THE W_N ADAPTIVE METHOD FOR NUMERICAL SOLUTION OF PARTICLE TRANSPORT PROBLEMS

A Dissertation

by

AARON MICHAEL WATSON

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2005

Major Subject: Nuclear Engineering

THE W_N ADAPTIVE METHOD FOR NUMERICAL SOLUTION OF PARTICLE TRANSPORT PROBLEMS

A Dissertation

by

AARON MICHAEL WATSON

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2005

Major Subject: Nuclear Engineering

ABSTRACT

The W_N Adaptive Method for Numerical Solution of Particle Transport Problems.

(December 2005)

Aaron Michael Watson, B.S., Texas A&M University;

M.S., Texas A&M University

Chair of Advisory Committee: Dr. Paul Nelson

The source and nature, as well as the history of ray-effects, is described. A benchmark code, using piecewise constant functions in angle and diamond differencing in space, is derived in order to analyze four sample problems. The results of this analysis are presented showing the ray effects and how increasing the resolution (number of angles) eliminates them. The theory of wavelets is introduced and the use of wavelets in multiresolution analysis is discussed. This multiresolution analysis is applied to the transport equation, and equations that can be solved to calculate the coefficients in the wavelet expansion for the angular flux are derived. The use of thresholding to eliminate wavelet coefficients that are not required to adequately solve a problem is then discussed. An iterative sweeping algorithm, called the S_N-W_N method, is derived to solve the wavelet-based equations. The convergence of the S_N-W_N method is discussed. An algorithm for solving the equations is derived, by solving a matrix within each cell directly for the expansion coefficients. This algorithm is called the CW-W_N method. The results of applying the CW-W_N method to the benchmark problems are presented. These results show that more research is needed to improve the convergence of the S_N-W_N method, and that the CW-W_N method is computationally too costly to be seriously considered.

DEDICATION

To Ashley

ACKNOWLEDGEMENTS

I would like to thank my committee chair, Dr. Nelson, and my committee members, Dr. Adams, Dr. Charlton, and Dr. Efendiev, for their guidance and support throughout the course of this research.

I also would like to extend thanks to the Department of Energy Nuclear Engineering Education Research program for their funding of this research, as well as to Texas A&M University and my colleagues in the Department of Nuclear Engineering for their support.

Finally, I would like to thank my family and friends for their encouragement, and my wife for her patience and her love.

TABLE OF CONTENTS

Page

ABSTRACT iii

DEDICATION v

ACKNOWLEDGEMENTS vi

TABLE OF CONTENTS vii

LIST OF FIGURES ix

LIST OF TABLES xii

CHAPTER

I INTRODUCTION 1

II RAY EFFECTS 4

III BENCHMARK CODE: S_N 7

Derivation 8
Algorithm 16

IV BENCHMARK PROBLEMS 19

The Box-in-Box Problem 19
The Brunner Problems 29

V WAVELETS AND MULTIRESOLUTION ANALYSIS 33

Wavelet Theory 33
Multiresolution Analysis 36
Haar Wavelets and Other Wavelet Bases 37

VI APPLICATION TO TRANSPORT COMPUTATIONS 47

VII THRESHOLDING 63

VIII S_N-LIKE W_N METHOD 67

CHAPTER Page

Derivation 67
Algorithm 70

IX CONVERGENCE OF THE S_N-W_N METHOD 75

X CELL-WISE W_N METHOD 89

Derivation 89
Algorithm 94

XI RESULTS 98

The Box-in-Box Problems 100
The Brunner Problems 108

XII CONCLUSIONS 113

REFERENCES 116

APPENDIX A 119

APPENDIX B 132

APPENDIX C 148

APPENDIX D 179

VITA 184

LIST OF FIGURES

FIGURE Page

III–1 Diamond differencing cell. 14

III–2 Algorithm for implementing the S_N method. 17

IV–1 Geometry for the test box problem. (Lengths in centimeters.) 19

IV–2 Solution of test box problem at specified locations. 20

IV–3 Contour solutions of the test box problem. 21

IV–4 Solutions of the test box problem along top edge. 22

IV–5 Angular flux at point E in test box problem. 23

IV–6 Geometry for the first box-in-box problem. (Lengths in centimeters.) 24

IV–7 Contour solutions of the first box-in-box problem. 24

IV–8 Solutions of the first box-in-box problem along top edge. 25

IV–9 Geometry for the second box-in-box problem. (Lengths in centimeters.) ... 26

IV–10 Contour solutions of the second box-in-box problem. 26

IV–11 Solutions of the second box-in-box problem along top edge. 27

IV–12 Geometry for the lattice problem. 28

IV–13 Contour solutions of the lattice problem. 28

IV–14 Solutions of the lattice problem along top edge. 29

IV–15 Geometry for the Hohlraum problem. 30

IV–16 Contour solutions of the Hohlraum problem. 30

IV–17 Solutions of the Hohlraum problem around the target. 31

V–1 Haar scaling function. 37

FIGURE Page

V–2 Haar wavelet function. 38

V–3 2nd-order Chui-Wang scaling function. 39

V–4 2nd-order Chui-Wang wavelet function. 40

V–5 3rd-order Chui-Wang scaling function. 41

V–6 3rd-order Chui-Wang wavelet function. 41

V–7 2nd-order Daubechies scaling function. 42

V–8 2nd-order Daubechies wavelet function. 43

V–9 Real part of the spherical Morlet wavelet at scales a = 0.03 (a) and a = 0.3 (b). 44

V–10 Real part of the spherical Morlet wavelet at scale a = 0.03 and centered at ($\pi/3$, $\pi/3$), with rotation 0 (a) and $\pi/2$ (b). 45

VIII–1 Algorithm for implementing the S_N-W_N method. 71

IX–1 The iteration eigenvalues for the S_N-W_N method 81

X–1 Algorithm for implementing the CW-W_N method. 95

XI–1 Benchmark solution of the test box problem. 98

XI–2 Solutions of the test box problem using CW-W_N. 99

XI–3 Solutions of the test box problem along top edge using CW-W_N. 99

XI–4 Angular flux at point (0.3 cm, 0.2 cm) in test box problem. 101

XI–5 Benchmark solution of the first box-in-box problem. 102

XI–6 Solutions of the first box-in-box problem using CW-W_N. 103

XI–7 Solutions of the first box-in-box problem along top edge using CW-W_N. 103

FIGURE Page

XI–8 Benchmark solution of the second box-in-box problem. 104

XI–9 Solutions of the second box-in-box problem using CW-W_N. 105

XI–10 Solutions of the second box-in-box problem along top edge using CW-W_N. 106

XI–11 Benchmark solution of the lattice problem. 107

XI–12 Solutions of the lattice problem using CW-W_N. 107

XI–13 Solutions of the lattice problem along top edge using CW-W_N. 108

XI–14 Benchmark solution of the Hohlraum problem. 109

XI–15 Solutions of the Hohlraum problem using CW-W_N. 110

XI–16 Solutions of the Hohlraum problem around the target using CW-W_N. 111

LIST OF TABLES

TABLE		Page
III–1	Reflecting azimuthal angles for S_N method.	15
VIII–1	Reflecting azimuthal angles for S_N-W_N method.	69
X–1	Reflecting azimuthal angles for CW-W_N method.	92
XI–1	Results from the test box problem.	100
XI–II	Results from the first box-in-box problem.	101
XI–III	Results from the second box-in-box problem.	105
XI–IV	Results from the lattice problem.	109
XI–V	Results from the Hohlraum problem.	110
XI–VI	Relative errors about the target for the Hohlraum problem.	111

CHAPTER I

INTRODUCTION

We will consider the neutral particle transport equation

$$\frac{1}{v(E)}\frac{\partial\psi}{\partial t}+\vec{\Omega}\cdot\nabla\psi\left(\vec{r},E,\vec{\Omega},t\right)+\sigma\left(\vec{r},E,t\right)\psi\left(\vec{r},E,\vec{\Omega},t\right)$$
$$=\int_{4\pi} d\Omega'\,\sigma_s\left(\vec{r},E,\vec{\Omega}'\cdot\vec{\Omega},t\right)\psi\left(\vec{r},E,\vec{\Omega}',t\right)+S_{ext}\left(\vec{r},E,\vec{\Omega},t\right). \tag{1.1}$$

Throughout the boundary conditions will be assumed to have the form of a specified incoming angular flux (ψ) at every point on the surface of the underlying spatial domain. In order to solve this equation, one must first discretize in each variable: spatial, angular, energy, and time. For the purposes of this dissertation, we will assume that energy and time are discretized such that they can be ignored. Thus, the equation we are interested in is the steady state, monoenergetic transport equation

$$\vec{\Omega}\cdot\nabla\psi\left(\vec{r},\vec{\Omega}\right)+\sigma\left(\vec{r}\right)\psi\left(\vec{r},\vec{\Omega}\right)=\int_{4\pi} d\Omega'\,\sigma_s\left(\vec{r},\vec{\Omega}'\cdot\vec{\Omega}\right)\psi\left(\vec{r},\vec{\Omega}'\right)+S_{ext}\left(\vec{r},\vec{\Omega}\right). \tag{1.2}$$

The angular variable is usually discretized in two ways, either using a function expansion (spherical harmonics method) or a quadrature sum (discrete-ordinates method). Both of these approximations present difficulties when the geometry is complicated with sources that are highly localized spatially. This is possibly the foremost current problem in computational particle transport theory. Many of these difficulties are associated with a phenomenon known as the "ray effect," which was first

---

This dissertation follows the style of *Nuclear Science and Engineering*.

identified almost four decades ago. See Chapter II below for further discussion and literature citations.

Multiscale and multiresolution techniques have been applied successfully to solve similar problems in other fields. The foremost of these techniques are wavelet representations.[1] The objective of this dissertation is to use wavelet expansions of the transport equation in the directional variable to assess the potential for efficient resolution of ray effects, with minimal alteration of the ray-tracing/source iteration algorithm used by the more traditional discrete-ordinates approximation in direction.

Chapter II outlines the source and nature, as well as the history of ray effects. In Chapter III, we derive a benchmark code using piecewise constant functions in angle and diamond differencing in space in order to analyze four sample problems. Chapter IV presents the results of this analysis, showing the ray effects and how increasing the resolution (number of angles) eliminates them. In Chapter V, we introduce the theory of wavelets and discuss how wavelets can be used in a multiresolution analysis. Chapter VI applies this multiresolution analysis to the transport equation, deriving equations that can be solved to calculate the coefficients in the wavelet expansion for the angular flux. In Chapter VII, we discuss how thresholding can be used to eliminate wavelet coefficients that are not required to adequately solve a problem.

Chapter VII derives an iterative sweeping algorithm, called the S_N-W_N method, to solve the equations derived in Chapter VI. In Chapter IX, we discuss the convergence of the S_N-W_N method. Chapter X derives an algorithm for solving the equations presented in Chapter VI, but solving a matrix within each cell directly for the expansion coefficients. This algorithm is called the CW-W_N method. Chapter XI presents the results of applying the CW-W_N method to the benchmark problems given in Chapter IV. We conclude and suggest the direction of future research in Chapter XII.

CHAPTER II

RAY EFFECTS

The ray effect occurs when the angular flux from a source is localized about a single direction at each point, or when the source is very small or very far away. They can also occur when a shield is compromised, as in a duct problem. When one of these conditions exists, the discrete-ordinates solution in the angular domain will be highly peaked in some directions, while it is practically nonexistent in others. Using discrete-ordinates, particle travel is restricted to a given set of directions, rather than the natural continuum of directions associated with analytic solutions. This discretization leads to a loss of invariance under infinitesimal rotations. The natural solution is simply to increase the number of ordinates, but this solution results in an increase in computational effort. Other solution techniques have been used, including angular finite element methods and ficticious source methods. However, ficticious source methods converge very slowly and, although angular finite element methods reduce ray effects, they do not eliminate them.[2]

The ray effect phenomenon was first described by Gelbard[3] during a panel discussion on outstanding problems in reactor mathematics in 1965. About the same time, Hansen[4] reported anomalous results obtained by using the discrete-ordinates approximation. Later that year, Davis and Kaplan[5] showed that this phenomenon was due to the discretization of the angular variable. In 1968, Lathrop[6] coined the term “ray effect” and showed that they were related to the nonequivalence of the discrete-ordinates equations and the spherical harmonics (P_N) equations in multiple dimensions. He proposed a method of correcting this, but noted that it might result in an unacceptable increase in computational cost. That same year, Kaplan[7] proposed the idea of space-angle synthesis, in which the ordinates are viewed as piecewise constants in angle instead of delta functions. In 1970, in order to quantify ray effects in a practical sense, Miller and Jarka[8] showed that ray-effect errors could be significant when discrete-ordinates approximations are used in reactor design.

In 1971, Natelson[9] applied piecewise constant trial functions to the second-order even-parity form of the transport equation using Kaplan's method. Later that year, Lathrop[10] showed how to formulate the P_N-equivalent discrete-ordinates equations. This was known as the ficticious source method. He concluded that the practice of adding specially chosen directions is the most practical remedy, but points out that P_N formulations may be a better method in difficult problems. The following year, Lathrop[11] extended this method to (r, z) cylindrical geometry. In 1972, Jung et al.[12] applied the Weinberg-Wigner P_N approximation technique for one dimension to the two-dimensional case to obtain a discrete-ordinates approximation equivalent to the P_N approximation. The same year, Reed[13] presented a method for obtaining the spherical harmonics solution by interpreting the discrete-ordinates method as a projection method. In 1973, Miller et al.[14] applied piecewise bilinear angular finite elements to the second order transport equation. In 1975, Briggs et al.[15] compared the continuous piecewise bilinear finite element, the piecewise constant finite element, and the discrete-ordinate forms. They showed that both finite element methods approximated the streaming term as an elliptic operator instead of a hyperbolic operator, which the discrete-ordinates does, and the streaming term is a hyperbolic operator (because it has real characteristics, which elliptic operators do not). This is the reason these finite element methods were able to mitigate ray effects. The next year, Seed[16] approximated the transport equation by representing the angular flux with an expansion of the angular dependence in the piecewise constant, orthogonal Walsh functions.

In 1977, Miller and Reed[17] proposed a scalable version of the ficticious source method in which discrete-ordinates is used in energy groups where ray effects are not expected and the ficticious source method in those groups where ray effects are expected. In 1983, Fletcher[18] showed that solving the P_N equations in the second-order self-adjoint form, rather than in the form of P_N-equivalent discrete-ordinates equations, is a good way to eliminate ray effects. More recently, in 2001, Brown et al.[19] used a Krylov subspace solution technique to solve the P_N-equivalent discrete-ordinates equations much more efficiently than the ficticious source method. In 2003, Morel et al.[2] showed that any angular discretization technique that produces a hyperbolic approximation for the directional gradient operator will yield discrete-ray solutions for a line source in a void, which is similar to the box-in-box problem.

CHAPTER III

BENCHMARK CODE: S_N

We will compare the results of adaptive wavelet codes to a benchmark code. This benchmark code, called S_N, uses a Haar scaling function expansion in the azimuthal direction, discrete ordinates in the polar direction, and diamond differencing in space. The term "scaling function" comes from the mathematics of wavelet analysis. It is also called the "father wavelet."

For our purposes, we will use the Haar wavelet basis described in You[1]. The scaling function in this basis is defined as a piecewise constant with value one on the interval $[0,1]$. We want to scale this interval to $[0,2\pi]$, since we are interested in the azimuthal direction, and add scaling and translation coefficients. Thus, the Haar scaling function on the interval $\theta \in [0,2\pi]$ is defined as

$$f_{r,\lambda}(\theta) = \begin{cases} 2^{r/2+1}, & 2^{-1-r}\pi(\lambda-1) \le \theta \le 2^{-1-r}\pi\lambda \\ 0, & \text{otherwise} \end{cases}, \tag{3.1}$$

for resolution level r ($r \ge 0$) and index λ ($1 \le \lambda \le 2^r$).

Thus, for resolution level zero, there are four scaling functions between zero and 2π, with indices from one to four. This is the coarsest meaningful resolution level for transport problems, since there is one scaling function per quadrant. If there were any fewer than one per quadrant, the problem would not be well-posed, and convergence would not be guaranteed in the spatial direction. In addition, the traditional sweep method could not be implemented. For resolution level one, there are two scaling functions per quadrant, and each successive resolution level increase doubles the number of scaling functions per quadrant.

Derivation

Consider the steady-state, monoenergetic transport equation

$$\vec{\Omega}\cdot\nabla\psi\left(\vec{r},\vec{\Omega}\right)+\sigma\left(\vec{r}\right)\psi\left(\vec{r},\vec{\Omega}\right)=\int_{4\pi}d\Omega'\,\sigma_s\left(\vec{r},\vec{\Omega}'\cdot\vec{\Omega}\right)\psi\left(\vec{r},\vec{\Omega}'\right)+S_{ext}\left(\vec{r},\vec{\Omega}\right) \tag{3.2}$$

In two-dimensional Cartesian geometry, and with isotropic scattering and sources, this becomes

$$\Omega_x\frac{\partial\psi}{\partial x}+\Omega_y\frac{\partial\psi}{\partial y}+\sigma\left(x,y\right)\psi\left(x,y,\vec{\Omega}\right)=\frac{\sigma_s\left(x,y\right)}{4\pi}\int_{4\pi}d\Omega'\,\psi\left(x,y,\vec{\Omega}'\right)+\frac{S_{ext}\left(x,y\right)}{4\pi} \tag{3.3}$$

We will now split the direction vector $\vec{\Omega}$ into polar (ξ) and azimuthal (θ) components

$$\begin{aligned}\Omega_x&=\sin\xi\cos\theta\\ \Omega_y&=\sin\xi\sin\theta\\ \Omega_z&=\mu:=\cos\xi\end{aligned}$$

Thus, our transport equation now becomes

$$\begin{aligned}&\sqrt{1-\mu^2}\cos\theta\frac{\partial\psi}{\partial x}+\sqrt{1-\mu^2}\sin\theta\frac{\partial\psi}{\partial y}+\sigma\left(x,y\right)\psi\left(x,y,\mu,\theta\right)\\ &\qquad=\frac{\sigma_s\left(x,y\right)}{4\pi}\int_{-1}^{1}d\mu'\int_{2\pi}d\theta'\,\psi\left(x,y,\mu',\theta'\right)+\frac{S_{ext}\left(x,y\right)}{4\pi}\end{aligned} \tag{3.4}$$

Next, we will look only at some discrete polar cosine ($\mu_m,\ m=1,\ldots,M$) and replace the integral over polar cosine with a Gauss-Legendre quadrature sum. Also, we will define $\psi_m\left(x,y,\theta\right):=\psi\left(x,y,\mu_m,\theta\right)$. Thus

$$\sqrt{1-\mu_m^2}\cos\theta\frac{\partial\psi_m}{\partial x}+\sqrt{1-\mu_m^2}\sin\theta\frac{\partial\psi_m}{\partial y}+\sigma(x,y)\psi_m(x,y,\theta) \\ =\frac{\sigma_s(x,y)}{4\pi}\int_{2\pi}d\theta'\sum_{n=1}^{M}\omega_n\psi_n(x,y,\theta')+\frac{S_{ext}(x,y)}{4\pi}. \tag{3.5}$$

Finally, we will define the scalar flux as $\phi(x,y):=\int_{2\pi}d\theta\sum_{m=1}^{M}\omega_m\psi_m(x,y,\theta)$ and divide through by $\sqrt{1-\mu_m^2}$. Then, our transport equation is

$$\cos\theta\frac{\partial\psi_m}{\partial x}+\sin\theta\frac{\partial\psi_m}{\partial y}+\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi_m(x,y,\theta)=\frac{\sigma_s(x,y)}{4\pi\sqrt{1-\mu_m^2}}\phi(x,y)+\frac{S_{ext}(x,y)}{4\pi\sqrt{1-\mu_m^2}}. \tag{3.6}$$

Using the Haar scaling function, we expand the angular flux as

$$\psi_m(x,y,\theta)=\sum_{\lambda=1}^{2^r}\psi_{r,\lambda,m}(x,y)f_{r,\lambda}(\theta),\qquad r\geq 0. \tag{3.7}$$

Substituting this expansion into equation (3.6), we find

$$\sum_{\lambda=1}^{2^r}\left\{\frac{\partial\psi_{r,\lambda,m}}{\partial x}f_{r,\lambda}(\theta)\cos\theta+\frac{\partial\psi_{r,\lambda,m}}{\partial y}f_{r,\lambda}(\theta)\sin\theta \right. \\ \left. +\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi_{r,\lambda,m}(x,y)f_{r,\lambda}(\theta)=\frac{\sigma_s(x,y)}{4\pi\sqrt{1-\mu_m^2}}\phi^r(x,y)+\frac{S_{ext}(x,y)}{4\pi\sqrt{1-\mu_m^2}}\right\} \tag{3.8}$$

We will now multiply through this equation by a scaling function $f_{n,p}(\theta)$

$$\sum_{\lambda=1}^{2^r}\left\{\frac{\partial\psi_{r,\lambda,m}}{\partial x}f_{r,\lambda}(\theta)f_{n,p}(\theta)\cos\theta+\frac{\partial\psi_{r,\lambda,m}}{\partial y}f_{r,\lambda}(\theta)f_{n,p}(\theta)\sin\theta \right. \\ +\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi_{r,\lambda,m}(x,y)f_{r,\lambda}(\theta)f_{n,p}(\theta) \\ \left. =\frac{\sigma_s(x,y)}{4\pi\sqrt{1-\mu_m^2}}\phi^r(x,y)f_{n,p}(\theta)+\frac{S_{ext}(x,y)}{4\pi\sqrt{1-\mu_m^2}}f_{n,p}(\theta)\right\} \tag{3.9}$$

Since the scaling functions are orthogonal, this becomes

$$\sum_{\lambda=1}^{2^r}\left\{\frac{\partial\psi_{r,\lambda,m}}{\partial x}f_{r,\lambda}^2(\theta)\cos\theta+\frac{\partial\psi_{r,\lambda,m}}{\partial y}f_{r,\lambda}^2(\theta)\sin\theta+\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi_{r,\lambda,m}(x,y)f_{r,\lambda}^2(\theta)\right.$$
$$\left.=\frac{\sigma_s(x,y)}{4\pi\sqrt{1-\mu_m^2}}\phi^r(x,y)f_{r,\lambda}(\theta)+\frac{S_{ext}(x,y)}{4\pi\sqrt{1-\mu_m^2}}f_{r,\lambda}(\theta)\right\} \tag{3.10}$$

Next, we will integrate over the interval $\theta\in[0,2\pi]$. Then, we find

$$\sum_{\lambda=1}^{2^r}\left\{\tilde{\mu}_{r,\lambda}\frac{\partial\psi_{r,\lambda,m}}{\partial x}+\tilde{\eta}_{r,\lambda}\frac{\partial\psi_{r,\lambda,m}}{\partial y}+A_{r,\lambda}\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi_{r,\lambda,m}(x,y)\right.$$
$$\left.=B_{r,\lambda}\frac{\sigma_s(x,y)}{4\pi\sqrt{1-\mu_m^2}}\phi^r(x,y)+B_{r,\lambda}\frac{S_{ext}(x,y)}{4\pi\sqrt{1-\mu_m^2}}\right\} \tag{3.11}$$

where

$$\tilde{\mu}_{r,\lambda}=\int_0^{2\pi}d\theta\, f_{r,\lambda}^2(\theta)\cos\theta=2^{r+2}\int_{2^{-1-r}\pi(\lambda-1)}^{2^{-1-r}\pi\lambda}d\theta\,\cos\theta=2^{r+2}\left[\sin\left(2^{-1-r}\pi\lambda\right)-\sin\left(2^{-1-r}\pi(\lambda-1)\right)\right]$$

$$\tilde{\eta}_{r,\lambda}=\int_0^{2\pi}d\theta\, f_{r,\lambda}^2(\theta)\sin\theta=2^{r+2}\int_{2^{-1-r}\pi(\lambda-1)}^{2^{-1-r}\pi\lambda}d\theta\,\sin\theta=2^{r+2}\left[-\cos\left(2^{-1-r}\pi\lambda\right)+\cos\left(2^{-1-r}\pi(\lambda-1)\right)\right]$$

$$A_{r,\lambda}=\int_0^{2\pi}d\theta\, f_{r,\lambda}^2(\theta)=2^{r+2}\int_{2^{-1-r}\pi(\lambda-1)}^{2^{-1-r}\pi\lambda}d\theta\;=2^{r+2}\left[2^{-1-r}\pi\lambda-2^{-1-r}\pi(\lambda-1)\right]=2\pi$$

$$B_{r,\lambda}=\int_0^{2\pi}d\theta\, f_{r,\lambda}(\theta)=2^{r/2+1}\int_{2^{-1-r}\pi(\lambda-1)}^{2^{-1-r}\pi\lambda}d\theta\;=2^{r/2+1}\left[2^{-1-r}\pi\lambda-2^{-1-r}\pi(\lambda-1)\right]=2^{-r/2}\pi$$

Dividing through by 2π, we find

$$\sum_{\lambda=1}^{2^r}\left\{\mu_{r,\lambda}\frac{\partial\psi_{r,\lambda,m}}{\partial x}+\eta_{r,\lambda}\frac{\partial\psi_{r,\lambda,m}}{\partial y}+\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi_{r,\lambda,m}(x,y)\right.$$
$$\left.=2^{-r/2-1}\frac{\sigma_s(x,y)}{4\pi\sqrt{1-\mu_m^2}}\phi^r(x,y)+2^{-r/2-1}\frac{S_{ext}(x,y)}{4\pi\sqrt{1-\mu_m^2}}\right\} \tag{3.12}$$

where

$$\mu_{r,\lambda} = \frac{\tilde{\mu}_{r,\lambda}}{2\pi} \qquad \text{and} \qquad \eta_{r,\lambda} = \frac{\tilde{\eta}_{r,\lambda}}{2\pi}.$$

Next, we substitute the expansion (3.7) into the definition of the scalar flux, and we find

$$\phi^r(x,y) = 2^{-r/2}\pi \sum_{\lambda=1}^{2^r}\sum_{m=1}^{M} \omega_m \psi_{r,\lambda,m}(x,y) \tag{3.13}$$

Now, we will address boundary conditions. In this dissertation, we will employ vacuum, reflective, and incident flux boundary conditions only. For vacuum boundaries, the condition is

$$\psi\left(\vec{r},\vec{\Omega}\right) = 0, \qquad \vec{r} \in \partial D,\ \vec{\Omega}\cdot\vec{n} < 0,$$

where $\vec{n}$ is the outward normal vector on the boundary of the domain *D*. Substituting the expansion (3.7), we find

$$\sum_{\lambda=1}^{2^r} \psi_{r,\lambda,m}(x,y) f_{r,\lambda}(\theta) = 0, \qquad (x,y) \in \partial D,\ \vec{\Omega}_m\cdot\vec{n} < 0$$

Because the scaling functions form a basis for the space $\theta \in [0,2\pi]$, for this summation to be zero, all of the coefficients $\psi_{r,\lambda,m}(x,y)$ must be zero. Thus,

$$\psi_{r,\lambda,m}(x,y) = 0, \qquad (x,y) \in \partial D,\ \vec{\Omega}_m\cdot\vec{n} < 0 \tag{3.14}$$

For reflective boundary conditions, we have

$$\psi\left(\vec{r},\vec{\Omega}\right) = \psi\left(\vec{r},-\vec{\Omega}\right), \qquad \vec{r} \in \partial D,\ \vec{\Omega}\cdot\vec{n} < 0.$$

Substituting the expansion (3.7), we find

$$\sum_{\lambda=1}^{2^r} \psi_{r,\lambda,m}(x,y) f_{r,\lambda}(\theta) = \sum_{\lambda=1}^{2^r} \psi_{r,\lambda',m'}(x,y) f_{r,\lambda'}(\theta), \quad (x,y) \in \partial D,\ \vec{\Omega}_m \cdot \vec{n} < 0$$

Integrating over the angular interval $\theta \in [0, 2\pi]$

$$\sum_{\lambda=1}^{2^r} \psi_{r,\lambda,m}(x,y)\pi = \sum_{\lambda=1}^{2^r} \psi_{r,\lambda',m'}(x,y)\pi, \quad (x,y) \in \partial D,\ \vec{\Omega}_m \cdot \vec{n} < 0$$

One solution to this equation is

$$\psi_{r,\lambda,m}(x,y) = \psi_{r,\lambda',m'}(x,y), \qquad (x,y) \in \partial D,\ m',\lambda' : \vec{\Omega}' = -\vec{\Omega} \tag{3.15}$$

where (x, y) is on the boundary of *D*, and m' and λ' are such that $\vec{\Omega}' = -\vec{\Omega}$.

