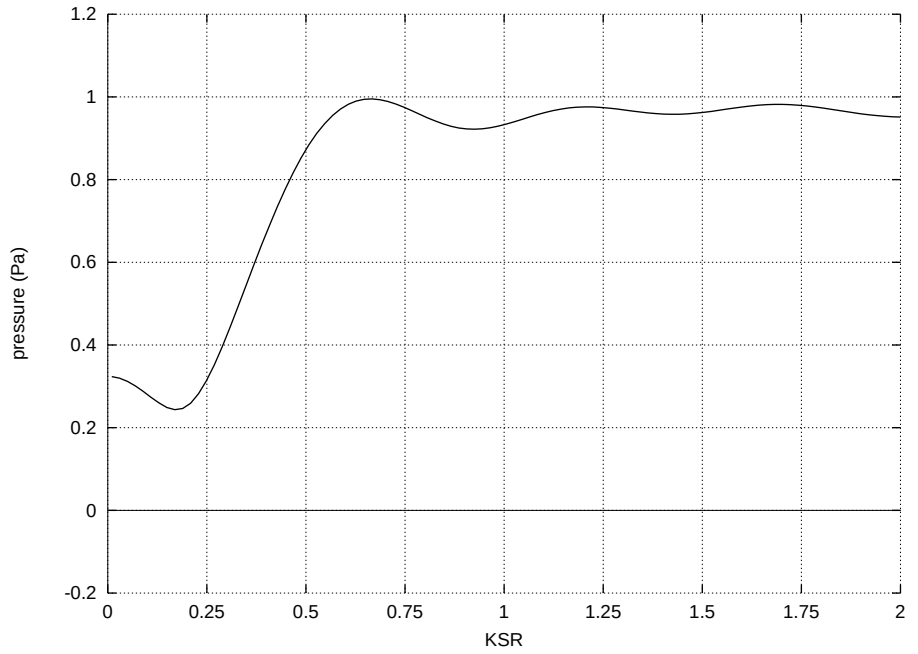
The minimum size of the computational grid is limited by a condition imposed on the Kirchhoff surface; it must surround any nonlinear effects. The Kirchhoff Integral Theorem is based on the linear wave equation and therefore cannot account for nonlinear pressures. The computational Kirchhoff formulation has a geometrical limitation which affects the minimum grid size. Figure 4.19.b shows the behavior of the far-field pressure calculations when the radius of a spherical computational grid approaches 0.

4.4. Grid Shape

The shape of the computational grid can have an effect on the far-field pressure calculations. Figure 4.20 shows a progression of grids that starts with a cylinder and becomes more conical and the difference between the field generated from an analytical quadrupole and the far-field pressure calculations. Figure 4.21 shows the directivity produced by the Kirchhoff method using the grids in figure 4.20 along with the local error directivities. Figure 4.22 shows the maximum local error arranged by inlet end-cap radius.



a. r_s as it extends beyond R $\quad (n_{ax} = 70,\ n_{cr} = 70,\ n_t = 80,\ R = 2.75\lambda)$. KSR is $\frac{r_s}{\lambda}$.



b. Pressure predictions when r_s is near 0 $\quad (n_{ax} = 30,\ n_{cr} = 40,\ n_t = 60,\ R = 10.3\lambda)$.

Figure 4.19: Pressure calculations showing the limits of r_s. Predictions are made along the $+y$ axis using analytical terms for a quadrupole and a spherical grid.

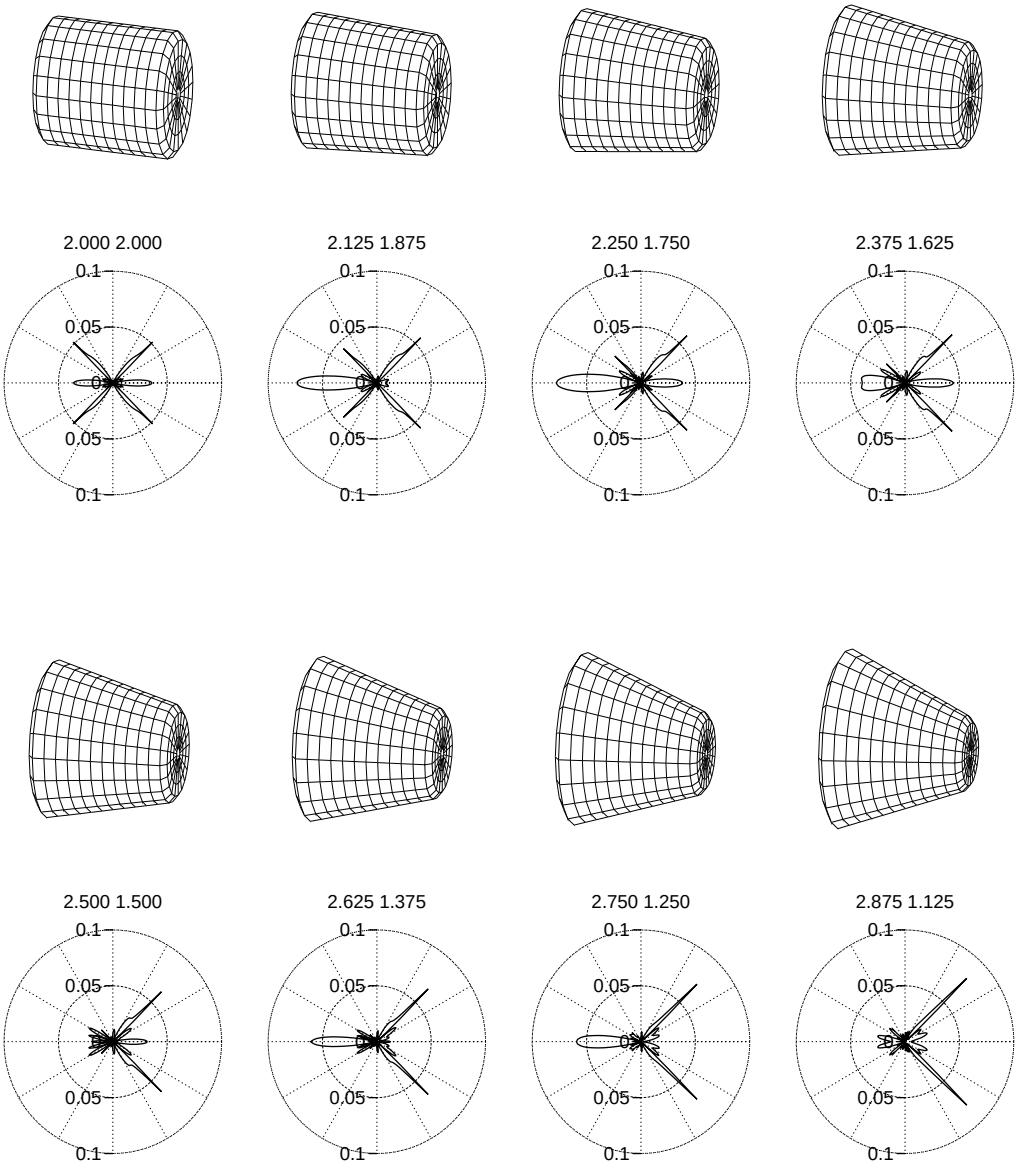


Figure 4.20: Grids and error plots for a succession of grids which become more conical ($d_l = 0.01\lambda$, $n_{ax} = 46$, $n_{cr} = 64$, $n_t = 60$, $R = 10.3\lambda$). Each grid and error plot is shown with corresponding r_{otl} and r_{inl} in wavelength.

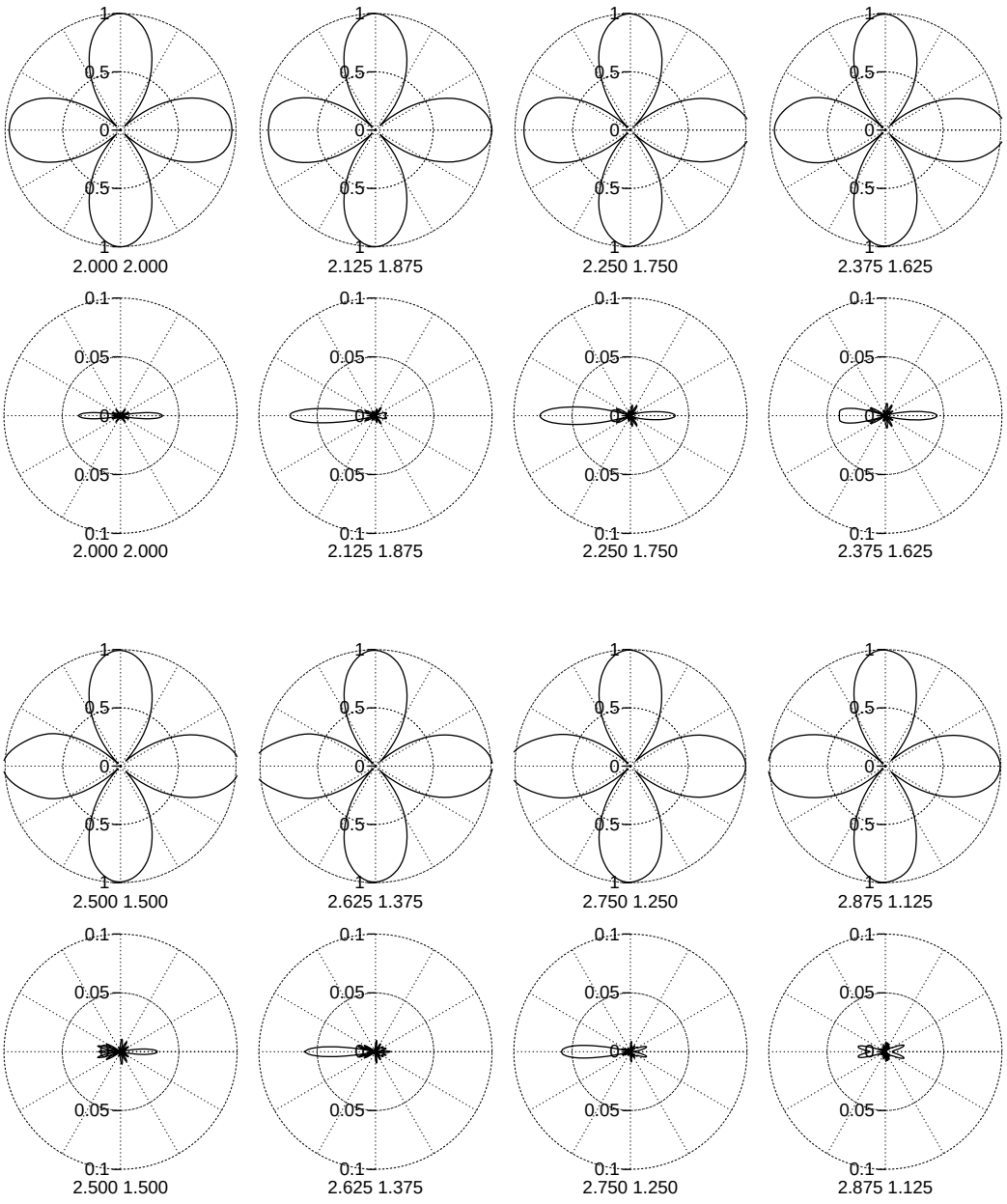


Figure 4.21: Directivity and local error plots for a succession of grids which become more conical ($d_l = 0.01\lambda$, $n_{ax} = 46$, $n_{cr} = 64$, $n_t = 60$, $R = 10.3\lambda$).

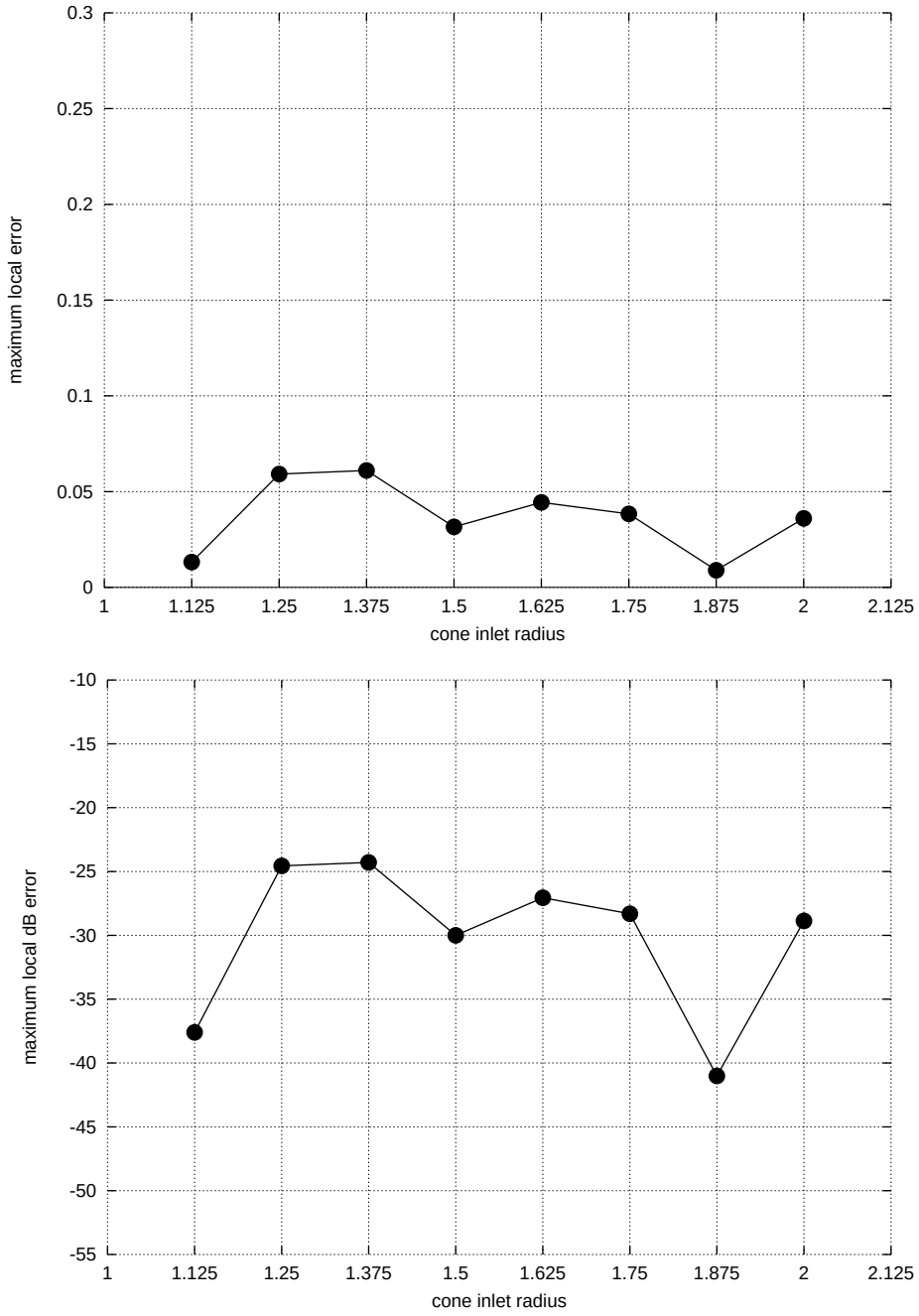


Figure 4.22: Maximum local error for a succession of grids which become more conical ($d_l = 0.01\lambda$, $n_{ax} = 46$, $n_{cr} = 64$, $n_t = 60$, $R = 10.3\lambda$).

4.5. Summary

The computational grid is one of the most important elements of a working computational Kirchhoff method. It must meet requirements based on grid point spacing, grid layer separation, and grid size and shape. Discretization in time is also an important element. The grid point spacing depends separately upon the number of axial and circumferential grid points. The shape of the grid can affect this dependence. The grid layer separation has a strong effect on the far-field pressure calculations so it is both beneficial because it allows the $\frac{\partial p}{\partial \vec{n}}$ to be calculated from only pressure values, and detrimental because it can cause such significant error. The sampling of the oscillations of the source at all of the points on the grid controls what frequencies can be calculated by the Fourier algorithm. The size of the grid is limited by the specifications of the Kirchhoff Integral Theorem. The grid shape can have an effect on the error of the far-field pressure calculations. The minimum values for these requirements to maintain a local error of 10% are presented in this chapter.

Chapter 5.

CONCLUSIONS AND RECOMMENDATIONS

5.1. Conclusions

Despite the fact that the Kirchhoff Integral Theorem was formulated over 100 years ago, it is only with the aid of modern computing resources that it has become useful for calculating sound from complex sources. When using the theorem to calculate pressures from sources with no known far-field pressure expression, the numerical techniques must be trusted for validity as well as accuracy. This thesis presents enough information about the sensitivity of a computational Kirchhoff method to ensure accurate results from the method.

A Kirchhoff surface is represented by a computational grid so that the pressure values at grid points are known analytically or are calculated numerically. The Kirchhoff terms are integrated over the grid to find far-field pressures. These pressures are found at many points in space and time to simulate a continuous pressure history in a plane. This method provides a technique of numerically determining the far-field pressure field for a complex source.

The validation of the computational formulations of the Kirchhoff Integral Theorem shows that some error is unavoidable, but depending on the application this error can be controlled to produce results of the desired accuracy. This method, therefore, is useful in predicting the sound radiation from a source that is otherwise difficult to determine. Proper specification of parameters is imperative in order to achieve accurate results, as shown by the sensitivity of the calculations to certain grid and time parameters. The research conducted indicates circumstances in which grid array dimensions or time sampling can be reduced in order to save on computational resources.

The grid layer separation distance, shown to affect the calculations, is the strictest requirement which must be fulfilled to achieve accurate calculations. A small change in the separation distance produces a large error in the pressure calculations. This requirement is significant for generating grids from computational fluid dynamics simulations because it specifies a dense computational fluid dynamics grid at a distance away from the source.