Finally, for incident boundary conditions, we have

$$\psi(\vec{r},\vec{\Omega}) = f(\vec{r},\vec{\Omega}), \qquad \vec{r} \in \partial D,\ \vec{\Omega} \cdot \vec{n} > 0. \tag{3.16}$$

We will only consider isotropic incident fluxes in this dissertation, so

$$\psi(\vec{r},\vec{\Omega}) = f(\vec{r}), \qquad \vec{r} \in \partial D,\ \vec{\Omega} \cdot \vec{n} > 0.$$

Substituting the expansion (3.7), we find

$$\sum_{\lambda=1}^{2^r} \psi_{r,\lambda,m}(x,y) f_{r,\lambda}(\theta) = f(x,y), \qquad (x,y) \in \partial D,\ \vec{\Omega}_m \cdot \vec{n} > 0$$

Integrating over the angular interval $\theta \in [0, 2\pi]$

$$\sum_{\lambda=1}^{2^r} \psi_{r,\lambda,m}(x,y)\pi = f(x,y)2\pi, \qquad (x,y) \in \partial D,\ \vec{\Omega}_m \cdot \vec{n} > 0$$

Thus, we find

$$\psi_{r,\lambda,m}(x,y) = 2^{1-r} f(x,y), \qquad (x,y) \in \partial D,\ \vec{\Omega}_m \cdot \vec{n} > 0 \tag{3.17}$$

So, in summary, our equations are

$$\sum_{\lambda=1}^{2^r}\left\{\mu_{r,\lambda}\frac{\partial\psi_{r,\lambda,m}}{\partial x}+\eta_{r,\lambda}\frac{\partial\psi_{r,\lambda,m}}{\partial y}+\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi_{r,\lambda,m}(x,y)\right.$$
$$\left.=2^{-r/2-1}\frac{\sigma_s(x,y)}{4\pi\sqrt{1-\mu_m^2}}\phi^r(x,y)+2^{-r/2-1}\frac{S_{ext}(x,y)}{4\pi\sqrt{1-\mu_m^2}}\right\},\quad r\ge 0,\ m\ge 1$$

where

$$\mu_{r,\lambda}=\frac{2^{r+2}}{2\pi}\left[\sin\left(2^{-1-r}\pi\lambda\right)-\sin\left(2^{-1-r}\pi(\lambda-1)\right)\right]$$
$$\eta_{r,\lambda}=\frac{2^{r+2}}{2\pi}\left[\cos\left(2^{-1-r}\pi(\lambda-1)\right)-\cos\left(2^{-1-r}\pi\lambda\right)\right]$$
$$\phi^r(x,y)=2^{-r/2}\pi\sum_{\lambda=1}^{2^r}\sum_{m=1}^{M}\omega_m\psi_{r,\lambda,m}(x,y)$$

with boundary conditions

$$\begin{aligned}&\text{vacuum:} && \psi_{r,\lambda,m}(x,y)=0\\ &\text{reflective:} && \psi_{r,\lambda,m}(x,y)=\psi_{r,\lambda',m'}(x,y)\\ &\text{incident:} && \psi_{r,\lambda,m}(x,y)=2^{1-r}f(x,y).\end{aligned}$$

Now, we will address the spatial discretization. We will use diamond-differencing, where the angular flux is defined in terms of the cell edge fluxes as

$$\psi_{i,j}=\tfrac{1}{2}\left[\psi_{i-1/2,j}+\psi_{i+1/2,j}\right]=\tfrac{1}{2}\left[\psi_{i,j-1/2}+\psi_{i,j+1/2}\right]$$

A schematic of a cell is shown in Fig. III–1. Substituting this into the system, we find

$$\psi_{i,j,m}^{r,\lambda}=\frac{Q_{i,j,m}^r+2\frac{|\mu_{r,\lambda}|}{\Delta x_{i,j}}\psi_{i\text{-inc},j,m}^{r,\lambda}+2\frac{|\eta_{r,\lambda}|}{\Delta y_{i,j}}\psi_{i,j\text{-inc},m}^{r,\lambda}}{\frac{\sigma_{i,j}}{\sqrt{1-\mu_m^2}}+2\frac{|\mu_{r,\lambda}|}{\Delta x_{i,j}}+2\frac{|\eta_{r,\lambda}|}{\Delta y_{i,j}}},\tag{3.18}$$

where $\psi_{i\text{-inc},j,m}^{r,\lambda}$ and $\psi_{i,j\text{-inc},m}^{r,\lambda}$ are the quadrant-dependent incident fluxes and

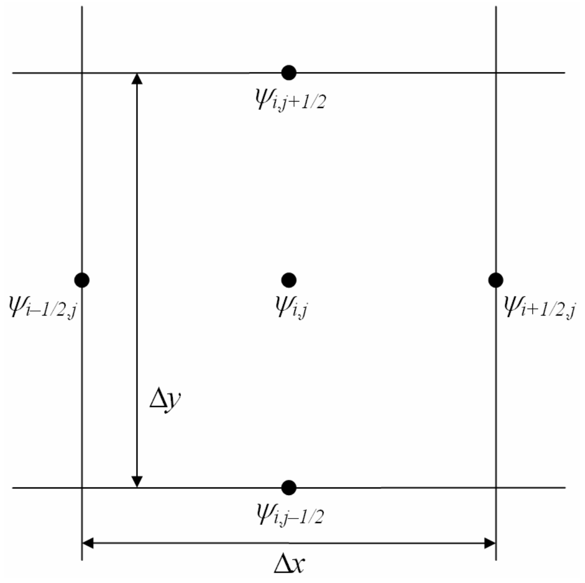


Figure III–1: Diamond differencing cell.

$$Q^r_{i,j,m} = \frac{2^{-r/2-1}}{4\pi\sqrt{1-\mu_m^2}}\left(\sigma_{s,i,j}\phi^r_{i,j} + S_{ext,i,j}\right).$$

The scalar flux is then

$$\phi^r_{i,j} = 2^{-r/2}\pi\sum_{\lambda=1}^{2^r}\sum_{m=1}^{M}\omega_m\psi^{r,\lambda}_{i,j,m}$$

and the boundary conditions are

vaccum:	$\psi^{r,\lambda}_{i,j,m} = 0$
reflective:	$\psi^{r,\lambda}_{i,j,m} = \psi^{r,\lambda'}_{i,j,m'}$
incident:	$\psi^{r,\lambda}_{i,j,m} = 2^{1-r} f_{i,j}$.

where *i* and *j* are located on the boundary. If we define the quadrants as

Quadrant 1:	$\mu > 0,\ \eta > 0$
Quadrant 2:	$\mu < 0,\ \eta > 0$
Quadrant 3:	$\mu < 0,\ \eta < 0$
Quadrant 4:	$\mu > 0,\ \eta < 0$

Table III–1: Reflecting azimuthal angles for S_N method.

Face	Quadrant 1	Quadrant 2	Quadrant 3	Quadrant 4
Left	$\frac{1}{2}2^r-\lambda+1$	-	-	$\frac{3}{2}2^r-\lambda+1$
Right	-	$\frac{1}{2}2^r-\lambda+1$	$\frac{3}{2}2^r-\lambda+1$	-
Top	-	-	$2^r-\lambda+1$	$2^r-\lambda+1$
Bottom	$2^r-\lambda+1$	$2^r-\lambda+1$	-	-

then, for the reflecting boundary conditions, for the azimuthal angles, λ' are shown in Table III–1, and for the polar angles, $m' = M - m + 1$.

The final consideration is convergence of the source iteration. We will define the error vector as

$$e^{(k)} = \phi^{(k)} - \phi^{(k-1)}$$

Then, we will calculate an estimate to the spectral radius of the iteration as

$$\rho^{(k)} = \frac{\left\|e^{(k)}\right\|_2}{\left\|e^{(k-1)}\right\|_2}$$

Then, we will consider the method converged if

$$\left\|e^{(k)}\right\|_2 \le \varepsilon\left(1-\rho^{(k)}\right)\left\|\phi^{(k)}\right\|_2$$

The term $(1-\rho)$ will prevent false convergence. Since this method only uses source iteration, its convergence is guaranteed for $c := \iint dxdy\,\sigma(x,y) \big/ \iint dxdy\,\sigma_s(x,y)$ less than one. This result differs from the results presented in Chapter IX since the S_N method does not iterate upon wavelets.

Algorithm

The following algorithm will solve the equations given above for the scalar flux in benchmark problems given in the next chapter. It will solve any steady-state, monoenergetic, two-dimensional problem that can be described in Cartesian coordinates, with isotropic scattering. The solution using this algorithm should be identical to that using the wavelet-based algorithms that will be presented later in this dissertation. Thus, it will be used as a benchmark to verify those algorithms.

Now, we will present pseudocode for an algorithm for implementing the S_N method. The values after the descriptions are operation counts assuming an $N \times N$ spatial mesh, with M polar angles and an azimuthal resolution of r. The algorithm is shown in Figure III–2. In total, we have $O\left(N^4\right)+kO\left(M2^rN^2\right)$ operations computed, with $O\left(N^4\right)$ for the input section and $O\left(M2^rN^2\right)$ operations per source iteration. For our benchmark problems, this is about 40,000 operations for the input section, plus 1,280,000 operations per source iteration.

The input file is composed of regional inputs, so the problem simply has to be divided into regions. The code also has built-in material specifications, so that one only needs to read the number corresponding to a particular material. The available materials are:

1. Void ($\sigma = 0.0$, $\sigma_s = 0.0$)
2. Box1 Problem ($\sigma = 0.75$, $\sigma_s = 0.5$)
3. Box2 Problem ($\sigma = 0.375$, $\sigma_s = 0.25$)

4. Weak pure scatterer ($\sigma = 1.0$, $\sigma_s = 1.0$)

5. Weak absorber ($\sigma = 10.0$, $\sigma_s = 0.0$)

6. Strong absorber ($\sigma = 100.0$, $\sigma_s = 0.0$)

7. Pure scatterer ($\sigma = 100.0$, $\sigma_s = 100.0$)

```
1) Read input file and allocate memory, $O(N^4)$
2) Calculate polar angles and weights, $O(M^3)$
3) Initialize variables, $O(M 2^r N^2)$
4) Begin source iteration, $k$
    a) Calculate source matrix Q, $O(N^2)$
    b) Initialize convergence variables, $O(N^2)$
    c) Loop over polar and azimuthal indices, $M 2^r$
        i) Calculate mu and eta, $O(1)$
        ii) Determine quadrant, $O(1)$
        iii) Calculate reflective indices, $O(1)$
        iv) Read y-incident values, $O(N)$
        v) Loop over y-position index, $N$
            (1) Read x-incident values, $O(N)$
            (2) Loop over x-position index, $N$
                (a) Calculate angular flux, $O(1)$
                (b) Save next incident flux values, $O(1)$
                (c) Save x-exit fluxes, $O(1)$
                (d) Increment scalar flux, $O(1)$
            (3) Save y-exit fluxes, $O(N)$
    d) Calculate error vectors and determine convergence, $O(N^2)$
5) Write to output files, $O(N^2)$
```

Figure III–2: Algorithm for implementing the S_N method.

8. Weak absorber ($\sigma = 1.0$, $\sigma_s = 0.0$)

Material indices (2) and (3) are used as described in Section IV.A. The source code for S_N is in Appendix A, as well as an example input file. The identifier in the incident fluxes section of the input file specifies the type of boundary condition to be implemented (vac = vacuum, ref = reflecting, inc = incident). The boundary conditions are listed in the order left, right, top, and bottom. The incident flux values are listed to the right of the identifier. The iteration parameters are the maximum number of iterations and the convergence criterion. Finally, the angular resolution section contains the number of polar angles and the value of r in the azimuthal discretization. All codes described in this dissertation are written in Fortran 90.

CHAPTER IV

BENCHMARK PROBLEMS

In this chapter, we will present five benchmark problems. These problems will be solved using the benchmark algorithm described in Chapter III, as well as the two wavelet-based algorithms that will be presented later in this dissertation. The results for these problems will be presented and discussed in Chapter X. In this section, the results presented use the benchmark code S_N.

The Box-in-Box Problem

One example problem that illustrates the phenomenon of the ray-effect is the box-in-box problem.[6] This is a two-dimensional problem that contains only a square source in one corner of the domain, with no incident fluxes and no scattering. An illustration of the problem is shown in Figure IV–1. The domain is 1 cm square, and the

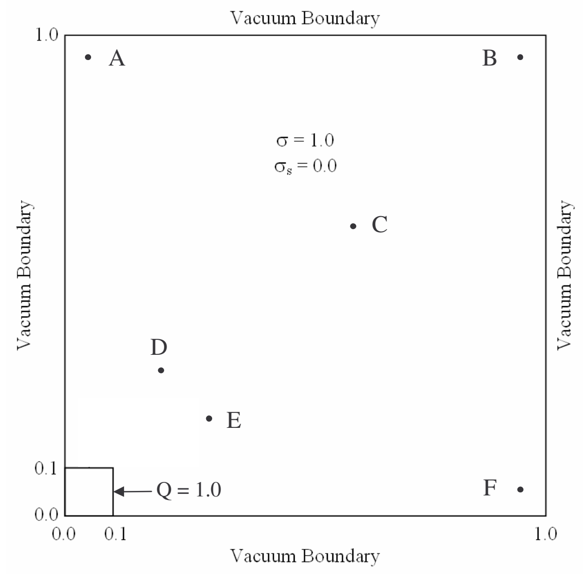


Figure IV–1: Geometry for the test box problem. (Lengths in centimeters.)

source is 0.1 cm square. We will refer to this problem as the “test box” problem. The points labeled A through F have the following coordinates:

A: (0.05 cm, 0.95 cm)

B: (0.95 cm, 0.95 cm)

C: (0.6 cm, 0.6 cm)

D: (0.2 cm, 0.3 cm)

E: (0.3 cm, 0.2 cm)

F: (0.95 cm, 0.05 cm)

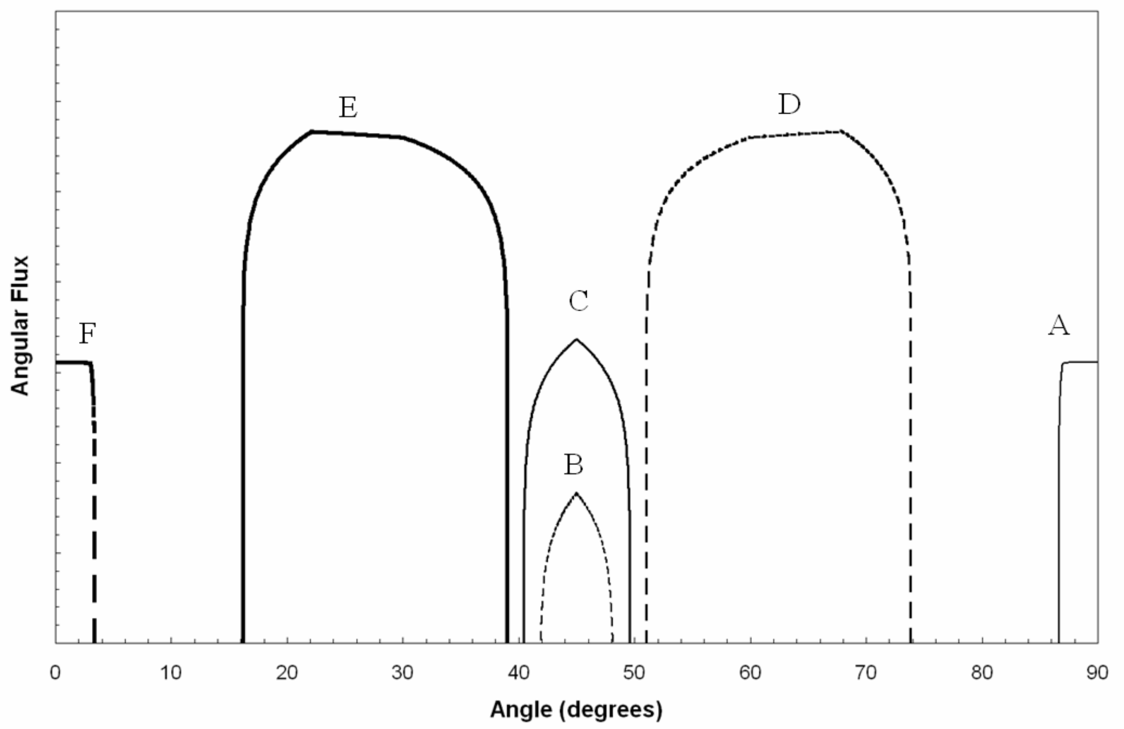


Figure IV–2: Solution of test box problem at specified locations.

The solution of this problem can be obtained analytically using the integral form of the transport equation

$$\psi\left(\vec{r},\vec{\Omega}\right)=\psi\left(\vec{r}-\tau\vec{\Omega},\vec{\Omega}\right)e^{-\alpha\left(\vec{r}-\tau\vec{\Omega},\vec{r}\right)}+\int_0^\tau ds\ S_{ext}\left(\vec{r}-s\vec{\Omega},\vec{\Omega}\right)e^{-\alpha\left(\vec{r}-s\vec{\Omega},\vec{r}\right)} \qquad (3.1)$$

at each of the points specified. In this equation, $\vec{r}-\tau\vec{\Omega}$ is on the boundary of the problem. Thus, the equation becomes

$$\psi\left(\vec{r},\vec{\Omega}\right)=\int_0^\tau ds\ S_{ext}\left(\vec{r}-s\vec{\Omega},\vec{\Omega}\right)e^{-\alpha\left(\vec{r}-s\vec{\Omega},\vec{r}\right)} \qquad (3.2)$$

The analytic angular solution of this problem at the specified points is shown in Figure IV–2. The angle variable is measured from along the x-axis in Figure IV–1.

If we consider the S_2 quadrature set, we examine the flux in four directions at each spatial point. These directions correspond to -135°, -45°, 45°, and 135°. Therefore, the scalar flux reported at points A, C, E, and F would be zero, while that at

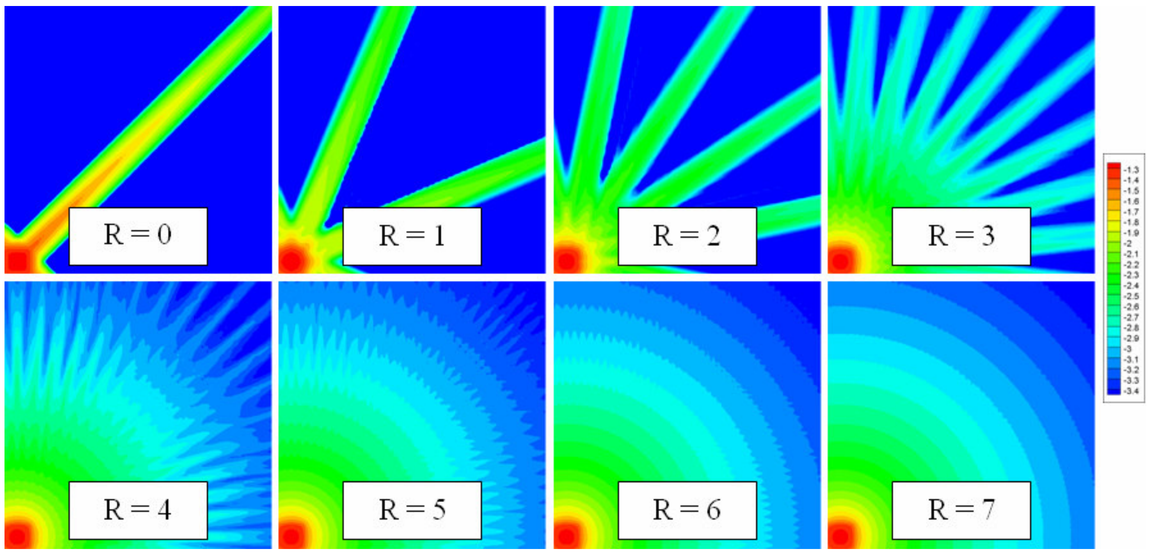


Figure IV–3: Contour solutions of the test box problem.

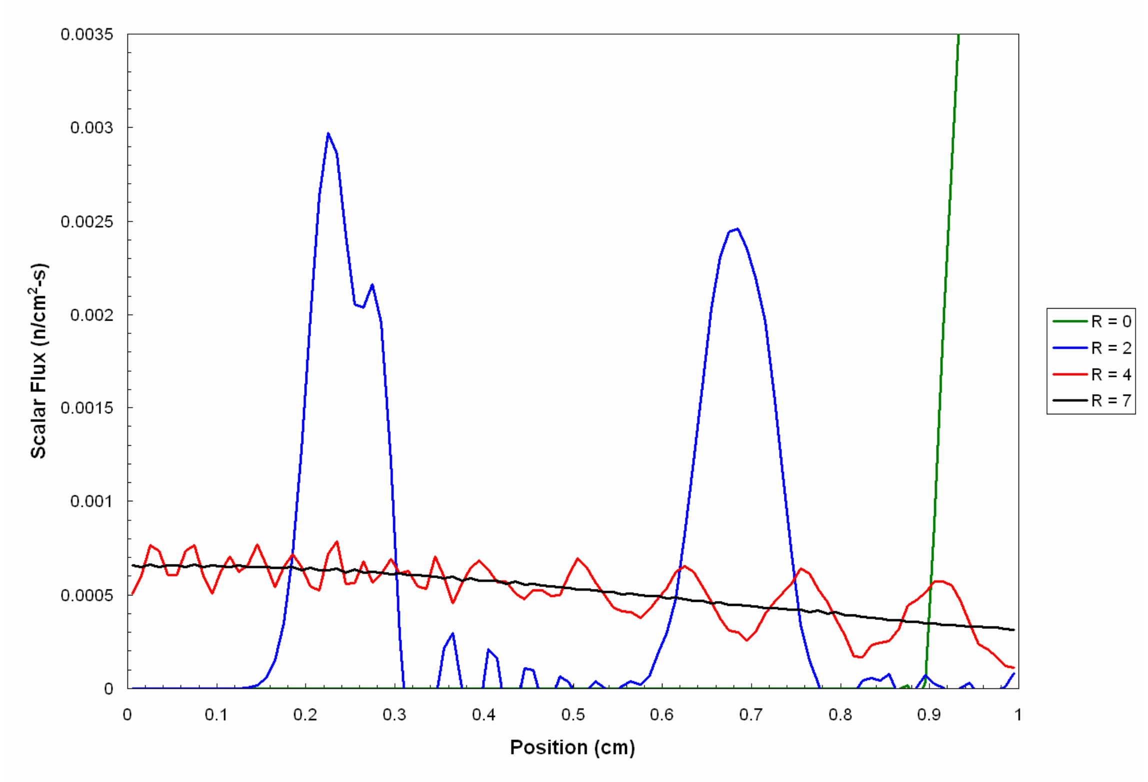


Figure IV–4: Solutions of the test box problem along top edge.

points B and D would be the indicated angular fluxes. Thus, the approximation would be entirely incorrect.

Several logarithmic contour plots of the scalar flux are shown in Figure IV–3 at different resolution levels (R). The solution at resolution level seven is close to the analytic solution, but does still display some ray effects. The ray effects are easily visible at resolution levels lower than seven. Figure IV–4 shows the computed scalar flux along the top edge of the domain. One can see how the ray effects play a predominant role in the level zero and two solutions, but a lesser role in levels four and seven. The highest resolution solution shown (R = 7) uses 128 angles per quadrant per

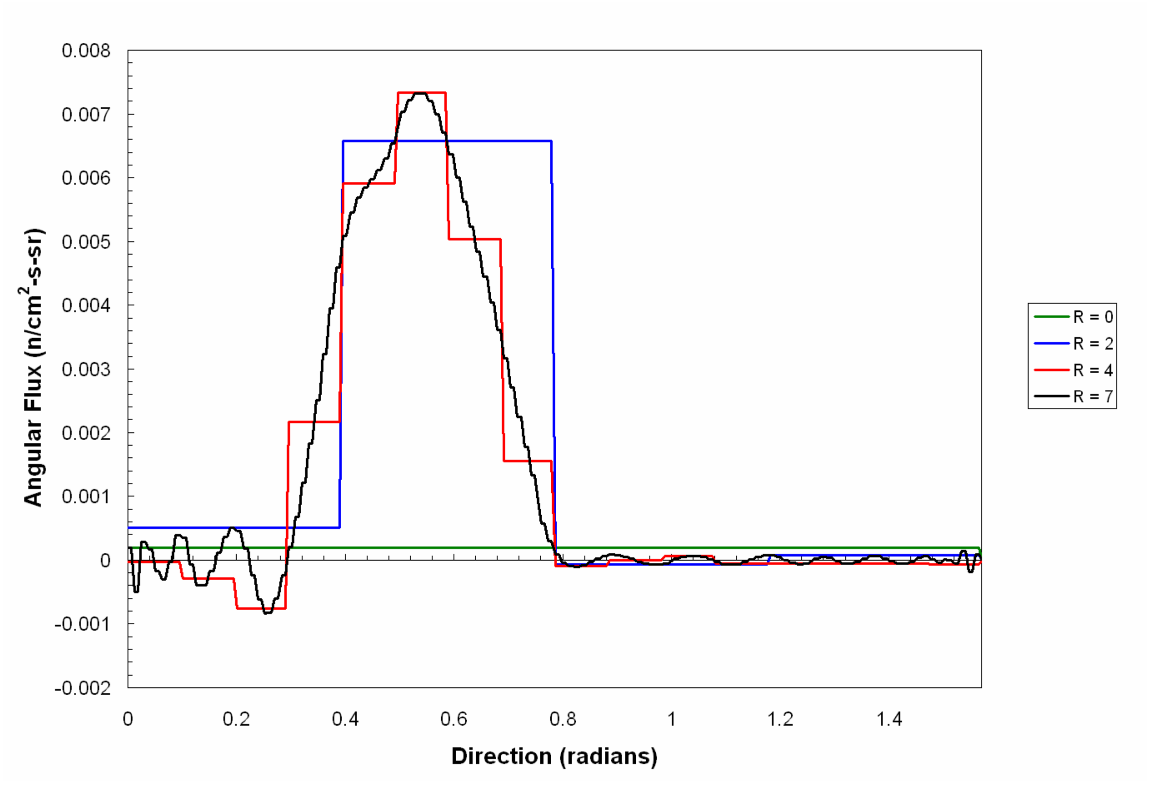


Figure IV–5: Angular flux at point E in test box problem.

cell and a 100×100 cell spatial mesh, for a total of 5,120,000 unknowns. We will use the solution at R = 9 as the reference solution.

The angular flux at point E is shown in Figure IV–5. As R is increased, one can see how the shape of the peak resembles that given in Figure IV–2. One can also get an idea about how wavelets approximate functions at differing resolution levels.

We will consider two additional box-in-box type problems in this dissertation as benchmark problems. Both were taken from Morel.[2] The first is shown in Figure IV–6, and will be referred to as the “first box problem.” It resembles the classic ray-effect problem described by Lathrop,[6] which has subsequently been used in many papers on ray-effect mitigation. It consists of a square domain 2.0 cm in length on each side, with

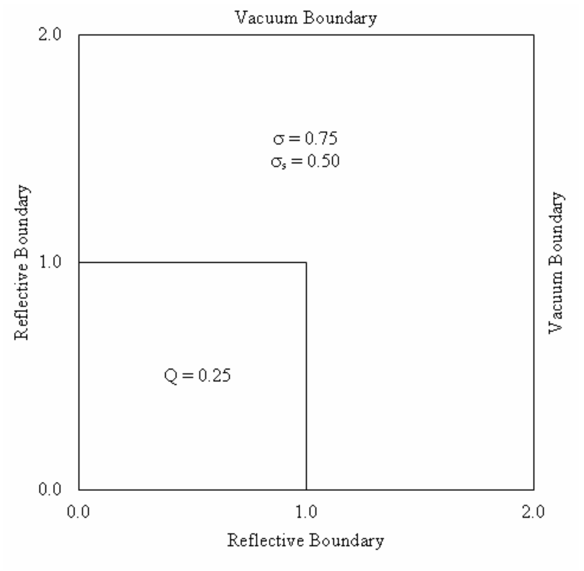


Figure IV–6: Geometry for the first box-in-box problem. (Lengths in centimeters.)

reflective boundary conditions on the left and on the bottom of the domain, and vacuum boundary conditions on the top and right sides. The cross-sections are constant throughout the domain with $\sigma = 0.75$ cm^{-1} and $\sigma_s = 0.5$ cm^{-1}. There is a source located

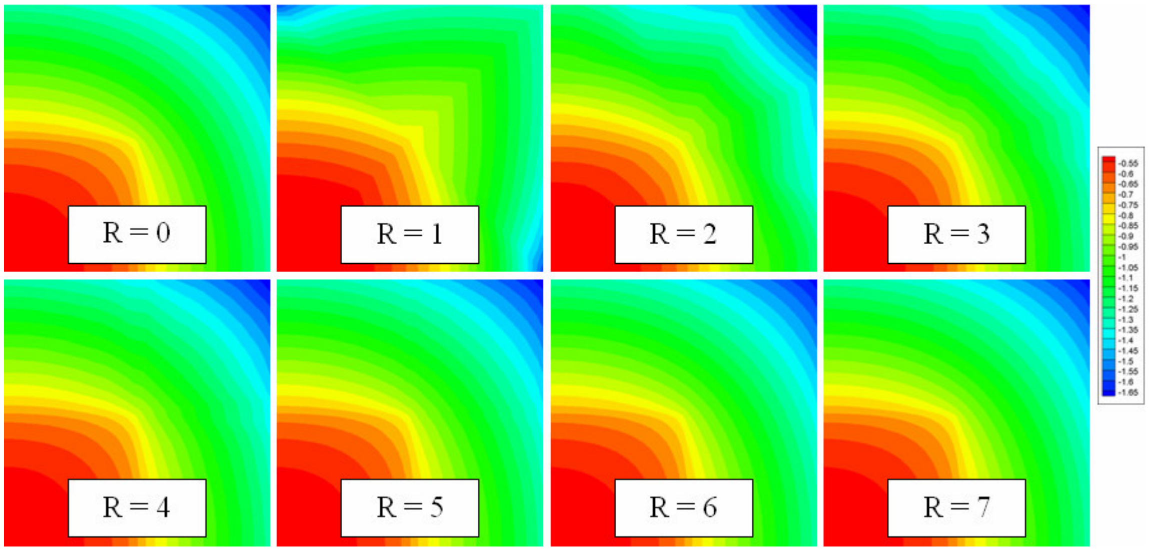


Figure IV–7: Contour solutions of the first box-in-box problem.

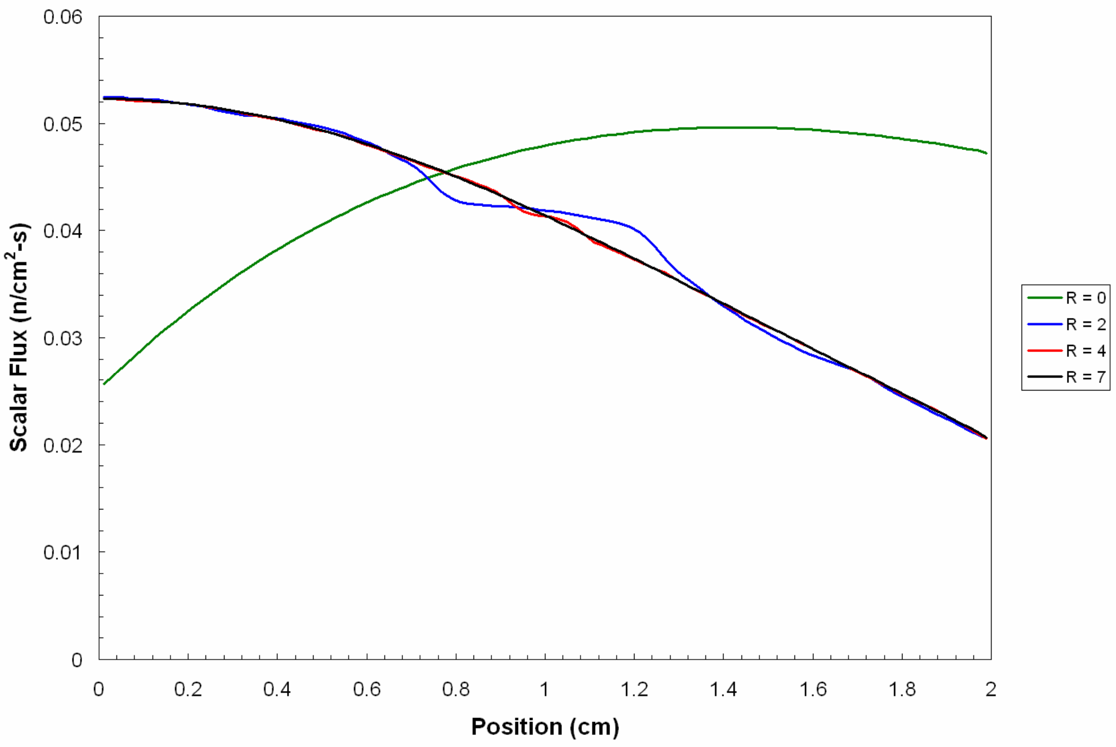


Figure IV–8: Solutions of the first box-in-box problem along top edge.

in the bottom left corner of the domain that is square in shape with dimensions 1.0 cm on each side. This source is isotropic with an intensity of 0.25 particles/sec.

Several logarithmic contour plots of the scalar flux are shown in Figure IV–7 at different resolution levels (R). The solution at resolution level seven is practically identical to the analytic solution. The ray effects are easily visible at resolution levels lower than four. Figure IV–8 shows the computed scalar flux along the top edge of the domain. One can see how the ray effects play a predominant role in the level zero and two solutions, but a lesser role in levels four and six. The highest resolution solution shown (R = 7) uses 128 angles per quadrant per cell and a 100×100 cell spatial mesh, for

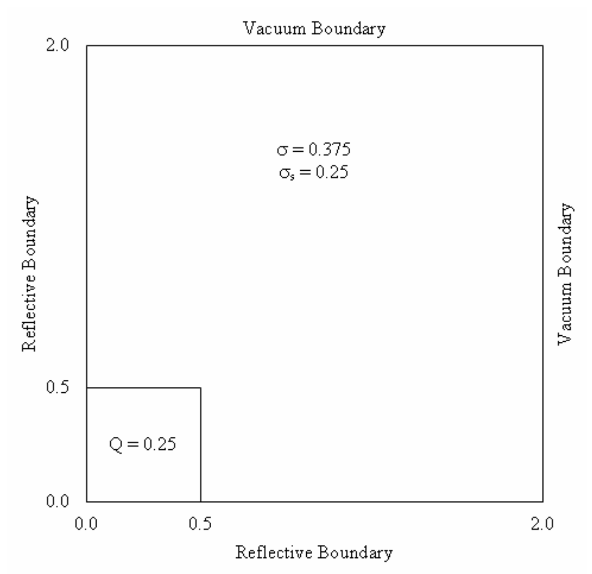


Figure IV–9: Geometry for the second box-in-box problem. (Lengths in centimeters.)

a total of 5,120,000 unknowns per iteration. We will use the solution at R = 5 as the refence solution.

The second box-in-box problem is shown in Figure IV–9. It has been shown[2] that ray effects become more pronounced as a problem approached a line source in a

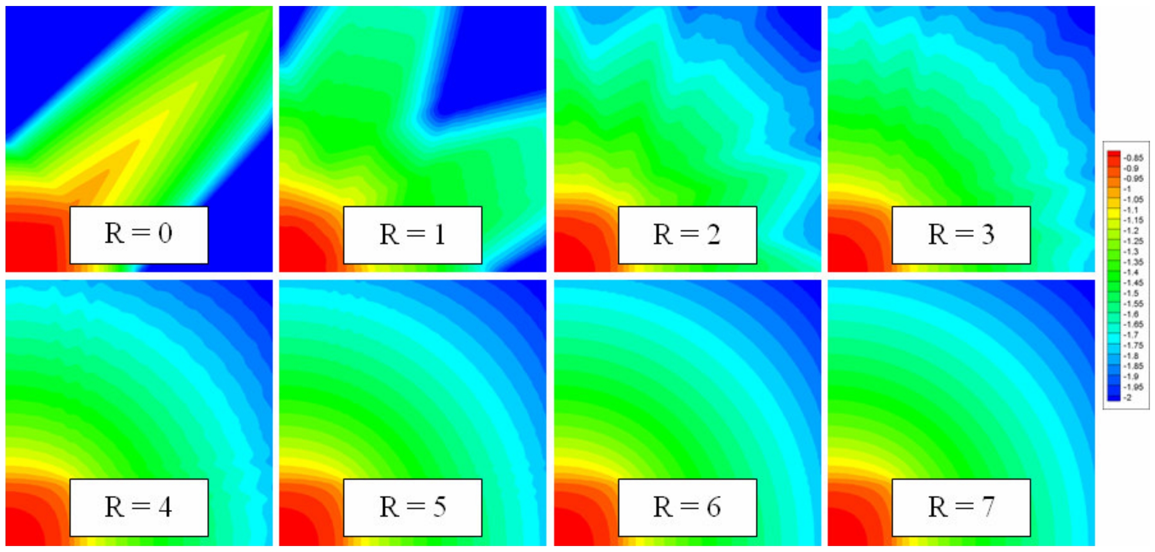


Figure IV–10: Contour solutions of the second box-in-box problem.

void. Thus, to approximate this effect, the second box-in-box problem has a smaller source spatially and a smaller total cross-section. The spatial domain is a square 2.0 cm in width. It has reflective boundary conditions on the left and bottom faces, and vacuum boundary conditions on the top and right faces, as does the first problem. The cross sections are constant throughout the domain with $\sigma = 0.375\ \text{cm}^{-1}$ and $\sigma_s = 0.25\ \text{cm}^{-1}$. The source is a square located in the bottom left corner, and it is 0.5 cm in width. The source is isotropic with an intensity of 0.25 particles/sec.

Several logarithmic contour plots of the scalar flux, using the S_N algorithm, are shown in Figure IV–10 at different resolution levels. The solution at resolution level

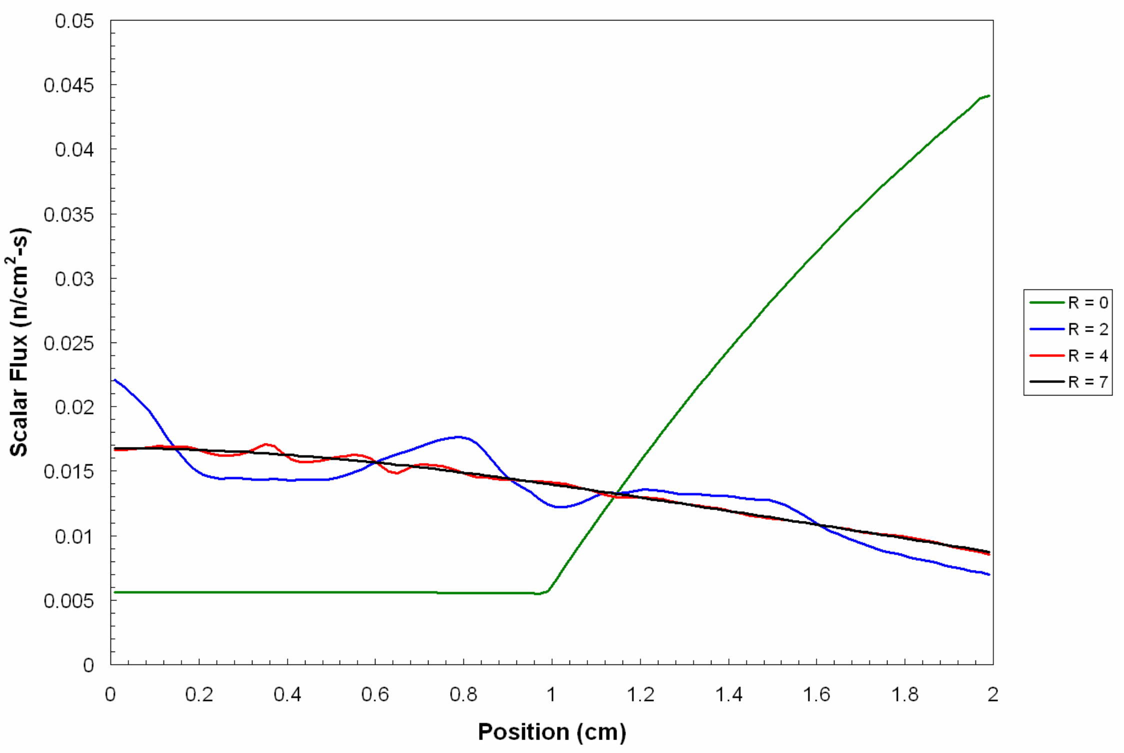


Figure IV–11: Solutions of the second box-in-box problem along top edge.

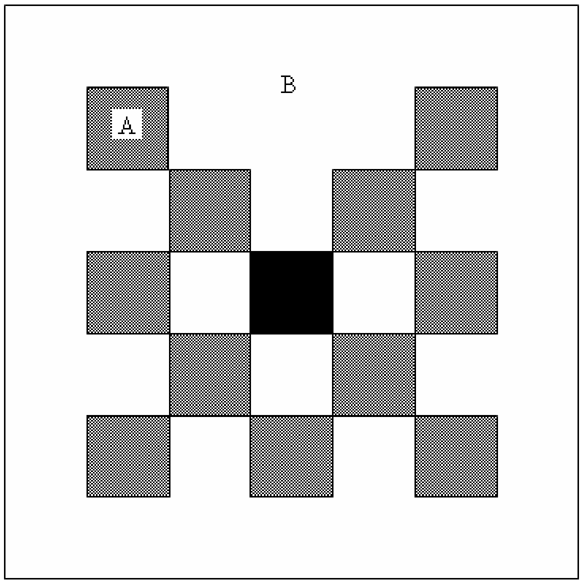


Figure IV–12: Geometry for the lattice problem.

seven is practically identical to the analytic solution. In this problem, ray effects are easily visible up to a resolution level of five. Figure IV–11 shows the scalar flux along the top edge of the domain. The prominent ray effects are visible at levels zero and two. The highest resolution solution shown (R = 7) uses 128 angles per quadrant per cell and

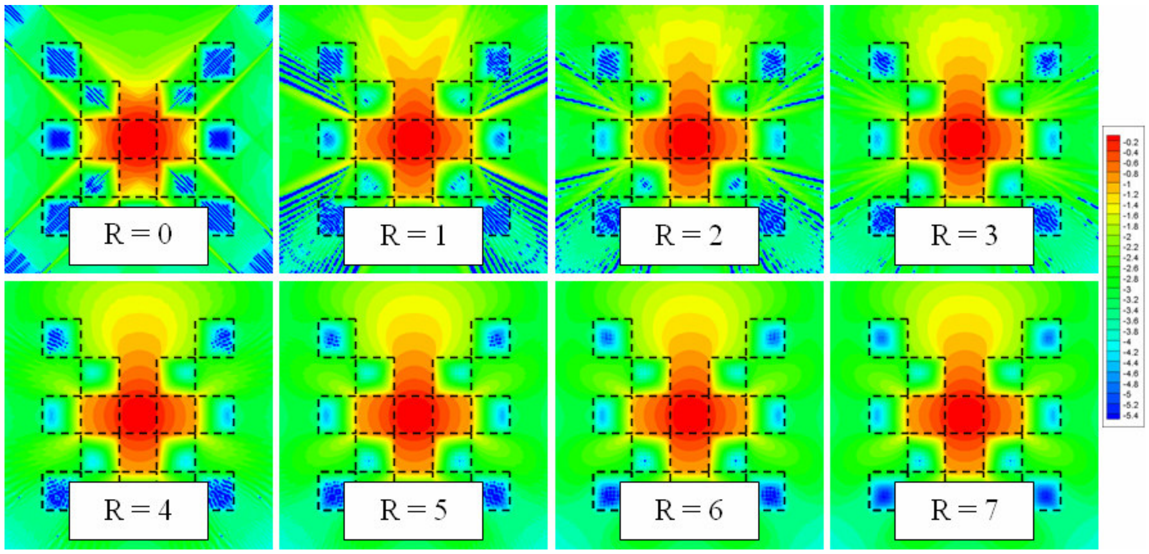


Figure IV–13: Contour solutions of the lattice problem.

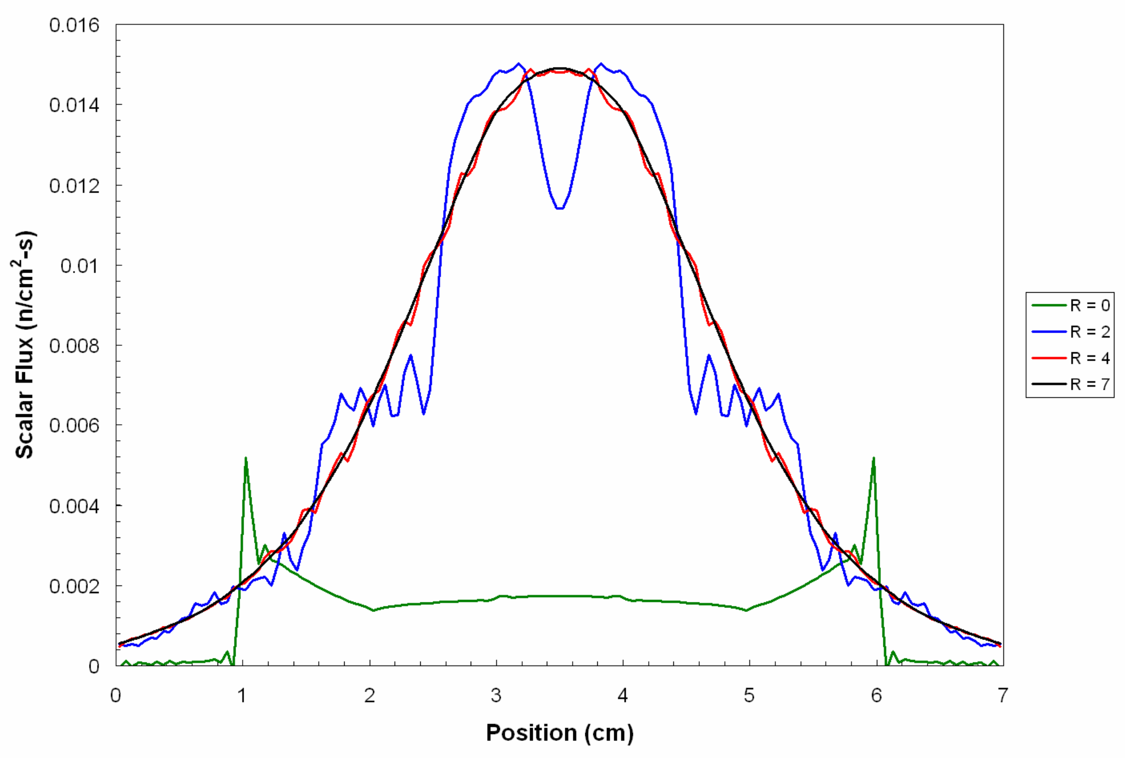


Figure IV–14: Solutions of the lattice problem along top edge.

a 100×100 cell spatial mesh, for a total of 5,120,000 unknowns per iteration. We will use the solution at R = 7 as the reference solution.

The Brunner Problems

Two other, and more realistic, examples that demonstrate the ray-effect phenomenon were created by Thomas Brunner[20] of Sandia National Laboratory. The first example, referred to as the "Lattice problem," is shown in Figure IV–12. The domain is square and the lattice is composed of squares of width 1.0 cm. The lattice is made up of squares (A) of pure absorbers, with $\sigma = 10$ cm^{-1}. The rest of the domain (B) is purely scattering with $\sigma = \sigma_s = 1$ cm^{-1}. The source is in the center square of the lattice

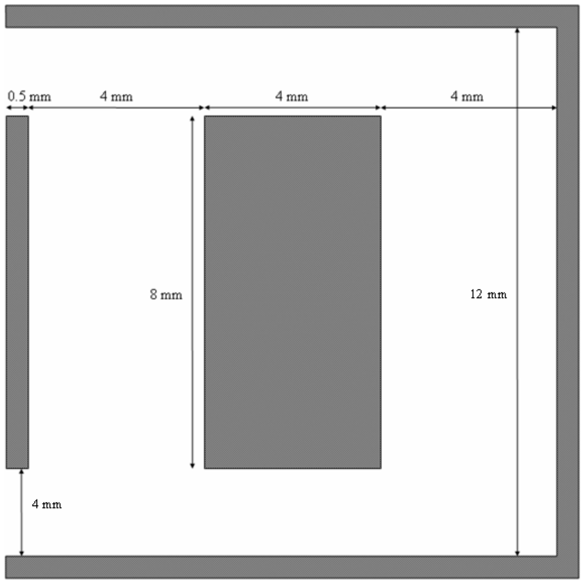


Figure IV–15: Geometry for the Hohlraum problem.

and is isotropic with an intensity of 1.0 particles/sec. The source square is also of material A.

Several logarithmic contour plots of the scalar flux are shown in Figure IV–13 at

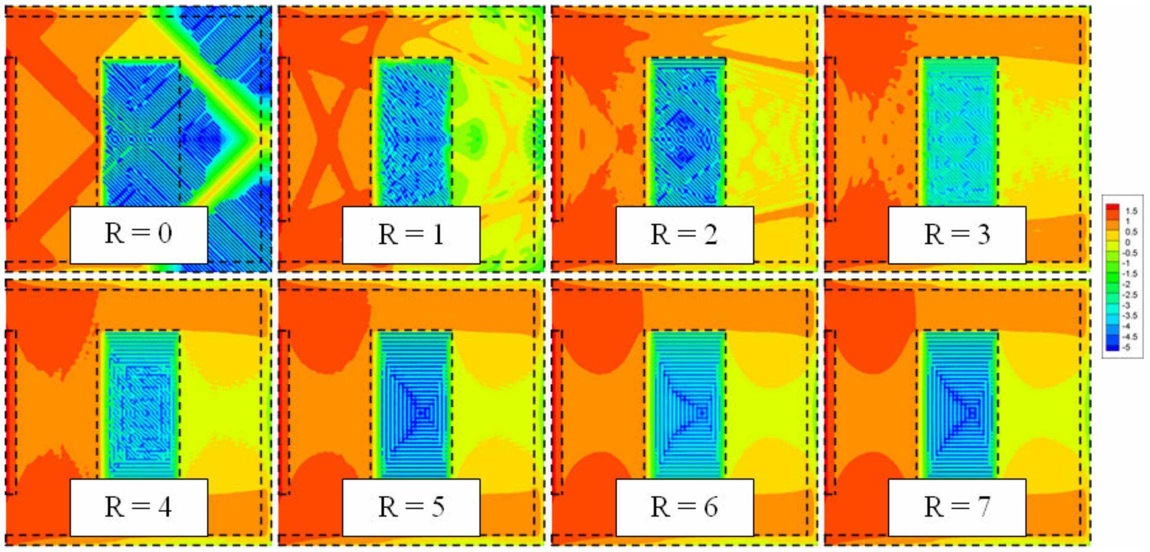


Figure IV–16: Contour solutions of the Hohlraum problem.

different resolution levels. In this problem, when compared to resolution level seven, ray effects are easily visible up to a resolution level of four. Figure IV–14 shows the scalar flux along the top edge of the domain. The prominent ray effects are visible at levels zero and two. The highest resolution solution shown (R = 7) uses 128 angles per quadrant per cell and a 140×140 cell spatial mesh, for a total of 10,035,200 unknowns per iteration. We will use the solution at R = 6 as the reference solution.

The second Brunner example, referred to as the "Hohlraum problem," is shown in Figure IV–15. The domain is square with width of 13 mm. The walls of the hohlraum are purely scattering, with $\sigma = \sigma_s = 100 \text{ cm}^{-1}$, and the target (the large,

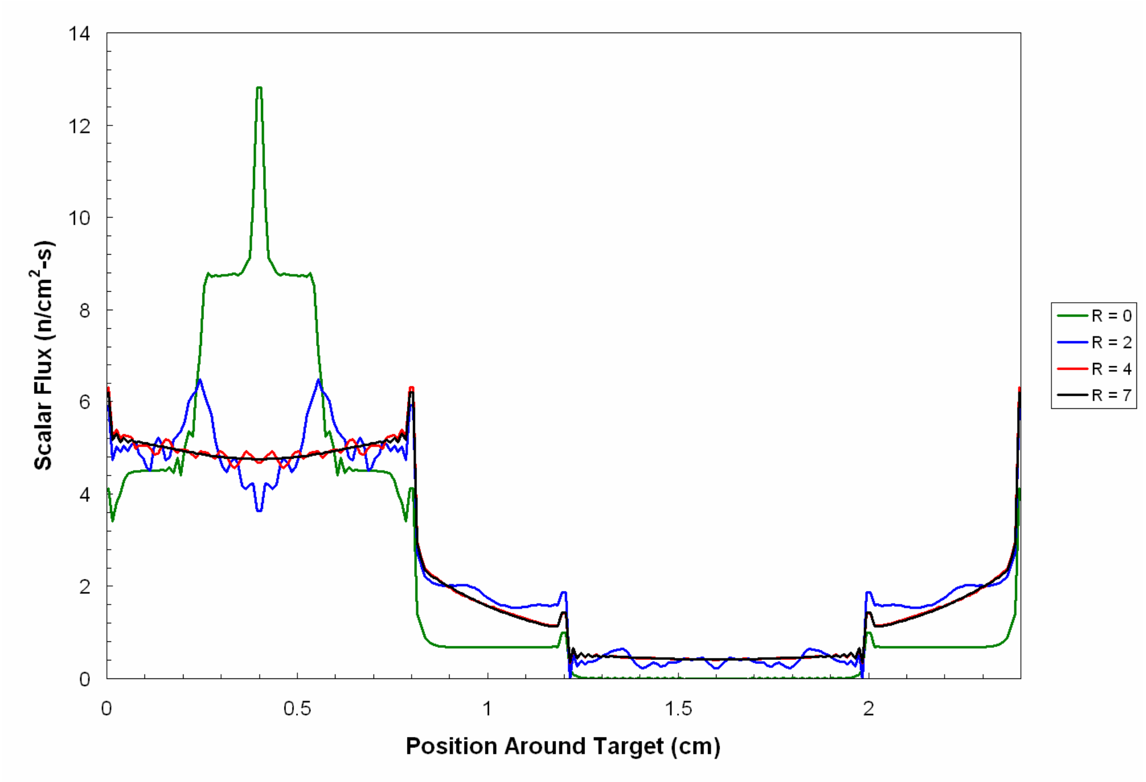


Figure IV–17: Solutions of the Hohlraum problem around the target.

centrally located absorber) is a pure absorber with $\sigma = 100$ cm^{-1}. The rest of the domain is a void. There is an isotropic incident flux with intensity 1.0 particles/sec along the entire length of the left wall.

Several logarithmic contour plots of the scalar flux are shown in Figure IV–16 at different resolution levels. In this problem, when compared to resolution level seven, ray effects may be seen above, left, and below the target up to a resolution level of six. Figure IV–17 shows the scalar flux around the perimeter of the target area, with zero being in the lower left corner of the target and the coordinates running clockwise to 2.4 cm. The prominent ray effects are visible at levels zero and two. The highest resolution solution shown (R = 7) uses 128 angles per quadrant per cell and a 130×130 cell spatial mesh, for a total of 8,652,800 unknowns. We will use the solution at R = 7 as the reference solution.

CHAPTER V

WAVELETS AND MULTIRESOLUTION ANALYSIS

In this chapter, we will define the scaling and wavelet functions, and describe the multiresolution analysis that they can generate. We will also describe differing forms of 1-D wavelets, including Haar, Chui-Wang, and Daubechies wavelets, as well as 3-D spherical wavelets. This theory will be used in Chapter VI to derive equations using wavelets in angle for the transport equation.

Wavelet Theory

A way to attack the problem of ray effects is by using an adaptive discretization in the angular variable. At many positions, there are certain angles that provide no information regarding the angular shape of the flux. This can be seen by looking at Figure IV–2, at point C, any angle smaller than 40.5° or larger than 49.5° does not contribute to the value of the scalar flux. An adaptive method would be able to recognize these positions and only use those directions that contribute to the scalar flux calculation, while ignoring those that do not.

Wavelet theory involves representing general functions in terms of simpler, fixed building blocks at different scales and positions. The following account of the history of wavelets is taken from the paper by Jawerth and Sweldens[21]:

> In abstract mathematics, is has been known for quite some time that techniques based on Fourier series and Fourier transforms are not quite adequate for many problems and so-called *Littlewood-Paley techniques* often are effective substitutes. These techniques were initially developed in the 30's to understand, among other things, summability

properties of Fourier series and boundary behavior of analytic functions. In the 50's and 60's, these developed into powerful tools for studying other things, such as solutions of partial differential equations and integral equations. It was realized that they fit into *Calderón-Zygmund theory*, an area of harmonic analysis that is still very heavily researched.

In the early 80's, Strömberg discovered the first orthogonal wavelets[22]....In the early to mid 80's, several groups…independently realized…that tools from Calderón-Zygmund theory have discrete analogs that could give a unified view of many of the results in harmonic analysis. Also, one started to understand that these techniques could be effective substitutes for Fourier series in numerical applications.

As the emphasis shifted more towards the representations themselves, and the building blocks involved, the name of the theory also shifted. Grossmann and Morlet suggested the word "wavelet" for the building blocks, and what earlier had been referred to as Littlewood-Paley theory, now started to be called wavelet theory.

With the notion of multiresolution analysis, introduced by Mallat and Meyer, a systematic framework for understanding these orthogonal expansions was developed....Soon, Daubechies[23] gave a construction of wavelets, nonzero only on a finite interval and with arbitrarily high, but fixed, regularity.

Wavelets as a solution methodology were first applied to partial differential equations by Maday, Perrier, and Ravel[24] in 1991. They proposed the concept of adaptivity for the approximation of partial differential equations using decomposition in a wavelet basis. They then generalized their method for the case of Dirichlet boundary conditions and for multidimensional problems.[25] However, they only considered using wavelets in the spatial domain. Most previous applications of wavelets to solving PDEs have followed a similar evolution.

Little work has been done involving the use of wavelet bases to solve hyperbolic partial differential equations. One such method[26] is based on an interpolating wavelet transform using polynomial interpolation on dyadic grids. This is equivalent to classical wavelet approximations. The conclusion is that wavelet-based expansion works for hyperbolic partial differential equations, with a significant improvement in terms of computational time. However, the wavelet expansion was only used in the spatial domain. There has been no work done on using wavelets in direction adaptively to solve hyperbolic transport equations.

Wavelet-based solution methods have been used successfully, if only recently, in many other areas of mathematics and engineering to solve differential equations. Wavelets exhibit two unique properties:

- Both wavelet functions and scaling functions are locally supported. This means they can be used to evaluate sharp local variations in angular flux at a given angular position without requiring a large resolution throughout the angular domain.
- The zero-order moment of a scaling function is nonzero, while that of a wavelet function is zero. Thus, for the application to transport problems, the scalar flux can be completely determined by the scaling function's magnitude. The wavelet functions then only describe the variation in angle of the angular flux.

Using these properties, the angular flux at point C in Figure IV–2, for example, could be represented using only three scaling functions that are piecewise-constant. This could mean a large reduction in the number of unknowns required to resolve the solution of a given problem adequately.

A functional expansion similar to wavelets was used in particle transport by Thomas Seed in 1976[16]. He used Walsh functions in a functional expansion of the transport equation. Walsh functions are similar to wavelets in that they are composed of piecewise-constant functions and form a complete basis for the angular space at a given resolution level. However, it is not obvious how Walsh functions can be used in an adaptive discretization of the transport equation.