5.2. Recommendations for Computational Fluid Dynamicists

The information provided by this research can aid in the generation of computational grids for use in the Kirchhoff Integral Theorem from computational fluid dynamics simulations. By looking at the trends shown for grid point spacing in the circumferential and axial directions, an appropriate number of grid points can be specified so error levels are below desired levels and the time for computational work is not longer than needed because of unnecessarily high point density.

Small changes in the grid layer separation distance cause significant error, so it is important to specify a fine computational fluid dynamics grid in the area that will be used for the Kirchhoff surface. The region for the Kirchhoff surface has to be far enough away from the source to surround nonlinear effects, so the computational fluid dynamics grid is usually coarse at this distance. The different density needs of these grids has to be resolved to achieve useful results from the Kirchhoff Integral Theorem.

The shape of the grid can affect the error in pressure calculations from the Kirchhoff Integral Theorem. Meadows and Atkins[7] discuss some issues with irregular grid shapes.

5.3. Recommendations for Further Research

Furthering research of the Kirchhoff method for turbomachinery simulations can allow turbomachine manufacturers to build quieter engines at a lower production cost. This could provide an increase in tractable land near airports due to reduced noise pollution.

There are many characteristics of turbomachinery which could be added to the Kirchhoff method used in this study which would provide more accurate noise calculations for turbomachinery. Lyrintzis[6] provides steps for adding subsonic and supersonic flow to the classical Kirchhoff formula. The addition of flow has been derived by Morgans[9] and Farassat and Myers.[2] Morgans formulation was questioned by Ffowcs-Williams and Hawkings.[3] Because turbomachinery has rotating parts, geometries and surfaces which move along with the blades of the turbomachine could also be added.

Several areas could also be examined which are outside of the scope of this thesis. Nonperiodic time histories could be used with a different formulation of the $\frac{\partial p}{\partial t}$ term. A method of specifying point placement on the grid so point density stays relatively constant over the grid surface would aid in determining error based on density rather than axial and

circumferential point spacing. Meadows and Atkins[7] discuss the effects of increasing the order of the finite difference schemes used to find the $\frac{\partial p}{\partial t}$ and $\frac{\partial p}{\partial \vec{n}}$ terms.

A variation in pressure calculated by the Kirchhoff Integral Method can be seen in figure 4.19.b as the Kirchhoff surface radius varies for a spherical grid. A similar variation in pressure occurs as the observer point changes in distance from the source. The cause of these unexpected variations in calculated pressure could be important and should be researched.

Another project for future research is a full simulation of a real turbomachine which has been measured experimentally. The Kirchhoff method could be linked to the pressure calculations of a computational fluid dynamics simulation to calculate the far-field pressure. The computational results could be compared to experimental results for a real-world validation of a computational version of Kirchhoff's method.

Since the computational Kirchhoff method provides pressure histories at far-field points, it is possible to use sound manipulation tools on computers to repeat the periods generated and actually listen to the sound that a source makes. The sources used in this thesis would emit single tones or combinations of single tones, while a source with a more complex set of pressure histories, such as the output of a computational fluid dynamics simulation, would produce a broader range of frequencies. Because the sound created by the simulation is calculated at different directivity angles, it would be possible to put on headphones and experience "walking around" the simulated turbomachine.

REFERENCES

1. W. H. Beyer, ed., *CRC Standard Mathematical Tables*, (CRC Press, Inc., 1987), 28th ed.
2. F. Farassat and M. K. Myers, "Extension of Kirchhoff's Formula to Radiation from Moving Surfaces," Journal of Sound and Vibration, **123**(3):451–460 (1988).
3. J. E. Ffowcs-Williams and D. L. Hawkings, "Sound Generation by Turbulence and Surfaces in Arbitrary Motion," Philosophical Transactions of The Royal Society, **246A**:321–342 (1969).
4. G. R. Kirchhoff, "Zur Theorie der Lichtstrahlen," Annalen der Physik und Chemie, **18**:663–695 (1883).
5. E. Kreyszig, *Advanced Engineering Mathematics*, (John Wiley & Sons, Inc., 1988), 6th ed.
6. A. S. Lyrintzis, "Review: The Use of Kirchhoff's Method in Computational Aeroacoustics," Journal of Fluids Engineering **116**, pp. 665–675 (1994).
7. K. R. Meadows and H. L. Atkins, "Towards a Highly Accurate Implementation of the Kirchhoff Approach for Computational Aeroacoustics," Journal of Computational Acoustics **4**(2):225–241 (June 1996).
8. A. Mizrahi and M. Sullivan, *Calculus and Analytic Geometry*, (Wadsworth Publishing Company, 1986), 2nd ed.
9. R. P. Morgans, "The Kirchhoff Formula Extended to a Moving Surface," Philosophical Magazine, **9**(7)(55):141-161 (1930).
10. L. Morino, B. K. Bharadvaj, M. I. Freedman, and K. Tseng, "BEM for Wave Equation With Boundary in Arbitrary Motion and Applications to Compressible Potential Aerodynamics of Aeroplanes and Helicopters," *Advanced Boundary Element Methods*, T. A. Cruse, ed., (Springer-Verlag, 1988).
11. A. D. Pierce, *Acoustics: An Introduction to Its Physical Principles and Applications*, (Acoustical Society of America, 1989).
12. P. R. Prentice, "The Acoustic Ring Source and Its Application to Propeller Noise," Proc. R. Soc. Lond. A **437**, pp. 629–644 (1992).
13. J. A. Stratton, *Electromagnetic Theory*, (McGraw-Hill, 1941).

Appendix A.

SOURCE

A.1. Grid Program

A.1.1. Grid.const

```
* ----------------------------------------------------------------------
*
*   Grid.const
*
*   quest
*   nodog
*   05may97
*
*   RCS info
*   $Id: Grid.const,v 1.1 1997/05/07 03:57:26 nodog Exp nodog $
*
*   Grid.const holds several constants for Grid.f
*
* ----------------------------------------------------------------------

*     gridshape constants
      INTEGER sphere, cylinder
      PARAMETER ( sphere = 1, cylinder = 2 )

* ----------------------------------------------------------------------
```

A.1.2. Grid.rc

```
* ----------------------------------------------------------------------
* Grid.rc
* is read by gridcyl.f to get many of its constant values.

* gridshape: [INTEGER] 1=sphere where radius=radinl  2=cylinder
1

* dstinl: [DOUBLE PRECISION] distance of the inlet plane from origin
2.0

* dstotl: [DOUBLE PRECISION] distance of the outlet plane from origin
-2.0

* dstgrdlvl: [DOUBLE PRECISION] distance between grid levels
0.01

* radinl: [DOUBLE PRECISION] radius of the inlet plane circle
```

```
2.0

* radotl: [DOUBLE PRECISION] radius of the outlet plane circle
2.0

* nptinlrad: [INTEGER] number of points on an inlet radius
11

* nptotlrad: [INTEGER] number of points on an outlet radius
11

* beautgrid: [LOGICAL] beautify the grid for gnuplot
FALSE

* ---------------------------------------------------------------------
```

A.1.3. Grid.f

```
* =====================================================================
*
      PROGRAM Grid
*
* ================== Prologue ========================================
*
* $Id: Grid.f,v 1.8 1997/06/09 04:48:33 nodog Exp nodog $
*
* Original program written by
* Anderson Mills <nodog@sabine.acs.psu.edu>
* at the Graduate Program in Acoustics
* at Pennsylvania State University
*
* Purpose:
*   Grid generates a grid (supposedly around a
*   turbomachine) which will be used by a pressure generating
*   program and a Kirchhoff surface solver to predict far field
*   noise.
*
* Processing:
*   Parameters from Kirch3.dmn, Grid.const, and Grid.rc are
*   used to generate the grid.  Output is stored in the file
*   Kirch3.grid.
*
* Special requirements:
*   Just that Kirch3.dmn, Grid.const, and Grid.rc exist.
*
* ------------------ File Numbers -----------------------------------

*    20   Grid.rc
*    29   Kirch3.grid

* ------------------ Constant declarations --------------------------

      INCLUDE 'Grid.const'
```

```
* ------------------ Argument declarations -----------------------------

*     Constants from Grid.rc --- see file for description
      INTEGER gridshape, nptinlrad, nptotlrad
      DOUBLE PRECISION dstinl, dstotl, dstgrdlvl, radinl, radotl
      LOGICAL beautgrid

* ------------------ Code ----------------------------------------------

* Main Program

      CALL Banner()

*     Read in constants from Grid.rc --------------------------------
      CALL GetGrConst (gridshape, dstinl, dstotl, dstgrdlvl, radinl,
     &   radotl, nptinlrad, nptotlrad, beautgrid)

*     Based on the value of gridshape, call the correct grid subroutine

      IF ( gridshape .EQ. sphere ) THEN

         CALL GridSph (dstinl, dstotl, dstgrdlvl, radinl, radotl,
     &      nptinlrad, nptotlrad, beautgrid)

      ELSEIF ( gridshape .EQ. cylinder ) THEN

         CALL GridCyl (dstinl, dstotl, dstgrdlvl, radinl, radotl,
     &      nptinlrad, nptotlrad, beautgrid)

      ELSE

         WRITE ( *, * ) 'Grid shape is undefined.'

      ENDIF

      CALL Banner()

      STOP
      END

* Main Program

* ====================================================================


* ====================================================================
*
      SUBROUTINE Banner()
*
* ================== Prologue ========================================
*
* Purpose: Show the beginning and ending of the program.
*
* ------------------ Code ----------------------------------------------
```

```
      WRITE (*, *) '                         '
      WRITE (*, *) '  88888                  '
      WRITE (*, *) ' 8     8 88888  8 88888  '
      WRITE (*, *) ' 8       8    8 8 8    8 '
      WRITE (*, *) ' 8  8888 8    8 8 8    8 '
      WRITE (*, *) ' 8     8 88888  8 8    8 '
      WRITE (*, *) ' 8     8 8   8  8 8    8 '
      WRITE (*, *) '  88888  8    8 8 88888  '
      WRITE (*, *) '                         '


      RETURN
      END

* ====================================================================


* ====================================================================
*
      SUBROUTINE GetGrConst (gridshape, dstinl, dstotl, dstgrdlvl,
     &   radinl, radotl, nptinlrad, nptotlrad, beautgrid)
*
* ================== Prologue ========================================
*
* Purpose:
*    To read in the constants from the Grid.rc file
*
* Processing:
*    Uses READ's to get the info in.
*
* Special requirements:
*    Gotta have something to read, so the Grid.rc file must exist.
*
* ------------------ Argument declarations ----------------------------

*     Constants from Grid.rc --- see file for description
      INTEGER gridshape, nptinlrad, nptotlrad
      DOUBLE PRECISION dstinl, dstotl, dstgrdlvl, radinl, radotl
      LOGICAL beautgrid

* ------------------ Local declarations -------------------------------

      CHARACTER Dummy

* ------------------ Code ---------------------------------------------

      OPEN (UNIT=20, FILE='Grid.rc', STATUS='UNKNOWN')

*     Three lines of comments at the top.
      READ (20, 1000) Dummy
      READ (20, 1000) Dummy
      READ (20, 1000) Dummy

*     Read constants from Grid.rc
```

```
*     One blank line, one label line, then the constant.
      READ (20, 1000) Dummy
      READ (20, 1000) Dummy
      READ (20, 1010) gridshape
      READ (20, 1000) Dummy
      READ (20, 1000) Dummy
      READ (20, 1020) dstinl
      READ (20, 1000) Dummy
      READ (20, 1000) Dummy
      READ (20, 1020) dstotl
      READ (20, 1000) Dummy
      READ (20, 1000) Dummy
      READ (20, 1020) dstgrdlvl
      READ (20, 1000) Dummy
      READ (20, 1000) Dummy
      READ (20, 1020) radinl
      READ (20, 1000) Dummy
      READ (20, 1000) Dummy
      READ (20, 1020) radotl
      READ (20, 1000) Dummy
      READ (20, 1000) Dummy
      READ (20, 1010) nptinlrad
      READ (20, 1000) Dummy
      READ (20, 1000) Dummy
      READ (20, 1010) nptotlrad
      READ (20, 1000) Dummy
      READ (20, 1000) Dummy
      READ (20, 1030) beautgrid


      CLOSE (20)

 1000 FORMAT (A)
 1010 FORMAT (I10)
 1020 FORMAT (F24.17)
 1030 FORMAT (L5)

      RETURN
      END

* ====================================================================


* ====================================================================
*
      SUBROUTINE GridSph (dstinl, dstotl, dstgrdlvl, radinl, radotl,
     &   nptinlrad, nptotlrad, beautgrid)
*
* ================== Prologue ========================================
*
* Purpose: Generate a spherical grid and output it to file.
*
* Processing: Basically run through the array of the grid and figure
*    out the X, Y, and Z coords of each point and output that to file.
*
```

```
* Special requirements:
*
* ------------------ Include files ------------------------------------

*     ncirc: [INTEGER] Number of points around the cylindrical grid?
*     naxis: [INTEGER] Number of axial points in the grid?
*     ntimes: [INTEGER] The number of timesteps?
*     nlevels: [INTEGER] Number of grid levels?
      INCLUDE 'Kirch3.dmn'

* ------------------ Argument declarations ----------------------------

*     Constants from Grid.rc --- see file for description
      INTEGER nptinlrad, nptotlrad
      DOUBLE PRECISION dstinl, dstotl, dstgrdlvl, radinl, radotl
      LOGICAL beautgrid

* ------------------ Local declarations -------------------------------

      DOUBLE PRECISION pi

*     Axidex: Index in the axial direction.
*     Cirdex: Index in the circuferenctial direction.
*     Lvldex: Level index
      INTEGER Lvldex, Cirdex, Axidex

*     Sigma: Angle around X axis from Y towards Z
      DOUBLE PRECISION Sigma

*     Theta: Angle with the X axis.
      DOUBLE PRECISION Theta

*     TempRad: Working radius
      DOUBLE PRECISION TempRad

*     X: X coord of the grid
*     Y: Y coord of the grid
*     Z: Z coord of the grid
      DOUBLE PRECISION X(nlevels, 0:ncirc, naxis)
      DOUBLE PRECISION Y(nlevels, 0:ncirc, naxis)
      DOUBLE PRECISION Z(nlevels, 0:ncirc, naxis)

*     CrStart: Starting count for circum. direction
*     LvlEnd: Ending count for level index
      INTEGER CrStart, LvlEnd


* ------------------ Code ---------------------------------------------

      pi = 4.0*ATAN (1.0)

      OPEN (UNIT=29, FILE='Kirch3.grid', STATUS='UNKNOWN')

      IF (beautgrid) THEN
         CrStart = 0
```

```
         LvlEnd = 1
      ELSE
         CrStart = 1
         LvlEnd = nlevels
      END IF

      DO Axidex = 1, naxis

         Theta = ( pi*( Axidex - 0.5 ) )/( naxis )

         DO Cirdex = CrStart, ncirc

            DO Lvldex = 1, LvlEnd

               Sigma = ( 2.0*pi*Cirdex )/( ncirc )
               TempRad = ABS (radinl) + dstgrdlvl*( Lvldex - 1)

               X(Lvldex, Cirdex, Axidex) = TempRad*COS (Theta)
               Y(Lvldex, Cirdex, Axidex) = TempRad*SIN (Theta) *
     &            COS (Sigma)
               Z(Lvldex, Cirdex, Axidex) = TempRad*SIN (Theta) *
     &            SIN (Sigma)

               WRITE (29, 2000) X(Lvldex, Cirdex, Axidex),
     &            Y(Lvldex, Cirdex, Axidex),
     &            Z(Lvldex, Cirdex, Axidex)
 2000          FORMAT (3 ( 2X, E24.17 ))

            END DO
*           Lvldex loop

         END DO
*        Cirdex loop

         WRITE (*, 2100) Axidex, naxis
 2100    FORMAT ('Finshed with step', I5, ' of', I5)

*        For beautifying the grid (gnuplot's splot).
         IF (beautgrid) THEN
            WRITE (29, 2200) ''
 2200       FORMAT (A)
         END IF

      END DO
*     Axidex loop

      CLOSE (29)

      RETURN
      END

* ======================================================================


* ======================================================================
```

```
*
      SUBROUTINE GridCyl (dstinl, dstotl, dstgrdlvl, radinl, radotl,
     &   nptinlrad, nptotlrad, beautgrid)
*
* ================== Prologue =========================================
*
* Purpose: Generate the cylindrical grid and output it to file.
*
* Processing: Basically run through the array of the grid and figure
*    out the X, Y, and Z coords of each point and output that to file.
*
* Special requirements:
*    Kirch3.dmn must exist.
*
* ------------------ Include files ------------------------------------

*     ncirc: [INTEGER] Number of points around the cylindrical grid?
*     naxis: [INTEGER] Number of axial points in the grid?
*     ntimes: [INTEGER] The number of timesteps?
*     nlevels: [INTEGER] Number of grid levels?
      INCLUDE 'Kirch3.dmn'

* ------------------ Argument declarations ----------------------------

*     Constants from Grid.rc --- see file for description
      INTEGER nptinlrad, nptotlrad
      DOUBLE PRECISION dstinl, dstotl, dstgrdlvl, radinl, radotl
      LOGICAL beautgrid

* ------------------ Local declarations -------------------------------

      DOUBLE PRECISION pi

*     dsttot: Total dist from inlet to outlet plane
      DOUBLE PRECISION dsttot

*     Axidex: Index in the axial direction.
*     Cirdex: Index in the circuferenctial direction.
*     Lvldex: Level index
      INTEGER Lvldex, Cirdex, Axidex

*     Sigma: Angle around X going from Y towards Z
      DOUBLE PRECISION Sigma

*     TempRad: Working radius
      DOUBLE PRECISION TempRad

*     X: X coord of the grid
*     Y: Y coord of the grid
*     Z: Z coord of the grid
      DOUBLE PRECISION X(nlevels, 0:ncirc, naxis)
      DOUBLE PRECISION Y(nlevels, 0:ncirc, naxis)
      DOUBLE PRECISION Z(nlevels, 0:ncirc, naxis)

*     CrStart: Starting count of the circum index
```

```
*     LvlEnd: Ending count of the level index
      INTEGER CrStart, LvlEnd

* ------------------ Code ----------------------------------------------

      pi = 4.0*ATAN (1.0)
      dsttot = dstotl - dstinl
      radiff = radinl - radotl
      sidelen = SQRT ( dsttot**2.0 + radiff**2.0 )

      OPEN (UNIT=29, FILE='Kirch3.grid', STATUS='UNKNOWN')

      IF (beautgrid) THEN
         CrStart = 0
         LvlEnd = 1
      ELSE
         CrStart = 1
         LvlEnd = nlevels
      END IF

      DO Axidex = 1, naxis
         DO Cirdex = CrStart, ncirc
            DO Lvldex = 1, LvlEnd

               Sigma = ( 2.0*pi*Cirdex )/( ncirc )

*              Inlet plane calculations
               IF ( Axidex.LE.nptinlrad ) THEN

                  X(Lvldex, Cirdex, Axidex) = dstinl + dstgrdlvl*
     &               SIGN (1.0, dstinl)*( Lvldex - 1 )
                  TempRad = ( Axidex - 0.5 )*radinl/nptinlrad

*              Outlet plane grid points
               ELSE IF ( Axidex.GT.( naxis - nptotlrad ) ) THEN

                  X(Lvldex, Cirdex, Axidex) = dstotl + dstgrdlvl*
     &               SIGN (1.0, dstotl)*( Lvldex - 1 )
                  TempRad = ( ( naxis - Axidex + 1) - 0.5 )*
     &               radotl/nptotlrad

*              Otherwise, must be a point along the side
               ELSE

*                 X here is adjusted by a delta to make the grid
*                 layers be normal for conical grids.
                  X(Lvldex, Cirdex, Axidex) = dstinl +
     &               dsttot*( ( Axidex - nptinlrad ) - 0.5 )/
     &               ( naxis - nptotlrad - nptinlrad ) +
     &               ( ( dstgrdlvl*( Lvldex - 1 ) )*
     &               dsttot*radiff/( sidelen**2.0 ) )
                  TempRad = ( dstotl - X(Lvldex, Cirdex, Axidex) )*
     &               radinl/dsttot +
     &               ( X(Lvldex, Cirdex, Axidex) - dstinl )*
     &               radotl/dsttot +
```

```
     &               dstgrdlvl*( Lvldex - 1 )

                END IF

                Y(Lvldex, Cirdex, Axidex) = TempRad*COS (Sigma)
                Z(Lvldex, Cirdex, Axidex) = TempRad*SIN (Sigma)

                WRITE (29, 3000) X(Lvldex, Cirdex, Axidex),
     &             Y(Lvldex, Cirdex, Axidex),
     &             Z(Lvldex, Cirdex, Axidex)
 3000           FORMAT (3 ( 2X, E24.17 ))

             END DO
          END DO

          WRITE (*, 3100) Axidex, naxis
 3100     FORMAT ('Finshed with step', I5, ' of', I5)

*         For beautifying the grid (gnuplot's splot format).
          IF (beautgrid) THEN
             WRITE (29, 3200) ''
 3200        FORMAT (A)
          END IF

       END DO

       CLOSE (29)

       RETURN
       END

* ======================================================================
```

A.2. Pressure Program

A.2.1. Press.const

```
* ----------------------------------------------------------------------
*
*   Press.const
*
*   quest
*   nodog
*   05may97
*
*   RCS info
*   $Id: Press.const,v 1.4 1997/05/16 04:18:14 nodog Exp nodog $
*
```

```
*   Press.const holds several constants for Press.f
*
* ----------------------------------------------------------------------

*     sourcetype constants
      INTEGER dipole, quadrupole, ring, diquad
      PARAMETER ( dipole = 1, quadrupole = 2, ring = 3, diquad = 4 )

*     square root of -1.0
      DOUBLE COMPLEX i
      PARAMETER ( i = ( 0.0, 1.0 ) )

* ----------------------------------------------------------------------
```

A.2.2. Press.rc

```
* ----------------------------------------------------------------------
* Press.rc
* is read by Press.f to get runtime conditions.

* gencompfile: [LOGICAL] generate pressure comparison file
FALSE

* sourcetype: [INTEGER] 1=dipole 2=quadrupole 3=ring 4=dipole&(quadrupole@2f)
2

* ringsources: [INTEGER] number of monopole sources in the ring
16

* ringperiods: [INTEGER] number of periods around the ring
1

* ringradius: [DOUBLE PRECISION] radius of the ring in meters
0.125
* ----------------------------------------------------------------------
```