Multiresolution Analysis

A wavelet basis is defined in the following way: If *f* is a real-valued function, then $f_{r,\lambda}(\theta) := 2^{r/2} f\left(2^r\theta - \lambda\right)$ is a (binary) scaled (r) translate (λ) of *f*. If for each "resolution level" *r* (a non-negative integer), *f* satisfies the "refinement condition"

$$f(\theta) \in \mathbf{A}_r := \operatorname{span}\left\{ f_{r,\lambda}(\theta) : \lambda \in I_r \right\} \tag{5.1}$$

for some finite set I_r of "localization indices" λ, then *f* is a "scaling function." As $\mathbf{A}_{r-1} \subseteq \mathbf{A}_r$, and all spaces involved are finite-dimensional, there exists some direct complement of $\mathbf{A}_{r-1}$ in $\mathbf{A}_r$, say $\mathbf{W}_{r-1}$. The $\mathbf{W}_r$ are called "wavelet spaces," and their elements are called "wavelets."

The conditions of the preceding paragraph seem suited to angular approximation in transport computations. Increasing indices *r* provide finer resolution in the angular variable. Suppose there exists some coarsest meaningful resolution 0, and some finest resolution level *N* such that the angular flux can be adequately resolved in $\mathbf{A}_N$, but doing so in the "natural" basis suggested by (5.1) will be computationally costly. Possibly by appropriately choosing the wavelet spaces, and bases in those spaces, then resolving the angular flux pursuant to the representation

$$\mathbf{A}_N = \mathbf{W}_{N-1} \oplus \mathbf{W}_{N-2} \oplus \cdots \oplus \mathbf{W}_0 \oplus \mathbf{A}_0, \tag{5.2}$$

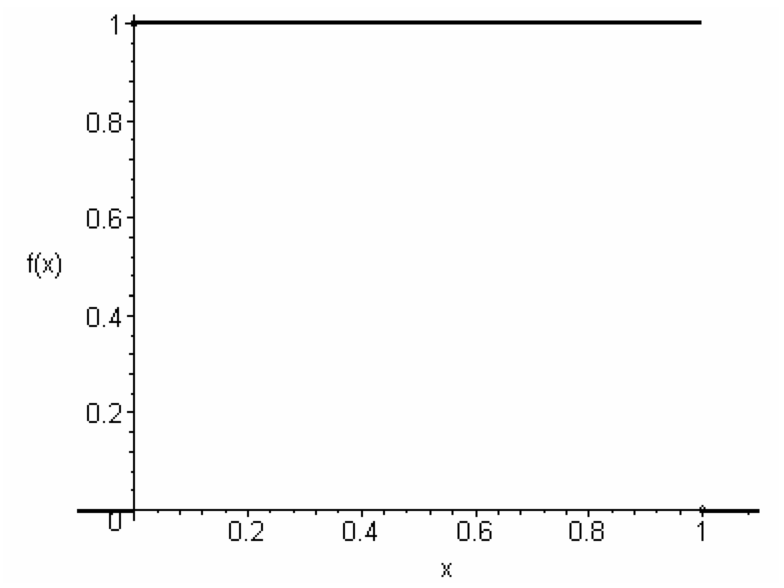


Figure V–1: Haar scaling function.

the required resolution can be attained with fewer coefficients and less computational effort. Any success in achieving even the first of these objectives depends upon the choice of wavelet spaces and bases within those spaces. We choose to specify the wavelet spaces $\mathbf{W}_{r-1}$ uniquely as orthogonal to $\mathbf{A}_{r-1}$. We refer to the use of wavelet bases in such a way as a W_N method.

Haar Wavelets and Other Wavelet Bases

The simplest wavelet basis is the Haar[27] basis composed of piecewise constant functions. This basis consists of the scaling function

$$f(x) = \begin{cases} 1, & 0 \le x < 1 \\ 0, & \text{otherwise} \end{cases}.$$

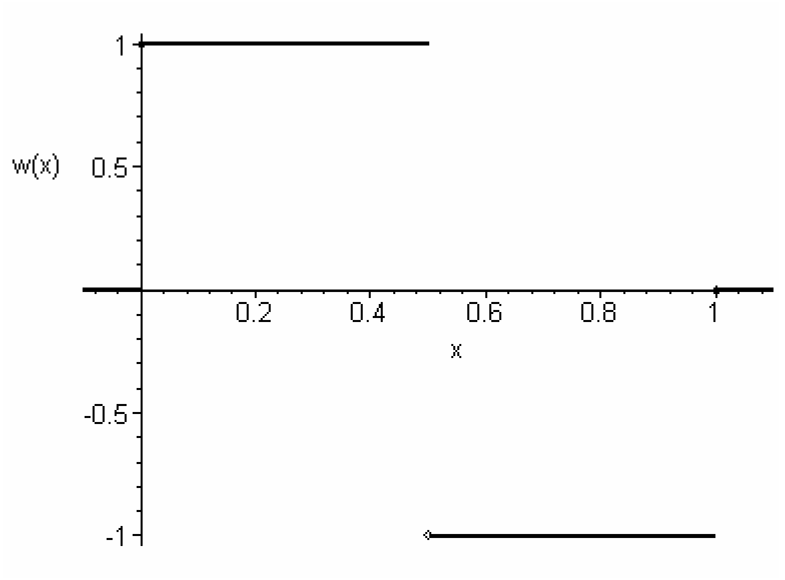


Figure V–2: Haar wavelet function.

The corresponding wavelet function is

$$w(x) = \begin{cases} 1, & 0 \le x < \frac{1}{2} \\ -1, & \frac{1}{2} \le x < 1 \\ 0, & \text{otherwise} \end{cases} .$$

The Haar scaling function is shown in Figure V–1 and the corresponding wavelet function in Figure V–2.

We have chosen to use the Haar basis for our applications in this dissertation. This formulation of the wavelet basis is chosen because of its mathematical simplicity in formulating a working solution methodology. As such, it is only suitable for 2-D Cartesian and 1-D geometries. This version of the W_N method using the Haar basis scaling and wavelet functions will be referred to as the Haar W_N method.

There do exist more complicated bases than the Haar basis. The Chui-Wang B-Spline scaling functions are defined by the recursive formula

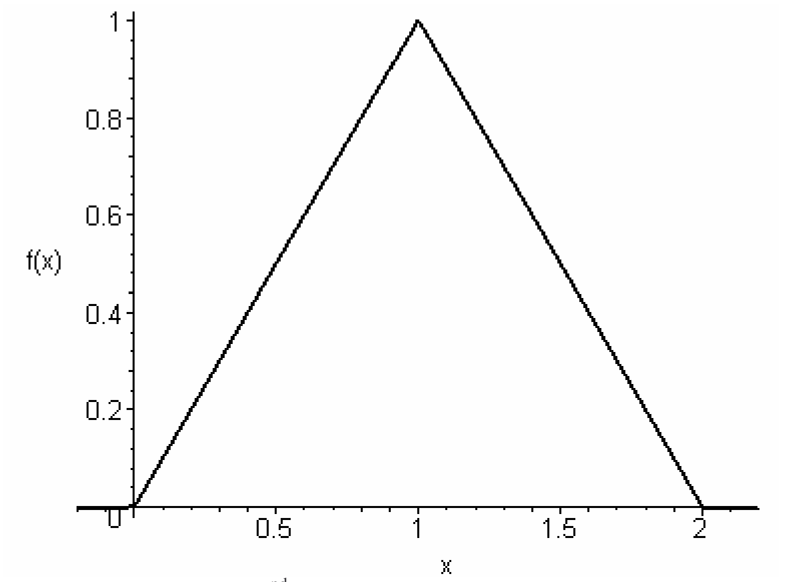


Figure V–3: 2nd-order Chui-Wang scaling function.

$$f(x) = N_m(x) = \int_0^1 N_{m-1}(x-t)\,dt \tag{5.3}$$

where $N_0(x)$ is the Haar scaling function. The corresponding wavelet function is given by

$$w(x) = 2^{-m+1} \sum_{k=0}^{2m-2} (-1)^k N_{2m}(k+1) N_{2m}^{(m)}(2x-k) \tag{5.4}$$

where

$$N_{2m}^{(m)}(x) = \sum_{k=0}^{m} (-1)^k \binom{m}{k} f(x-k). \tag{5.5}$$

The 2nd-order Chui-Wang scaling function is then defined as

$$N_2(x) = \begin{cases} x, & 0 \le x < 1 \\ 2-x, & 1 \le x < 2 \\ 0, & \text{otherwise} \end{cases}$$

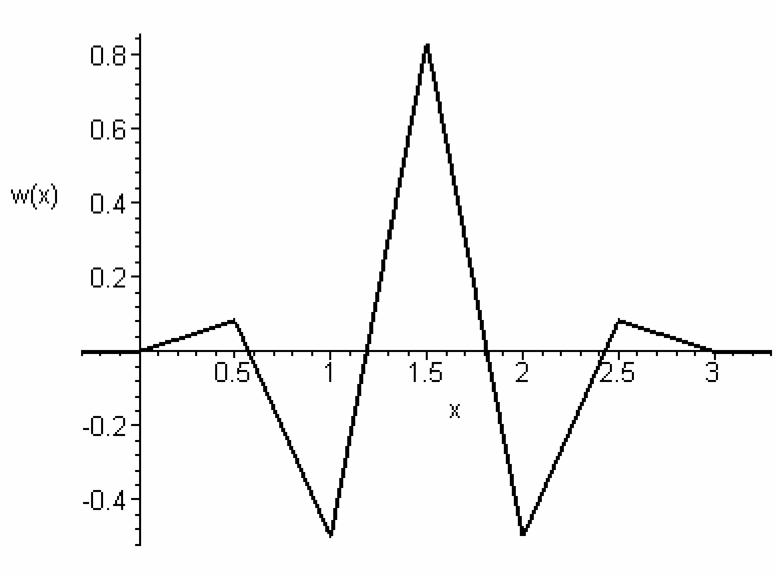


Figure V–4: 2nd-order Chui-Wang wavelet function.

with associated wavelet

$$w(x)=\begin{cases} \frac{1}{6}x, & 0\le x<\frac{1}{2} \\ -\frac{7}{6}x+\frac{2}{3}, & \frac{1}{2}\le x<1 \\ \frac{8}{3}x-\frac{19}{6}, & 1\le x<\frac{3}{2} \\ -\frac{8}{3}x+\frac{29}{6}, & \frac{3}{2}\le x<2 \\ \frac{7}{6}x-\frac{17}{6}, & 2\le x<\frac{5}{2} \\ -\frac{1}{6}x+\frac{1}{2}, & \frac{5}{2}\le x<3 \\ 0, & \text{otherwise} \end{cases}.$$

The 2nd-order Chui-Wang scaling function is shown in Figure V–3 and the corresponding wavelet function in Figure V–4. Likewise, the 3rd-order Chui-Wang scaling function is shown in Figure V–5 and the corresponding wavelet function in Figure V–6.

A more powerful wavelet basis was introduced by Daubechies[28]. The Daubechies bases do not have analytic expressions. Rather, they are defined recursively by solving the equation

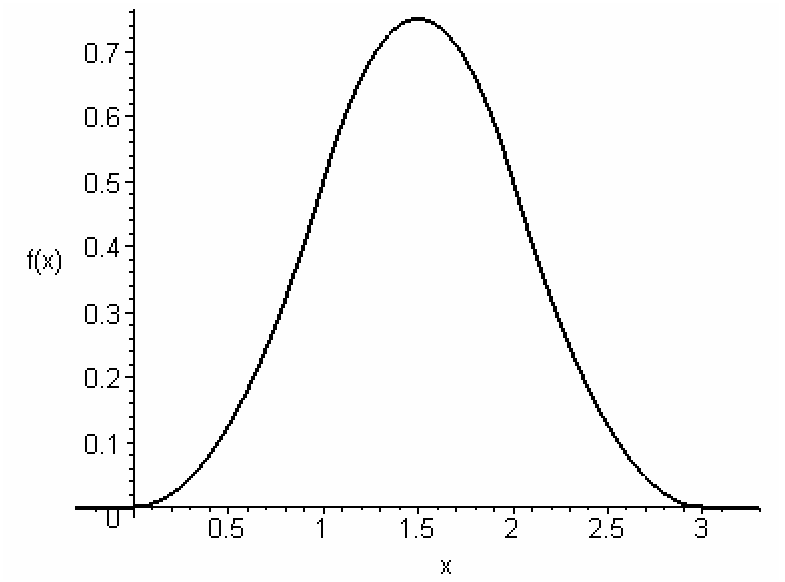


Figure V–5: 3rd-order Chui-Wang scaling function.

$$\phi(x) = \sum_k p_k \phi(2x - k) \tag{5.6}$$

given the coefficients p_k and initial values of the scaling function at integer values of *x*. To find these initial values, one must solve the system

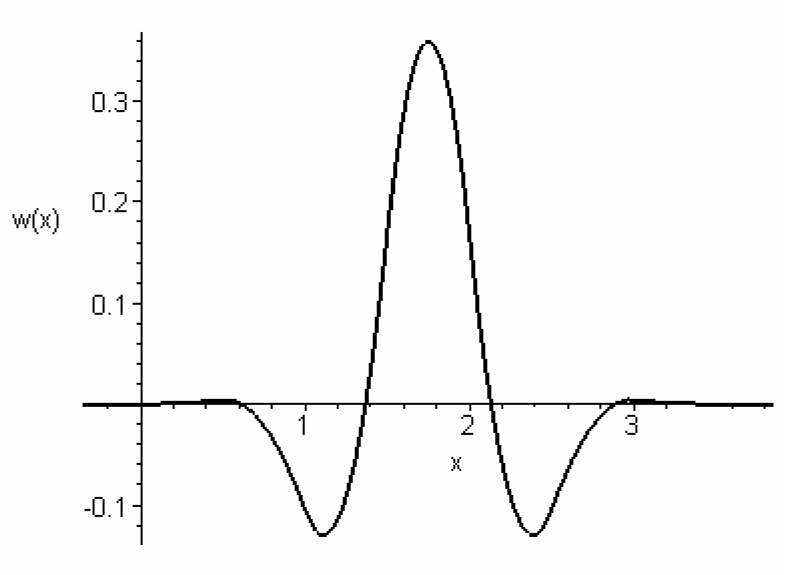


Figure V–6: 3rd-order Chui-Wang wavelet function.

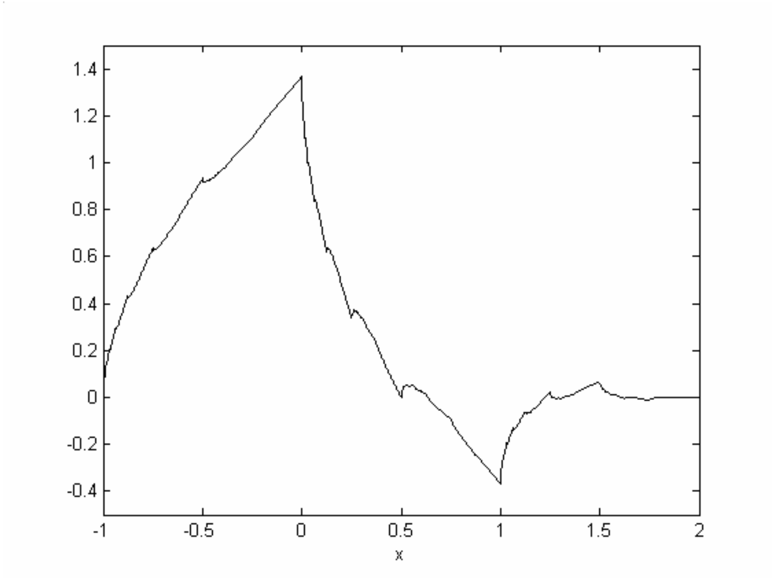


Figure V–7: 2nd-order Daubechies scaling function.

$$\phi(j)=\sum_{k=0}^{2N-2} p_k \phi(2j-k), \qquad j=1,2,\ldots,2N-2 . \tag{5.7}$$

However, this system is singular, so one of the equations must be replaced by

$$\sum_{k=0}^{2N-2} \phi(k)=1 . \tag{5.8}$$

Once equations (5.7) and (5.8) are solved, one can then use equation (5.6) to solve for the values at dyadic points (i.e. $x=k\cdot 2^{-n}$) recursively. Daubechies[28] gave numerical values for the coefficients p_k, which are not unique. The wavelet functions are then calculated from the scaling function, using the relation

$$\psi(x)=\sum_k q_k \phi(2x-k), \tag{5.9}$$

where

$$q_k=(-1)^k p_{1-k} . \tag{5.10}$$

The Daubechies 2nd-order scaling function is shown in Figure V–7. The corresponding wavelet function is shown in Figure V–8.

Finally, there exist wavelets on the sphere[29,30] that are continuous in angle and would be able to solve the transport equation while adapting in both polar and azimuthal directions. Because they are located on the surface of a sphere, there are three manipulations that are possible: dilations, translations, and rotations. Since translations and rotations are both motion operators, we will define[30] a motion as

$$\left(R_\rho f\right)(\theta,\xi)=f\left(\rho^{-1}\theta,\rho^{-1}\xi\right)=\left(U_{qr}\left(\rho\right)f\right)(\theta,\xi), \tag{5.11}$$

and a dilation as

$$\left(D_a f\right)(\theta,\xi)=f_a\left(\theta,\xi\right)=\lambda\left(a,\theta\right)^{1/2} f\left(\theta_{1/a},\xi\right), \tag{5.12}$$

where $\tan\left(\frac{1}{2}\theta_{1/a}\right)=\frac{1}{a}\tan\left(\frac{1}{2}\theta\right)$ and

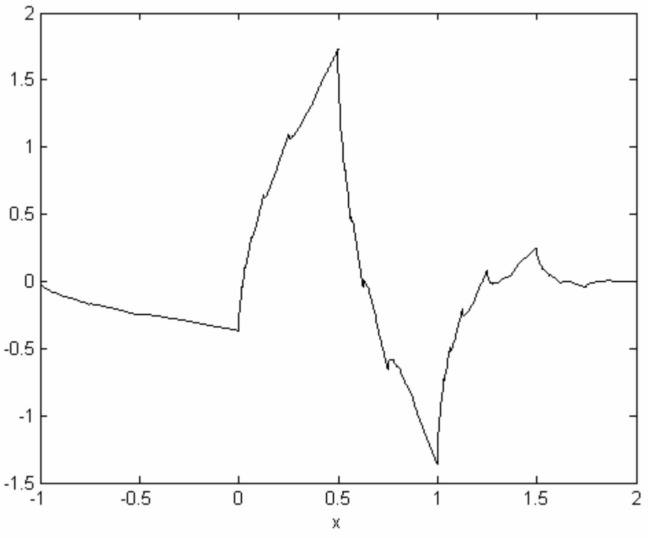


Figure V–8: 2nd-order Daubechies wavelet function.

$$\lambda(a,\theta) = \frac{4a^2}{\left[\left(a^2-1\right)\cos\theta + \left(a^2+1\right)\right]^2}.$$

The details of these definitions are given in the references cited above. Once one has determined the scaling function, the wavelet may be computed using the difference formula

$$\psi^{(\alpha)}(\theta,\xi) = \phi(\theta,\xi) - \tfrac{1}{\alpha} D_\alpha \phi(\theta,\xi), \qquad \alpha > 1. \tag{5.13}$$

The most well-known spherical wavelet is the spherical DOG wavelet, which is derived from the scaling function $\phi(\theta,\xi) = \exp\left(-\tan^2(\theta/2)\right)$, $\theta \in [-\pi,\pi]$. In addition, it is also possible to construct a spherical wavelet or scaling function from a two-dimensional Euclidean wavelet or scaling function. The transform is

$$\psi_S(\theta,\xi) = \frac{2\psi\left(2\tan(\theta/2),\xi\right)}{1+\cos\theta}, \tag{5.14}$$

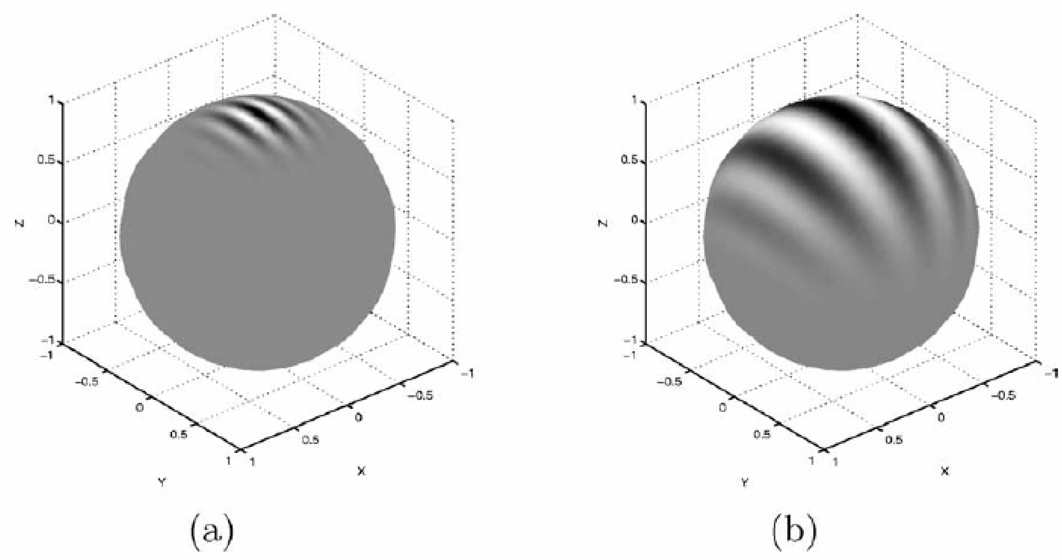


Figure V–9: Real part of the spherical Morlet wavelet at scales a = 0.03 (a) and a = 0.3 (b).

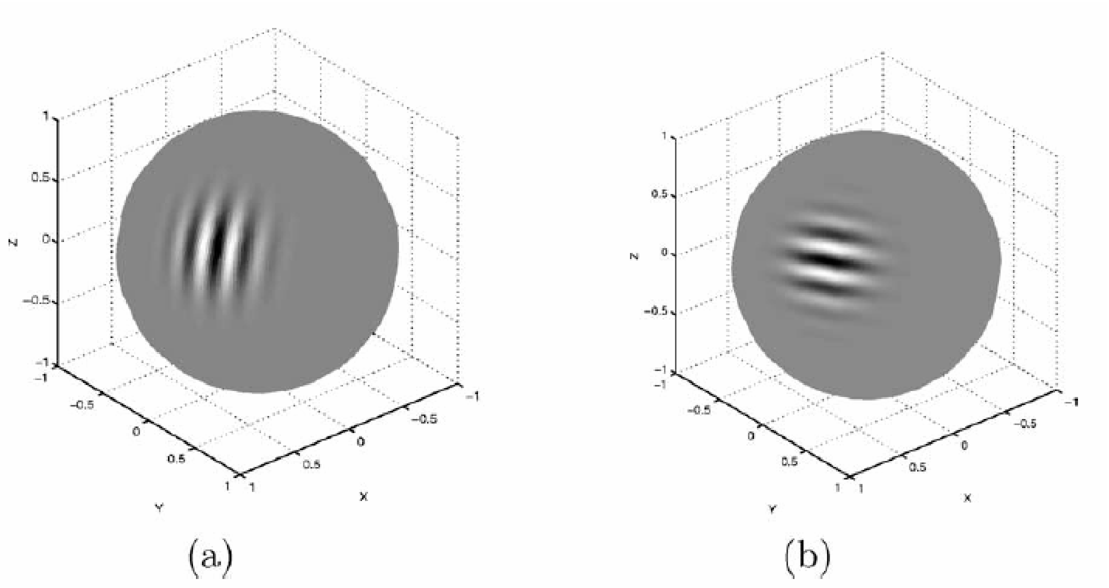


Figure V–10: Real part of the spherical Morlet wavelet at scale a = 0.03 and centered at (π/3, π/3), with rotation 0 (a) and π/2 (b).

where ψ_S is the spherical wavelet and ψ is the Euclidean wavelet. An example, taken from our reference paper[30], uses the Euclidean Morlet wavelet, defined as

$$\psi(\vec{x}) = e^{i\vec{k}_0\cdot\vec{x}} e^{-\|\vec{x}\|^2}.$$

The corresponding spherical wavelet, using (5.14), is then

$$\psi(\theta,\xi) = \frac{e^{ik_0\tan(\theta/2)\cos(\xi_0-\xi)} e^{-\frac{1}{2}\tan^2(\theta/2)}}{1+\cos\theta}.$$

This wavelet is shown in Figures V–9 and V–10 at different resolution levels, centers, and rotations.

We use the Haar basis in the codes presented in this dissertation in order to assess the potential of wavelets in adaptively solving the transport equation in angle. However, for adaptation in the azimuthal angle, either Chui-Wang or Daubechies wavelets would be ideal. For adaptation in both azimuthal and polar angles, wavelets on the sphere would be the best choice.

CHAPTER VI

APPLICATION TO TRANSPORT COMPUTATIONS

In this chapter we will apply the definition of a multiresolution analysis using wavelets to the transport equation. From the resulting system of equations, we will build the S_N-W_N algorithm in Chapter VIII and the CW-W_N algorithm in Chapter X.