A.2.3. Press.f

```
* ======================================================================
*
      PROGRAM Press
*
* ================== Prologue ==========================================
*
* $Id: Press.f,v 1.13 1997/06/09 05:01:53 nodog Exp nodog $
*
* Original program written by
* Anderson Mills <nodog@sabine.acs.psu.edu>
* at the Graduate Program in Acoustics
* at Pennsylvania State University
*
```

```
* Purpose:
*    Press reads in a grid, Kirch3.grid, and generates the
*    appropriate values for pressure on all grid points.
*
* Processing:
*    Reads in values from Kirch3.dmn and Kirch3.rc and
*    the grid from Kirch3.grid.  Output is stored in the file
*    Kirch3.pres.
*
* Special requirements:
*        Just that Press.rc, Press.const, Kirch3.dmn, Kirch3.rc,
*       and Kirch3.grid all exist.
*
*
* ----------------- File Numbers -------------------------------------

* 29    Kirch3.grid
* 39    Kirch3.pres
* 40    Kirch3.rc

* 32        Press.dir
* 37    Press.const
* 38    Press.rc

* ----------------- Include files ------------------------------------

*     ncirc: [INTEGER] Number of points around the cylindrical grid?
*     naxis: [INTEGER] Number of axial points in the grid?
*     ntimes: [INTEGER] The number of timesteps?
*     nlevels: [INTEGER] Number of grid levels?
      INCLUDE 'Kirch3.dmn'

* ----------------- Argument declarations ----------------------------

*     Constants from Press.rc --- see Press.rc for description.
      LOGICAL gencompfile
      INTEGER sourcetype
      INTEGER ringsources
      INTEGER ringperiods
      DOUBLE PRECISION ringradius

*     Constants from Kirch3.rc --- see Kirch3.rc for description.
      LOGICAL anglefiles
      LOGICAL directivity
      LOGICAL geometry
      LOGICAL eqnparts
      DOUBLE PRECISION density
      DOUBLE PRECISION soundspeed
      DOUBLE PRECISION deltatime
      DOUBLE PRECISION ffpdist
      DOUBLE PRECISION mach
      DOUBLE PRECISION aspstart
      DOUBLE PRECISION aspend
      DOUBLE PRECISION aspint
```

```
*     X: X coord of the grid
*     Y: Y coord of the grid
*     Z: Z coord of the grid
      DOUBLE PRECISION X(nlevels, ncirc, naxis)
      DOUBLE PRECISION Y(nlevels, ncirc, naxis)
      DOUBLE PRECISION Z(nlevels, ncirc, naxis)

* ----------------- Code ---------------------------------------------

* Main Program

      CALL Banner ()

*     Read in constants from Press.rc ----------------------------------
      CALL ReadPrConst (gencompfile, sourcetype, ringsources,
     &   ringperiods, ringradius)

*     Read in constants from Kirch3.rc ---------------------------------
      CALL GetK3Const (anglefiles, directivity,
     &   geometry, eqnparts, density, soundspeed, deltatime,
     &   ffpdist, mach, aspstart, aspend, aspint)

*     Read the grid file -----------------------------------------------
      CALL ReadGrid (X, Y, Z)

*     Generate and write the Kirch3 pressure file ----------------------
      CALL K3PressOut (sourcetype, ringsources, ringperiods,
     &   ringradius, deltatime, ffpdist, soundspeed, X, Y, Z)

      IF ( gencompfile ) THEN

*        Generate and write the directivity pressure file -------------
         CALL DirPressOut (sourcetype, ringsources, ringperiods,
     &      ringradius, deltatime, ffpdist, soundspeed)

      ENDIF

      CALL Banner ()

      STOP
      END

* Main Program

* ====================================================================


* ====================================================================
*
      SUBROUTINE Banner()
*
* ================== Prologue ========================================
*
* Purpose: Show the beginning and ending of the program.
*
```

```
* ------------------ Code ----------------------------------------------

      WRITE (*, *) '                                    '
      WRITE (*, *) ' 888888                             '
      WRITE (*, *) ' 8     8 88888  888888  8888   8888 '
      WRITE (*, *) ' 8     8 8    8 8      8      8     '
      WRITE (*, *) ' 888888  8    8 88888   8888   8888 '
      WRITE (*, *) ' 8       88888  8           8      8'
      WRITE (*, *) ' 8       8   8  8      8    8 8    8'
      WRITE (*, *) ' 8       8    8 888888  8888   8888 '
      WRITE (*, *) '                                    '

      RETURN
      END

* ====================================================================


* ====================================================================
*
      SUBROUTINE ReadPrConst (gencompfile, sourcetype, ringsources,
     &   ringperiods, ringradius)

* ================== Prologue ========================================
*
* Purpose:
*    To read in the constants from the Press.rc file
*
* Processing:
*    Uses READ's to get the info.
*
* Special requirements:
*    Gotta have something to read, so the Press.rc file must exist.
*
* ------------------ Argument declarations ----------------------------

*     Constants from Press.rc --- see Press.rc for description.
      LOGICAL gencompfile
      INTEGER sourcetype
      INTEGER ringsources
      INTEGER ringperiods
      DOUBLE PRECISION ringradius

* ------------------ Local declarations -------------------------------

      CHARACTER Dummy

* ------------------ Code ----------------------------------------------

      OPEN (UNIT=30, FILE='Press.rc', STATUS='UNKNOWN')

*     Read constants from Press.rc --- See Press.rc for description.
*     Three lines of comments at the top.
      READ (30, 6000) Dummy
      READ (30, 6000) Dummy
```

```
      READ (30, 6000) Dummy

*     One blank line, one label line, then the constant.
      READ (30, 6000) Dummy
      READ (30, 6000) Dummy
      READ (30, 6010) gencompfile
      READ (30, 6000) Dummy
      READ (30, 6000) Dummy
      READ (30, 6030) sourcetype
      READ (30, 6000) Dummy
      READ (30, 6000) Dummy
      READ (30, 6030) ringsources
      READ (30, 6000) Dummy
      READ (30, 6000) Dummy
      READ (30, 6030) ringperiods
      READ (30, 6000) Dummy
      READ (30, 6000) Dummy
      READ (30, 6020) ringradius

      CLOSE (30)

 6000 FORMAT (A)
 6010 FORMAT (L5)
 6020 FORMAT (F24.17)
 6030 FORMAT (I9)

      RETURN
      END

* ====================================================================


* ====================================================================
*
      SUBROUTINE ReadGrid (X, Y, Z)
*
* ================== Prologue ========================================
*
* Purpose: Read the grid file.
*
* Processing: Basically run through the array of the grid and figure
*    out the X, Y, and Z coords of each point.
*
* Special requirements:
*    Kirch3.grid must exist.
*
* ------------------ Include files ----------------------------------

*     ncirc: [INTEGER] Number of points around the cylindrical grid?
*     naxis: [INTEGER] Number of axial points in the grid?
*     ntimes: [INTEGER] The number of timesteps?
*     nlevels: [INTEGER] Number of grid levels?
      INCLUDE 'Kirch3.dmn'

* ------------------ Argument declarations --------------------------
```

```
*     X: X coord of the grid
*     Y: Y coord of the grid
*     Z: Z coord of the grid
      DOUBLE PRECISION X(nlevels, ncirc, naxis)
      DOUBLE PRECISION Y(nlevels, ncirc, naxis)
      DOUBLE PRECISION Z(nlevels, ncirc, naxis)

* ------------------ Local declarations ------------------------------

*     Axidex: Index in the axial direction.
*     Cirdex: Index in the circuferenctial direction.
*     Lvldex: Level index
      INTEGER Lvldex, Cirdex, Axidex

* ------------------ Code --------------------------------------------

      OPEN (UNIT=29, FILE='Kirch3.grid', STATUS='UNKNOWN')

      DO Axidex = 1, naxis
         DO Cirdex = 1, ncirc
            DO Lvldex = 1, nlevels

               READ (29, 3000) X(Lvldex, Cirdex, Axidex),
     &            Y(Lvldex, Cirdex, Axidex),
     &            Z(Lvldex, Cirdex, Axidex)
 3000          FORMAT (3 ( 2X, E24.17 ))


            END DO
         END DO

         WRITE (*, 3100) Axidex, naxis
 3100    FORMAT (' Read grid step', I5, ' of', I5)

      END DO

      CLOSE (29)

      RETURN
      END

* ====================================================================


* ====================================================================
*
      SUBROUTINE K3PressOut (sourcetype, ringsources, ringperiods,
     & ringradius, deltatime, ffpdist, soundspeed, X, Y, Z)
*
* ================== Prologue ========================================
*
* Purpose:
*    Generate the pressure file for Kirch3.
*
```

```
* Processing:
*    Uses analytic computations to determine the pressure at all
*    the grid points for each time step.
*
* Special requirements:
*    Kirch3.dmn and Press.const must exist.
*
* ------------------ Include files -----------------------------------

*     ncirc: [INTEGER] Number of points around the cylindrical grid?
*     naxis: [INTEGER] Number of axial points in the grid?
*     ntimes: [INTEGER] The number of timesteps?
*     nlevels: [INTEGER] Number of grid levels?
      INCLUDE 'Kirch3.dmn'
      INCLUDE 'Press.const'

* ------------------ Argument declarations ---------------------------

*     Constants from Press.rc --- see Press.rc for description.
      INTEGER sourcetype
      INTEGER ringsources
      INTEGER ringperiods
      DOUBLE PRECISION ringradius

*     Constants from Kirch3.rc --- see Kirch3.rc for description.
      DOUBLE PRECISION deltatime
      DOUBLE PRECISION ffpdist
      DOUBLE PRECISION soundspeed

*     X: X coord of the grid
*     Y: Y coord of the grid
*     Z: Z coord of the grid
      DOUBLE PRECISION X(nlevels, ncirc, naxis)
      DOUBLE PRECISION Y(nlevels, ncirc, naxis)
      DOUBLE PRECISION Z(nlevels, ncirc, naxis)

* ------------------ Local declarations ------------------------------

*     Axidex: Index in the axial direction.
*     Cirdex: Index in the circuferenctial direction.
*     Lvldex: Level index
*     Timdex: Time index
      INTEGER Lvldex, Cirdex, Axidex, Timdex

*     P: Temporary pressure variable
      DOUBLE PRECISION P(nlevels)

*     Declaring the functions because they're not just REAL
      DOUBLE PRECISION RingSrcPress
      DOUBLE PRECISION DipSrcPress
      DOUBLE PRECISION DiQuSrcPress
      DOUBLE PRECISION QuadSrcPress

* ------------------ Code --------------------------------------------
```

```
* generate and write the pressure file for Kirch3

      OPEN(UNIT=39, FILE='Kirch3.pres', STATUS='UNKNOWN')

      DO Timdex = 0, ntimes - 1
         DO Axidex = 1, naxis
            DO Cirdex = 1, ncirc
               DO Lvldex = 1, nlevels

*                 Use the correct function depending on what is
*                 specified in Press.rc.  If something is incorrectly
*                 specified, send a message to the screen.

                  IF ( sourcetype .EQ. dipole ) THEN

                     P(Lvldex) = DipSrcPress (
     &                  X(Lvldex, Cirdex, Axidex),
     &                  Y(Lvldex, Cirdex, Axidex),
     &                  Z(Lvldex, Cirdex, Axidex),
     &                  Timdex, deltatime, ffpdist, soundspeed )

                  ELSEIF ( sourcetype .EQ. quadrupole ) THEN

                     P(Lvldex) = QuadSrcPress (
     &                  X(Lvldex, Cirdex, Axidex),
     &                  Y(Lvldex, Cirdex, Axidex),
     &                  Z(Lvldex, Cirdex, Axidex),
     &                  Timdex, deltatime, ffpdist, soundspeed )

                  ELSEIF ( sourcetype .EQ. ring ) THEN

                     P(Lvldex) = RingSrcPress (
     &                  X(Lvldex, Cirdex, Axidex),
     &                  Y(Lvldex, Cirdex, Axidex),
     &                  Z(Lvldex, Cirdex, Axidex),
     &                  Timdex, ringsources, ringperiods, ringradius,
     &                  deltatime, ffpdist, soundspeed )

                  ELSEIF ( sourcetype .EQ. diquad ) THEN

                     P(Lvldex) = DiQuSrcPress (
     &                  X(Lvldex, Cirdex, Axidex),
     &                  Y(Lvldex, Cirdex, Axidex),
     &                  Z(Lvldex, Cirdex, Axidex),
     &                  Timdex, deltatime, ffpdist, soundspeed )

                  ELSE

                     WRITE ( *, * ) 'K3PressOut: Source type undefined.'

                  ENDIF

               END DO

               WRITE (39, 4000) (P(Lvldex), Lvldex = 1, 3)
```

```
 4000          FORMAT (3 ( 1x, E24.17 ) )

           END DO
         END DO

         WRITE (*, 4100) Timdex + 1, ntimes
 4100    FORMAT (' Finished with time step', I5, ' of', I5,
     &       ' of the Kirch3.pres file.')

*        Add this back in to format the output for gnuplot's splot.
*        WRITE (39, '(A)') ' '

       END DO

       CLOSE (39)

       RETURN
       END

* ======================================================================


* ======================================================================
*
       SUBROUTINE DirPressOut (sourcetype, ringsources, ringperiods,
     &   ringradius, deltatime, ffpdist, soundspeed)
*
* ================== Prologue ============================================
*
* Purpose:
*    Generate a comparison directivity pressure file.
*
* Processing:
*    Uses analytic computations to determine the pressure at all
*    the directivity angles (0-359).
*
* Special requirements:
*    Kirch3.dmn and Press.const must exist.
*
* ------------------ Include files -----------------------------------

*     ncirc: [INTEGER] Number of points around the cylindrical grid?
*     naxis: [INTEGER] Number of axial points in the grid?
*     ntimes: [INTEGER] The number of timesteps?
*     nlevels: [INTEGER] Number of grid levels?
       INCLUDE 'Kirch3.dmn'
       INCLUDE 'Press.const'

* ------------------ Argument declarations ----------------------------

*     Constants from Press.rc --- see Press.rc for description.
       INTEGER sourcetype
       INTEGER ringsources
       INTEGER ringperiods
       DOUBLE PRECISION ringradius
```

```
*     Constants from Kirch3.rc --- see Kirch3.rc for description.
      DOUBLE PRECISION deltatime
      DOUBLE PRECISION ffpdist
      DOUBLE PRECISION soundspeed

* ------------------ Local declarations ------------------------------

      DOUBLE PRECISION pi

*     Timdex: Time index
      INTEGER Timdex

*     Xc: Temporary X variable.
*     Yc: Temporary Y variable.
*     Zc: Temporary Z variable.
      DOUBLE PRECISION Xc, Yc, Zc

*     Pd: [DOUBLE PRECISION] Temporary pressure variable
      DOUBLE PRECISION Pd(0:ntimes - 1)

*     Thetadex: Temp angle variable
*     ThetaRad: Temp angle variable
      INTEGER Thetadex
      DOUBLE PRECISION ThetaRad

*     Fourdex: Index through Fourier components
      INTEGER Fourdex

*     FourCN: Fourier coefficient
*     FourFreq: frequency of Fourier component
*     FourPhi: phase of Fourier components
      DOUBLE PRECISION FourPhi, FourCN, FourFreq

*     FourComp: Fourier components
      DOUBLE PRECISION FourComp(1:nfouriercomps)

*     FourdBComp: Fourier components in dB
      DOUBLE PRECISION FourdBComp(1:nfouriercomps)

      DOUBLE PRECISION RingSrcPress
      DOUBLE PRECISION DipSrcPress
      DOUBLE PRECISION DiQuSrcPress
      DOUBLE PRECISION QuadSrcPress


* ------------------ Code --------------------------------------------
* generate and write the directivity pressure file

      pi = 4.0*ATAN (1.0)

* directivity output

      OPEN(UNIT=32, FILE='Press.dir', STATUS='UNKNOWN')
```

```
*     Run through all angles.
      DO Thetadex = 0, 359

*        Convert them to radians.
         ThetaRad = ( pi/180.0 )*Thetadex

*        Run through all time steps.
         DO Timdex = 0, ntimes - 1


*           Farfield points
            Xc = ffpdist*COS (ThetaRad)
            Yc = ffpdist*SIN (ThetaRad)
            Zc = 0.0

*           Use the correct function depending on what is
*           specified in Press.rc.  If something is incorrectly
*           specified, send a message to the screen.

            IF ( sourcetype .EQ. dipole ) THEN

               Pd(Timdex) = DipSrcPress (Xc, Yc, Zc, Timdex,
     &            deltatime, ffpdist, soundspeed )

            ELSEIF ( sourcetype .EQ. quadrupole ) THEN

               Pd(Timdex) = QuadSrcPress (Xc, Yc, Zc, Timdex,
     &            deltatime, ffpdist, soundspeed )

            ELSEIF ( sourcetype .EQ. ring ) THEN

               Pd(Timdex) = RingSrcPress (Xc, Yc, Zc, Timdex,
     &            ringsources, ringperiods, ringradius,
     &            deltatime, ffpdist, soundspeed )

            ELSEIF ( sourcetype .EQ. diquad ) THEN

               Pd(Timdex) = DiQuSrcPress (Xc, Yc, Zc, Timdex,
     &            deltatime, ffpdist, soundspeed )

            ELSE

               WRITE ( *, * ) 'DirPressOut: Source type undefined.'

            ENDIF

         END DO

*        Fourier coefficient of pressure ------------------------------

*        Run through all Fourier components
         DO Fourdex = 1, nfouriercomps

*           NOTE: freq = (1/period) * which Fourier component
            FourFreq = ( Fourdex*1.0 )/( ntimes*deltatime )
```

```
            CALL Fourier (Pd, FourPhi, FourCN, FourFreq, deltatime)

            FourComp(Fourdex) = FourCN
            FourdBComp(Fourdex) = 20.0*LOG10 (FourCN/2.0E-5)

         END DO

         WRITE (32, 5000) Thetadex,
     &      ( FourdBComp(Fourdex), FourComp(Fourdex),
     &      Fourdex = 1, nfouriercomps )
 5000    FORMAT (I4, 2X, 20 ( 1X, E24.17 ))

         WRITE (*, 5100) Thetadex
 5100    FORMAT ('Finished with', I5,
     &      ' of 360 degrees of the Press.dir comparison file.')

      END DO

      CLOSE(UNIT=32)

      RETURN
      END

* ====================================================================


* ====================================================================
*
      DOUBLE PRECISION FUNCTION DipSrcPress (Xp, Yp, Zp, TimeStep,
     &   deltatime, ffpdist, soundspeed)
*
* ================== Prologue ========================================
*
* Purpose:
*    Generate the pressure for one X, Y, Z location for a dipole source.
*
* Processing:
*    Uses analytic computations to determine the pressure at one point.
*
* Special requirements:
*    Kirch3.dmn and Press.cont must exist.
*
* ------------------ Include files -----------------------------------

*     ncirc: [INTEGER] Number of points around the cylindrical grid?
*     naxis: [INTEGER] Number of axial points in the grid?
*     ntimes: [INTEGER] The number of timesteps?
*     nlevels: [INTEGER] Number of grid levels?
      INCLUDE 'Kirch3.dmn'
      INCLUDE 'Press.const'

* ------------------ Argument declarations ---------------------------

*     Xp: X coord of the point
```

```
*     Yp: Y coord of the point
*     Zp: Z coord of the point
      DOUBLE PRECISION Xp, Yp, Zp

*     TimeStep: The particular time step in the range of 0 to ntimes-1
      INTEGER TimeStep

*     Constants from Kirch3.rc --- see Kirch3.rc for description.
      DOUBLE PRECISION deltatime
      DOUBLE PRECISION ffpdist
      DOUBLE PRECISION soundspeed

* ------------------ Local declarations ------------------------------

      DOUBLE PRECISION pi

*     Omega: Radial frequency of the source
*     WaveNum: Wave number of the source
*     Strength: Strength of the source
      DOUBLE PRECISION Omega, WaveNum, Strength

*     Distance: Distance from the point to the KSP
*     CosTheta: cosine of the angle from X-Z plane
      DOUBLE PRECISION Distance, CosTheta

*     DP: temporary pressure variable
      DOUBLE PRECISION DP

* ------------------ Code --------------------------------------------

* generate and write the pressure

      pi = 4.0*ATAN (1.0)
      Omega = ( 2.0*pi )/( deltatime*ntimes )
      WaveNum = Omega/soundspeed

*     The following strength gives 1 Pa at the far-field point
      Strength = ffpdist

      Distance = SQRT ( Xp**2.0 + Yp**2.0 + Zp**2.0 )

      CosTheta = Xp/Distance

*     Dipole with highest pressure along the x axis
      DP = CosTheta*(Strength/Distance)*
     &   DBLE ( EXP ( i*( WaveNum*Distance -
     &   Omega*deltatime*TimeStep ) ) )

      DipSrcPress = DP

      RETURN
      END

* ====================================================================
```

```
* ================================================================
*
      DOUBLE PRECISION FUNCTION QuadSrcPress (Xp, Yp, Zp, TimeStep,
     &   deltatime, ffpdist, soundspeed)
*
* ================== Prologue =========================================
*
* Purpose:
*    Generate the pressure for one X, Y, Z location for a
*    quadrupole source
*
* Processing:
*    Uses analytic computations to determine the pressure at one point.
*
* Special requirements:
*    Kirch3.dmn and Press.cont must exist.
*
* ------------------ Include files ----------------------------------

*     ncirc: [INTEGER] Number of points around the cylindrical grid?
*     naxis: [INTEGER] Number of axial points in the grid?
*     ntimes: [INTEGER] The number of timesteps?
*     nlevels: [INTEGER] Number of grid levels?
      INCLUDE 'Kirch3.dmn'
      INCLUDE 'Press.const'

* ------------------ Argument declarations ----------------------------

*     Xp: X coord of the point
*     Yp: Y coord of the point
*     Zp: Z coord of the point
      DOUBLE PRECISION Xp, Yp, Zp

*     TimeStep: The particular time step in the range of 0 to ntimes-1
      INTEGER TimeStep

*     Constants from Kirch3.rc --- see Kirch3.rc for description.
      DOUBLE PRECISION deltatime
      DOUBLE PRECISION ffpdist
      DOUBLE PRECISION soundspeed

* ------------------ Local declarations -----------------------------

      DOUBLE PRECISION pi

*     Omega: Radial frequency of the source
*     WaveNum: Wave number of the source
*     Strength: Strength of the source
      DOUBLE PRECISION Omega, WaveNum, Strength

*     Distance: Distance from the point to the KSP
*     SinTheta: sine of the angle from the X-Z plane
*     Cos2Theta: cosine of the 2 times the angle from X-Z plane
      DOUBLE PRECISION Distance, SinTheta, Cos2Theta
```

```
*     DP: temporary pressure variable
      DOUBLE PRECISION DP

* ------------------ Code ------------------------------------------

* generate and write the pressure

      pi = 4.0*ATAN (1.0)
      Omega = ( 2.0*pi )/( deltatime*ntimes )
      WaveNum = Omega/soundspeed

*     The following strength gives 1 Pa at the far-field point
      Strength = ffpdist

      Distance = SQRT ( Xp**2.0 + Yp**2.0 + Zp**2.0 )

      SinTheta = Yp/Distance
      Cos2Theta = 1 - 2.*SinTheta**2.

*     Quadrupole with highest pressure along the x and y axis
      DP = Cos2Theta*(Strength/Distance)*
     &   DBLE ( EXP ( i*( WaveNum*Distance -
     &   Omega*deltatime*TimeStep ) ) )

      QuadSrcPress = DP

      RETURN
      END

* ========================================================================


* ========================================================================
*
      DOUBLE PRECISION FUNCTION DiQuSrcPress (Xp, Yp, Zp, TimeStep,
     &   deltatime, ffpdist, soundspeed)
*
* ================= Prologue =============================================
*
* Purpose:
*    Generate the pressure for one X, Y, Z location for a dipole source
*    at the lowest frequency and a quadrupole source at 2 times the
*    lowest frequency.
*
* Processing:
*    Uses analytic computations to determine the pressure at one point.
*
* Special requirements:
*    Kirch3.dmn and Press.cont must exist.
*
* ------------------ Include files -----------------------------------

*     ncirc: [INTEGER] Number of points around the cylindrical grid?
*     naxis: [INTEGER] Number of axial points in the grid?
```

```
*     ntimes: [INTEGER] The number of timesteps?
*     nlevels: [INTEGER] Number of grid levels?
      INCLUDE 'Kirch3.dmn'
      INCLUDE 'Press.const'

* ------------------ Argument declarations -----------------------------

*     Xp: X coord of the point
*     Yp: Y coord of the point
*     Zp: Z coord of the point
      DOUBLE PRECISION Xp, Yp, Zp

*     TimeStep: The particular time step in the range of 0 to ntimes-1
      INTEGER TimeStep

*     Constants from Kirch3.rc --- see Kirch3.rc for description.
      DOUBLE PRECISION deltatime
      DOUBLE PRECISION ffpdist
      DOUBLE PRECISION soundspeed

* ------------------ Local declarations ------------------------------

      DOUBLE PRECISION pi

*     Omega: Radial frequency of the source
*     WaveNum: Wave number of the source
*     Strength: Strength of the source
      DOUBLE PRECISION Omega, WaveNum, Strength

*     Distance: Distance from the point to the KSP
*     CosTheta: cosine of the angle from X-Z plane
*     SinTheta: sine of the angle from the X-Z plane
*     Cos2Theta: cosine of the 2 times the angle from X-Z plane
      DOUBLE PRECISION Distance, CosTheta, SinTheta, Cos2Theta

*     DP: temporary pressure variable
      DOUBLE PRECISION DP

* ------------------ Code ----------------------------------------------

* generate and write the pressure

      pi = 4.0*ATAN (1.0)
      Omega = ( 2.0*pi )/( deltatime*ntimes )
      WaveNum = Omega/soundspeed

*     The following strength gives 1 Pa at the far-field point
      Strength = ffpdist

      Distance = SQRT ( Xp**2.0 + Yp**2.0 + Zp**2.0 )

      CosTheta = Xp/Distance
      SinTheta = Yp/Distance
      Cos2Theta = 1 - 2.*SinTheta**2.
```

```
*     Dipole with highest pressure along the x axis plus a quadrupole
*     at 2*freq with pressure antinodes along the x and y axis
      DP = CosTheta*(Strength/Distance)*
     &   DBLE ( EXP ( i*( WaveNum*Distance -
     &   Omega*deltatime*TimeStep ) ) ) +
     &   Cos2Theta*(Strength/Distance)*
     &   DBLE ( EXP ( i*( 2.*WaveNum*Distance -
     &   2.*Omega*deltatime*TimeStep ) ) )


      DiQuSrcPress = DP

      RETURN
      END

* ======================================================================


* ======================================================================
*
      DOUBLE PRECISION FUNCTION RingSrcPress (Xp, Yp, Zp, TimeStep,
     &   ringsources, ringperiods, ringradius,
     &   deltatime, ffpdist, soundspeed)
*
* ================== Prologue ============================================
*
* Purpose:
*    Generate the pressure for one X, Y, Z location for a ring source.
*
* Processing:
*    Uses analytic computations to determine the pressure at one point.
*
* Special requirements:
*    Kirch3.dmn and Press.const must exist.
*
* ------------------ Include files -----------------------------------

*     ncirc: [INTEGER] Number of points around the cylindrical grid?
*     naxis: [INTEGER] Number of axial points in the grid?
*     ntimes: [INTEGER] The number of timesteps?
*     nlevels: [INTEGER] Number of grid levels?
      INCLUDE 'Kirch3.dmn'
      INCLUDE 'Press.const'

* ------------------ Argument declarations ----------------------------

*     Xp: X coord of the point
*     Yp: Y coord of the point
*     Zp: Z coord of the point
      DOUBLE PRECISION Xp, Yp, Zp

*     TimeStep: The particular time step in the range of 0 to ntimes-1
      INTEGER TimeStep

*     Constants from Press.rc --- see Press.rc for description.
```

```
      INTEGER ringsources
      INTEGER ringperiods
      DOUBLE PRECISION ringradius

*     Constants from Kirch3.rc --- see Kirch3.rc for description.
      DOUBLE PRECISION deltatime
      DOUBLE PRECISION ffpdist
      DOUBLE PRECISION soundspeed

* ------------------ Local declarations ------------------------------

      DOUBLE PRECISION pi

*     Omega: Radial frequency of the source
*     WaveNum: Wave number of the source
*     Strength: Strength of the source
      DOUBLE PRECISION Omega, WaveNum, Strength

*     Distance: Distance from source to point.
      DOUBLE PRECISION Distance

*     SumOfP: Temporary pressure variable
      DOUBLE PRECISION SumOfP

*     Srcdex: Index of sources
      INTEGER Srcdex

*     SigmaRad: Radian angle of individual source
      DOUBLE PRECISION SigmaRad

*     Xsrc: X coord of the source
*     Ysrc: Y coord of the source
*     Zsrc: Z coord of the source
      DOUBLE PRECISION Xsrc, Ysrc, Zsrc

* ------------------ Code --------------------------------------------

* generate and write the pressure

      pi = 4.0*ATAN (1.0)
      Omega = ( 2.0*pi )/( deltatime*ntimes )
      WaveNum = Omega/soundspeed

*     The following strength gives 1 Pa at the far-field point for
*     each source.  The combination of sources will probably have
*     a higher pressure.
      Strength = ffpdist

      SumOfP = 0.0

*     Look at all the sources.
      DO Srcdex = 1, ringsources

*        Find the X, Y, and Z coordinate of each source.
         SigmaRad = 2.0*pi*(1.0*(Srcdex - 1))/(1.0*ringsources)
```

```
            Xsrc = 0.0
            Ysrc = ringradius*COS (SigmaRad)
            Zsrc = ringradius*SIN (SigmaRad)

            Distance = SQRT ( ( Xsrc - Xp )**2.0 + ( Ysrc - Yp )**2.0 +
     &         ( Zsrc - Zp )**2.0 )

*           Monopole source with the phase shifted by the last EXP.
            SumOfP = SumOfP + (Strength/Distance)*
     &         DBLE ( EXP ( -i*Omega*TimeStep*deltatime )*
     &         EXP ( i*WaveNum*Distance )*
     &         EXP ( i*SigmaRad*( 1.0*ringperiods ) ) )

         END DO

         RingSrcPress = SumOfP

         RETURN
         END

* ======================================================================
```

A.3. Fully Numerical Kirchhoff Program

A.3.1. Kirch3.dmn

```
* ----------------------------------------------------------------------
*
*   Kirch3.dmn
*
*   quest
*   nodog
*   01apr96
*
*   RCS info
*   $Id: Kirch3.dmn,v 1.51 1997/07/17 15:06:49 nodog Exp nodog $
*
*   Kirch3.dmn is a file which holds the spatial and time array
*   dimensions of a grid used to calculate far field noise.
*
* ----------------------------------------------------------------------

*     Number of axial points in the grid?
      INTEGER naxis
      PARAMETER (naxis = 10)

*     Number of points around the cylindrical grid?
      INTEGER ncirc
      PARAMETER (ncirc = 22)
```

```
*     The number of timesteps?
      INTEGER ntimes
      PARAMETER (ntimes = 60)

*     The number of grid levels?
      INTEGER nlevels
      PARAMETER (nlevels = 3)

*     The number of Fourier components?
*     maximum is set by 7020 FORMAT statement in Kirch3.f.
      INTEGER nfouriercomps
      PARAMETER (nfouriercomps = 1)

* ---------------------------------------------------------------------
```