For our application to (azimuthal angular representations in) transport computations the scaling functions are

$$f_{2,\lambda}(\theta) = \begin{cases} 2, & \frac{\pi}{2}(\lambda-1) \le \theta < \frac{\pi}{2}\lambda \\ 0, & \text{otherwise} \end{cases},$$

for quadrant λ $(=1,2,3,4)$. The corresponding wavelet spaces are defined as $\mathbf{W}_0 = \operatorname{span}\{w_{2,\lambda}\}$, where

$$w_{r,k}(\theta) = \begin{cases} 2^{r/2}, & 2^{1-r}\pi(k-1) \le \theta < 2^{1-r}\pi\left(k-\frac{1}{2}\right) \\ -2^{r/2}, & 2^{1-r}\pi\left(k-\frac{1}{2}\right) \le \theta < 2^{1-r}\pi k \\ 0, & \text{otherwise} \end{cases},$$

is the "root wavelet" and $\mathbf{W}_r := \operatorname{span}\{w_{r+2,k} : 1 \le k \le 2^{r+2}\}$, $r \ge 1$. The corresponding angular flux representation is

$$\psi(x,y,\mu,\theta) = \sum_{\lambda=1}^{4}\left[\psi^{s}_{2,\lambda}(x,y,\mu) f_{2,\lambda}(\theta)\right] + \sum_{r=2}^{N+1}\sum_{k=1}^{2^r}\left[\psi^{w}_{r,k}(x,y,\mu) w_{r,k}(\theta)\right]. \tag{6.1}$$

Consider the steady-state, monoenergetic transport equation

$$\vec{\Omega}\cdot\nabla\psi(\vec{r},\vec{\Omega}) + \sigma(\vec{r})\psi(\vec{r},\vec{\Omega}) = \int_{4\pi} d\Omega'\,\sigma_s(\vec{r},\vec{\Omega}'\cdot\vec{\Omega})\psi(\vec{r},\vec{\Omega}') + S_{ext}(\vec{r},\vec{\Omega}) \tag{6.2}$$

In two-dimensional Cartesian geometry, with isotropic scattering and sources, this becomes

$$\Omega_x \frac{\partial \psi}{\partial x} + \Omega_y \frac{\partial \psi}{\partial y} + \sigma(x,y)\psi(x,y,\vec{\Omega}) = \frac{\sigma_s(x,y)}{4\pi}\int_{4\pi} d\Omega'\,\psi(x,y,\vec{\Omega}') + \frac{S_{ext}(x,y)}{4\pi}. \quad (6.3)$$

We will now split the direction vector $\vec{\Omega}$ into polar (ξ) and azimuthal (θ) components

$$\begin{aligned}\Omega_x &= \sin\xi\cos\theta \\ \Omega_y &= \sin\xi\sin\theta \\ \Omega_z &= \mu := \cos\xi\end{aligned}$$

Thus, our transport equation becomes

$$\begin{aligned}&\sqrt{1-\mu^2}\cos\theta\frac{\partial \psi}{\partial x} + \sqrt{1-\mu^2}\sin\theta\frac{\partial \psi}{\partial y} + \sigma(x,y)\psi(x,y,\mu,\theta) \\ &\qquad = \frac{\sigma_s(x,y)}{4\pi}\int_{-1}^{1} d\mu' \int_{2\pi} d\theta'\,\psi(x,y,\mu',\theta') + \frac{S_{ext}(x,y)}{4\pi}\end{aligned} \quad (6.4)$$

Next, we will consider some discrete polar cosine (μ_m) and replace the integral over polar cosine with a quadrature sum. Also, we will define $\psi_m(x,y,\theta) := \psi(x,y,\mu_m,\theta)$. Thus (6.4) becomes

$$\begin{aligned}&\sqrt{1-\mu_m^2}\cos\theta\frac{\partial \psi_m}{\partial x} + \sqrt{1-\mu_m^2}\sin\theta\frac{\partial \psi_m}{\partial y} + \sigma(x,y)\psi_m(x,y,\theta) \\ &\qquad = \frac{\sigma_s(x,y)}{4\pi}\int_{2\pi} d\theta' \sum_n \omega_n\psi_n(x,y,\theta') + \frac{S_{ext}(x,y)}{4\pi}.\end{aligned} \quad (6.5)$$

Finally, we will define the scalar flux as $\phi(x,y) := \int_{2\pi} d\theta \sum_n \omega_n\psi_n(x,y,\theta)$ and divide through by $\sqrt{1-\mu_m^2}$. Then, our transport equation is

$$\cos\theta\frac{\partial\psi_m}{\partial x}+\sin\theta\frac{\partial\psi_m}{\partial y}+\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi_m(x,y,\theta)=\frac{\sigma_s(x,y)}{4\pi\sqrt{1-\mu_m^2}}\phi(x,y)+\frac{S_{ext}(x,y)}{4\pi\sqrt{1-\mu_m^2}}. \quad (6.6)$$

We will now substitute the expansion (6.1) into (6.6)

$$\begin{aligned}&\sum_{\lambda=1}^{4}\left[\frac{\partial\psi^s_{2,\lambda,m}}{\partial x}f_{2,\lambda}(\theta)\cos\theta+\frac{\partial\psi^s_{2,\lambda,m}}{\partial y}f_{2,\lambda}(\theta)\sin\theta+\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi^s_{2,\lambda,m}f_{2,\lambda}(\theta)\right]\\&+\sum_{r=2}^{N+1}\sum_{k=1}^{2^r}\left[\frac{\partial\psi^w_{r,k,m}}{\partial x}w_{r,k}(\theta)\cos\theta+\frac{\partial\psi^w_{r,k,m}}{\partial y}w_{r,k}(\theta)\sin\theta+\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi^w_{r,k,m}w_{r,k}(\theta)\right]\\&=\frac{\sigma_s(x,y)}{4\pi\sqrt{1-\mu_m^2}}\phi(x,y)+\frac{S_{ext}(x,y)}{4\pi\sqrt{1-\mu_m^2}}.\end{aligned} \quad (6.7)$$

In order to obtain an equation for the scaling function coefficients, $\psi^s_{2,\lambda,m}$, we will multiply (6.7) by a scaling function $f_{2,p}(\theta)$

$$\begin{aligned}&\sum_{\lambda=1}^{4}\left[\frac{\partial\psi^s_{2,\lambda,m}}{\partial x}f_{2,\lambda}(\theta)f_{2,p}(\theta)\cos\theta+\frac{\partial\psi^s_{2,\lambda,m}}{\partial y}f_{2,\lambda}(\theta)f_{2,p}(\theta)\sin\theta\right.\\&\left.+\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi^s_{2,\lambda,m}f_{2,\lambda}(\theta)f_{2,p}(\theta)\right]+\sum_{r=2}^{N+1}\sum_{k=1}^{2^r}\left[\frac{\partial\psi^w_{r,k,m}}{\partial x}w_{r,k}(\theta)f_{2,p}(\theta)\cos\theta\right.\\&\left.+\frac{\partial\psi^w_{r,k,m}}{\partial y}w_{r,k}(\theta)f_{2,p}(\theta)\sin\theta+\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi^w_{r,k,m}w_{r,k}(\theta)f_{2,p}(\theta)\right]\\&=\frac{\sigma_s(x,y)}{4\pi\sqrt{1-\mu_m^2}}\phi(x,y)f_{2,p}(\theta)+\frac{S_{ext}(x,y)}{4\pi\sqrt{1-\mu_m^2}}f_{2,p}(\theta).\end{aligned} \quad (6.8)$$

Because the scaling functions are orthogonal, this becomes

$$\begin{aligned}
&\sum_{\lambda=1}^{4}\left[\frac{\partial\psi^{s}_{2,\lambda,m}}{\partial x}f^{2}_{2,\lambda}(\theta)\cos\theta+\frac{\partial\psi^{s}_{2,\lambda,m}}{\partial y}f^{2}_{2,\lambda}(\theta)\sin\theta+\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi^{s}_{2,\lambda,m}f^{2}_{2,\lambda}(\theta)\right]\\
&\quad+\sum_{r=2}^{N+1}\sum_{k=1}^{2^r}\left[\frac{\partial\psi^{w}_{r,k,m}}{\partial x}w_{r,k}(\theta)f_{2,p}(\theta)\cos\theta+\frac{\partial\psi^{w}_{r,k,m}}{\partial y}w_{r,k}(\theta)f_{2,p}(\theta)\sin\theta\right.\\
&\qquad\left.+\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi^{w}_{r,k,m}w_{r,k}(\theta)f_{2,p}(\theta)\right]=\frac{\sigma_s(x,y)}{4\pi\sqrt{1-\mu_m^2}}\phi(x,y)f_{2,p}(\theta)\\
&\qquad\qquad+\frac{S_{ext}(x,y)}{4\pi\sqrt{1-\mu_m^2}}f_{2,p}(\theta).
\end{aligned} \tag{6.9}$$

Next, we will integrate over the interval $\theta\in[0,2\pi]$, which yields

$$\begin{aligned}
&\sum_{\lambda=1}^{4}\left[\tilde{\mu}_{2,\lambda}\frac{\partial\psi^{s}_{2,\lambda,m}}{\partial x}+\tilde{\eta}_{2,\lambda}\frac{\partial\psi^{s}_{2,\lambda,m}}{\partial y}+\tilde{\chi}_{2,\lambda}\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi^{s}_{2,\lambda,m}\right]\\
&\quad+\sum_{r=2}^{N+1}\sum_{k=1}^{2^r}\left[\tilde{\alpha}_{r,k,p}\frac{\partial\psi^{w}_{r,k,m}}{\partial x}+\tilde{\beta}_{r,k,p}\frac{\partial\psi^{w}_{r,k,m}}{\partial y}+\tilde{\delta}_{r,k,p}\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi^{w}_{r,k,m}\right]\\
&\qquad=\tilde{\varepsilon}_{2,p}\frac{\sigma_s(x,y)}{4\pi\sqrt{1-\mu_m^2}}\phi(x,y)+\tilde{\varepsilon}_{2,p}\frac{S_{ext}(x,y)}{4\pi\sqrt{1-\mu_m^2}}.
\end{aligned} \tag{6.10}$$

where

$$\tilde{\mu}_{2,\lambda}=\int_0^{2\pi}d\theta\ f^{2}_{2,\lambda}(\theta)\cos\theta=2\int_{\frac{\pi}{2}(\lambda-1)}^{\frac{\pi}{2}\lambda}d\theta\ \cos\theta=2\left[\sin\left(\tfrac{\pi}{2}\lambda\right)-\sin\left(\tfrac{\pi}{2}(\lambda-1)\right)\right]$$

$$\tilde{\eta}_{2,\lambda}=\int_0^{2\pi}d\theta\ f^{2}_{2,\lambda}(\theta)\sin\theta=2\int_{\frac{\pi}{2}(\lambda-1)}^{\frac{\pi}{2}\lambda}d\theta\ \sin\theta=2\left[\cos\left(\tfrac{\pi}{2}(\lambda-1)\right)-\cos\left(\tfrac{\pi}{2}\lambda\right)\right]$$

$$\tilde{\chi}_{2,\lambda}=\int_0^{2\pi}d\theta\ f^{2}_{2,\lambda}(\theta)=4\int_{\frac{\pi}{2}(\lambda-1)}^{\frac{\pi}{2}\lambda}d\theta\ =2\pi$$

$$\begin{aligned}
\tilde{\alpha}_{r,k,p} &= \int_0^{2\pi} d\theta\, w_{r,k}(\theta)\, f_{2,p}(\theta)\cos\theta \\
&= 2^{r/2+1} \int_{2^{1-r}\pi(k-1)}^{2^{1-r}\pi\left(k-\frac{1}{2}\right)} d\theta\ \cos\theta - 2^{r/2+1} \int_{2^{1-r}\pi\left(k-\frac{1}{2}\right)}^{2^{1-r}\pi k} d\theta\ \cos\theta \\
&= 2^{r/2+1}\left[-\sin\left(2^{1-r}\pi(k-1)\right) + 2\sin\left(2^{1-r}\pi\left(k-\tfrac{1}{2}\right)\right) - \sin\left(2^{1-r}\pi k\right)\right]
\end{aligned}$$

$$\begin{aligned}
\tilde{\beta}_{r,k,p} &= \int_0^{2\pi} d\theta\, w_{r,k}(\theta)\, f_{2,p}(\theta)\sin\theta \\
&= 2^{r/2+1} \int_{2^{1-r}\pi(k-1)}^{2^{1-r}\pi\left(k-\frac{1}{2}\right)} d\theta\ \sin\theta - 2^{r/2+1} \int_{2^{1-r}\pi\left(k-\frac{1}{2}\right)}^{2^{1-r}\pi k} d\theta\ \sin\theta \\
&= 2^{r/2+1}\left[\cos\left(2^{1-r}\pi(k-1)\right) - 2\cos\left(2^{1-r}\pi\left(k-\tfrac{1}{2}\right)\right) + \cos\left(2^{1-r}\pi k\right)\right]
\end{aligned}$$

$$\tilde{\delta}_{r,k,p} = \int_0^{2\pi} d\theta\, w_{r,k}(\theta)\, f_{2,p}(\theta) = 2^{r/2+1} \int_{2^{1-r}\pi(k-1)}^{2^{1-r}\pi\left(k-\frac{1}{2}\right)} d\theta\ - 2^{r/2+1} \int_{2^{1-r}\pi\left(k-\frac{1}{2}\right)}^{2^{1-r}\pi k} d\theta\ = 0$$

$$\tilde{\varepsilon}_{2,p} = \int_0^{2\pi} d\theta\ f_{2,p}(\theta) = 2\int_{\frac{\pi}{2}(\lambda-1)}^{\frac{\pi}{2}\lambda} d\theta\ = \pi$$

Thus, we have

$$\begin{aligned}
&\sum_{\lambda=1}^{4}\left[\tilde{\mu}_{2,\lambda}\frac{\partial\psi^{s}_{2,\lambda,m}}{\partial x} + \tilde{\eta}_{2,\lambda}\frac{\partial\psi^{s}_{2,\lambda,m}}{\partial y} + \frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi^{s}_{2,\lambda,m}\right] \\
&\qquad + \sum_{r=2}^{N+1}\sum_{k=1}^{2^r}\left[\tilde{\alpha}_{r,k,p}\frac{\partial\psi^{w}_{r,k,m}}{\partial x} + \tilde{\beta}_{r,k,p}\frac{\partial\psi^{w}_{r,k,m}}{\partial y}\right] \\
&\qquad\qquad = \frac{\sigma_s(x,y)}{4\sqrt{1-\mu_m^2}}\phi(x,y) + \frac{S_{ext}(x,y)}{4\sqrt{1-\mu_m^2}}.
\end{aligned} \tag{6.11}$$

Dividing through by 2π, we find

$$\sum_{\lambda=1}^{4}\left[\mu_{2,\lambda}\frac{\partial\psi^{s}_{2,\lambda,m}}{\partial x}+\eta_{2,\lambda}\frac{\partial\psi^{s}_{2,\lambda,m}}{\partial y}+\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi^{s}_{2,\lambda,m}\right] + \sum_{r=2}^{N+1}\sum_{k=1}^{2^r}\left[\alpha_{r,k,p}\frac{\partial\psi^{w}_{r,k,m}}{\partial x}+\beta_{r,k,p}\frac{\partial\psi^{w}_{r,k,m}}{\partial y}\right] = \frac{\sigma_s(x,y)}{8\pi\sqrt{1-\mu_m^2}}\phi(x,y)+\frac{S_{ext}(x,y)}{8\pi\sqrt{1-\mu_m^2}}. \tag{6.12}$$

where

$$\mu_{2,\lambda}=\frac{\tilde{\mu}_{2,\lambda}}{2\pi},\quad \eta_{2,\lambda}=\frac{\tilde{\eta}_{2,\lambda}}{2\pi},\quad \alpha_{r,k,p}=\frac{\tilde{\alpha}_{r,k,p}}{2\pi},\quad \text{and}\quad \beta_{r,k,p}=\frac{\tilde{\beta}_{r,k,p}}{2\pi}.$$

Since the coarsest resolution level consists of one scaling function in each quadrant, these equations can be decoupled by quadrant. Thus, we have finally, for quadrant λ,

$$\left[\mu_{2,\lambda}\frac{\partial\psi^{s}_{2,\lambda,m}}{\partial x}+\eta_{2,\lambda}\frac{\partial\psi^{s}_{2,\lambda,m}}{\partial y}+\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi^{s}_{2,\lambda,m}\right] + \sum_{r=2}^{N+1}\sum_{k=2^{r-2}(\lambda-1)+1}^{2^{r-2}\lambda}\left[\alpha_{r,k,\lambda}\frac{\partial\psi^{w}_{r,k,m}}{\partial x}+\beta_{r,k,\lambda}\frac{\partial\psi^{w}_{r,k,m}}{\partial y}\right] = \frac{\sigma_s(x,y)}{8\pi\sqrt{1-\mu_m^2}}\phi(x,y)+\frac{S_{ext}(x,y)}{8\pi\sqrt{1-\mu_m^2}},\quad m\geq 1 \tag{6.13}$$

Next, in order to obtain an equation for the wavelet function coefficients, $\psi^{w}_{r,k,m}$, we will multiply (6.7) by a wavelet function $w_{n,p}(\theta)$. There results

$$\sum_{\lambda=1}^{4}\Bigg[\frac{\partial\psi^{s}_{2,\lambda,m}}{\partial x}f_{2,\lambda}(\theta)w_{n,p}(\theta)\cos\theta+\frac{\partial\psi^{s}_{2,\lambda,m}}{\partial y}f_{2,\lambda}(\theta)w_{n,p}(\theta)\sin\theta$$
$$+\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi^{s}_{2,\lambda,m}f_{2,\lambda}(\theta)w_{n,p}(\theta)\Bigg]+\sum_{r=2}^{N+1}\sum_{k=1}^{2^r}\Bigg[\frac{\partial\psi^{w}_{r,k,m}}{\partial x}w_{r,k}(\theta)w_{n,p}(\theta)\cos\theta$$
$$+\frac{\partial\psi^{w}_{r,k,m}}{\partial y}w_{r,k}(\theta)w_{n,p}(\theta)\sin\theta+\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi^{w}_{r,k,m}w_{r,k}(\theta)w_{n,p}(\theta)\Bigg] \tag{6.14}$$
$$=\frac{\sigma_s(x,y)}{4\pi\sqrt{1-\mu_m^2}}\phi(x,y)w_{n,p}(\theta)+\frac{S_{ext}(x,y)}{4\pi\sqrt{1-\mu_m^2}}w_{n,p}(\theta).$$

If, we integrate this over the interval $\theta\in[0,2\pi]$, we find

$$\sum_{\lambda=1}^{4}\Bigg[\tilde{\alpha}_{n,p,\lambda}\frac{\partial\psi^{s}_{2,\lambda,m}}{\partial x}+\tilde{\beta}_{n,p,\lambda}\frac{\partial\psi^{s}_{2,\lambda,m}}{\partial y}+\tilde{\chi}_{n,p,\lambda}\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi^{s}_{2,\lambda,m}\Bigg]$$
$$+\sum_{r=2}^{N+1}\sum_{k=1}^{2^r}\Bigg[\tilde{A}^{n,p}_{r,k}\frac{\partial\psi^{w}_{r,k,m}}{\partial x}+\tilde{B}^{n,p}_{r,k}\frac{\partial\psi^{w}_{r,k,m}}{\partial y}+\tilde{C}^{n,p}_{r,k}\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi^{w}_{r,k,m}\Bigg] \tag{6.15}$$
$$=\tilde{\delta}_{n,p}\frac{\sigma_s(x,y)}{4\pi\sqrt{1-\mu_m^2}}\phi(x,y)+\tilde{\delta}_{n,p}\frac{S_{ext}(x,y)}{4\pi\sqrt{1-\mu_m^2}}.$$

where

$$\tilde{\alpha}_{n,p,\lambda}=\int_0^{2\pi}d\theta\ f_{2,\lambda}(\theta)w_{n,p}(\theta)\cos\theta$$
$$=2^{n/2+1}\int_{2^{1-n}\pi(p-1)}^{2^{1-n}\pi\left(p-\frac{1}{2}\right)}d\theta\ \cos\theta-2^{n/2+1}\int_{2^{1-n}\pi\left(p-\frac{1}{2}\right)}^{2^{1-n}\pi p}d\theta\ \cos\theta$$
$$=2^{n/2+1}\left[-\sin\left(2^{1-n}\pi(p-1)\right)+2\sin\left(2^{1-n}\pi\left(p-\tfrac{1}{2}\right)\right)-\sin\left(2^{1-n}\pi p\right)\right]$$

$$\tilde{\beta}_{n,p,\lambda}=\int_0^{2\pi}d\theta\ f_{2,\lambda}(\theta)w_{n,p}(\theta)\sin\theta$$
$$=2^{n/2+1}\int_{2^{1-n}\pi(p-1)}^{2^{1-n}\pi\left(p-\frac{1}{2}\right)}d\theta\ \sin\theta-2^{n/2+1}\int_{2^{1-n}\pi\left(p-\frac{1}{2}\right)}^{2^{1-n}\pi p}d\theta\ \sin\theta$$
$$=2^{n/2+1}\left[\cos\left(2^{1-n}\pi(p-1)\right)-2\cos\left(2^{1-n}\pi\left(p-\tfrac{1}{2}\right)\right)+\cos\left(2^{1-n}\pi p\right)\right]$$

$$\tilde{A}_{r,k}^{n,p} = \int_0^{2\pi} d\theta \, w_{r,k}(\theta) w_{n,p}(\theta) \cos\theta$$

$$= \left\{ \begin{array}{l} 2^{n/2}\left[2^{r/2} \int_{2^{1-r}\pi(k-1)}^{2^{1-r}\pi(k-\frac{1}{2})} d\theta \, \cos\theta - 2^{r/2} \int_{2^{1-r}\pi(k-\frac{1}{2})}^{2^{1-r}\pi k} d\theta \, \cos\theta \right], \\ \qquad r > n, k \in \left[2^{r-n}(p-1)+1, 2^{r-n}(p-\tfrac{1}{2}) \right] \\ -2^{n/2}\left[2^{r/2} \int_{2^{1-r}\pi(k-1)}^{2^{1-r}\pi(k-\frac{1}{2})} d\theta \, \cos\theta - 2^{r/2} \int_{2^{1-r}\pi(k-\frac{1}{2})}^{2^{1-r}\pi k} d\theta \, \cos\theta \right], \\ \qquad r > n, k \in \left[2^{r-n}(p-\tfrac{1}{2})+1, 2^{r-n} p \right] \\ 2^{n/2}\left[2^{r/2} \int_{2^{1-r}\pi(k-1)}^{2^{1-r}\pi(k-\frac{1}{2})} d\theta \, \cos\theta + 2^{r/2} \int_{2^{1-r}\pi(k-\frac{1}{2})}^{2^{1-r}\pi k} d\theta \, \cos\theta \right], \quad r = n, k = p \\ 2^{r/2}\left[2^{n/2} \int_{2^{1-n}\pi(p-1)}^{2^{1-n}\pi(p-\frac{1}{2})} d\theta \, \cos\theta - 2^{n/2} \int_{2^{1-n}\pi(p-\frac{1}{2})}^{2^{1-n}\pi p} d\theta \, \cos\theta \right], \\ \qquad r < n, \ p \in \left[2^{n-r}(k-1)+1, 2^{n-r}(k-\tfrac{1}{2}) \right] \\ -2^{r/2}\left[2^{n/2} \int_{2^{1-n}\pi(p-1)}^{2^{1-n}\pi(p-\frac{1}{2})} d\theta \, \cos\theta - 2^{n/2} \int_{2^{1-n}\pi(p-\frac{1}{2})}^{2^{1-n}\pi p} d\theta \, \cos\theta \right], \\ \qquad r < n, \ p \in \left[2^{r-n}(k-\tfrac{1}{2})+1, 2^{r-n} k \right] \end{array} \right\}$$

$$= \left\{ \begin{array}{l} 2^{n/2+r/2}\left[-\sin\left(2^{1-r}\pi(k-1)\right) + 2\sin\left(2^{1-r}\pi(k-\tfrac{1}{2})\right) - \sin\left(2^{1-r}\pi k\right) \right], \\ \qquad r > n, k \in \left[2^{r-n}(p-1)+1, 2^{r-n}(p-\tfrac{1}{2}) \right] \\ -2^{n/2+r/2}\left[-\sin\left(2^{1-r}\pi(k-1)\right) + 2\sin\left(2^{1-r}\pi(k-\tfrac{1}{2})\right) - \sin\left(2^{1-r}\pi k\right) \right], \\ \qquad r > n, k \in \left[2^{r-n}(p-\tfrac{1}{2})+1, 2^{r-n} p \right] \\ 2^{n}\left[-\sin\left(2^{1-n}\pi(p-1)\right) + \sin\left(2^{1-n}\pi p\right) \right], \quad r = n, k = p \\ 2^{n/2+r/2}\left[-\sin\left(2^{1-n}\pi(p-1)\right) + 2\sin\left(2^{1-n}\pi(p-\tfrac{1}{2})\right) - \sin\left(2^{1-n}\pi p\right) \right], \\ \qquad r < n, \ p \in \left[2^{n-r}(k-1)+1, 2^{n-r}(k-\tfrac{1}{2}) \right] \\ -2^{n/2+r/2}\left[-\sin\left(2^{1-n}\pi(p-1)\right) + 2\sin\left(2^{1-n}\pi(p-\tfrac{1}{2})\right) - \sin\left(2^{1-n}\pi p\right) \right], \\ \qquad r < n, \ p \in \left[2^{r-n}(k-\tfrac{1}{2})+1, 2^{r-n} k \right] \end{array} \right.$$

$$\tilde{B}_{r,k}^{n,p} = \int_0^{2\pi} d\theta\, w_{r,k}(\theta)\, w_{n,p}(\theta) \sin\theta$$

$$= \begin{cases} 2^{n/2}\left[2^{r/2}\int_{2^{1-r}\pi(k-1)}^{2^{1-r}\pi\left(k-\frac{1}{2}\right)} d\theta\,\sin\theta - 2^{r/2}\int_{2^{1-r}\pi\left(k-\frac{1}{2}\right)}^{2^{1-r}\pi k} d\theta\,\sin\theta\right], \\ \qquad r>n,\, k\in\left[2^{r-n}(p-1)+1,\, 2^{r-n}\left(p-\tfrac{1}{2}\right)\right] \\ -2^{n/2}\left[2^{r/2}\int_{2^{1-r}\pi(k-1)}^{2^{1-r}\pi\left(k-\frac{1}{2}\right)} d\theta\,\sin\theta - 2^{r/2}\int_{2^{1-r}\pi\left(k-\frac{1}{2}\right)}^{2^{1-r}\pi k} d\theta\,\sin\theta\right], \\ \qquad r>n,\, k\in\left[2^{r-n}\left(p-\tfrac{1}{2}\right)+1,\, 2^{r-n}p\right] \\ 2^{n/2}\left[2^{r/2}\int_{2^{1-r}\pi(k-1)}^{2^{1-r}\pi\left(k-\frac{1}{2}\right)} d\theta\,\sin\theta + 2^{r/2}\int_{2^{1-r}\pi\left(k-\frac{1}{2}\right)}^{2^{1-r}\pi k} d\theta\,\sin\theta\right], \quad r=n,\, k=p \\ 2^{r/2}\left[2^{n/2}\int_{2^{1-n}\pi(p-1)}^{2^{1-n}\pi\left(p-\frac{1}{2}\right)} d\theta\,\sin\theta - 2^{n/2}\int_{2^{1-n}\pi\left(p-\frac{1}{2}\right)}^{2^{1-n}\pi p} d\theta\,\sin\theta\right], \\ \qquad r<n,\, p\in\left[2^{n-r}(k-1)+1,\, 2^{n-r}\left(k-\tfrac{1}{2}\right)\right] \\ -2^{r/2}\left[2^{n/2}\int_{2^{1-n}\pi(p-1)}^{2^{1-n}\pi\left(p-\frac{1}{2}\right)} d\theta\,\sin\theta - 2^{n/2}\int_{2^{1-n}\pi\left(p-\frac{1}{2}\right)}^{2^{1-n}\pi p} d\theta\,\sin\theta\right], \\ \qquad r<n,\, p\in\left[2^{r-n}\left(k-\tfrac{1}{2}\right)+1,\, 2^{r-n}k\right] \end{cases}$$

$$= \begin{cases} 2^{n/2+r/2}\left[\cos\left(2^{1-r}\pi(k-1)\right) - 2\cos\left(2^{1-r}\pi\left(k-\tfrac{1}{2}\right)\right) + \cos\left(2^{1-r}\pi k\right)\right], \\ \qquad r>n,\, k\in\left[2^{r-n}(p-1)+1,\, 2^{r-n}\left(p-\tfrac{1}{2}\right)\right] \\ -2^{n/2+r/2}\left[\cos\left(2^{1-r}\pi(k-1)\right) - 2\cos\left(2^{1-r}\pi\left(k-\tfrac{1}{2}\right)\right) + \cos\left(2^{1-r}\pi k\right)\right], \\ \qquad r>n,\, k\in\left[2^{r-n}\left(p-\tfrac{1}{2}\right)+1,\, 2^{r-n}p\right] \\ 2^{n}\left[\cos\left(2^{1-n}\pi(p-1)\right) - \cos\left(2^{1-n}\pi p\right)\right], \quad r=n,\, k=p \\ 2^{n/2+r/2}\left[\cos\left(2^{1-n}\pi(p-1)\right) - 2\cos\left(2^{1-n}\pi\left(p-\tfrac{1}{2}\right)\right) + \cos\left(2^{1-n}\pi p\right)\right], \\ \qquad r<n,\, p\in\left[2^{n-r}(k-1)+1,\, 2^{n-r}\left(k-\tfrac{1}{2}\right)\right] \\ -2^{n/2+r/2}\left[\cos\left(2^{1-n}\pi(p-1)\right) - 2\cos\left(2^{1-n}\pi\left(p-\tfrac{1}{2}\right)\right) + \cos\left(2^{1-n}\pi p\right)\right], \\ \qquad r<n,\, p\in\left[2^{r-n}\left(k-\tfrac{1}{2}\right)+1,\, 2^{r-n}k\right] \end{cases}$$

$$\tilde{C}_{r,k}^{n,p} = \int_0^{2\pi} d\theta\, w_{r,k}(\theta)\, w_{n,p}(\theta) = \begin{cases} 2\pi, & r = n, k = p \\ 0, & \text{otherwise} \end{cases}$$

$$\tilde{\chi}_{n,p,\lambda} = \int_0^{2\pi} d\theta\, f_{2,\lambda}(\theta)\, w_{n,p}(\theta) = 2^{n/2+1} \int_{2^{1-n}\pi(p-1)}^{2^{1-n}\pi\left(p-\frac{1}{2}\right)} d\theta \; - 2^{n/2+1} \int_{2^{1-n}\pi\left(p-\frac{1}{2}\right)}^{2^{1-n}\pi p} d\theta \; = 0$$

$$\tilde{\delta}_{n,p} = \int_0^{2\pi} d\theta\, w_{n,p}(\theta) = 2^{n/2} \int_{2^{1-n}\pi(p-1)}^{2^{1-n}\pi\left(p-\frac{1}{2}\right)} d\theta \; - 2^{n/2} \int_{2^{1-n}\pi\left(p-\frac{1}{2}\right)}^{2^{1-n}\pi p} d\theta \; = 0$$

We will split equation (6.15) so that the unknowns $\psi_{n,p,m}^{w}$ for the equation *n*, *p* are out of the summation. So, we have

$$\sum_{\lambda=1}^{4}\left[\tilde{\alpha}_{n,p,\lambda}\frac{\partial \psi_{2,\lambda,m}^{s}}{\partial x} + \tilde{\beta}_{n,p,\lambda}\frac{\partial \psi_{2,\lambda,m}^{s}}{\partial y}\right] + \sum_{\substack{r=2 \\ r\neq n}}^{N+1}\sum_{k=1}^{2^r}\left[\tilde{A}_{r,k}^{n,p}\frac{\partial \psi_{r,k,m}^{w}}{\partial x} + \tilde{B}_{r,k}^{n,p}\frac{\partial \psi_{r,k,m}^{w}}{\partial y}\right]$$
$$+\sum_{p=1}^{2^n}\left[\tilde{A}_{n,p}^{n,p}\frac{\partial \psi_{n,p,m}^{w}}{\partial x} + \tilde{B}_{n,p}^{n,p}\frac{\partial \psi_{n,p,m}^{w}}{\partial y} + \tilde{C}_{n,p}^{n,p}\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi_{n,p,m}^{w}\right] = 0. \tag{6.16}$$

Dividing through by 2π, we find

$$\sum_{\lambda=1}^{4}\left[\alpha_{n,p,\lambda}\frac{\partial \psi_{2,\lambda,m}^{s}}{\partial x} + \beta_{n,p,\lambda}\frac{\partial \psi_{2,\lambda,m}^{s}}{\partial y}\right] + \sum_{\substack{r=2 \\ r\neq n}}^{N+1}\sum_{k=1}^{2^r}\left[A_{r,k}^{n,p}\frac{\partial \psi_{r,k,m}^{w}}{\partial x} + B_{r,k}^{n,p}\frac{\partial \psi_{r,k,m}^{w}}{\partial y}\right]$$
$$+\sum_{p=1}^{2^n}\left[\mu_{n,p}\frac{\partial \psi_{n,p,m}^{w}}{\partial x} + \eta_{n,p}\frac{\partial \psi_{n,p,m}^{w}}{\partial y} + \frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi_{n,p,m}^{w}\right] = 0. \tag{6.17}$$

where

$$\alpha_{n,p,\lambda} = \frac{\tilde{\alpha}_{n,p,\lambda}}{2\pi}, \qquad \beta_{n,p,\lambda} = \frac{\tilde{\beta}_{n,p,\lambda}}{2\pi},$$
$$A_{r,k}^{n,p} = \frac{\tilde{A}_{r,k}^{n,p}}{2\pi}, \qquad B_{r,k}^{n,p} = \frac{\tilde{B}_{r,k}^{n,p}}{2\pi}$$
$$\mu_{n,p} = \frac{\tilde{A}_{n,p}^{n,p}}{2\pi}, \qquad \eta_{n,p} = \frac{\tilde{B}_{n,p}^{n,p}}{2\pi}$$