A.3.2. Kirch3.rc

```
* ---------------------------------------------------------------------
* Kirch3.rc
* is read by Kirch3.f to get many of its constant values.

* anglefiles: [LOGICAL] pressure and Fourier files at every angle
FALSE

* directivity: [LOGICAL] look at directivity and Fourier output
TRUE

* geometry: [LOGICAL] output the grid geometry
FALSE

* eqnparts: [LOGICAL] look at the separate parts of the equation
FALSE

* density: [DOUBLE PRECISION]  density in kg/m^3 (future use)
1.21

* soundspeed: [DOUBLE PRECISION] speed of sound in m/s
343.0

* frequency: [DOUBLE PRECISION] the frequency of the source
343.0

* ffpdist: [DOUBLE PRECISION] the distance to the far field point
10.3

* mach: [DOUBLE PRECISION] the Mach number (avg v/C0) (future use)
0.0

* aspstart: [INTEGER] the beginning aspect angle in degrees
0.0

* aspend: [INTEGER] the ending aspect angle in degrees
360.0
```

```
* aspint: [INTEGER] the interval between angles in degrees
1.0
* ----------------------------------------------------------------------
```

A.3.3. Kirch3.f

```
* ======================================================================
*
      PROGRAM Kirch3
*
* ================== Prologue =========================================
*
*   $Id: Kirch3.f,v 1.33 1997/06/18 13:03:07 nodog Exp nodog $
*
* Original program written by
* Anderson Mills <nodog@sabine.acs.psu.edu>
* at the Graduate Program in Acoustics
* at Pennsylvania State University
*
* Purpose:
*   Kirch3 is meant to be used to calculate the far field noise
*   at a point some specified distance from a Kirchhoff surface.
*   It requires grid files and unsteady pressure files given by
*   other programs.
*
* Processing:
*
* Special requirements:
*
*----------------------------------------------------------------------
*   Unit Numbers
*
*   20   Grid.rc        Grid.f runtime conditions
*   29   Kirch3.grid    grid file
*
*   30   Press.rc       Press.f runtime conditions
*   32   Press.dir      comparison output of Press.f
*   39   Kirch3.pres    pressure input file
*
*   40   Kirch3.rc      Kirch3.f runtime conditions
*
*   50   Kirch3.dir     directivity output
*   52   P.XXX          pressure output file
*   53   F.XXX          fourier output file
*   56   PartAFile      eqn part A file
*   57   PartBFile      eqn part B file
*   58   PartCFile      eqn part C file
*
*   64   dPKirch3.out   dPdN and dPdT from Kirch3.f
*   66   geom.XXX       R, dRdN, and dS for angles
*
* ------------------ Include files -----------------------------------
```

```
*     ncirc: [INTEGER] Number of points around the cylindrical grid?
*     naxis: [INTEGER] Number of axial points in the grid?
*     ntimes: [INTEGER] The number of timesteps?
*     nlevels: [INTEGER] Number of grid levels?
      INCLUDE 'Kirch3.dmn'

* ------------------ Argument declarations -----------------------------

      DOUBLE PRECISION pi

*     Timdex: current timestep
      INTEGER Timdex

*     X: X coord of the grid
*     Y: Y coord of the grid
*     Z: Z coord of the grid
      DOUBLE PRECISION X(naxis, ncirc, nlevels)
      DOUBLE PRECISION Y(naxis, ncirc, nlevels)
      DOUBLE PRECISION Z(naxis, ncirc, nlevels)

*     dN: distance between two gridpoints away from the KS
*        in the normal direction
      DOUBLE PRECISION dN(naxis, ncirc, nlevels - 1)

*     RadAng: radian angle of current calculation
      DOUBLE PRECISION RadAng

*     FileDegAng: Angle used for filenames
      DOUBLE PRECISION FileDegAng

*     R: distance between KS point and far field point
*     dRdN: change in R in normal direction from KS
*     dS: area of effect of pressure on the computational surface
      DOUBLE PRECISION R(naxis, ncirc, nlevels)
      DOUBLE PRECISION dRdN(naxis, ncirc)
      DOUBLE PRECISION dS(naxis, ncirc)

*     Lag: number of Timdexs of lag for any KS point
      DOUBLE PRECISION Lag(naxis, ncirc)

*     P: pressures on the KS
      DOUBLE PRECISION P(0:ntimes - 1, naxis, ncirc)

*     dPdN: the change in pressure on the normal away from the KS
      DOUBLE PRECISION dPdN(0:ntimes - 1, naxis, ncirc)

*     dPdT: change in pressure between time steps at any KS point
      DOUBLE PRECISION dPdT(0:ntimes - 1, naxis, ncirc)

*     FFPress: radiated pressure
      DOUBLE PRECISION FFPress(0: ntimes - 1)

*     Constants from Kirch3.rc --- see Kirch3.rc for description.
      LOGICAL anglefiles
```

```
      LOGICAL directivity
      LOGICAL geometry
      LOGICAL eqnparts
      DOUBLE PRECISION density
      DOUBLE PRECISION soundspeed
      DOUBLE PRECISION deltatime
      DOUBLE PRECISION ffpdist
      DOUBLE PRECISION mach
      DOUBLE PRECISION aspstart
      DOUBLE PRECISION aspend
      DOUBLE PRECISION aspint


* ------------------ Local declarations --------------------------------

*     RadStart: radian aspstart
*     RadInt: radian aspint
      DOUBLE PRECISION RadStart, RadInt

*     Angdex: Angle index
      INTEGER Angdex

      DOUBLE PRECISION FindRadPress

* ------------------ Code ----------------------------------------------

*     Main Loop

      pi = 4.0*ATAN (1.0)

      CALL Banner ()

*     Read in constants from Kirch3.rc ----------------------------------
      CALL GetK3Const (anglefiles, directivity,
     &   geometry, eqnparts, density, soundspeed, deltatime,
     &   ffpdist, mach, aspstart, aspend, aspint)

*     Read the grid file ------------------------------------------------
      CALL ReadGrid (X, Y, Z, dN, geometry)

*     Read the pressure file --------------------------------------------
      CALL ReadPress (P, dN, dPdN, dPdT, pi, density, soundspeed,
     &   deltatime, ffpdist, eqnparts)

*     Set up for directivity output. ------------------------------------
      IF (directivity) THEN
         OPEN (UNIT=50, FILE='Kirch3.dir', STATUS='UNKNOWN')
      END IF

*     Scroll over aspect angles. ----------------------------------------

      RadStart = pi*aspstart/180.0
      RadInt   = pi*aspint/180.0

      DO Angdex = 0, INT ( ( aspend - aspstart )/aspint )
```

```
         RadAng = RadStart + RadInt*( Angdex*1.0 )

         WRITE (*, 6000) INT ( Angdex*aspint )
6000     FORMAT ( ' Kirch3: Calculations for ', I3, ' degrees' )

*        Open the output files for each angle -------------------------
         IF (anglefiles) THEN
           CALL OpenAngleFiles (RadAng, FileDegAng, pi,
     &         directivity, geometry, eqnparts)
         END IF

*        Scroll over time. --------------------------------------------
         DO Timdex = 0, ntimes - 1

*           Reset the summed pressure.
            FFPress(Timdex) = 0.0

*           Calculate the distances and lags. ------------------------
            IF ( Timdex .EQ. 0 ) THEN

*              Find distances from ffp to KS points.
               CALL FindDistance (X, Y, Z, dN, dS, R, dRdN, Lag,
     &            RadAng, FileDegAng, pi, soundspeed, deltatime,
     &            ffpdist, geometry)

            END IF

*           Sum radiated pressure. ---------------------------------
            FFPress(Timdex) = FindRadPress (P, dPdN, dPdT,
     &         X, Y, Z, dS, R, dRdN, Lag, Timdex,
     &         eqnparts, pi, soundspeed, deltatime)

*           Output the radiated pressure for this time step. ----------
         IF (anglefiles) THEN
            WRITE (52, 150) (1.0*Timdex)*deltatime, FFPress(Timdex)
  150       FORMAT (2 ( 2X, E24.17 ))
         END IF

*        End time loop.
         END DO

*        Output directivity for this RadAng. --------------------------
         IF (directivity) THEN
            CALL OutDirect (RadAng, FileDegAng, FFPress, pi, anglefiles,
     &         deltatime)
         END IF
*
         IF (anglefiles) THEN
            CALL CloseAngleFiles (directivity, geometry, eqnparts)
         ENDIF

*     End aspectang loop.
      END DO

* --------------------------------------------------------------------
```

```
*     Close the directivity file.
      IF (directivity) THEN
         CLOSE (UNIT=50)
      END IF

      CALL Banner()

      STOP
      END

*     End of Main
*======================================================================


* ======================================================================
*
      SUBROUTINE Banner()
*
* ================== Prologue ==========================================
*
* Purpose:
*    To mark the begin and end of the program.
*
* Processing:
*    Just writes out a banner.
*
* Special requirements:
*
* ------------------ Code ----------------------------------------------

      WRITE (*, *) '                                      '
      WRITE (*, *) ' 8    8                          88888 '
      WRITE (*, *) ' 8   8  8 88888   8888  8    8 8     8'
      WRITE (*, *) ' 8  8   8 8    8 8    8 8    8       8'
      WRITE (*, *) ' 888    8 8    8 8      888888  88888 '
      WRITE (*, *) ' 8  8   8 88888  8      8    8       8'
      WRITE (*, *) ' 8   8  8 8   8  8    8 8    8 8     8'
      WRITE (*, *) ' 8    8 8 8    8  8888  8    8  88888 '
      WRITE (*, *) '                                      '

      RETURN
      END
* ======================================================================


* ======================================================================
*
      SUBROUTINE OpenAngleFiles (RadAng, FileDegAng, pi,
     &   directivity, geometry, eqnparts)
*
* ================== Prologue ==========================================
*
* Purpose:
*    Open all the files associated with one aspect angle.
```

```
*
* Processing:
*    Figure out file names and then open them.
*
* Special requirements:
*
* ------------------ Argument declarations -----------------------------

      DOUBLE PRECISION RadAng
      DOUBLE PRECISION FileDegAng
      DOUBLE PRECISION pi
      LOGICAL directivity
      LOGICAL geometry
      LOGICAL eqnparts

* ------------------ Local declarations --------------------------------

*     POutFile: full output filename
*     FourFile: fourier output filename
*     GeomFile: geomtery output filename
      CHARACTER*32 POutFile, FourFile, GeomFile

*     PartAFile: output filename for eqn part a
*     PartBFile: output filename for eqn part b
*     PartCFile: output filename for eqn part c
      CHARACTER*32 PartAFile, PartBFile, PartCFile

*     Huns: integer hundreds of current angle in degrees
*     Ones: integer ones of current angle in degrees
*     Tens: integer tens of current angle in degrees
      INTEGER Huns, Tens, Ones

*     LenFileName: length of the filename [integer]
      INTEGER LenFileName

* ------------------ Code ----------------------------------------------

*     Fix output filename so there are no negative angles.
      IF (RadAng .LT. 0.0) THEN
         FileDegAng = ( 2.0*pi + RadAng )*180.0/pi
      ELSE
         FileDegAng = ( RadAng + 1E-6 )*180.0/pi
      END IF

*     Create pressure vs. time output filename
      Huns = INT ( FileDegAng/100.0 )
      Tens = INT ( ( FileDegAng - ( Huns*100.0 ) )/10.0 )
      Ones = INT ( ( FileDegAng ) - ( Huns*100.0 + Tens*10.0 ) )
      POutFile = 'P'
      LenFileName = INDEX (POutFile, ' ') - 1
      POutFile = POutFile(:LenFileName) // '.' // CHAR (Huns + 48) //
     &   CHAR (Tens + 48) // CHAR (Ones + 48)

      OPEN (UNIT=52, FILE=POutFile, STATUS='UNKNOWN')
```

```
*     Open the Fourier output file. ---------------------------------
      IF (directivity) THEN

*        Create Fourier vs. freq output filename
         FourFile = 'F'
         LenFileName = INDEX (FourFile, ' ') - 1
         FourFile = FourFile(:LenFileName) // '.' // CHAR (Huns+ 48)
     &      // CHAR (Tens + 48) // CHAR (Ones + 48)

         OPEN (UNIT=53, FILE=FourFile, STATUS='UNKNOWN')

      END IF

*     Open the geom output file. -----------------------------------
      IF (geometry) THEN

         GeomFile = 'geom'
         LenFileName = INDEX (GeomFile, ' ') - 1
         GeomFile = GeomFile(:LenFileName) // '.' // CHAR (Huns+ 48)
     &      // CHAR (Tens + 48) // CHAR (Ones + 48)

         OPEN (UNIT=66, FILE=GeomFile, STATUS='UNKNOWN')

      END IF

*     Set up output filenames for the equation parts ----------------
      IF (eqnparts) THEN

         LenFileName = INDEX (POutFile,' ') - 1
         PartAFile = POutFile(:LenFileName) // '.P'
         PartBFile = POutFile(:LenFileName) // '.dPdT'
         PartCFile = POutFile(:LenFileName) // '.dPdN'

         OPEN (UNIT=56, FILE=PartAFile, STATUS='UNKNOWN')
         OPEN (UNIT=57, FILE=PartBFile, STATUS='UNKNOWN')
         OPEN (UNIT=58, FILE=PartCFile, STATUS='UNKNOWN')

      END IF

      RETURN
      END

* ====================================================================


* ====================================================================
*
      SUBROUTINE CloseAngleFiles (directivity, geometry, eqnparts)
*
* ================= Prologue ========================================
*
* Purpose:
*    Close all the files associated with one aspect angle.
*
* Processing:
```

```
*    Close files
*
* Special requirements:
*
* ------------------ Argument declarations -----------------------------

      LOGICAL directivity
      LOGICAL geometry
      LOGICAL eqnparts

* ------------------ Code ----------------------------------------------

*     Close the pressure output file.
      CLOSE (UNIT=52)

*     Close the Fourier analysis output file.
      IF (directivity) THEN
         CLOSE (UNIT=53)
      END IF

*     Close the geometry output file.
      IF (geometry) THEN
         CLOSE (UNIT=66)
      END IF

*     Close the equation parts files.
      IF (eqnparts) THEN
         CLOSE (UNIT=56)
         CLOSE (UNIT=57)
         CLOSE (UNIT=58)
      END IF

      RETURN
      END

* ====================================================================


* ====================================================================
*
      SUBROUTINE OutDirect (RadAng, FileDegAng, FFPress, pi, anglefiles,
     &   deltatime)
*
* ================== Prologue ========================================
*
* Purpose:
*    Write out the direcitivity and Fourier output.
*
* Processing:
*    Write write write
*
* Special requirements:
*
* ------------------ Include files -------------------------------------
```

```
*     ncirc: [INTEGER] Number of points around the cylindrical grid?
*     naxis: [INTEGER] Number of axial points in the grid?
*     ntimes: [INTEGER] The number of timesteps?
*     nlevels: [INTEGER] Number of grid levels?
      INCLUDE 'Kirch3.dmn'

* ------------------ Argument declarations -----------------------------

      DOUBLE PRECISION RadAng
      DOUBLE PRECISION FileDegAng
      DOUBLE PRECISION FFPress(0:ntimes - 1)
      DOUBLE PRECISION pi
      LOGICAL anglefiles
      DOUBLE PRECISION deltatime

* ------------------ Local declarations --------------------------------

*     Fourdex: Index through Fourier components
      INTEGER Fourdex

*     FourCN: Fourier coefficient
*     FourFreq: frequency of Fourier component
*     FourPhi: phase of Fourier components
      DOUBLE PRECISION FourPhi, FourCN, FourFreq

*     FourComp: Fourier components
      DOUBLE PRECISION FourComp(1:nfouriercomps)

*     FourdBComp: Fourier components in dB
      DOUBLE PRECISION FourdBComp(1:nfouriercomps)

* ------------------ Code ----------------------------------------------

*     Run through all Fourier components ----------------------------
      DO Fourdex = 1, nfouriercomps

*        Calculate the freq of interest
         FourFreq = ( Fourdex*1.0 )/( ntimes*deltatime )

*        Find the Fourier component at that freq
         CALL Fourier (FFPress, FourPhi, FourCN, FourFreq, deltatime)

*        Add it to the array of Fourier components
         FourComp(Fourdex) = FourCN
         FourdBComp(Fourdex) = 20.0*LOG10 (FourCN/2.0E-5)

*        Write that to the F.??? files (Fourier comp vs. freq)
         IF (anglefiles) THEN
            WRITE (53, 7010) FourFreq, FourdBComp(Fourdex),
     &         FourComp(Fourdex)
 7010       FORMAT (3 ( 1X, E24.17 ))
         ENDIF

      END DO
```

```
*     Write the components out to the directivity file.
      WRITE (50, 7020) RadAng*180.0/pi,
     &   ( FourdBComp(Fourdex), FourComp(Fourdex),
     &   Fourdex = 1, nfouriercomps )
*     The 21 in the following FORMAT statement sets the max number of
*     nfouriercomps.  If more than 10 are desired, 1+2*fouriercomps.
 7020 FORMAT (21 ( 1X, E24.17 ))

      RETURN
      END
* ====================================================================


* ====================================================================
*
      SUBROUTINE ReadGrid (X, Y, Z, dN, geometry)
*
* ================== Prologue ========================================
*
* Purpose: Read the cylindrical grid file and generate dN.
*
* Processing: Basically run through the array of the grid and figure
*    out the X, Y, and Z coords of each point then run through the
*    array again and calculate dN.
*
* Special requirements:
*
* ------------------ Include files ----------------------------------

*     ncirc: [INTEGER] Number of points around the cylindrical grid?
*     naxis: [INTEGER] Number of axial points in the grid?
*     ntimes: [INTEGER] The number of timesteps?
*     nlevels: [INTEGER] Number of grid levels?
      INCLUDE 'Kirch3.dmn'

* ------------------ Argument declarations --------------------------

*     X: X coord of the grid
*     Y: Y coord of the grid
*     Z: Z coord of the grid
      DOUBLE PRECISION X(naxis, ncirc, nlevels)
      DOUBLE PRECISION Y(naxis, ncirc, nlevels)
      DOUBLE PRECISION Z(naxis, ncirc, nlevels)

*     dN: distance between two gridpoints away from the KS
*        in the normal direction
      DOUBLE PRECISION dN(naxis, ncirc, nlevels - 1)

      LOGICAL geometry

* ------------------ Local declarations -----------------------------

*     Axidex: [INTEGER] Index in the axial direction.
*     Cirdex: [INTEGER] Index in the circuferenctial direction.
*     Lvldex: [INTEGER] Level index
```

```
      INTEGER Axidex, Cirdex, Lvldex

* ------------------ Code ----------------------------------------------

      OPEN (UNIT=29, FILE='Kirch3.grid', STATUS='UNKNOWN')

*     Set up for the geometry output
      IF (geometry) THEN
         OPEN (UNIT=50, FILE='geometry.K3', STATUS='UNKNOWN')
      END IF

      DO Axidex = 1, naxis
         DO Cirdex = 1, ncirc
            DO Lvldex = 1, nlevels

*              Read the coords of the grid from Kirch3.grid
               READ (29, 5000) X(Axidex, Cirdex, Lvldex),
     &            Y(Axidex, Cirdex, Lvldex),
     &            Z(Axidex, Cirdex, Lvldex)
 5000          FORMAT (3 ( 2X, E24.17 ))

*              Write out geometry information.
               IF (geometry) THEN
                  WRITE (50, 5000) X(Axidex, Cirdex, Lvldex),
     &            Y(Axidex, Cirdex, Lvldex),
     &            Z(Axidex, Cirdex, Lvldex)
               END IF

            END DO

*           Find dn's (the distances between normal blade points) ------

*           This assumes that the grid points in increasing levels are
*           along lines normal to the surface of the lowest level of
*           the grid.

            dN(Axidex, Cirdex, 1) = SQRT (
     &         ( X(Axidex, Cirdex, 1) - X(Axidex, Cirdex, 2) )**2.0 +
     &         ( Y(Axidex, Cirdex, 1) - Y(Axidex, Cirdex, 2) )**2.0 +
     &         ( Z(Axidex, Cirdex, 1) - Z(Axidex, Cirdex, 2) )**2.0 )

            dN(Axidex, Cirdex, 2) = SQRT (
     &         ( X(Axidex, Cirdex, 2) - X(Axidex, Cirdex, 3) )**2.0 +
     &         ( Y(Axidex, Cirdex, 2) - Y(Axidex, Cirdex, 3) )**2.0 +
     &         ( Z(Axidex, Cirdex, 2) - Z(Axidex, Cirdex, 3) )**2.0 )

         END DO

*        Let the user see progress
         WRITE (*, 2100) Axidex, naxis
 2100    FORMAT (' ReadGrid: step', I5, ' of', I5)

      END DO

      CLOSE (29)
```

```
*     Close the geometry output file.
      IF (geometry) THEN
         CLOSE (UNIT=50)
      END IF

      RETURN
      END

*   end ReadGrid subroutine

* ========================================================================


* ========================================================================
*
      SUBROUTINE ReadPress (P, dN, dPdN, dPdT, pi, density, soundspeed,
     &   deltatime, ffpdist, eqnparts)
*
* ================== Prologue ============================================
*
* Purpose:
*   This subroutine extracts information important to the prediction of
*   the radiated noise from the pressure file.
*
* Processing:
*
* Special requirements:
*
* ------------------ Include files -------------------------------------

*     ncirc: [INTEGER] Number of points around the cylindrical grid?
*     naxis: [INTEGER] Number of axial points in the grid?
*     ntimes: [INTEGER] The number of timesteps?
*     nlevels: [INTEGER] Number of grid levels?
      INCLUDE 'Kirch3.dmn'

* ------------------ Argument declarations -----------------------------

*     dN: distance between two gridpoints away from the KS
*        in the normal direction
      DOUBLE PRECISION dN(naxis, ncirc, nlevels - 1)

*     P: pressures on the KS
      DOUBLE PRECISION P(0:ntimes - 1, naxis, ncirc)

*     dPdN: the change in pressure on the normal away from the KS
      DOUBLE PRECISION dPdN(0:ntimes - 1, naxis, ncirc)

*     dPdT: change in pressure between time steps at any KS point
      DOUBLE PRECISION dPdT(0:ntimes - 1, naxis, ncirc)

      DOUBLE PRECISION pi
      DOUBLE PRECISION density
      DOUBLE PRECISION soundspeed
```

```
      DOUBLE PRECISION deltatime
      DOUBLE PRECISION ffpdist
      LOGICAL eqnparts

* ------------------ Local declarations --------------------------------

*     Axidex: Index in the axial direction.
*     Cirdex: Index in the circuferenctial direction.
      INTEGER Axidex, Cirdex

*     Timdex: time index
      INTEGER Timdex

*     Pr: pressures above the first gridlevel (temporary variable)
      DOUBLE PRECISION Pr(0:ntimes - 1, naxis, ncirc, nlevels - 1)

*     SOP: temp variable to remove the static press
      DOUBLE PRECISION SOP

* ------------------ Code ----------------------------------------------

      OPEN(UNIT=39, FILE='Kirch3.pres', STATUS='UNKNOWN')

      DO Timdex = 0, ntimes - 1
         DO Axidex = 1, naxis
            DO Cirdex = 1, ncirc

*              Read in the pressure
               READ (39, 3000) P(Timdex, Axidex, Cirdex),
     &            Pr(Timdex, Axidex, Cirdex, 1),
     &            Pr(Timdex, Axidex, Cirdex, 2)
 3000          FORMAT (3 ( 1x, E24.17 ) )

            END DO
         END DO

*        Let the user see progress
         WRITE (*, 3100) Timdex + 1, ntimes
 3100    FORMAT (' ReadPress: Finished with time step', I5, ' of', I5)

      END DO

      CLOSE (39)

*     Removing the static pressure term at each location on the blade --

*     Sum the pressures over the entire grid
      WRITE (*, *) 'ReadPress: Removing static P.'
      DO Timdex = 0, ntimes - 1
         DO Axidex = 1, naxis
            DO Cirdex = 1, ncirc

               SOP = SOP + P(Timdex, Axidex, Cirdex)

            END DO
```

```
         END DO
      END DO

*     Remove the average from everywhere on the grid
      DO Timdex = 0, ntimes - 1
         DO Axidex = 1, naxis
            DO Cirdex = 1, ncirc

               P(Timdex, Axidex, Cirdex) = P(Timdex, Axidex, Cirdex) -
     &            ( SOP/( 1.0*ntimes*naxis*ncirc ) )
               Pr(Timdex, Axidex, Cirdex, 1) =
     &            Pr(Timdex, Axidex, Cirdex, 1) -
     &            ( SOP/( 1.0*ntimes*naxis*ncirc ) )
               Pr(Timdex, Axidex, Cirdex, 2) =
     &            Pr(Timdex, Axidex, Cirdex, 2) -
     &            ( SOP/( 1.0*ntimes*naxis*ncirc ) )

            END DO
         END DO
      END DO

*     find dPdN term with a second order scheme ------------------------

      WRITE (*, *) 'ReadPress: Finding dPdN.'

      DO Timdex = 0, ntimes - 1
         DO Axidex = 1, naxis
            DO Cirdex = 1, ncirc

*              This is for a non-equally spaced grid,
*              but it works for equally spaced grids, too.
               dPdN(Timdex, Axidex, Cirdex) =
     &            ( ( -2.0*dN(Axidex, Cirdex, 1)*
     &            dN(Axidex, Cirdex, 2) -
     &            dN(Axidex, Cirdex, 2)**2.0 ) *
     &            P(Timdex, Axidex, Cirdex) +
     &            ( dN(Axidex, Cirdex, 1) +
     &            dN(Axidex, Cirdex, 2) )**2.0 *
     &            Pr(Timdex, Axidex, Cirdex, 1) -
     &            dN(Axidex, Cirdex, 1)**2.0 *
     &            Pr(Timdex, Axidex, Cirdex, 2) )/
     &            ( ( dN(Axidex, Cirdex, 1)*dN(Axidex, Cirdex, 2) )*
     &            ( dN(Axidex, Cirdex, 1) + dN(Axidex, Cirdex, 2) ) )


            END DO
         END DO
      END DO

*     calculate dPdT ---------------------------------------------------

      WRITE (*, *) 'ReadPress: Finding dPdT.'

      DO Axidex = 1, naxis
         DO Cirdex = 1, ncirc
```

```
*           Using a central differencing scheme and the assumption
*           that the pressures on the grid are exactly one period
            dPdT(0, Axidex, Cirdex) = ( P(1, Axidex, Cirdex) -
     &         P(ntimes - 1, Axidex, Cirdex) )/( 2.0*deltatime )
            dPdT(ntimes - 1, Axidex, Cirdex) = ( P(0, Axidex, Cirdex) -
     &         P(ntimes - 2, Axidex, Cirdex) )/( 2.0*deltatime )

         END DO
       END DO

       DO Timdex = 1, ntimes - 2
          DO Axidex = 1, naxis
             DO Cirdex = 1, ncirc

*               Using a central differencing scheme
                dPdT(Timdex, Axidex, Cirdex) =
     &          ( P(Timdex + 1, Axidex, Cirdex) -
     &          P(Timdex - 1, Axidex, Cirdex) )/( 2.0*deltatime )

             END DO
          END DO
       END DO

*      output dPKirch3.out ---------------------------------------------

*      This allows the user to look at the pressure values on the grid.
       IF (eqnparts) THEN

          WRITE (*, *) 'ReadPress: Writing P, dPdN, and dPdT.'

          OPEN (UNIT=64, FILE='dPKirch3.out', STATUS='UNKNOWN')

          DO Timdex = 0, ntimes - 1
             DO Axidex = 1, naxis
                Cirdex = 1

                   WRITE (64, 3200) P(Timdex, Axidex, Cirdex),
     &                dPdN(Timdex, Axidex, Cirdex),
     &                dPdT(Timdex, Axidex, Cirdex)
 3200              FORMAT (3 ( 1x, E24.17 ) )


             END DO
          END DO

          CLOSE (UNIT=64)

       END IF

       RETURN
       END

*    end ReadPress
```

```
* ================================================================


* ================================================================
*
      SUBROUTINE FindDistance (X, Y, Z, dN, dS, R, dRdN, Lag,
     &   RadAng, FileDegAng, pi, soundspeed, deltatime,
     &   ffpdist, geometry)
*
* ================== Prologue ============================================
*
* Purpose:
*   finddistance finds the distance between each point on the KS
*   and the far field point.
*
* Processing:
*
* Special requirements:
*
* ------------------ Include files -------------------------------------

*     ncirc: [INTEGER] Number of points around the cylindrical grid?
*     naxis: [INTEGER] Number of axial points in the grid?
*     ntimes: [INTEGER] The number of timesteps?
*     nlevels: [INTEGER] Number of grid levels?
      INCLUDE 'Kirch3.dmn'

* ------------------ Argument declarations -----------------------------

*     X: X coord of the grid
*     Y: Y coord of the grid
*     Z: Z coord of the grid
      DOUBLE PRECISION X(naxis, ncirc, nlevels)
      DOUBLE PRECISION Y(naxis, ncirc, nlevels)
      DOUBLE PRECISION Z(naxis, ncirc, nlevels)

*     dN: distance between two gridpoints away from the KS
*        in the normal direction
      DOUBLE PRECISION dN(naxis, ncirc, nlevels - 1)

*     R: distance between KS point and far field point
*     dRdN: change in R in normal direction from KS
*     dS: length of effect of pressure on the computational surface
      DOUBLE PRECISION R(naxis, ncirc, nlevels)
      DOUBLE PRECISION dRdN(naxis, ncirc)
      DOUBLE PRECISION dS(naxis, ncirc)

*     Lag: number of Timdexs of lag for any KS point
      DOUBLE PRECISION Lag(naxis, ncirc)

*     RadAng: radian angle of current calculation
      DOUBLE PRECISION RadAng

*     FileDegAng: Angle used for filenames
      DOUBLE PRECISION FileDegAng
```

```
      DOUBLE PRECISION pi
      DOUBLE PRECISION soundspeed
      DOUBLE PRECISION deltatime
      DOUBLE PRECISION ffpdist
      LOGICAL geometry

* ------------------ Local declarations --------------------------------

*     Axidex: Index in the axial direction.
*     Cirdex: Index in the circuferenctial direction.
*     Lvldex: Level index
      INTEGER Axidex, Cirdex, Lvldex

*     XSP: x coord of the surface point
*     YSP: y coord of the surface point
*     ZSP: z coord of the surface point
*     XFP: x coord of the far field point
*     YFP: y coord of the far field point
*     ZFP: z coord of the far field point
      DOUBLE PRECISION XSP, YSP, ZSP, XFP, YFP, ZFP

*     INTEGER Huns, Tens, Ones
*     CHARACTER*32 ROutFile

*     Radius: Radius of dS
*     AxiLen: "axial" length of dS
      DOUBLE PRECISION Radius, AxiLen

*     Xp1: x coordinate of the midpoint between Index - 1 and Index
*     Xp2: x coordinate of the midpoint between Index and Index + 1
*     Yp1: y coordinate of the midpoint between Index - 1 and Index
*     Yp2: y coordinate of the midpoint between Index and Index + 1
*     Zp1: z coordinate of the midpoint between Index - 1 and Index
*     Zp2: z coordinate of the midpoint between Index and Index + 1
      DOUBLE PRECISION Xp1, Xp2, Yp1, Yp2, Zp1, Zp2

* ------------------ Code ----------------------------------------------

* Calculate R's for all grid points. ----------------------------
      DO Axidex = 1, naxis
         DO Cirdex = 1, ncirc
            DO Lvldex = 1, nlevels

*              Farfield points
               XFP = ffpdist*COS (RadAng)
               YFP = ffpdist*SIN (RadAng)
               ZFP = 0.0

*              Surface points
               XSP = X(Axidex, Cirdex, Lvldex)
               YSP = Y(Axidex, Cirdex, Lvldex)
               ZSP = Z(Axidex, Cirdex, Lvldex)

               R(Axidex, Cirdex, Lvldex) =
```

```
     &            SQRT (( XFP - XSP )**2.0 +
     &            ( YFP - YSP )**2.0 +
     &            ( ZFP - ZSP )**2.0)

            END DO

*           Find dRdN term with a second order scheme ------------------

*           This is the formulation for a non-equally spaced grid,
*           but works just as well for an equally spaced one.
            dRdN(Axidex, Cirdex) =
     &         -( ( -2.0*dN(Axidex, Cirdex, 1)*
     &         dN(Axidex, Cirdex, 2) -
     &         dN(Axidex, Cirdex, 2)**2.0 ) *
     &         R(Axidex, Cirdex, 1) +
     &         ( dN(Axidex, Cirdex, 1) +
     &         dN(Axidex, Cirdex, 2) )**2.0 *
     &         R(Axidex, Cirdex, 2) -
     &         dN(Axidex, Cirdex, 1)**2.0 *
     &         R(Axidex, Cirdex, 3) )/
     &         ( ( dN(Axidex, Cirdex, 1)*dN(Axidex, Cirdex, 2) )*
     &         ( dN(Axidex, Cirdex, 1) + dN(Axidex, Cirdex, 2) ) )

*           Calculate Lag for this surface point.
*           This algorithm calculates the lag relative to the FFP
            Lag(Axidex, Cirdex) = -R(Axidex, Cirdex, 1)/
     &         ( soundspeed*deltatime )

         END DO
      END DO

*     Finding trapezoidal weights. -------------------------------------

      DO Cirdex = 1, ncirc

*        Once for the starting point.
         Xp1 = X(1, Cirdex, 1)
         Xp2 = ( X(1, Cirdex, 1) + X(2, Cirdex, 1) )/2.0
         Yp1 = 0.0
         Yp2 = ( Y(1, Cirdex, 1) + Y(2, Cirdex, 1) )/2.0
         Zp1 = 0.0
         Zp2 = ( Z(1, Cirdex, 1) + Z(2, Cirdex, 1) )/2.0
         Radius = SQRT ( Y(1, Cirdex, 1)**2.0 +
     &      Z(1, Cirdex, 1)**2.0 )
         AxiLen = SQRT ( ( Xp2 - Xp1 )**2.0 +
     &      ( Yp2 - Yp1 )**2.0 +
     &      ( Zp2 - Zp1 )**2.0 )
         dS(1, Cirdex) = 2.0*pi*Radius*AxiLen/ncirc

*        Once for the ending point.
         Xp1 = ( X(naxis - 1, Cirdex, 1) + X(naxis, Cirdex, 1) )/2.0
         Xp2 = X(naxis, Cirdex, 1)
         Yp1 = ( Y(naxis - 1, Cirdex, 1) + Y(naxis, Cirdex, 1) )/2.0
         Yp2 = 0.0
         Zp1 = ( Z(naxis - 1, Cirdex, 1) + Z(naxis, Cirdex, 1) )/2.0
```

```
         Zp2 = 0.0
         Radius = SQRT ( Y(naxis, Cirdex, 1)**2.0 +
     &      Z(naxis, Cirdex, 1)**2.0 )
         AxiLen = SQRT ( ( Xp2 - Xp1 )**2.0 +
     &      ( Yp2 - Yp1 )**2.0 +
     &      ( Zp2 - Zp1 )**2.0 )
         dS(naxis, Cirdex) = 2.0*pi*Radius*AxiLen/ncirc

      END DO

*     And once for everything in between
      DO Axidex = 2, naxis - 1
         DO Cirdex = 1, ncirc

            Xp1 = ( X(Axidex - 1, Cirdex, 1) + X(Axidex, Cirdex, 1) )
     &         /2.0
            Xp2 = ( X(Axidex, Cirdex, 1) + X(Axidex + 1, Cirdex, 1) )
     &         /2.0
            Yp1 = ( Y(Axidex - 1, Cirdex, 1) + Y(Axidex, Cirdex, 1) )
     &         /2.0
            Yp2 = ( Y(Axidex, Cirdex, 1) + Y(Axidex + 1, Cirdex, 1) )
     &         /2.0
            Zp1 = ( Z(Axidex - 1, Cirdex, 1) + Z(Axidex, Cirdex, 1) )
     &         /2.0
            Zp2 = ( Z(Axidex, Cirdex, 1) + Z(Axidex + 1, Cirdex, 1) )
     &         /2.0
            Radius = SQRT ( Y(Axidex, Cirdex, 1)**2.0 +
     &         Z(Axidex, Cirdex, 1)**2.0 )
            AxiLen = SQRT ( ( Xp2 - Xp1 )**2.0 +
     &         ( Yp2 - Yp1 )**2.0 +
     &         ( Zp2 - Zp1 )**2.0 )
            dS(Axidex, Cirdex) = 2.0*pi*Radius*AxiLen/ncirc

         END DO
      END DO

*     Write R, dRdN, and dS to the geometry output file. --------------
      IF (geometry) THEN

         WRITE (*,*) 'FindDistance: Writing geometry output.'

         DO Axidex = 1, naxis
            DO Cirdex = 1, ncirc

               WRITE (66, 9000) R(Axidex, Cirdex, 1),
     &            dRdN(Axidex, Cirdex), dS(Axidex, Cirdex),
     &            Lag(Axidex, Cirdex)
 9000          FORMAT (4( 2X, E24.17 ))

            END DO
         END DO

      END IF

      RETURN
```

```
      END

*   End finddistance subroutine

* ========================================================================


* ========================================================================
*
      DOUBLE PRECISION FUNCTION FindRadPress (P, dPdN, dPdT,
     &         X, Y, Z, dS, R, dRdN, Lag, Timdex,
     &         eqnparts, pi, soundspeed, deltatime)
*
* ================== Prologue =============================================
*
* Purpose:
*   This suborutine adds the different pressures around the KS
*   together for each time step and then returns it.
*
* Processing:
*
* Special requirements:
*
* ------------------ Include files -------------------------------------

*     ncirc: [INTEGER] Number of points around the cylindrical grid?
*     naxis: [INTEGER] Number of axial points in the grid?
*     ntimes: [INTEGER] The number of timesteps?
*     nlevels: [INTEGER] Number of grid levels?
      INCLUDE 'Kirch3.dmn'

* ------------------ Argument declarations -----------------------------

*     P: pressures on the KS
      DOUBLE PRECISION P(0:ntimes - 1, naxis, ncirc)

*     dPdN: the change in pressure on the normal away from the KS
      DOUBLE PRECISION dPdN(0:ntimes - 1, naxis, ncirc)

*     dPdT: change in pressure between time steps at any KS point
      DOUBLE PRECISION dPdT(0:ntimes - 1, naxis, ncirc)

*     X: X coord of the grid
*     Y: Y coord of the grid
*     Z: Z coord of the grid
      DOUBLE PRECISION X(naxis, ncirc, nlevels)
      DOUBLE PRECISION Y(naxis, ncirc, nlevels)
      DOUBLE PRECISION Z(naxis, ncirc, nlevels)

*     R: distance between KS point and far field point
*     dRdN: change in R in normal direction from KS
*     dS: length of effect of pressure on the computational surface
      DOUBLE PRECISION R(naxis, ncirc, nlevels)
      DOUBLE PRECISION dRdN(naxis, ncirc)
      DOUBLE PRECISION dS(naxis, ncirc)
```

```
*     Lag: number of Timdexs of lag for any KS point
      DOUBLE PRECISION Lag(naxis, ncirc)

*     Timdex: Time step
      INTEGER Timdex

      LOGICAL eqnparts
      DOUBLE PRECISION pi
      DOUBLE PRECISION soundspeed
      DOUBLE PRECISION deltatime

* ------------------ Local declarations --------------------------------

*     IT1: Timdex before actual time
*     IT2: Timdex after actual time
      INTEGER IT1, IT2

*     Axidex: [INTEGER] Index in the axial direction.
*     Cirdex: [INTEGER] Index in the circuferenctial direction.
      INTEGER Axidex, Cirdex

*     Y1: eqn value at IT1
*     Y2: eqn value at IT2
*     D1: distance from IT1 to TimeShifted
*     D2: distance from IT2 to TimeShifted
*     TempTime: temporary time calculation variable
*     TimeShifted: actual time of calculation
      DOUBLE PRECISION Y1, Y2, D1, D2, TempTime, TimeShifted

*     A: value of p term at TimeShifted
*     A1: value of p term at IT1
*     A2: value of p term at IT2
      DOUBLE PRECISION A, A1, A2

*     B: value of dPdT term at TimeShifted
*     B1: value of dPdT term at IT1
*     B2: value of dPdT term at IT2
      DOUBLE PRECISION B, B1, B2

*     C: value of dPdN term at TimeShifted
*     C1: value of dPdN term at IT1
*     C2: value of dPdN term at IT2
      DOUBLE PRECISION C, C1, C2

*     FrntConst: Constant to multiply integration terms.
      DOUBLE PRECISION FrntConst

*     SumOfP: Summing pressure variable
      DOUBLE PRECISION SumOfP

* ------------------ Code ----------------------------------------------

*     Front constant
      FrntConst = 1.0/( 4.0*pi )
```

```
*     Reset cummulative variables
      SumOfP = 0.0

      IF (eqnparts) THEN
         A = 0.0
         B = 0.0
         C = 0.0
      END IF

      DO Axidex = 1, naxis
         DO Cirdex = 1, ncirc

*           Add the lag to the time
*           (This creates the "retarded (emission) time")
            TempTime = 1.0*Timdex + Lag(Axidex, Cirdex)

*           I HATE GOTO statements, but it's the only way to make a
*           WHILE-DO loop in Fortran.  Ugh.
*           This makes sure that TempTime is >= 0
 4000       IF ( TempTime .LT. 0.0 ) THEN
               TempTime = TempTime + ntimes*1.0
            GOTO 4000
            END IF

*           This makes sure that TimeShifted is < ntimes and accurately
*           wrapped to the correct place in the period.
            TimeShifted = MOD ( TempTime, ntimes*1.0 )

*           TimeShifted should be between IT1 and IT2
            IT1 = INT (TimeShifted)
            IT2 = MOD (IT1 + 1, ntimes)

*           Where is TimeShifted placed between them?
            D1 = TimeShifted - 1.0*IT1
            D2 = 1.0 - D1

*           The pressure is calculated in parts and then summed.

            A1 = P(IT1, Axidex, Cirdex)*dRdN(Axidex, Cirdex)/
     &         R(Axidex, Cirdex, 1)**2.0
            B1 = dRdN(Axidex, Cirdex)*dPdT(IT1, Axidex, Cirdex)/
     &         ( soundspeed*R(Axidex, Cirdex, 1) )
            C1 = dPdN(IT1, Axidex, Cirdex)/R(Axidex, Cirdex, 1)
            Y1 = ( A1 + B1 - C1 )

            A2 = P(IT2, Axidex, Cirdex)*dRdN(Axidex, Cirdex)/
     &         R(Axidex, Cirdex, 1)**2.0
            B2 = dRdN(Axidex, Cirdex)*dPdT(IT2, Axidex, Cirdex)/
     &         ( soundspeed*R(Axidex, Cirdex, 1) )
            C2 = dPdN(IT2, Axidex, Cirdex)/R(Axidex, Cirdex, 1)
            Y2 = ( A2 + B2 - C2 )

*           Cummulative variable
            SumOfP = SumOfP + FrntConst*dS(Axidex, Cirdex)*
```

```
     &         ( Y1*D2 + Y2*D1 )

*           Accumulate the equation part sums
            IF (eqnparts) THEN
               A = A + FrntConst*dS(Axidex, Cirdex)*( A1*D2 + A2*D1 )
               B = B + FrntConst*dS(Axidex, Cirdex)*( B1*D2 + B2*D1 )
               C = C + FrntConst*dS(Axidex, Cirdex)*( C1*D2 + C2*D1 )
            END IF

         END DO
      END DO

*     Send out the value for the function
      FindRadPress = SumOfP

*     Output the equation parts
      IF (eqnparts) THEN
         WRITE (56, 4020) (1.0*Timdex)*deltatime, A
         WRITE (57, 4020) (1.0*Timdex)*deltatime, B
         WRITE (58, 4020) (1.0*Timdex)*deltatime, C
      END IF

 4020 FORMAT (2( 2X, E24.17 ))

      RETURN
      END

*     End FindRadPress procedure

* ======================================================================
```