Decoupling the quadrants, and switching the indices *n*,*p* with the indices *r*,*k*, we find, for wavelet *r*,*k* in quadrant λ,

$$\left[\alpha_{r,k,\lambda}\frac{\partial\psi^{s}_{2,\lambda,m}}{\partial x}+\beta_{r,k,\lambda}\frac{\partial\psi^{s}_{2,\lambda,m}}{\partial y}\right]+\sum_{\substack{n=2\\ n\neq r}}^{N+1}\sum_{p=2^{n-2}(\lambda-1)+1}^{2^{n-2}\lambda}\left[A^{r,k}_{n,p}\frac{\partial\psi^{w}_{n,p,m}}{\partial x}+B^{r,k}_{n,p}\frac{\partial\psi^{w}_{n,p,m}}{\partial y}\right]$$
$$+\left[\mu_{r,k}\frac{\partial\psi^{w}_{r,k,m}}{\partial x}+\eta_{r,k}\frac{\partial\psi^{w}_{r,k,m}}{\partial y}+\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi^{w}_{r,k,m}\right]=0, \qquad (6.18)$$
$$r\geq 2,\ k=2^{r-2}(\lambda-1)+1\ldots 2^{r-2}\lambda,\ m\geq 1$$

Next, we substitute the expansion (6.1) into the definition of the scalar flux and find

$$\phi(x,y)=\pi\sum_{\lambda=1}^{4}\sum_{n}\omega_n\psi^{s}_{2,\lambda,n}(x,y)$$

Thus, the scalar flux is defined by only the scaling coefficients. Now, we will address boundary conditions. In this dissertation, we will employ vacuum, reflective, and incident flux boundary conditions only. For vacuum boundaries, the condition is

$$\psi\left(\vec{r},\vec{\Omega}\right)=0, \qquad \vec{r}\in\partial D,\ \vec{\Omega}\cdot\vec{n}<0,$$

where $\vec{n}$ is the outward normal vector on the boundary of the domain *D*. Substituting the expansion (6.1), we find

$$\sum_{\lambda=1}^{4}\left[\psi^{s}_{2,\lambda,m}(x,y)f_{2,\lambda}(\theta)\right]+\sum_{r=2}^{N+1}\sum_{k=1}^{2^r}\left[\psi^{w}_{r,k,m}(x,y)w_{r,k}(\theta)\right]=0,$$
$$(x,y)\in\partial D,\ \vec{\Omega}_m\cdot\vec{n}<0$$

Since the scaling functions and wavelet functions form a basis for the space $\theta\in[0,2\pi]$, for this summation to be zero, all of the coefficients must be zero. Thus,

$$\psi^s_{2,\lambda,m}(x,y)=0,\ \psi^w_{r,k,m}(x,y)=0, \qquad (x,y)\in\partial D,\ \vec{\Omega}_m\cdot\vec{n}<0 \tag{6.19}$$

For reflective boundary conditions, we have

$$\psi(\vec{r},\vec{\Omega})=\psi(\vec{r},-\vec{\Omega}), \qquad \vec{r}\in\partial D,\ \vec{\Omega}\cdot\vec{n}<0.$$

Substituting the expansion (6.1), we find

$$\begin{aligned}\sum_{\lambda=1}^{4}\left[\psi^s_{2,\lambda,m}(x,y)f_{2,\lambda}(\theta)\right]+\sum_{r=2}^{N+1}\sum_{k=1}^{2^r}\left[\psi^w_{r,k,m}(x,y)w_{r,k}(\theta)\right]&\\ =\sum_{\lambda=1}^{4}\left[\psi^s_{2,\lambda',m'}(x,y)f_{2,\lambda'}(\theta)\right]+\sum_{r=2}^{N+1}\sum_{k=1}^{2^r}\left[\psi^w_{r,k',m'}(x,y)w_{r,k'}(\theta)\right],&\\ \vec{r}\in\partial D,\ \vec{\Omega}\cdot\vec{n}<0&\end{aligned}$$

One solution to this equation is

$$\begin{aligned}\psi^s_{2,\lambda,m}(x,y)=\psi^s_{2,\lambda',m'}(x,y),\ \psi^w_{r,k,m}(x,y)=\psi^w_{r,k',m'}(x,y)&\\ (x,y)\in\partial D,\quad m',\lambda':\vec{\Omega}'=-\vec{\Omega},\ r\geq 2,\ 1\leq\lambda\leq 2^r&\end{aligned} \tag{6.20}$$

where (x,y) is on the boundary of D, and m' and λ' are such that $\vec{\Omega}'=-\vec{\Omega}$.

Finally, for incident boundary conditions, we have

$$\psi(\vec{r},\vec{\Omega})=f(\vec{r},\vec{\Omega}), \qquad \vec{r}\in\partial D,\ \vec{\Omega}\cdot\vec{n}>0. \tag{6.21}$$

We will only consider isotropic incident fluxes in this dissertation, so

$$\psi(\vec{r},\vec{\Omega})=f(\vec{r}), \qquad \vec{r}\in\partial D,\ \vec{\Omega}\cdot\vec{n}>0.$$

Substituting the expansion (6.1), we find

$$\begin{aligned}&\sum_{\lambda=1}^{4}\left[\psi^s_{2,\lambda,m}(x,y)f_{2,\lambda}(\theta)\right]+\sum_{r=2}^{N+1}\sum_{k=1}^{2^r}\left[\psi^w_{r,k,m}(x,y)w_{r,k}(\theta)\right]=f(x,y),\\ &\qquad (x,y)\in\partial D,\ \vec{\Omega}_m\cdot\vec{n}>0\end{aligned}$$

Integrating over the angular interval $\theta\in[0,2\pi]$

$$\sum_{\lambda=1}^{4}\left[\psi^{s}_{2,\lambda,m}(x,y)\pi\right]=f(x,y)2\pi, \qquad (x,y)\in\partial D,\ \vec{\Omega}_m\cdot\vec{n}>0$$

Thus, we find

$$\psi_{r,\lambda,m}(x,y)=\tfrac{1}{2}f(x,y), \quad \psi^{w}_{r,k,m}(x,y)=0, \qquad (x,y)\in\partial D,\ \vec{\Omega}_m\cdot\vec{n}>0 \qquad (6.22)$$

So, in summary, the equation for the scaling coefficient, in quadrant λ, is

$$\left[\mu_{2,\lambda}\frac{\partial\psi^{s}_{2,\lambda,m}}{\partial x}+\eta_{2,\lambda}\frac{\partial\psi^{s}_{2,\lambda,m}}{\partial y}+\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi^{s}_{2,\lambda,m}\right]$$
$$+\sum_{r=2}^{N+1}\sum_{k=2^{r-2}(\lambda-1)+1}^{2^{r-2}\lambda}\left[\alpha_{r,k,\lambda}\frac{\partial\psi^{w}_{r,k,m}}{\partial x}+\beta_{r,k,\lambda}\frac{\partial\psi^{w}_{r,k,m}}{\partial y}\right]$$
$$=\frac{\sigma_s(x,y)}{8\pi\sqrt{1-\mu_m^2}}\phi(x,y)+\frac{S_{ext}(x,y)}{8\pi\sqrt{1-\mu_m^2}}, \quad N\geq 0,\ m\geq 1$$

and the equation for the wavelet coefficient r,k, in quadrant λ, is

$$\left[\alpha_{r,k,\lambda}\frac{\partial\psi^{s}_{2,\lambda,m}}{\partial x}+\beta_{r,k,\lambda}\frac{\partial\psi^{s}_{2,\lambda,m}}{\partial y}\right]+\sum_{\substack{n=2\\ n\neq r}}^{N+1}\sum_{p=2^{n-2}(\lambda-1)+1}^{2^{n-2}\lambda}\left[A^{r,k}_{n,p}\frac{\partial\psi^{w}_{n,p,m}}{\partial x}+B^{r,k}_{n,p}\frac{\partial\psi^{w}_{n,p,m}}{\partial y}\right]$$
$$+\left[\mu_{r,k}\frac{\partial\psi^{w}_{r,k,m}}{\partial x}+\eta_{r,k}\frac{\partial\psi^{w}_{r,k,m}}{\partial y}+\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi^{w}_{r,k,m}\right]=0,$$
$$r\geq 2,\ k=2^{r-2}(\lambda-1)+1\ldots 2^{r-2}\lambda,\ m\geq 1$$

with the constants defined above. The scalar flux is defined as

$$\phi(x,y)=\pi\sum_{\lambda=1}^{4}\sum_{n}\omega_n\psi^{s}_{2,\lambda,n}(x,y)$$

and the system includes the boundary conditions

vaccum:	$\psi^{s}_{2,\lambda,m}(x,y)=0, \quad \psi^{w}_{r,k,m}(x,y)=0$
reflective:	$\psi^{s}_{2,\lambda,m}(x,y)=\psi^{s}_{2,\lambda',m'}(x,y), \quad \psi^{w}_{r,k,m}(x,y)=\psi^{w}_{r,k',m'}(x,y)$
incident:	$\psi^{s}_{2,\lambda,m}(x,y)=\tfrac{1}{2}f(x,y), \quad \psi^{w}_{r,k,m}(x,y)=0.$

Two solution methods have been devised to solve these equations. The first involves solution of the system via a modified source iteration in which the off-diagonal terms are lagged. This will be referred to as the S_N-W_N method. Because the four scaling functions each have support in a single quadrant, the usual ray-tracing strategy can be employed to update the coefficients. The equations including iteration indices are

$$\begin{aligned}\left[\mu_{2,\lambda}\frac{\partial\psi_{2,\lambda,m}^{s(\ell+½)}}{\partial x}+\eta_{2,\lambda}\frac{\partial\psi_{2,\lambda,m}^{s(\ell+½)}}{\partial y}+\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi_{2,\lambda,m}^{s(\ell+½)}\right]&\\ =\frac{\sigma_s(x,y)}{8\pi\sqrt{1-\mu_m^2}}\phi^{(\ell)}(x,y)+\frac{S_{ext}(x,y)}{8\pi\sqrt{1-\mu_m^2}}&\\ -\sum_{r=2}^{N+1}\sum_{k=2^{r-2}(\lambda-1)+1}^{2^{r-2}\lambda}\left[\alpha_{r,k,\lambda}\frac{\partial\psi_{r,k,m}^{w(\ell)}}{\partial x}+\beta_{r,k,\lambda}\frac{\partial\psi_{r,k,m}^{w(\ell)}}{\partial y}\right],\quad N\geq 0,\ m\geq 1&\end{aligned}$$

for the scaling function in quadrant λ and

$$\begin{aligned}\left[\mu_{r,k}\frac{\partial\psi_{r,k,m}^{w(\ell+1)}}{\partial x}+\eta_{r,k}\frac{\partial\psi_{r,k,m}^{w(\ell+1)}}{\partial y}+\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi_{r,k,m}^{w(\ell+1)}\right]&\\ =-\left[\alpha_{r,k,\lambda}\frac{\partial\psi_{2,\lambda,m}^{s(\ell+½)}}{\partial x}+\beta_{r,k,\lambda}\frac{\partial\psi_{2,\lambda,m}^{s(\ell+½)}}{\partial y}\right]&\\ -\sum_{\substack{n=2\\ n\neq r}}^{N+1}\sum_{p=2^{n-2}(\lambda-1)+1}^{2^{n-2}\lambda}\left[A_{n,p}^{r,k}\frac{\partial\psi_{n,p,m}^{w(\ell+1)}}{\partial x}+B_{n,p}^{r,k}\frac{\partial\psi_{n,p,m}^{w(\ell+1)}}{\partial y}\right]&\\ r\geq 2,\ k=2^{r-2}(\lambda-1)+1,\ldots,2^{r-2}\lambda,\ m\geq 1&\end{aligned}$$

for the wavelet functions. It needs to be mentioned that the wavelet coefficients are solved for in increasing order of resolution level, i.e. we first solve for the scaling coefficients, then the r = 2 wavelet coefficients, then the r = 3 wavelet coefficients and so on, using the latest available data each time.

This iteration method can be seen more easily in operator notation:

$$D\psi_{2,\lambda}^{s(\ell+\frac{1}{2})} = Q\phi^{(\ell)} + S - \sum_{r=2}^{N+1} \sum_{k=2^{r-2}(\lambda-1)+1}^{2^{r-2}\lambda} W_{r,k}\psi_{r,k}^{w(\ell)}$$

$$D\psi_{r,k}^{w(\ell+1)} = -W_{r,k}\psi_{2,\lambda}^{s(\ell+\frac{1}{2})} - \sum_{\substack{n=2 \\ n\neq r}}^{N+1} \sum_{p=2^{n-2}(\lambda-1)+1}^{2^{n-2}\lambda} C_{n,p}^{r,k}\psi_{n,p}^{w(\ell+1)}$$

where *D*, *W,* and *C* are the scaling function, wavelet function, and wavelet-to-wavelet coupling operators, and *Q* and *S* are the scattering and extraneous source operators, respectively. We will discuss convergence of the S_N-W_N method in Chapter IX. This method results in one scaling function and $2^{N-2}-1$ wavelet functions per quadrant. Thus, we have a total of 2^N equations and unknowns per cell per iteration.

The second solution method likewise involves the usual ray-tracing strategy; however, all wavelet coefficients below each scaling coefficient are computed during each sweep. This is accomplished by applying a spatial discretization method and moving the incident fluxes to the source side of the equation. The resulting unknowns are then solved using a direct matrix inversion technique. This method is referred to as the CW-W_N method. This iteration method in operator notation is:

$$D\psi_{2,\lambda}^{s(\ell+1)} + \sum_{r=2}^{N+1} \sum_{k=2^{r-2}(\lambda-1)+1}^{2^{r-2}\lambda} W_{r,k}\psi_{r,k}^{w(\ell+1)} = Q\phi^{(\ell)} + S$$

$$D\psi_{r,k}^{w(\ell+1)} + W_{r,k}\psi_{2,\lambda}^{s(\ell+1)} + \sum_{\substack{n=2 \\ n\neq r}}^{N+1} \sum_{p=2^{n-2}(\lambda-1)+1}^{2^{n-2}\lambda} C_{n,p}^{r,k}\psi_{n,p}^{w(\ell+1)} = 0.$$

This method also has 2^N equations and unknowns per cell per iteration. Since there is a matrix inversion required, this method will take more computational time per cell than the S_N-W_N method. However, this iteration will converge since it is simply source iteration in this case. That is, we are not iterating upon wavelet coefficients.

The reason we use a multiresolution analysis based on wavelets is so that we will be able to "threshold" the angular flux in angle. This means that we are able to estimate the error in the angular flux, in a particular angular range, between two resolution levels, and only include those wavelet coefficients that are significant to our computations, while setting the insignificant wavelet coefficients to zero.

For the S_N-W_N method, we simply test the value of a particular wavelet coefficient once it is computed. If the value of that coefficient falls blow a threshold criterion, it is set to zero and its indices saved in a list. Before a wavelet coefficient is computed, it is checked against this list. If the domain of this wavelet is within the domain of any wavelet in the list, its coefficient value is set to zero. Else, it is computed as described above.

Using the CW-W_N method, when thresholding is included, in each cell, the scaling and first-level wavelet coefficients are first computed using the CW-W_N method described above. Then, each wavelet level is added to the matrix of unknowns until all coefficients at a given level have been thresholded out. The resulting matrix system is then solved directly for the unknown coefficients. Thresholding will be described in more detail in the next chapter.

CHAPTER VII

THRESHOLDING

In this chapter, we will discuss thresholding in more detail. In addition, we will derive a method of implementing the thresholding criterion and describe how this will be applied to the S_N-W_N and CW-W_N methods.

Thresholding is defined as efficient determination and exclusion of negligible coefficients. It can be seen that, for each wavelet in a given position within a resolution level, there are two wavelets in the next resolution level whose domains span the same angular range as the wavelet in the given resolution level. Thus, we can impose a binary tree structure on the wavelets having support in each quadrant, so that $w_{0,0}^{(i)}$ is the root for quadrant i, and the children of $w_{r,\lambda}^{(i)}$ are $w_{r+1,2\lambda}^{(i)}$ and $w_{r+1,2\lambda+1}^{(i)}$. Initially each node in this tree is marked as "active." The tree is subsequently pruned to designate as "inactive" nodes whose associated coefficients, after two passes of ray-tracing, fall below some designated "threshold fraction" ε of the maximum coefficient for any coarser-resolution wavelet, and all descendants of such nodes. In the S_N-W_N method, there is a binary tree in each quadrant, and in the CW-W_N method, there is a binary tree in each spatial cell.

We chose to threshold after two iterations in order to allow the scattering source to adequately develop. We then threshold once and only compute the non-thresholded coefficients each subsequent iteration. This is simply one of many strategies of implementing the thresholding with source iteration. Other strategies may include thresholding every M iterations, where M is some integer constant.

The quantity of interest is the angular flux, $\psi(x,y,\mu,\theta)$. The angular flux expansion is

$$\psi(x,y,\mu,\theta)=\sum_{\lambda=1}^{4}\left[\psi^{s}_{2,\lambda}(x,y,\mu)\,f_{2,\lambda}(\theta)\right]+\sum_{r=2}^{N+1}\sum_{k=1}^{2^r}\left[\psi^{w}_{r,k}(x,y,\mu)\,w_{r,k}(\theta)\right]. \tag{7.1}$$

If we have solved the system for all the wavelet coefficients up to some level *R*, then the angular flux is

$$\psi(x,y,\mu,\theta)=\sum_{\lambda=1}^{4}\left[\psi^{s}_{2,\lambda}(x,y,\mu)\,f_{2,\lambda}(\theta)\right]+\sum_{r=2}^{R}\sum_{k=1}^{2^r}\left[\psi^{w}_{r,k}(x,y,\mu)\,w_{r,k}(\theta)\right]. \tag{7.2}$$

Now, assume we solve for wavelet coefficient $\psi^{w}_{R',k'}(x,y,\mu)$. Then, the angular flux will be

$$\begin{aligned}\psi'(x,y,\mu,\theta)=\sum_{\lambda=1}^{4}\left[\psi^{s}_{2,\lambda}(x,y,\mu)\,f_{2,\lambda}(\theta)\right]+\sum_{r=2}^{R}\sum_{k=1}^{2^r}\left[\psi^{w}_{r,k}(x,y,\mu)\,w_{r,k}(\theta)\right]\\ +\psi^{w}_{R',k'}(x,y,\mu)\,w_{R',k'}(\theta)=\psi(x,y,\mu,\theta)+\psi^{w}_{R',k'}(x,y,\mu)\,w_{R',k'}(\theta).\end{aligned} \tag{7.3}$$

Then, we define the relative error between these two angular fluxes as

$$e(x,y,\mu)=\frac{\left\|\psi'(x,y,\mu,\theta)-\psi(x,y,\mu,\theta)\right\|_{L^2(\theta)}}{\left\|\psi(x,y,\mu,\theta)\right\|_{L^2(\theta)}} \tag{7.4}$$

Thus

$$\begin{aligned}e(x,y,\mu)&=\frac{\left\|\psi(x,y,\mu,\theta)+\psi^{w}_{R',k'}(x,y,\mu)\,w_{R',k'}(\theta)-\psi(x,y,\mu,\theta)\right\|_{L^2(\theta)}}{\left\|\psi(x,y,\mu,\theta)\right\|_{L^2(\theta)}}\\ &=\frac{\left\|\psi^{w}_{R',k'}(x,y,\mu)\,w_{R',k'}(\theta)\right\|_{L^2(\theta)}}{\left\|\psi(x,y,\mu,\theta)\right\|_{L^2(\theta)}}\\ &=\frac{\left|\psi^{w}_{R',k'}(x,y,\mu)\right|\left\|w_{R',k'}(\theta)\right\|_{L^2(\theta)}}{\left\|\psi(x,y,\mu,\theta)\right\|_{L^2(\theta)}}\end{aligned}$$

By the definition of $w_{R',k'}(\theta)$:

$$\left\| w_{R',k'}(\theta) \right\|_{L^2(\theta)} = 2\pi \tag{7.5}$$

Thus

$$e(x,y,\mu) = 2\pi \frac{\left| \psi^{w}_{R',k'}(x,y,\mu) \right|}{\left\| \psi(x,y,\mu,\theta) \right\|_{L^2(\theta)}} \tag{7.6}$$

Finally, we compute

$$\left\| \psi(x,y,\mu,\theta) \right\|_{L^2(\theta)} = \sqrt{2\pi} \sqrt{\sum_{\lambda=1}^{4} \left[\psi^{s}_{2,\lambda}(x,y,\mu) \right]^2 + \left[\psi^{w}_{2,\lambda}(x,y,\mu) \right]^2} \tag{7.7}$$

If this error is less than ε, the wavelet coefficient $\psi^{w}_{R',k'}(x,y,\mu)$ is thresholded out. Thus, the wavelet coefficient is set to zero if

$$\left| \psi^{w}_{R',k'}(x,y,\mu) \right| < \frac{\varepsilon}{\sqrt{2\pi}} \sqrt{\sum_{\lambda=1}^{4} \left[\psi^{s}_{2,\lambda}(x,y,\mu) \right]^2 + \left[\psi^{w}_{2,\lambda}(x,y,\mu) \right]^2} \tag{7.8}$$

and the tree is pruned. So, when we seek to calculate a wavelet coefficient, we first check to see if it lives on a pruned branch. If so, we set its value to zero. If not, we calculate its value using (6.17) and, if is meets the criterion of (7.8), set its value to zero and prune the tree. If it does not meet the threshold criterion, we save its value and move on to the next coefficient.

In both of the above solution methods, if the scattering cross section is nonzero, we will perform two source iterations without thresholding. Then, on the third iteration, we threshold. We then continue the source iteration to convergence, while performing no additional thresholding of the wavelet coefficients, but calculating the coefficients that survived the thresholding on the third iteration.

CHAPTER VIII

S_N-LIKE W_N METHOD

In this chapter, we will use the diamond-difference spatial approximation to derive an algorithm for implementing the S_N-W_N method.

Derivation

The S_N-W_N scaling and wavelet coefficient equations are

$$\begin{aligned}\left[\mu_{2,\lambda}\frac{\partial\psi^{s(\ell+½)}_{2,\lambda,m}}{\partial x}+\eta_{2,\lambda}\frac{\partial\psi^{s(\ell+½)}_{2,\lambda,m}}{\partial y}+\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi^{s(\ell+½)}_{2,\lambda,m}\right]&\\ =\frac{\sigma_s(x,y)}{8\pi\sqrt{1-\mu_m^2}}\phi^{(\ell)}(x,y)+\frac{S_{ext}(x,y)}{8\pi\sqrt{1-\mu_m^2}}&\\ -\sum_{r=2}^{N+1}\sum_{k=2^{r-2}(\lambda-1)+1}^{2^{r-2}\lambda}\left[\alpha_{r,k,\lambda}\frac{\partial\psi^{w(\ell)}_{r,k,m}}{\partial x}+\beta_{r,k,\lambda}\frac{\partial\psi^{w(\ell)}_{r,k,m}}{\partial y}\right],\quad N\ge 0,\ m\ge 1&\end{aligned}\tag{8.1}$$

and

$$\begin{aligned}\left[\mu_{r,k}\frac{\partial\psi^{w(\ell+1)}_{r,k,m}}{\partial x}+\eta_{r,k}\frac{\partial\psi^{w(\ell+1)}_{r,k,m}}{\partial y}+\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi^{w(\ell+1)}_{r,k,m}\right]&\\ =-\left[\alpha_{r,k,\lambda}\frac{\partial\psi^{s(\ell+½)}_{2,\lambda,m}}{\partial x}+\beta_{r,k,\lambda}\frac{\partial\psi^{s(\ell+½)}_{2,\lambda,m}}{\partial y}\right]&\\ -\sum_{\substack{n=2\\ n\ne r}}^{N+1}\sum_{p=2^{n-2}(\lambda-1)+1}^{2^{n-2}\lambda}\left[A^{r,k}_{n,p}\frac{\partial\psi^{w(\ell+1)}_{n,p,m}}{\partial x}+B^{r,k}_{n,p}\frac{\partial\psi^{w(\ell+1)}_{n,p,m}}{\partial y}\right]&\\ r\ge 2,\ k=2^{r-2}(\lambda-1)+1\ldots 2^{r-2}\lambda,\ m\ge 1&\end{aligned}\tag{8.2}$$

In order to solve this system, we will use the diamond-difference spatial approximation, where the angular flux is defined in terms of the cell edge fluxes as

$$\psi_{i,j} = \tfrac{1}{2}\left[\psi_{i-1/2,j} + \psi_{i+1/2,j}\right] = \tfrac{1}{2}\left[\psi_{i,j-1/2} + \psi_{i,j+1/2}\right] \tag{8.3}$$

A schematic of a cell is shown in Figure III–1. For this approach, we will delay the contributions of the wavelets in solving for the scaling function coefficients. Likewise, we will delay the contributions of higher-resolution wavelets in calculating the wavelet function coefficients.

Substituting the diamond difference approximation into the system, we find, for the scaling function coefficients,

$$\psi^{s}_{\lambda,i,j,m} = \frac{Q_{i,j,m} + 2\frac{|\mu_{2,\lambda}|}{\Delta x_{i,j}}\psi^{s}_{\lambda,i\text{-inc},j,m} + 2\frac{|\eta_{2,\lambda}|}{\Delta y_{i,j}}\psi^{s}_{\lambda,i,j\text{-inc},m}}{\frac{\sigma_{i,j}}{\sqrt{1-\mu_m^2}} + 2\frac{|\mu_{2,\lambda}|}{\Delta x_{i,j}} + 2\frac{|\eta_{2,\lambda}|}{\Delta y_{i,j}}} \tag{8.4a}$$

where $\psi^{s}_{\lambda,i\text{-inc},j,m}$ and $\psi^{s}_{\lambda,i,j\text{-inc},m}$ are the quadrant-dependent incident scaling function coefficients,

$$\begin{aligned} Q_{i,j,m} &= \frac{\sigma_{s,i,j}}{8\pi\sqrt{1-\mu_m^2}}\phi_{i,j} + \frac{S_{ext,i,j}}{8\pi\sqrt{1-\mu_m^2}} \\ &\quad -\sum_{r=2}^{N+1}\sum_{k=1}^{2^r}\left[2a\alpha_{r,k,\lambda}\frac{\psi^{w}_{r,k,i,j,m} - \psi^{w}_{r,k,i\text{-inc},j,m}}{\Delta x_{i,j}} + 2b\beta_{r,k,\lambda}\frac{\psi^{w}_{r,k,i,j,m} - \psi^{w}_{r,k,i,j\text{-inc},m}}{\Delta y_{i,j}}\right], \end{aligned} \tag{8.4b}$$

and

$$a = \begin{cases} 1, & \lambda = \{1,4\} \\ -1, & \lambda = \{2,3\} \end{cases} \qquad b = \begin{cases} 1, & \lambda = \{1,2\} \\ -1, & \lambda = \{3,4\} \end{cases}. \tag{8.4c}$$

The scalar flux is then

$$\phi_{i,j} = \pi\sum_{\lambda=1}^{4}\sum_{n}\omega_n\psi^{s}_{\lambda,i,j,n} \tag{8.5}$$

and the boundary conditions are

$$\begin{aligned} &\text{vaccum:} && \psi^{s}_{\lambda,i,j,m} = 0 \\ &\text{reflective:} && \psi^{s}_{\lambda,i,j,m} = \psi^{s}_{\lambda',i,j,m'} \\ &\text{incident:} && \psi^{s}_{\lambda,i,j,m} = \tfrac{1}{2} f_{i,j}. \end{aligned}$$

where *i* and *j* are located on the boundary. If we define the quadrants as

$$\begin{aligned} &\text{Quadrant 1:} && \mu > 0,\ \eta > 0 \\ &\text{Quadrant 2:} && \mu < 0,\ \eta > 0 \\ &\text{Quadrant 3:} && \mu < 0,\ \eta < 0 \\ &\text{Quadrant 4:} && \mu > 0,\ \eta < 0 \end{aligned}$$

then, for the reflecting boundary conditions, for the azimuthal angles, λ' are shown in Table VIII–1, and for the polar angles, $m' = M - m + 1$.

For wavelet coefficient *r*,*k*, in quadrant λ, the equation is

$$\psi^{w}_{r,k,i,j,m} = \frac{Q_{r,k,\lambda,i,j,m} + 2\frac{|\mu_{r,k}|}{\Delta x_{i,j}}\psi^{w}_{r,k,i\text{-inc},j,m} + 2\frac{|\eta_{r,k}|}{\Delta y_{i,j}}\psi^{w}_{r,k,i,j\text{-inc},m}}{\frac{\sigma_{i,j}}{\sqrt{1-\mu_m^2}} + 2\frac{|\mu_{r,k}|}{\Delta x_{i,j}} + 2\frac{|\eta_{r,k}|}{\Delta y_{i,j}}}, \tag{8.6a}$$

where $\psi^{w}_{r,k,i\text{-inc},j,m}$ and $\psi^{w}_{r,k,i,j\text{-inc},m}$ are the incident wavelet function coefficients and

Table VIII–1: Reflecting azimuthal angles for S_N-W_N method.

Face	Quadrant 1	Quadrant 2	Quadrant 3	Quadrant 4
Left	$\frac{1}{2}2^r - \lambda + 1$	-	-	$\frac{3}{2}2^r - \lambda + 1$
Right	-	$\frac{1}{2}2^r - \lambda + 1$	$\frac{3}{2}2^r - \lambda + 1$	-
Top	-	-	$2^r - \lambda + 1$	$2^r - \lambda + 1$
Bottom	$2^r - \lambda + 1$	$2^r - \lambda + 1$	-	-

$$Q_{r,k,\lambda,i,j,m} = -\left[2a\alpha_{r,k,\lambda} \frac{\psi^{s}_{\lambda,i,j,m} - \psi^{s}_{\lambda,i\text{-inc},j,m}}{\Delta x_{i,j}} + 2b\beta_{r,k,\lambda} \frac{\psi^{s}_{\lambda,i,j,m} - \psi^{s}_{\lambda,i,j\text{-inc},m}}{\Delta y_{i,j}} \right]$$
$$- \sum_{\substack{n=2 \\ n\neq r}}^{N+1} \sum_{p=2^{n-2}(\lambda-1)+1}^{2^{n-2}\lambda} \left[2aA^{r,k}_{n,p} \frac{\psi^{w}_{n,p,i,j,m} - \psi^{w}_{n,p,i\text{-inc},j,m}}{\Delta x_{i,j}} + 2bB^{r,k}_{n,p} \frac{\psi^{w}_{n,p,i,j,m} - \psi^{w}_{n,p,i,j\text{-inc},m}}{\Delta y_{i,j}} \right]. \tag{8.6b}$$

The boundary conditions are

$$\begin{array}{ll} \text{vacuum:} & \psi^{w}_{r,k,i,j,m} = 0 \\ \text{reflective:} & \psi^{w}_{r,k,i,j,m} = \psi^{w}_{r,k',i,j,m'} \\ \text{incident:} & \psi^{w}_{r,k,i,j,m} = 0 \end{array}$$

The final consideration is convergence of the iteration. We will define the error vector as

$$e^{(k)} = \phi^{(k)} - \phi^{(k-1)}$$

Then, we will calculate an estimate to the spectral radius of the iteration as

$$\rho^{(k)} = \frac{\left\| e^{(k)} \right\|_2}{\left\| e^{(k-1)} \right\|_2}$$

Then, we will consider the method converged if

$$\left\| e^{(k)} \right\|_2 \leq \varepsilon \left(1 - \rho^{(k)}\right) \left\| \phi^{(k)} \right\|_2$$

The term $(1-\rho)$ will prevent false convergence.

Algorithm

Now, we will present an algorithm for implementing the S_N-W_N method. To do this, we will use a special Fortran 90 structure called a "linked list." Details about linked list implementation in Fortran 90 are given in Appendix D. The algorithm follows. The values after the descriptions are operation counts assuming an $N \times N$ spatial mesh, with

M polar angles and an azimuthal resolution of *r*. The algorithm is shown in Figure VIII–1. In total, we have $O\left(N^2\right)+kO\left(4M\,2^{2r}N^2\right)$ operations computed, with $O\left(N^2\right)$ for the input section and $O\left(4M\,2^{2r}N^2\right)$ operations per source iteration. For our benchmark problems, this is about 10,000 operations for the input section, plus 655,360,000 operations per source iteration, with no thresholding. This large number of operations is due to the program having to compute coupling to all other directions at each "discrete" direction (represented by a wavelet index). Because of this coupling, the S_N-W_N is not

```
1) Read input file and allocate memory, $O\left(N^2\right)$
2) Calculate polar angles and weights, $O\left(M^3\right)$
3) Initialize variables, $O\left(r2^r N^2\right)$
4) Initialize list, $O\left(N^2\right)$
5) Begin source iteration, $k$
   a) Initialize convergence variables, $O\left(N^2\right)$
   b) Loop over polar angles, $M$
      i) Loop over quadrant, 4
         (1) Calculate reflective indices, $O(1)$
         (2) Calculate source matrix Q, $O\left(N^2\right)$
         (3) Set up sweep directions, $O(1)$
         (4) Construct y-boundary conditions, $O(N)$
         (5) Loop over y-position index, $N$
            (a) Construct x-boundary conditions, $O(1)$
            (b) Loop over x-position index, $N$
               (i) Calculate scaling coefficients, $O(1)$
               (ii) Calculate scaling coefficient spatial derivatives, $O(1)$
               (iii)Calculate cell exiting fluxes, $O(1)$
               (iv)Increment scalar flux, $O(1)$
               (v) Save x-boundary fluxes, $O(1)$
            (c) Save y-boundary fluxes, $O(N)$
        ii) Initialize wavelet coupling data, $O(N^2)$
        iii) Loop over resolution level, $r$
            (1) Loop over wavelet index, $2^r$
                (a) Determine quadrant, $O(1)$
                (b) Compute wavelet-to-wavelet coupling, $O(2^r N^2)$
                (c) Calculate source, $O(N^2)$
                (d) Set up sweep directions, $O(1)$
                (e) Construct y-boundary conditions, $O(2^{r+1} N)$
                (f) Loop over y-position index, $N$
                    (i) Construct x-boundary conditions, $O(2^{r+1})$
                    (ii) Loop over x-position index, $N$
                        1. Is $r$, $l$ a descendant of a wavelet in the list? , $O(r)$
                            a. Yes: set wavelet coefficient to zero, $O(1)$
                            b. No: compute wavelet coefficient, $O(1)$
                        2. Compute threshold criterion, $O(1)$
                        3. If wavelet coefficient is less than threshold, set coefficient
                           to zero and add wavelet to list, $O(1)$
                        4. Calculate wavelet coefficient spatial derivatives, $O(1)$
                        5. Calculate cell exiting fluxes, $O(1)$
                        6. Increment scaling-to-wavelet coupling, $O(1)$
                        7. Save x-boundary fluxes, $O(1)$
                    (iii)Save y-boundary fluxes, $O(N)$
    c) Calculate error vectors and determine convergence, $O(N^2)$
6) Write to output files, $O(N^2)$
```

Figure VIII–1: Algorithm for implementing the S_N-W_N method.

the same computationally as discrete ordinates, but takes $2^{1-r} + 2^r - 2$ times longer than discrete ordinates. Thus, in order to be competitive, we would need to be able to threshold all but two of the wavelet coefficients. Since there are only 2^r wavelet coefficients per quadrant, and as we cannot threshold all of them and expect a reasonable answer, this goal cannot be attained. However, we can improve this problem by only computing coupling to the coefficients that we know have non-zero coupling, instead of the entire quadrant. We will not attempt this in this dissertation, but we know that a coefficient at level r is coupled to $r-2+2^{R-r+1}$ other wavelet coefficients, where R is the azimuthal resolution level. Thus, if this were done, the S_N-W_N method would only take $\frac{3}{2}R$ times longer than discrete ordinates. Then, the number of coefficients that would have to be thresholded to be competitive would be 1–2%, depending upon the value of R. Thus, this method could potentially be made competitive in terms of computational time.

The input file is composed of regional inputs, so the problem simply has to be divided into regions. The code also has built-in material specifications, so that one only needs to read the number corresponding to a particular material. The available materials are:

1. Void ($\sigma = 0.0$, $\sigma_s = 0.0$)
2. Box1 Problem ($\sigma = 0.75$, $\sigma_s = 0.5$)
3. Box2 Problem ($\sigma = 0.375$, $\sigma_s = 0.25$)
4. Weak pure scatterer ($\sigma = 1.0$, $\sigma_s = 1.0$)
5. Weak absorber ($\sigma = 10.0$, $\sigma_s = 0.0$)
6. Strong absorber ($\sigma = 100.0$, $\sigma_s = 0.0$)
7. Pure scatterer ($\sigma = 100.0$, $\sigma_s = 100.0$)

8. Weak absorber ($\sigma = 1.0$, $\sigma_s = 0.0$)

Material indices (2) and (3) are used as described in Section IV.A. The source code for S_N-W_N is in Appendix B, as well as an example input file. The identifier in the incident fluxes section of the input file specifies the type of boundary condition to be implemented (vac = vacuum, ref = reflecting, inc = incident). The boundary conditions are listed in the order left, right, top, and bottom. The incident flux values are listed to the right of the identifier. The iteration parameters are the maximum number of iterations and the convergence criterion. Finally, the angular resolution section contains the number of polar angles and the value of *r* in the azimuthal discretization. All codes described in this dissertation are written in Fortran 90.

CHAPTER IX

CONVERGENCE OF THE S_N-W_N METHOD

In this chapter, we will investigate convergence of the S_N-W_N method. In order to do this, we will perform a Fourier error mode analysis on the discretized equations obtained by substituting the diamond-difference approximation into equations (8.4) and (8.6). First, we define $h_{x,i,j} = \sigma_{i,j}\Delta x_{i,j}$ and $h_{y,i,j} = \sigma_{i,j}\Delta y_{i,j}$. The scaling function coefficient equation, for quadrant λ, is then

$$\begin{aligned}
&\tfrac{\mu_{2,\lambda}}{h_{x,i,j}}\left(\psi^{s(\ell+1/2)}_{2,\lambda,i+1/2,j,m} - \psi^{s(\ell+1/2)}_{2,\lambda,i-1/2,j,m}\right) + \tfrac{\eta_{2,\lambda}}{h_{y,i,j}}\left(\psi^{s(\ell+1/2)}_{2,\lambda,i,j+1/2,m} - \psi^{s(\ell+1/2)}_{2,\lambda,i,j-1/2,m}\right) + \frac{1}{\sqrt{1-\mu_m^2}}\psi^{s(\ell+1/2)}_{2,\lambda,i,j,m} \\
&\qquad = \frac{c_{i,j}}{8\pi\sqrt{1-\mu_m^2}}\phi^{(\ell)}_{i,j} + \frac{S_{ext,i,j}}{8\pi\sigma_{i,j}\sqrt{1-\mu_m^2}} \qquad (9.1)\\
&\qquad -\sum_{r=2}^{N+1}\sum_{k=2^{r-2}(\lambda-1)+1}^{2^{r-2}\lambda}\left[\tfrac{\alpha_{r,k,\lambda}}{h_{x,i,j}}\left(\psi^{w(\ell)}_{r,k,i+1/2,j,m} - \psi^{w(\ell)}_{r,k,i-1/2,j,m}\right) + \tfrac{\beta_{r,k,\lambda}}{h_{y,i,j}}\left(\psi^{w(\ell)}_{r,k,i,j+1/2,m} - \psi^{w(\ell)}_{r,k,i,j-1/2,m}\right)\right],
\end{aligned}$$

the wavelet function coefficient equation, for coefficient r, k, is

$$\begin{aligned}
&\left[\tfrac{\mu_{r,k}}{h_{x,i,j}}\left(\psi^{w(\ell+1)}_{r,k,i+1/2,j,m} - \psi^{w(\ell+1)}_{r,k,i-1/2,j,m}\right) + \tfrac{\eta_{r,k}}{h_{y,i,j}}\left(\psi^{w(\ell+1)}_{r,k,i,j+1/2,m} - \psi^{w(\ell+1)}_{r,k,i,j-1/2,m}\right) + \frac{1}{\sqrt{1-\mu_m^2}}\psi^{w(\ell+1)}_{r,k,i,j,m}\right] \\
&\qquad = -\left[\tfrac{\alpha_{r,k,\lambda}}{h_{x,i,j}}\left(\psi^{s(\ell+1/2)}_{2,\lambda,i+1/2,j,m} - \psi^{s(\ell+1/2)}_{2,\lambda,i-1/2,j,m}\right) + \tfrac{\beta_{r,k,\lambda}}{h_{y,i,j}}\left(\psi^{s(\ell+1/2)}_{2,\lambda,i,j+1/2,m} - \psi^{s(\ell+1/2)}_{2,\lambda,i,j-1/2,m}\right)\right] \qquad (9.2)\\
&\qquad -\sum_{\substack{n=2\\ n\neq r}}^{N+1}\sum_{p=2^{n-2}(\lambda-1)+1}^{2^{n-2}\lambda}\left[\tfrac{A^{r,k}_{n,p}}{h_{x,i,j}}\left(\psi^{w(\ell+1)}_{n,p,i+1/2,j,m} - \psi^{w(\ell+1)}_{n,p,i-1/2,j,m}\right) + \tfrac{B^{r,k}_{n,p}}{h_{y,i,j}}\left(\psi^{w(\ell+1)}_{n,p,i,j+1/2,m} - \psi^{w(\ell+1)}_{n,p,i,j-1/2,m}\right)\right],
\end{aligned}$$

and the equation for the scalar flux is

$$\phi^{(\ell+1)}_{i,j} = \pi\sum_{\lambda=1}^{4}\sum_{n}\omega_n\psi^{s(\ell+1/2)}_{2,\lambda,i,j,n}. \qquad (9.3)$$

First, we will write the equations for the converged solutions: The equation for the converged scaling function coefficient is

$$\frac{\mu_{2,\lambda}}{h_{x,i,j}}\left(\psi^{s,con}_{2,\lambda,i+1/2,j,m}-\psi^{s,con}_{2,\lambda,i-1/2,j,m}\right)+\frac{\eta_{2,\lambda}}{h_{y,i,j}}\left(\psi^{s,con}_{2,\lambda,i,j+1/2,m}-\psi^{s,con}_{2,\lambda,i,j-1/2,m}\right)+\frac{1}{\sqrt{1-\mu_m^2}}\psi^{s,con}_{2,\lambda,i,j,m}$$
$$=\frac{c_{i,j}}{8\pi\sqrt{1-\mu_m^2}}\phi^{con}_{i,j}+\frac{S_{ext,i,j}}{8\pi\sigma_{i,j}\sqrt{1-\mu_m^2}} \tag{9.4}$$
$$-\sum_{r=2}^{N+1}\sum_{k=2^{r-2}(\lambda-1)+1}^{2^{r-2}\lambda}\left[\frac{\alpha_{r,k,\lambda}}{h_{x,i,j}}\left(\psi^{w,con}_{r,k,i+1/2,j,m}-\psi^{w,con}_{r,k,i-1/2,j,m}\right)+\frac{\beta_{r,k,\lambda}}{h_{y,i,j}}\left(\psi^{w,con}_{r,k,i,j+1/2,m}-\psi^{w,con}_{r,k,i,j-1/2,m}\right)\right],$$

and the equations for the converged wavelet function coefficients are

$$\left[\frac{\mu_{r,k}}{h_{x,i,j}}\left(\psi^{w,con}_{r,k,i-1/2,j,m}-\psi^{w,con}_{r,k,i-1/2,j,m}\right)+\frac{\eta_{r,k}}{h_{y,i,j}}\left(\psi^{w,con}_{r,k,i,j+1/2,m}-\psi^{w,con}_{r,k,i,j-1/2,m}\right)+\frac{1}{\sqrt{1-\mu_m^2}}\psi^{w,con}_{r,k,i,j,m}\right]$$
$$=-\left[\frac{\alpha_{r,k,\lambda}}{h_{x,i,j}}\left(\psi^{s,con}_{2,\lambda,i+1/2,j,m}-\psi^{s,con}_{2,\lambda,i-1/2,j,m}\right)+\frac{\beta_{r,k,\lambda}}{h_{y,i,j}}\left(\psi^{s,con}_{2,\lambda,i,j+1/2,m}-\psi^{s,con}_{2,\lambda,i,j-1/2,m}\right)\right] \tag{9.5}$$
$$-\sum_{\substack{n=2\\n\neq r}}^{N+1}\sum_{p=2^{n-2}(\lambda-1)+1}^{2^{n-2}\lambda}\left[\frac{A^{r,k}_{n,p}}{h_{x,i,j}}\left(\psi^{w,con}_{n,p,i+1/2,j,m}-\psi^{w,con}_{n,p,i-1/2,j,m}\right)+\frac{B^{r,k}_{n,p}}{h_{y,i,j}}\left(\psi^{w,con}_{n,p,i,j+1/2,m}-\psi^{w,con}_{n,p,i,j-1/2,m}\right)\right],$$

The equation for the converged scalar flux is

$$\phi^{con}_{i,j}=\pi\sum_{\lambda=1}^{4}\sum_{n}\omega_n\psi^{s,con}_{2,\lambda,i,j,n}. \tag{9.6}$$

We will define the iteration errors as

$$\Psi^{s(\ell)}_{2,\lambda,i,j,m}:=\psi^{s,con}_{2,\lambda,i,j,m}-\psi^{s(\ell)}_{2,\lambda,i,j,m}$$
$$\Psi^{w(\ell)}_{r,k,i,j,m}:=\psi^{w,con}_{r,k,i,j,m}-\psi^{w(\ell)}_{r,k,i,j,m}$$
$$\Phi^{(\ell)}_{i,j}:=\phi^{con}_{i,j}-\phi^{(\ell)}_{i,j}$$

Subtracting equation (9.1) from equation (9.3), we obtain the equation for the scaling function coefficient iteration error, in quadrant λ:

$$\begin{aligned}\frac{\mu_{2,\lambda}}{h_{x,i,j}}\left(\Psi^{s(\ell+1/2)}_{2,\lambda,i+1/2,j,m}-\Psi^{s(\ell+1/2)}_{2,\lambda,i-1/2,j,m}\right)+\frac{\eta_{2,\lambda}}{h_{y,i,j}}\left(\Psi^{s(\ell+1/2)}_{2,\lambda,i,j+1/2,m}-\Psi^{s(\ell+1/2)}_{2,\lambda,i,j-1/2,m}\right)\\ +\frac{1}{\sqrt{1-\mu_m^2}}\Psi^{s(\ell+1/2)}_{2,\lambda,i,j,m}=\frac{c_{i,j}}{8\pi\sqrt{1-\mu_m^2}}\Phi^{(\ell)}_{i,j}\\ -\sum_{r=2}^{N+1}\sum_{k=2^{r-2}(\lambda-1)+1}^{2^{r-2}\lambda}\left[\frac{\alpha_{r,k,\lambda}}{h_{x,i,j}}\left(\Psi^{w(\ell)}_{r,k,i+1/2,j,m}-\Psi^{w(\ell)}_{r,k,i-1/2,j,m}\right)+\frac{\beta_{r,k,\lambda}}{h_{y,i,j}}\left(\Psi^{w(\ell)}_{r,k,i,j+1/2,m}-\Psi^{w(\ell)}_{r,k,i,j-1/2,m}\right)\right].\end{aligned}\tag{9.7}$$

Likewise, subtracting equation (9.2) from equation (9.4), we obtain the equation for the wavelet function coefficient iteration error *r*, *k*:

$$\begin{aligned}\left[\frac{\mu_{r,k}}{h_{x,i,j}}\left(\Psi^{w(\ell+1)}_{r,k,i+1/2,j,m}-\Psi^{w(\ell+1)}_{r,k,i-1/2,j,m}\right)+\frac{\eta_{r,k}}{h_{y,i,j}}\left(\Psi^{w(\ell+1)}_{r,k,i,j+1/2,m}-\Psi^{w(\ell+1)}_{r,k,i,j-1/2,m}\right)+\frac{1}{\sqrt{1-\mu_m^2}}\Psi^{w(\ell+1)}_{r,k,i,j,m}\right]\\ =-\left[\frac{\alpha_{r,k,\lambda}}{h_{x,i,j}}\left(\Psi^{s(\ell+1/2)}_{2,\lambda,i+1/2,j,m}-\Psi^{s(\ell+1/2)}_{2,\lambda,i-1/2,j,m}\right)+\frac{\beta_{r,k,\lambda}}{h_{y,i,j}}\left(\Psi^{s(\ell+1/2)}_{2,\lambda,i,j+1/2,m}-\Psi^{s(\ell+1/2)}_{2,\lambda,i,j-1/2,m}\right)\right]\\ -\sum_{\substack{n=2\\ n\neq r}}^{N+1}\sum_{p=2^{n-2}(\lambda-1)+1}^{2^{n-2}\lambda}\left[\frac{A^{r,k}_{n,p}}{h_{x,i,j}}\left(\Psi^{w(\ell+1)}_{n,p,i+1/2,j,m}-\Psi^{w(\ell+1)}_{n,p,i-1/2,j,m}\right)+\frac{B^{r,k}_{n,p}}{h_{y,i,j}}\left(\Psi^{w(\ell+1)}_{n,p,i,j+1/2,m}-\Psi^{w(\ell+1)}_{n,p,i,j-1/2,m}\right)\right].\end{aligned}\tag{9.8}$$

Finally, subtracting equation (9.3) from equation (9.6), the equation for the scalar flux iteration error is

$$\Phi^{(\ell+1)}_{i,j}=\pi\sum_{\lambda=1}^{4}\sum_{n}\omega_n\Psi^{s(\ell+1/2)}_{2,\lambda,i,j,n}.\,. \tag{9.9}$$

Next, we make the ansatz that the iteration errors are of the form

$$\begin{aligned}
\Psi^{s(\ell+1/2)}_{2,\lambda,i,j,m} &= \omega^{\ell} a_{\lambda,m} e^{i\sigma\left(\lambda_x x_i + \lambda_y y_j\right)} \\
\Psi^{s(\ell+1/2)}_{2,\lambda,i+1/2,j,m} &= \omega^{\ell} b_{\lambda,m} e^{i\sigma\left(\lambda_x x_{i+1/2} + \lambda_y y_j\right)} \\
\Psi^{s(\ell+1/2)}_{2,\lambda,i,j+1/2,m} &= \omega^{\ell} c_{\lambda,m} e^{i\sigma\left(\lambda_x x_i + \lambda_y y_{j+1/2}\right)} \\
\Psi^{w(\ell)}_{r,k,i,j,m} &= \omega^{\ell} d_{r,k,m} e^{i\sigma\left(\lambda_x x_i + \lambda_y y_j\right)} \\
\Psi^{w(\ell)}_{r,k,i+1/2,j,m} &= \omega^{\ell} e_{r,k,m} e^{i\sigma\left(\lambda_x x_{i+1/2} + \lambda_y y_j\right)} \\
\Psi^{w(\ell)}_{r,k,i,j+1/2,m} &= \omega^{\ell} f_{r,k,m} e^{i\sigma\left(\lambda_x x_i + \lambda_y y_{j+1/2}\right)} \\
\Phi^{(\ell)}_{i,j} &= \omega^{\ell} A e^{i\sigma\left(\lambda_x x_i + \lambda_y y_j\right)}
\end{aligned} \quad , \tag{9.10}$$

where ω is the iteration eigenvalue. Substituting this ansatz into equation (9.7), we find

$$\begin{aligned}
& b_{\lambda,m} \tfrac{\mu_{2,\lambda}}{h_{x,i,j}} \left(e^{\frac{1}{2} i \lambda_x h_{x,i,j}} - e^{-\frac{1}{2} i \lambda_x h_{x,i,j}} \right) + c_{\lambda,m} \tfrac{\eta_{2,\lambda}}{h_{y,i,j}} \left(e^{\frac{1}{2} i \lambda_y h_{y,i,j}} - e^{-\frac{1}{2} i \lambda_y h_{y,i,j}} \right) \\
& \qquad + \frac{1}{\sqrt{1-\mu_m^2}} a_{\lambda,m} = \frac{c_{i,j}}{8\pi\sqrt{1-\mu_m^2}} A \\
& - \sum_{r=2}^{N+1} \sum_{k=2^{r-2}(\lambda-1)+1}^{2^{r-2}\lambda} \left[e_{r,k,m} \tfrac{\alpha_{r,k,\lambda}}{h_{x,i,j}} \left(e^{\frac{1}{2} i \lambda_x h_{x,i,j}} - e^{-\frac{1}{2} i \lambda_x h_{x,i,j}} \right) + f_{r,k,m} \tfrac{\beta_{r,k,\lambda}}{h_{y,i,j}} \left(e^{\frac{1}{2} i \lambda_y h_{y,i,j}} - e^{-\frac{1}{2} i \lambda_y h_{y,i,j}} \right) \right].
\end{aligned} \tag{9.11}$$

Substituting the ansatz into the diamond-difference approximation, we find

$$\begin{aligned}
a_{\lambda,m} &= \tfrac{1}{2} b_{\lambda,m} \left[e^{\frac{1}{2} i \lambda_x h_{x,i,j}} + e^{-\frac{1}{2} i \lambda_x h_{x,i,j}} \right] = \tfrac{1}{2} c_{\lambda,m} \left[e^{\frac{1}{2} i \lambda_y h_{y,i,j}} + e^{-\frac{1}{2} i \lambda_y h_{y,i,j}} \right] \\
d_{r,k,m} &= \tfrac{1}{2} e_{r,k,m} \left[e^{\frac{1}{2} i \lambda_x h_{x,i,j}} + e^{-\frac{1}{2} i \lambda_x h_{x,i,j}} \right] = \tfrac{1}{2} f_{r,k,m} \left[e^{\frac{1}{2} i \lambda_y h_{y,i,j}} + e^{-\frac{1}{2} i \lambda_y h_{y,i,j}} \right]
\end{aligned} . \tag{9.12}$$

Thus,

$$2a_{\lambda,m}\tfrac{\mu_{2,\lambda}}{h_{x,i,j}}\frac{e^{\frac{1}{2}i\lambda_x h_{x,i,j}}-e^{-\frac{1}{2}i\lambda_x h_{x,i,j}}}{e^{\frac{1}{2}i\lambda_x h_{x,i,j}}+e^{-\frac{1}{2}i\lambda_x h_{x,i,j}}}+2a_{\lambda,m}\tfrac{\eta_{2,\lambda}}{h_{y,i,j}}\frac{e^{\frac{1}{2}i\lambda_y h_{y,i,j}}-e^{-\frac{1}{2}i\lambda_y h_{y,i,j}}}{e^{\frac{1}{2}i\lambda_y h_{y,i,j}}+e^{-\frac{1}{2}i\lambda_y h_{y,i,j}}}$$
$$+\frac{1}{\sqrt{1-\mu_m^2}}a_{\lambda,m}=\frac{c_{i,j}}{8\pi\sqrt{1-\mu_m^2}}A \tag{9.13}$$
$$-\sum_{r=2}^{N+1}\sum_{k=2^{r-2}(\lambda-1)+1}^{2^{r-2}\lambda}\left[2d_{r,k,m}\tfrac{\alpha_{r,k,\lambda}}{h_{x,i,j}}\frac{e^{\frac{1}{2}i\lambda_x h_{x,i,j}}-e^{-\frac{1}{2}i\lambda_x h_{x,i,j}}}{e^{\frac{1}{2}i\lambda_x h_{x,i,j}}+e^{-\frac{1}{2}i\lambda_x h_{x,i,j}}}+2d_{r,k,m}\tfrac{\beta_{r,k,\lambda}}{h_{y,i,j}}\frac{e^{\frac{1}{2}i\lambda_y h_{y,i,j}}-e^{-\frac{1}{2}i\lambda_y h_{y,i,j}}}{e^{\frac{1}{2}i\lambda_y h_{y,i,j}}+e^{-\frac{1}{2}i\lambda_y h_{y,i,j}}}\right].$$

Next, we will use the property:

$$\frac{e^{ix}-e^{-ix}}{e^{ix}+e^{-ix}}=i\tan(x) \tag{9.14}$$

So,

$$a_{\lambda,m}\left[\mu_{2,\lambda}\tfrac{2}{h_{x,i,j}}i\tan\left(\tfrac{1}{2}\lambda_x h_{x,i,j}\right)+\eta_{2,\lambda}\tfrac{2}{h_{y,i,j}}i\tan\left(\tfrac{1}{2}\lambda_y h_{y,i,j}\right)+\frac{1}{\sqrt{1-\mu_m^2}}\right]$$
$$=\frac{c_{i,j}}{8\pi\sqrt{1-\mu_m^2}}A-\sum_{r=2}^{N+1}\sum_{k=2^{r-2}(\lambda-1)+1}^{2^{r-2}\lambda}d_{r,k,m}\left[\alpha_{r,k,\lambda}\tfrac{2}{h_{x,i,j}}i\tan\left(\tfrac{1}{2}\lambda_x h_{x,i,j}\right)+\beta_{r,k,\lambda}\tfrac{2}{h_{y,i,j}}i\tan\left(\tfrac{1}{2}\lambda_y h_{y,i,j}\right)\right]. \tag{9.15}$$

Finally, we define

$$\theta_{x,i,j}:=\lambda_x h_{x,i,j}\quad\text{and}\quad\theta_{y,i,j}:=\lambda_y h_{y,i,j} \tag{9.16}$$

Therefore, we obtain, for the scaling coefficient errors,

$$a_{\lambda,m}\left[2\tfrac{\mu_{2,\lambda}}{h_{x,i,j}}i\tan\left(\tfrac{1}{2}\theta_{x,i,j}\right)+2\tfrac{\eta_{2,\lambda}}{h_{y,i,j}}i\tan\left(\tfrac{1}{2}\theta_{y,i,j}\right)+\frac{1}{\sqrt{1-\mu_m^2}}\right]$$
$$=\frac{c_{i,j}}{8\pi\sqrt{1-\mu_m^2}}A-\sum_{r=2}^{N+1}\sum_{k=2^{r-2}(\lambda-1)+1}^{2^{r-2}\lambda}d_{r,k,m}\left[2\tfrac{\alpha_{r,k,\lambda}}{h_{x,i,j}}i\tan\left(\tfrac{1}{2}\theta_{x,i,j}\right)+2\tfrac{\beta_{r,k,\lambda}}{h_{y,i,j}}i\tan\left(\tfrac{1}{2}\theta_{y,i,j}\right)\right]. \tag{9.17}$$

Substituting the ansatz into equation (9.8), we find

$$\begin{aligned}
&\left[\omega e_{r,k,m}\tfrac{\mu_{r,k}}{h_{x,i,j}}\left(e^{\frac{1}{2}i\sigma\lambda_x h_{x,i,j}}-e^{-\frac{1}{2}i\sigma\lambda_x h_{x,i,j}}\right)+\omega f_{r,k,m}\tfrac{\eta_{r,k}}{h_{y,i,j}}\left(e^{\frac{1}{2}i\sigma\lambda_y h_{y,i,j}}-e^{-\frac{1}{2}i\sigma\lambda_y h_{y,i,j}}\right)+\frac{1}{\sqrt{1-\mu_m^2}}\omega d_{r,k,m}\right] \\
&\quad=-\left[b_{\lambda,m}\tfrac{\alpha_{r,k,\lambda}}{h_{x,i,j}}\left(e^{\frac{1}{2}i\sigma\lambda_x h_{x,i,j}}-e^{-\frac{1}{2}i\sigma\lambda_x h_{x,i,j}}\right)+c_{\lambda,m}\tfrac{\beta_{r,k,\lambda}}{h_{y,i,j}}\left(e^{\frac{1}{2}i\sigma\lambda_y h_{y,i,j}}-e^{-\frac{1}{2}i\sigma\lambda_y h_{y,i,j}}\right)\right] \\
&\quad-\sum_{\substack{n=2\\n\neq r}}^{N+1}\sum_{p=2^{n-2}(\lambda-1)+1}^{2^{n-2}\lambda}\left[\omega e_{n,p,m}\tfrac{A_{n,p}^{r,k}}{h_{x,i,j}}\left(e^{\frac{1}{2}i\sigma\lambda_x h_{x,i,j}}-e^{-\frac{1}{2}i\sigma\lambda_x h_{x,i,j}}\right)+\omega f_{n,p,m}\tfrac{B_{n,p}^{r,k}}{h_{y,i,j}}\left(e^{\frac{1}{2}i\sigma\lambda_y h_{y,i,j}}-e^{-\frac{1}{2}i\sigma\lambda_y h_{y,i,j}}\right)\right].
\end{aligned} \tag{9.18}$$

Following the same pattern as above, we obtain, for the wavelet coefficient errors,

$$\begin{aligned}
&\omega d_{r,k,m}\left[\mu_{r,k}\tfrac{2}{h_{x,i,j}}i\tan\left(\tfrac{1}{2}\theta_{x,i,j}\right)+\eta_{r,k}\tfrac{2}{h_{y,i,j}}i\tan\left(\tfrac{1}{2}\theta_{y,i,j}\right)+\frac{1}{\sqrt{1-\mu_m^2}}\right] \\
&\quad=-a_{\lambda,m}\left[\alpha_{r,k,\lambda}\tfrac{2}{h_{x,i,j}}i\tan\left(\tfrac{1}{2}\theta_{x,i,j}\right)+\beta_{r,k,\lambda}\tfrac{2}{h_{x,i,j}}i\tan\left(\tfrac{1}{2}\theta_{x,i,j}\right)\right] \\
&\quad-\sum_{\substack{n=2\\n\neq r}}^{N+1}\sum_{p=2^{n-2}(\lambda-1)+1}^{2^{n-2}\lambda}\omega d_{n,p,m}\left[A_{n,p}^{r,k}\tfrac{2}{h_{x,i,j}}i\tan\left(\tfrac{1}{2}\theta_{x,i,j}\right)+B_{n,p}^{r,k}\tfrac{2}{h_{x,i,j}}i\tan\left(\tfrac{1}{2}\theta_{x,i,j}\right)\right].
\end{aligned} \tag{9.19}$$

Substituting the ansatz into equation (9.9), we find

$$\omega A=\pi\sum_{\lambda=1}^{4}\sum_{n}\omega_n a_{\lambda,n}. \tag{9.20}$$

Using the above equations, we will present estimates of the iteration eigenvalue for the cases where N = 0 and N = 1.