A.4. Analytical-Numerical Kirchhoff Program

A.4.1. K3A.rc

```
* ----------------------------------------------------------------------
* K3A.rc is read by K3A.f to get runtime conditions.
* NOTE: ksrbegin cannot = 0.0, n???steps = 0 sets ??? to ???begin

* beautygrid: [LOGICAL] beautify output grid for gnuplot's splot
FALSE

* ksrbegin: [DOUBLE PRECISION] starting Kirchhoff surface radius (KSRad)
2.0

* ksrend: [DOUBLE PRECISION] ending Kirchhoff surface radius (KSRad)
2.0

* nksrsteps: [INTEGER] number of steps between ksrbegin and ksrend
```

```
0

* ffdbegin: [DOUBLE PRECISION] starting far-field distance (FFDist)
10.0

* ffdend: [DOUBLE PRECISION] ending far-field distance (FFDist)
10.5

* nffdsteps: [INTEGER] number of steps between ffdbegin and ffdend
20

* ------------------------------------------------------------------------
```

A.4.2. K3A.f

```
* ========================================================================
*
      PROGRAM K3A
*
* ================== Prologue ============================================
*
* $Id: K3A.f,v 1.25 1997/05/30 14:55:50 nodog Exp nodog $
*
* Original program written by
* Anderson Mills <nodog@sabine.acs.psu.edu>
* at the Graduate Program in Acoustics
* at Pennsylvania State University
*
* Purpose:
*    Gives analytical results for Kirch3 based on a spherical geometry.
*
* Special requirements:
*    Kirch3.dmn, Kirch3.rc, and Press.rc must exist.
*
* ------------------ Unit Numbers ---------------------------------------

*   40   Kirch3.rc

*   70   K3A.rc
*   78   K3A.out

* ------------------ Argument declarations ------------------------------

*     Constants from K3A.rc --- See K3A.rc for descriptions
      LOGICAL beautygrid
      DOUBLE PRECISION ksrbegin
      DOUBLE PRECISION ksrend
      INTEGER nksrsteps
      DOUBLE PRECISION ffdbegin
      DOUBLE PRECISION ffdend
      INTEGER nffdsteps

*     Constants from Kirch3.rc --- see Kirch3.rc for description.
```

```
      LOGICAL anglefiles
      LOGICAL directivity
      LOGICAL geometry
      LOGICAL eqnparts
      DOUBLE PRECISION density
      DOUBLE PRECISION soundspeed
      DOUBLE PRECISION deltatime
      DOUBLE PRECISION ffpdist
      DOUBLE PRECISION mach
      DOUBLE PRECISION aspstart
      DOUBLE PRECISION aspend
      DOUBLE PRECISION aspint


* ------------------ Code ----------------------------------------------

* Main Program

      CALL Banner ()

*     Read in constants from K3A.rc ----------------------------------
      CALL GetAnConst (beautygrid, ksrbegin, ksrend, nksrsteps,
     &   ffdbegin, ffdend, nffdsteps)

*     Read in constants from Kirch3.rc -------------------------------
      CALL GetK3Const (anglefiles, directivity,
     &   geometry, eqnparts, density, soundspeed, deltatime,
     &   ffpdist, mach, aspstart, aspend, aspint)

*     Do the Analytical calculations ---------------------------------
      CALL Analytical (beautygrid, ksrbegin, ksrend, nksrsteps,
     &   ffdbegin, ffdend, nffdsteps, soundspeed, deltatime)

      CALL Banner ()

      STOP
      END

* Main Program

* ======================================================================


* ======================================================================
*
      SUBROUTINE Banner()
*
* ================== Prologue ==========================================
*
* Purpose: Show the beginning and ending of the program.
*
* ------------------ Code ----------------------------------------------

      WRITE (*, *) '                        '
      WRITE (*, *) ' 8    8  88888     8    '
```

```
      WRITE (*, *) ' 8   8  8     8   8 8   '
      WRITE (*, *) ' 8  8         8  8   8  '
      WRITE (*, *) ' 888     88888  8     8 '
      WRITE (*, *) ' 8  8         8 8888888 '
      WRITE (*, *) ' 8   8  8     8 8     8 '
      WRITE (*, *) ' 8    8  88888  8     8 '
      WRITE (*, *) '                        '

      RETURN
      END

* ======================================================================


* ======================================================================
*
      SUBROUTINE GetAnConst (beautygrid, ksrbegin, ksrend, nksrsteps,
     &   ffdbegin, ffdend, nffdsteps)
*
* ================== Prologue ==========================================
*
* Purpose:
*    To read in the constants from the K3A.rc file
*
* Processing:
*    Uses READ's to get the info in.
*
* Special requirements:
*    Gotta have something to read, so the K3A.rc file must exist.
*
* ------------------ Argument declarations -----------------------------

*     Constants from K3A.rc --- See K3A.rc for descriptions
      LOGICAL beautygrid
      DOUBLE PRECISION ksrbegin
      DOUBLE PRECISION ksrend
      INTEGER nksrsteps
      DOUBLE PRECISION ffdbegin
      DOUBLE PRECISION ffdend
      INTEGER nffdsteps

* ------------------ Local declarations --------------------------------

      CHARACTER Dummy

* ------------------ Code ----------------------------------------------

      OPEN (UNIT=70, FILE='K3A.rc', STATUS='UNKNOWN')

*     Three lines of comments at the top.
      READ (70, 5000) Dummy
      READ (70, 5000) Dummy
      READ (70, 5000) Dummy

*     Read constants from K3A.rc
```

```
*     One blank line, one label line, then the constant.
      READ (70, 5000) Dummy
      READ (70, 5000) Dummy
      READ (70, 5010) beautygrid
      READ (70, 5000) Dummy
      READ (70, 5000) Dummy
      READ (70, 5020) ksrbegin
      READ (70, 5000) Dummy
      READ (70, 5000) Dummy
      READ (70, 5020) ksrend
      READ (70, 5000) Dummy
      READ (70, 5000) Dummy
      READ (70, 5030) nksrsteps
      READ (70, 5000) Dummy
      READ (70, 5000) Dummy
      READ (70, 5020) ffdbegin
      READ (70, 5000) Dummy
      READ (70, 5000) Dummy
      READ (70, 5020) ffdend
      READ (70, 5000) Dummy
      READ (70, 5000) Dummy
      READ (70, 5030) nffdsteps

      CLOSE (70)

 5000 FORMAT (A)
 5010 FORMAT (L5)
 5020 FORMAT (F24.17)
 5030 FORMAT (I9)

      RETURN
      END

* ======================================================================


* ======================================================================

      SUBROUTINE Analytical (beautygrid, ksrbegin, ksrend, nksrsteps,
     &   ffdbegin, ffdend, nffdsteps, soundspeed, deltatime)

* ================== Prologue =============================================
*
* Purpose:
*    To calculate far-field pressure for a dipole in a
*    spherical Kirchhoff surface using analytical values for the
*    equation terms.  This is effectively an analytical version of
*    the Grid/Press/Kirch3 combination.
*
* Processing:
*    Uses values from Kirch3.rc to find the frequency and soundspeed
*    and uses values from Kirch3.dmn for array sizes.  Output is
*    several files giving output pressure over several Kirchhoff
*    surface radii (KSR) and/or far-field distances (FFDist).
*
```

```
* Special requirements:
*    Kirch3.rc, Kirch3.dmn, and K3A.rc must all exist.
*
* ------------------ Include files -------------------------------------

*     ncirc: [INTEGER] Number of points around the cylindrical grid?
*     naxis: [INTEGER] Number of axial points in the grid?
*     ntimes: [INTEGER] The number of timesteps?
*     nlevels: [INTEGER] Number of grid levels?
      INCLUDE 'Kirch3.dmn'

* ------------------ Argument declarations -----------------------------

*     Variables from K3A.rc --- See K3A.rc for descriptions
      LOGICAL beautygrid
      DOUBLE PRECISION ksrbegin
      DOUBLE PRECISION ksrend
      INTEGER nksrsteps
      DOUBLE PRECISION ffdbegin
      DOUBLE PRECISION ffdend
      INTEGER nffdsteps

*     Constants from Kirch3.rc --- see Kirch3.rc for description.
      DOUBLE PRECISION soundspeed
      DOUBLE PRECISION deltatime

* ------------------ Local declarations --------------------------------

      DOUBLE PRECISION pi

*     frntconst: 1/(4pi)
      DOUBLE PRECISION frntconst

*     Omega: The angular frequency = 2pif
*     K: The wavenumber = Omega/soundspeed
*     Strength: The strength of the dipole (currently = FFDist to
*               give 1 Pa at the FFDist
      DOUBLE PRECISION Omega, K, Strength

*     KSRad: The Kirchhoff surface radius
      DOUBLE PRECISION KSRad

*     Sigma: Angle around X going from Y towards Z
*     Theta: Angle from the X axis
      DOUBLE PRECISION Sigma, Theta

*     Phi: Angle of the far-field point with the X axis
      DOUBLE PRECISION Phi

*     Time: The time in seconds at a particular Timedex
      DOUBLE PRECISION Time

*     dS: The area of a trapezoidal patch for integration
      DOUBLE PRECISION dS
```

```
*     R: Distance from KSP to FFP
*     dRdN: normal derivative of Distance from KSP to FFP
      DOUBLE PRECISION R
      DOUBLE PRECISION dRdN

*     P: pressure
      DOUBLE PRECISION P

*     dPdN: normal derivative of the pressure
      DOUBLE PRECISION dPdN

*     dPdT: time derivative of the pressure
      DOUBLE PRECISION dPdT

*     Term?: Value for the ? term of the Kirchhoff equation
      DOUBLE PRECISION TermA, TermB, TermC

*     Part?: Storage of the ? part of the Kirchhoff equation
      DOUBLE PRECISION PartA, PartB, PartC

*     DipDirect: dipole directivity
      DOUBLE PRECISION DipDirect

*     Xf, Yf: FFP coords
*     Xk, Yk, Zk: KSP coords
      DOUBLE PRECISION Xf, Yf, Xk, Yk, Zk

*     h: height for dS calculations
      DOUBLE PRECISION h

*     Tau: DOUBLE PRECISION time for calculations
      DOUBLE PRECISION Tau

*     I: square root of -1
*     Etothe: E**i(kr-wt)
      DOUBLE COMPLEX I, Etothe

*     PFFP: Pressure contribution at the far-field point
      DOUBLE PRECISION PFFP(0:ntimes - 1)

*     PFFPContrib: Pressure term contribution at the far-field point
      DOUBLE PRECISION PFFPContrib(0:ntimes - 1)

*     dPdTFFPContrib: dPdT term contribution at the far-field point
      DOUBLE PRECISION dPdTFFPContrib(0:ntimes - 1)

*     dPdNFFPContrib: dPdN term contribution at the far-field point
      DOUBLE PRECISION dPdNFFPContrib(0:ntimes - 1)

*     PFour, PFourdB: Pa and dB Fourier coeff of whole integral
*     PC, PCdB: Pa and dB Fourier coeff of pressure term
      DOUBLE PRECISION PFour, PFourdB, PC, PCdB

*     dPdTC, dPdTCdB: Pa and dB Fourier coeff of dPdT term
*     dPdNC, dPdNCdB: Pa and dB Fourier coeff of dPdN term
```

```
      DOUBLE PRECISION dPdTC, dPdTCdB, dPdNC, dPdNCdB

*     Freq: Operating frequency
      DOUBLE PRECISION Freq

*     FourPhi: Phase of Fourier component
*     FourCN: Magnitude of Fourier component
      DOUBLE PRECISION FourPhi, FourCN

*     ???dex: Indexes for various loops
      INTEGER Axidex, Cirdex, Raddex, Timdex, Distdex

* ------------------ Code ----------------------------------------------

      pi = 4.0*ATAN (1.0)
      I  = (0.0, 1.0)

      WRITE (*, *) '   constants of interest'
      WRITE (*, *) '   ---------------------'
      Freq = 1.0/( ntimes*deltatime )
      WRITE (*, *) '        Freq =', Freq
      Omega = (2.0*pi)/(deltatime*ntimes)
      WRITE (*, *) '       Omega =', Omega
      K = Omega/soundspeed
      WRITE (*, *) '           K =', K
      Phi = 6.283
      WRITE (*, *) '         Phi =', Phi
      frntconst = 1.0/(4.0*pi)
      WRITE (*, *) '   frntconst =', frntconst
      WRITE (*, *) ' '

      OPEN (UNIT=78, FILE='K3A.out', STATUS='UNKNOWN')

*     Radius loop
      DO Raddex = 0, nksrsteps

*        Distance loop
         DO Distdex = 0, nffdsteps

*           Calculate KSRad, set to ksrbegin if nksrsteps = 0
            IF ( nksrsteps .EQ. 0 ) THEN
               KSRad = ksrbegin
            ELSE
               KSRad = ( 1.0*Raddex )*( ( ksrend - ksrbegin )/
     &            ( 1.0*nksrsteps ) )+ ksrbegin
            ENDIF

*           Calculate FFDist, set to ffdbegin if nffdsteps = 0
            IF ( nffdsteps .EQ. 0 ) THEN
               FFDist = ffdbegin
            ELSE
               FFDist = ( 1.0*Distdex )*( ( ffdend - ffdbegin )/
     &            ( 1.0*nffdsteps ) ) + ffdbegin
            ENDIF
```

```
*           Set the strength of the dipole to the FFDist so output
*           will always be 1 Pa at whatever FFDist is chosen
            Strength = FFDist

*           Calculate the far-field point coords
            Xf = FFDist*COS (Phi)
            Yf = FFDist*SIN (Phi)

*           Give and indication to the user of progress
            WRITE (*, 1000) KSRad, FFDist
 1000       FORMAT ('        KSRad = ', F6.3, '   FFDist = ', F6.3)

*           Time loop
            DO Timdex = 0, ntimes - 1

*              Calculate the time and reset cumulative variables
               Time = Timdex*deltatime
               TermA = 0.0
               TermB = 0.0
               TermC = 0.0

*              Axis loop
               DO Axidex = 1, naxis

*                 Calculate angle from X axis for a spherical grid.
                  Theta = ( pi*( Axidex - 0.5 ) )/naxis

*                 Calculate the area of the integration patch.
                  h = KSRad*ABS ( ( COS (pi*( Axidex - 1 )/naxis) ) -
     &               ( COS (pi*Axidex/naxis) ) )
                  dS = 2.0*pi*KSRad*h/ncirc

*                 Source directivity
                  DipDirect = COS (2.*Theta)

*                 Circumferential loop
                  DO Cirdex = 1, ncirc

*                    Find the angle around the X axis
                     Sigma = ( 2.0*pi*Cirdex )/( ncirc )

*                    Calculate the Kirchhoff surface point coords
                     Xk = KSRad*COS (Theta)
                     Yk = KSRad*SIN (Theta)*COS (Sigma)
                     Zk = KSRad*SIN (Theta)*SIN (Sigma)

*                    Helping constants for the Kirchhoff equation
                     R = SQRT ( ( Xk - Xf )**2.0 + ( Yk - Yf )**2.0 +
     &                  Zk**2.0 )
                     dRdN = ( Xk*( Xf - Xk ) + Yk*( Yf - Yk ) -
     &                  Zk**2.0 )/ABS ( KSRad*R )
                     Tau = Time - R/soundspeed
                     Etothe = EXP ( I*( K*KSRad - Omega*Tau ) )
                     P = DBLE ( Strength*DipDirect*Etothe/KSRad )
                     dPdN = DBLE ( Strength*DipDirect*Etothe*
```

```
     &                  ( ( I*K*KSRad - 1.0 )/( KSRad**2.0 ) ) )
                      dPdT = DBLE ( ( -I*Strength*DipDirect*Omega/
     &                  KSRad )*Etothe )

*                     Storage of the parts
                      PartA = P*dRdN/R**2.0
                      PartB = dPdT*dRdN/( R*soundspeed )
                      PartC = - dPdN/R

*                     Cumulative value variables
                      TermA = TermA + ( frntconst*dS*PartA )
                      TermB = TermB + ( frntconst*dS*PartB )
                      TermC = TermC + ( frntconst*dS*PartC )

                   END DO
*                  Cirdex loop

                END DO
*               Axidex loop

*               Set the time history variable to this Timdex's cum. var.
                PFFPContrib(Timdex) = TermA
                dPdTFFPContrib(Timdex) = TermB
                dPdNFFPContrib(Timdex) = TermC
                PFFP(Timdex) = TermA + TermB + TermC

             END DO
*            Timdex loop

* ----------Fourier calculations---------------------------------------

             CALL Fourier (PFFP, FourPhi, FourCN, Freq, deltatime)

             PFour = FourCN
             PFourdB = 20.0*LOG10 (FourCN/2.0E-5)

             CALL Fourier (PFFPContrib, FourPhi, FourCN, Freq,
     &          deltatime)

             PC = FourCN
             PCdB = 20.0*LOG10 (FourCN/2.0E-5)

             CALL Fourier (dPdTFFPContrib, FourPhi, FourCN, Freq,
     &          deltatime)

             dPdTC = FourCN
             dPdTCdB = 20.0*LOG10 (FourCN/2.0E-5)

             CALL Fourier (dPdNFFPContrib, FourPhi, FourCN, Freq,
     &          deltatime)

             dPdNC = FourCN
             dPdNCdB = 20.0*LOG10 (FourCN/2.0E-5)

*            Output to the output file.
```

```
            WRITE (78, 1040) KSRad*Freq/soundspeed, FFDist, PFour,
     &         PFourdB, PC, PCdB, dPdTC, dPdTCdB, dPdNC, dPdNCdB
 1040       FORMAT (10 ( 1X, E24.17 ))

         END DO
*        Distdex loop

*        Beautify the grid for gnuplot's splot.
         IF ( beautygrid ) THEN
            WRITE (78, 1050) ''
 1050       FORMAT (A)
         ENDIF

      END DO
*     Raddex loop

      CLOSE (UNIT=78)

      RETURN
      END

* Analytical

* ========================================================================
```

A.5. Common Subroutines

A.5.1. Fourier.f

```
* ========================================================================
*
      SUBROUTINE Fourier (Funct, FourPhi, FourCN, FourFreq, deltatime)
*
* ================== Prologue ============================================
*
* Purpose:
*     determines the Fourier coefficient, Cn, and the phase shift,
*     FourPhi, given a function, f, at kp discrete times, t, where
*     t(k) is the time sample size and n is the frequency bin number.
*
* Processing:
*
* Special requirements:
*
* ------------------ Include files -------------------------------------

*     ncirc: [INTEGER] Number of points around the cylindrical grid?
*     naxis: [INTEGER] Number of axial points in the grid?
*     ntimes: [INTEGER] The number of timesteps?
*     nlevels: [INTEGER] Number of grid levels?
      INCLUDE 'Kirch3.dmn'
```

```
* ------------------ Argument declarations -----------------------------

      DOUBLE PRECISION Funct(0: ntimes - 1)
      DOUBLE PRECISION FourPhi, FourCN, FourFreq
      DOUBLE PRECISION deltatime

* ------------------ Local declarations --------------------------------

      INTEGER Timdex
      DOUBLE PRECISION Omega
      DOUBLE PRECISION pi
      DOUBLE COMPLEX CN
      DOUBLE COMPLEX F(0:ntimes - 1)
      DOUBLE COMPLEX i

* ------------------ Code ----------------------------------------------

      pi = 4.0*ATAN ( 1.0 )
      i = ( 0.0, 1.0 )
      Omega = 2.0*pi*FourFreq

*     set up for integration ----------------------------------------
      DO Timdex = 0, ntimes - 1
         F(Timdex) = Funct(Timdex)*EXP ( i*Omega*Timdex*deltatime )
      END DO

      CN = ( 0.0, 0.0 )

*     Integration ---------------------------------------------------
      DO Timdex = 0, ntimes - 2
         CN = CN + ( F(Timdex) + F(Timdex + 1) )/( ntimes*1.0 )
      END DO

*     phase calculation ---------------------------------------------
      FourPhi = 0.0
      IF ( DBLE ( CN ) .EQ. 0d0 ) THEN
         IF ( IMAG ( CN ) .GT. 0d0 ) THEN
            FourPhi = pi/2.0
         ELSEIF ( IMAG ( CN ) .LT. 0d0 ) THEN
            FourPhi = -pi/2.0
         ENDIF
      ELSE
         FourPhi = ATAN2 ( -IMAG ( CN ), DBLE ( CN ) )
      ENDIF

*     Fourier component coefficient ---------------------------------
      FourCN = ABS ( CN )

      RETURN
      END

*   End of Fourier procedure.

* =====================================================================
```

A.5.2. GetK3Const.f

```
* ====================================================================
*
      SUBROUTINE GetK3Const (anglefiles, directivity,
     &   geometry, eqnparts, density, soundspeed, deltatime,
     &   ffpdist, mach, aspstart, aspend, aspint)
*
* ================== Prologue ========================================
*
* Purpose:
*    To read in the constants from the Kirch3.rc file
*
* Processing:
*    Uses READ's to get the info in.
*
* Special requirements:
*    Gotta have something to read, so the Kirch3.rc file must exist.
*
* ------------------ Include files -----------------------------------

*     ncirc: [INTEGER] Number of points around the cylindrical grid?
*     naxis: [INTEGER] Number of axial points in the grid?
*     ntimes: [INTEGER] The number of timesteps?
*     nlevels: [INTEGER] Number of grid levels?
      INCLUDE 'Kirch3.dmn'

* ------------------ Argument declarations ---------------------------

*     Constants from Kirch3.rc --- See Kirch3.rc for description.
      LOGICAL anglefiles
      LOGICAL directivity
      LOGICAL geometry
      LOGICAL eqnparts
      DOUBLE PRECISION density
      DOUBLE PRECISION soundspeed
      DOUBLE PRECISION deltatime
      DOUBLE PRECISION ffpdist
      DOUBLE PRECISION mach
      DOUBLE PRECISION aspstart
      DOUBLE PRECISION aspend
      DOUBLE PRECISION aspint

* ------------------ Local declarations ------------------------------

      CHARACTER Dummy
      DOUBLE PRECISION Frequency

* ------------------ Code --------------------------------------------

      OPEN (UNIT=40, FILE='Kirch3.rc', STATUS='UNKNOWN')
```

```
*     Three lines of comments at the top.
      READ (40, 2000) Dummy
      READ (40, 2000) Dummy
      READ (40, 2000) Dummy

*     Read constants from Kirch3.rc
*     One blank line, one label line, then the constant.
      READ (40, 2000) Dummy
      READ (40, 2000) Dummy
      READ (40, 2010) anglefiles
      READ (40, 2000) Dummy
      READ (40, 2000) Dummy
      READ (40, 2010) directivity
      READ (40, 2000) Dummy
      READ (40, 2000) Dummy
      READ (40, 2010) geometry
      READ (40, 2000) Dummy
      READ (40, 2000) Dummy
      READ (40, 2010) eqnparts
      READ (40, 2000) Dummy
      READ (40, 2000) Dummy
      READ (40, 2020) density
      READ (40, 2000) Dummy
      READ (40, 2000) Dummy
      READ (40, 2020) soundspeed
      READ (40, 2000) Dummy
      READ (40, 2000) Dummy
      READ (40, 2020) Frequency
      READ (40, 2000) Dummy
      READ (40, 2000) Dummy
      READ (40, 2020) ffpdist
      READ (40, 2000) Dummy
      READ (40, 2000) Dummy
      READ (40, 2020) mach
      READ (40, 2000) Dummy
      READ (40, 2000) Dummy
      READ (40, 2020) aspstart
      READ (40, 2000) Dummy
      READ (40, 2000) Dummy
      READ (40, 2020) aspend
      READ (40, 2000) Dummy
      READ (40, 2000) Dummy
      READ (40, 2020) aspint

      CLOSE (40)

      deltatime = 1d0/(Frequency*ntimes)
      WRITE ( *, * ) 'GetK3Const: Frequency  = ', Frequency,
     &    '  deltatime = ', deltatime, '  ntimes = ', ntimes

 2000 FORMAT (A)
 2010 FORMAT (L5)
 2020 FORMAT (F24.17)

      RETURN
```

```
      END
* =====================================================================
```

LIST OF FIGURES

Figure page

LIST OF TABLES

Table page

LIST OF VARIABLES

A	Source strength
c	Speed of sound
d	Distance between monopole sources for dipoles and quadrupoles
d_{ax}	Maximum axial grid point distance
d_{cr}	Maximum circumferential grid point distance
d_l	Grid layer separation distance
dS	Area of surface integration
e_r	Radial unit vector (Pierce's derivation)
F_c	Fourier component
$f(x)$	Arbitrary function of x
G	Green's function (Pierce's derivation)
h	Height of a frustum
i	$\sqrt{-1}$
I_R	Surface integration over outer sphere (Pierce's derivation)
k	Wavenumber
l_s	Length of the side of a cylindrical grid
m	Index of summation
$\vec{n}$	Unit normal pointing out of the surface
n_{ax}	Number of axial grid points
n_{cr}	Number of circumferential grid points
n_t	Number of time samples in one oscillation
`naxis`	Program variable representing number of axial points
`ncirc`	Program variable representing number of circumferential points
`nptinlrad`	Program variable representing the number of points on the inlet end-cap
`nptotlrad`	Program variable representing the number of points on the outlet end-cap
p	Acoustic pressure
$\hat{p}$	time harmonic pressure amplitude (Pierce's derivation)
p_a	Pressure calculated by analytical means
$\max p_a$	Maximum value of pressure calculated by analytical means

p_f	Pressure calculated by a computational formulation
$p_{t-\Delta t}$	Pressure one time sample before current time sample
$p_{t+\Delta t}$	Pressure one time sample after current time sample
R	Distance from source to observer point
R	Distance from origin to sphere of large radius (Pierce's derivation)
R_f	Average radius of a frustum
r	Distance from surface to observer point
r_s	Radius of a spherical Kirchhoff surface
r_c	Radius of a cylindrical Kirchhoff surface
r_{inl}	Radius of the inlet end-cap
r_{otl}	Radius of the outlet end-cap
r_x	Radius of the Kirchhoff surface at a constant x
$\vec{r}$	Vector from surface to observer point
S	Kirchhoff surface
T	Period of oscillation
t	Time
Δt	Length of time between samples
V	Volume (Pierce's derivation)
$\hat{v}_n$	Time harmonic normal velocity (Pierce's derivation)
$\mathbf{x}$	Observer location
x_n	Point on the nth grid layer
Δx_n	Distance between x_n and x_{n+1}
Δx_t	Distance between x_1 and x_3
$\mathbf{y}$	Surface location
δ	Dirac delta function (Pierce's derivation)
λ	Wavelength
Φ	Wave equation quantity
ϕ	Spherical coordinate (Pierce's derivation)
τ	Retarded time
θ	Angle in the x-y plane from the $+x$ axis towards the $+y$ axis
θ	Spherical coordinate (Pierce's derivation)

ω	Angular frequency
ω_c	Discrete angular frequency for Fourier algorithm

CONTENTS