If we let N = 0, for one polar angle, we find,

$$\omega=\tfrac{c_{i,j}}{4}\sum_{\lambda=1}^{4}\frac{1}{\left[\tfrac{2}{h_{x,i,j}}\mu_{2,\lambda}\tan\left(\tfrac{1}{2}\theta_{x,i,j}\right)+\tfrac{2}{h_{y,i,j}}\eta_{2,\lambda}\tan\left(\tfrac{1}{2}\theta_{y,i,j}\right)\right]^2+1}.$$

Figure IX–1 shows a 3-D plot of $\omega/c_{i,j}$ for $\theta_{x,i,j}$ and $\theta_{y,i,j}$ between 0 and 2π, and for various $h_{i,j}$ between 10^{-2} (grey) and 10^{2} (turquoise). Looking at the above equation, it can be seen that the term inside the summation ranges between zero and one, since, for

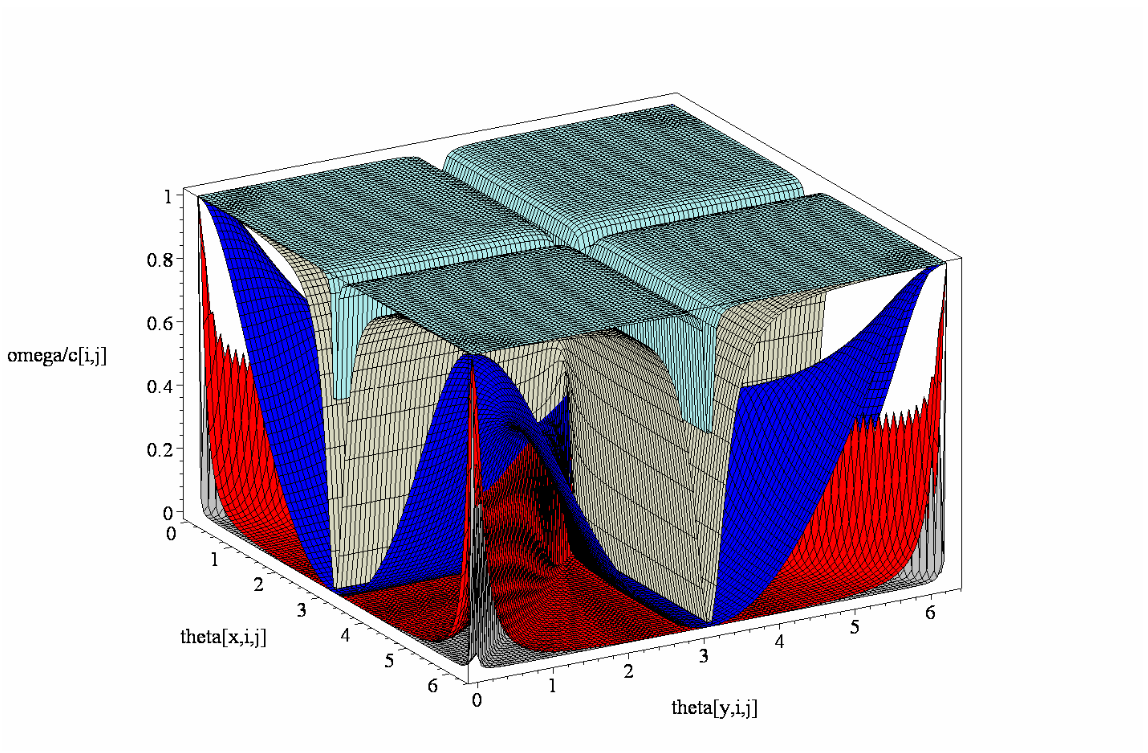


Figure IX–1: The iteration eigenvalues for the S_N-W_N method.

the θ terms between zero and 2π, the tangent terms range between zero and infinity, thus, the denominator ranges between one and infinity. Since this is true, the summation is always less than or equal to four, so *w* is always less than or equal to *c*. Thus, for the N = 0 case, the eigenvalue of the iteration is always less than the scattering ratio, *c*, and always greater then zero. Convergence is guaranteed for eigenvalues less than one, so, since *c* is always less than one, the iteration converges.

Letting N = 1, for one polar angle, we find, for the scaling function coefficient iteration error

$$a_\lambda\left[\mu_{2,\lambda}\tfrac{2}{h_{x,i,j}}i\tan\left(\tfrac{1}{2}\theta_{x,i,j}\right)+\eta_{2,\lambda}\tfrac{2}{h_{y,i,j}}i\tan\left(\tfrac{1}{2}\theta_{y,i,j}\right)+1\right]$$
$$=\frac{c_{i,j}}{8\pi}A-d_{2,\lambda}\left[\alpha_{2,\lambda,\lambda}\tfrac{2}{h_{x,i,j}}i\tan\left(\tfrac{1}{2}\theta_{x,i,j}\right)+\beta_{2,\lambda,\lambda}\tfrac{2}{h_{y,i,j}}i\tan\left(\tfrac{1}{2}\theta_{y,i,j}\right)\right],$$

and for the wavelet function coefficient iteration error

$$\omega d_{2,\lambda}\left[\mu_{2,\lambda}\tfrac{2}{h_{x,i,j}}i\tan\left(\tfrac{1}{2}\theta_{x,i,j}\right)+\eta_{2,\lambda}\tfrac{2}{h_{y,i,j}}i\tan\left(\tfrac{1}{2}\theta_{y,i,j}\right)+1\right]$$
$$=-a_\lambda\left[\alpha_{2,\lambda,\lambda}\tfrac{2}{h_{x,i,j}}i\tan\left(\tfrac{1}{2}\theta_{x,i,j}\right)+\beta_{2,\lambda,\lambda}\tfrac{2}{h_{x,i,j}}i\tan\left(\tfrac{1}{2}\theta_{x,i,j}\right)\right].$$

For simplicity in the following analysis, we will define

$$\Gamma_{s,\lambda,i,j}:=\mu_{2,\lambda}\tfrac{2}{h_{x,i,j}}i\tan\left(\tfrac{1}{2}\theta_{x,i,j}\right)+\eta_{2,\lambda}\tfrac{2}{h_{y,i,j}}i\tan\left(\tfrac{1}{2}\theta_{y,i,j}\right)+1$$
$$\Gamma_{w,\lambda,i,j}:=\alpha_{2,\lambda,\lambda}\tfrac{2}{h_{x,i,j}}i\tan\left(\tfrac{1}{2}\theta_{x,i,j}\right)+\beta_{2,\lambda,\lambda}\tfrac{2}{h_{y,i,j}}i\tan\left(\tfrac{1}{2}\theta_{y,i,j}\right)$$

Thus, our scaling function coefficient iteration error equation is

$$a_\lambda\Gamma_{s,\lambda,i,j}=\frac{c_{i,j}}{8\pi}A-d_{2,\lambda}\Gamma_{w,\lambda,i,j},$$

and, for the wavelet function coefficient

$$\omega d_{2,\lambda}\Gamma_{s,\lambda,i,j}=-a_\lambda\Gamma_{w,\lambda,i,j}.$$

The wavelet equation forms the 4×4 matrix system

$$\omega\begin{bmatrix}d_{2,1}\\d_{2,2}\\d_{2,3}\\d_{2,4}\end{bmatrix}=\begin{bmatrix}-\frac{\Gamma_{w,1,i,j}}{\Gamma_{s,1,i,j}}&&&\\&-\frac{\Gamma_{w,2,i,j}}{\Gamma_{s,2,i,j}}&&\\&&-\frac{\Gamma_{w,3,i,j}}{\Gamma_{s,3,i,j}}&\\&&&-\frac{\Gamma_{w,4,i,j}}{\Gamma_{s,4,i,j}}\end{bmatrix}\begin{bmatrix}a_1\\a_2\\a_3\\a_4\end{bmatrix}$$

Combining this with the scalar flux iteration error equation, we find

$$\omega\begin{bmatrix} A \\ d_{2,1} \\ d_{2,2} \\ d_{2,3} \\ d_{2,4} \end{bmatrix} = \begin{bmatrix} 2\pi & 2\pi & 2\pi & 2\pi \\ -\frac{\Gamma_{w,1,i,j}}{\Gamma_{s,1,i,j}} & & & \\ & -\frac{\Gamma_{w,2,i,j}}{\Gamma_{s,2,i,j}} & & \\ & & -\frac{\Gamma_{w,3,i,j}}{\Gamma_{s,3,i,j}} & \\ & & & -\frac{\Gamma_{w,4,i,j}}{\Gamma_{s,4,i,j}} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}$$

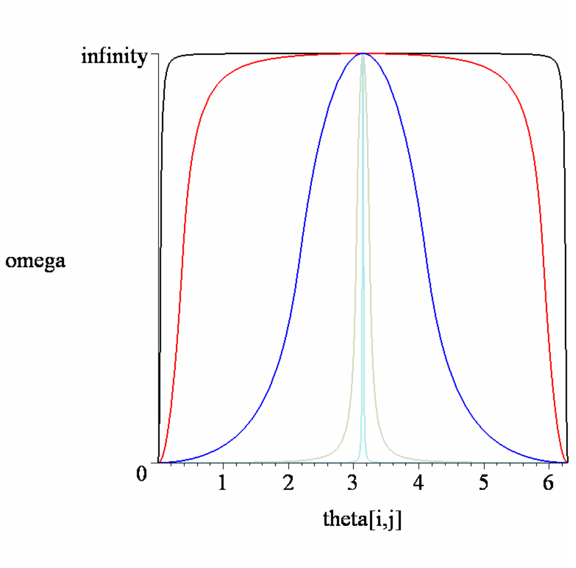
From the scaling function coefficient equation, we find

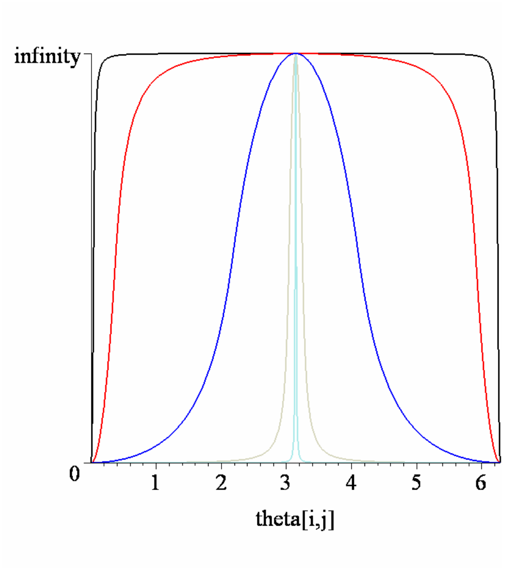
$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} \frac{\frac{c_{i,j}}{8\pi}}{\Gamma_{s,1,i,j}} & -\frac{\Gamma_{w,1,i,j}}{\Gamma_{s,1,i,j}} & & & \\ \frac{\frac{c_{i,j}}{8\pi}}{\Gamma_{s,2,i,j}} & & -\frac{\Gamma_{w,2,i,j}}{\Gamma_{s,2,i,j}} & & \\ \frac{\frac{c_{i,j}}{8\pi}}{\Gamma_{s,3,i,j}} & & & -\frac{\Gamma_{w,3,i,j}}{\Gamma_{s,3,i,j}} & \\ \frac{\frac{c_{i,j}}{8\pi}}{\Gamma_{s,4,i,j}} & & & & -\frac{\Gamma_{w,4,i,j}}{\Gamma_{s,4,i,j}} \end{bmatrix} \begin{bmatrix} A \\ d_{2,1} \\ d_{2,2} \\ d_{2,3} \\ d_{2,4} \end{bmatrix}$$

Combining these two matrix systems, we find an eigenvalue equation for ω:

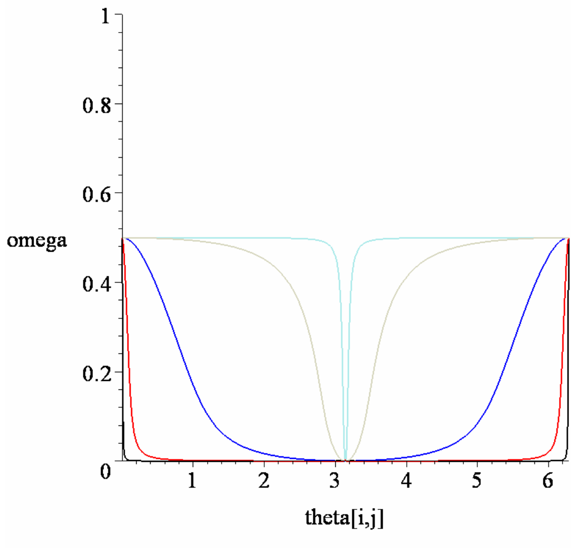
$$\omega\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} \frac{\frac{c_{i,j}}{4}}{\Gamma_{s,1,i,j}} + \frac{\Gamma^2_{w,1,i,j}}{\Gamma^2_{s,1,i,j}} & \frac{\frac{c_{i,j}}{4}}{\Gamma_{s,1,i,j}} & \frac{\frac{c_{i,j}}{4}}{\Gamma_{s,1,i,j}} & \frac{\frac{c_{i,j}}{4}}{\Gamma_{s,1,i,j}} \\ \frac{\frac{c_{i,j}}{4}}{\Gamma_{s,2,i,j}} & \frac{\frac{c_{i,j}}{4}}{\Gamma_{s,2,i,j}} + \frac{\Gamma^2_{w,2,i,j}}{\Gamma^2_{s,2,i,j}} & \frac{\frac{c_{i,j}}{4}}{\Gamma_{s,2,i,j}} & \frac{\frac{c_{i,j}}{4}}{\Gamma_{s,2,i,j}} \\ \frac{\frac{c_{i,j}}{4}}{\Gamma_{s,3,i,j}} & \frac{\frac{c_{i,j}}{4}}{\Gamma_{s,3,i,j}} & \frac{\frac{c_{i,j}}{4}}{\Gamma_{s,3,i,j}} + \frac{\Gamma^2_{w,3,i,j}}{\Gamma^2_{s,3,i,j}} & \frac{\frac{c_{i,j}}{4}}{\Gamma_{s,3,i,j}} \\ \frac{\frac{c_{i,j}}{4}}{\Gamma_{s,4,i,j}} & \frac{\frac{c_{i,j}}{4}}{\Gamma_{s,4,i,j}} & \frac{\frac{c_{i,j}}{4}}{\Gamma_{s,4,i,j}} & \frac{\frac{c_{i,j}}{4}}{\Gamma_{s,4,i,j}} + \frac{\Gamma^2_{w,4,i,j}}{\Gamma^2_{s,4,i,j}} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}$$

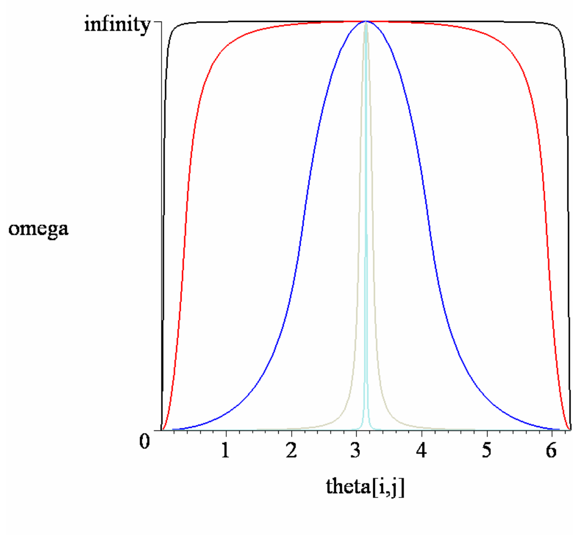
There are four solutions to this eigenvalue equation, but one of them is always zero. For $c = 0$, one of the remaining three eigenvalues is zero, and the third and fourth are plotted below for $\theta_{i,j} = \theta_{x,i,j} = \theta_{y,i,j}$ between 0 and 2π, and for various $h_{i,j} = h_{x,i,j} = h_{y,i,j}$ between 10^{-2} (black) and 10^{2} (turquoise):

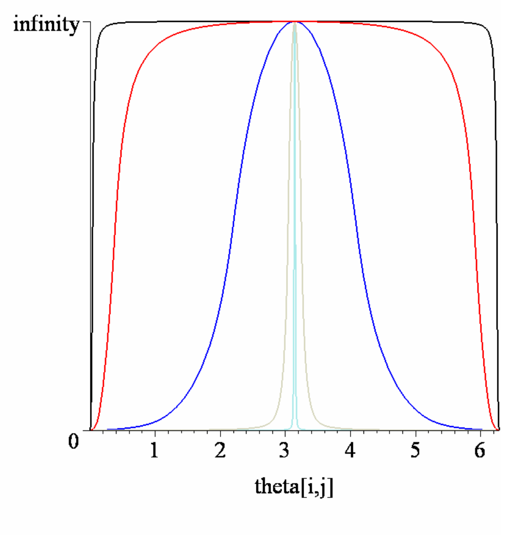




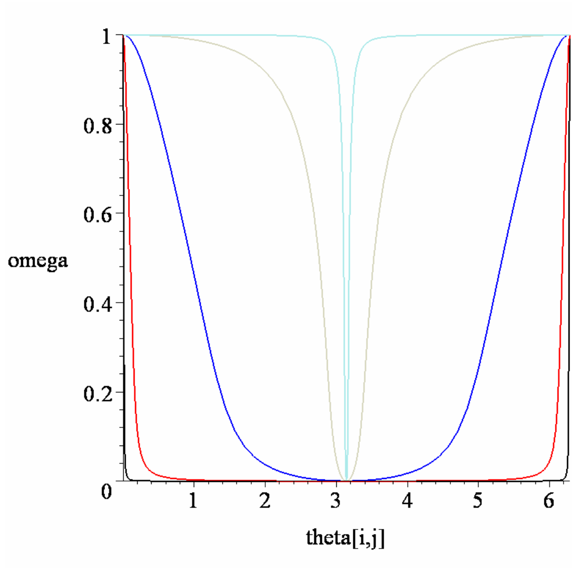
For $c = 0.5$, the three eigenvalues are plotted below for $\theta_{i,j} = \theta_{x,i,j} = \theta_{y,i,j}$ between 0 and 2π, and for various $h_{i,j} = h_{x,i,j} = h_{y,i,j}$ between 10^{-2} (black) and 10^{2} (turquoise):

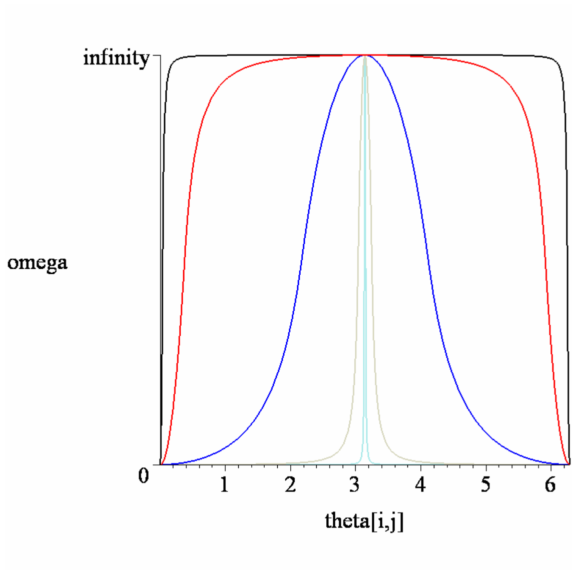


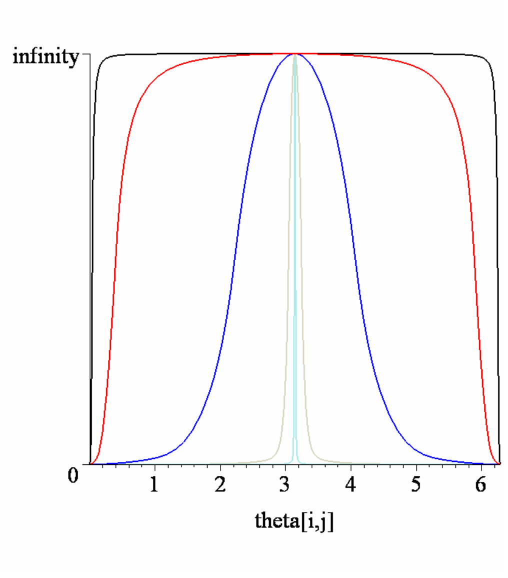




Finally, for c = 0.5, the three eigenvalues are plotted below for $\theta_{i,j} = \theta_{x,i,j} = \theta_{y,i,j}$ between 0 and 2π, and for various $h_{i,j} = h_{x,i,j} = h_{y,i,j}$ between 10^{-2} (black) and 10^{2} (turquoise):







It appears that first eigenvalue is always less than c. However, the second and third eigenvalues increase to infinity at $\theta_{i,j} = \pi$ regardless of the values of c and $h_{i,j}$. To show this, we will take the limit as $\theta_{i,j} \to \pi$ in the eigenvalue equation:

$$\omega \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} \left(\frac{\alpha_{2,1,1}+\beta_{2,1,1}}{\mu_{2,1}+\eta_{2,1}}\right)^2 & 0 & 0 & 0 \\ 0 & \left(\frac{\alpha_{2,2,2}+\beta_{2,2,2}}{\mu_{2,2}+\eta_{2,2}}\right)^2 & 0 & 0 \\ 0 & 0 & \left(\frac{\alpha_{2,3,3}+\beta_{2,3,3}}{\mu_{2,3}+\eta_{2,3}}\right)^2 & 0 \\ 0 & 0 & 0 & \left(\frac{\alpha_{2,4,4}+\beta_{2,4,4}}{\mu_{2,4}+\eta_{2,4}}\right)^2 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}$$

Substituting the values for μ, η, α, and β, we find:

$$\left(\frac{\alpha_{2,1,1}+\beta_{2,1,1}}{\mu_{2,1}+\eta_{2,1}}\right)^2 = 0$$

$$\left(\frac{\alpha_{2,2,2}+\beta_{2,2,2}}{\mu_{2,2}+\eta_{2,2}}\right)^2 = \infty$$

$$\left(\frac{\alpha_{2,3,3}+\beta_{2,3,3}}{\mu_{2,3}+\eta_{2,3}}\right)^2 = 0$$

$$\left(\frac{\alpha_{2,4,4}+\beta_{2,4,4}}{\mu_{2,4}+\eta_{2,4}}\right)^2 = \infty$$

Since this is a diagonal matrix, its eigenvalues are the values on the diagonal. Thus, when $\theta_{x,i,j} = \theta_{y,i,j} = \pi$, two of the iteration eigenvalues are zero, and two are infinity.

Thus, regardless of the values of $h_{i,j}$ or $c_{i,j}$, at least one of the iteration eigenvalues is infinite. Since the iteration eigenvalue is the maximum of the magnitudes of the eigenvalues, it becomes greater than c at $\theta_{i,j} = \pi$. Thus, the iteration will not converge for N > 0. For this reason, the next chapter will construct the CW-W_N method, which will not rely on iterative convergence other than source iteration. Since the maximum eigenvalue for source iteration is c, we know that, for $c < 1$, the CW-W_N method will converge. Thus, we will not include results from the S_N-W_N method in evaluating the use of wavelets to solve the transport equation.

CHAPTER X

CELL-WISE W_N METHOD

In this chapter we will derive an algorithm for solving the scaling function and wavelet function coefficients using the CW-W_N method. This method is similar to the S_N-W_N approach given in Chapter VIII, except that it solves for the wavelet coefficients in each cell directly using a matrix system solving routine (Gaussian elimination, Gauss-Seidel, conjugate gradient, etc.).

Derivation

In order to derive this method, we will first implement the diamond differencing spatial approximation, where the angular flux is defined in terms of the cell edge fluxes as

$$\psi_{i,j} = \tfrac{1}{2}\left[\psi_{i-1/2,j} + \psi_{i+1/2,j}\right] = \tfrac{1}{2}\left[\psi_{i,j-1/2} + \psi_{i,j+1/2}\right] \tag{10.1}$$

A schematic of a cell is shown in Figure III–1. For this approach, we will delay the contributions of the incident fluxes in solving for the scaling and wavelet function coefficients. The equation for the scaling function coefficients, in quadrant λ, is

$$\begin{aligned}&\left[\mu_{2,\lambda}\frac{\partial \psi^{s}_{2,\lambda,m}}{\partial x} + \eta_{2,\lambda}\frac{\partial \psi^{s}_{2,\lambda,m}}{\partial y} + \frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi^{s}_{2,\lambda,m}\right] \\ &\qquad + \sum_{r=2}^{N+1}\sum_{k=2^{r-2}(\lambda-1)+1}^{2^{r-2}\lambda}\left[\alpha_{r,k,\lambda}\frac{\partial \psi^{w}_{r,k,m}}{\partial x} + \beta_{r,k,\lambda}\frac{\partial \psi^{w}_{r,k,m}}{\partial y}\right] \\ &\qquad\qquad = \frac{\sigma_s(x,y)}{8\pi\sqrt{1-\mu_m^2}}\phi(x,y) + \frac{S_{ext}(x,y)}{8\pi\sqrt{1-\mu_m^2}}, \quad N \ge 0,\ m \ge 1\end{aligned} \tag{10.2}$$

and the equation for the wavelet coefficient *r*,*k*, in quadrant λ, is

$$\left[\alpha_{r,k,\lambda}\frac{\partial\psi^{s}_{2,\lambda,m}}{\partial x}+\beta_{r,k,\lambda}\frac{\partial\psi^{s}_{2,\lambda,m}}{\partial y}\right]+\sum_{\substack{n=2\\n\neq r}}^{N+1}\sum_{p=2^{n-2}(\lambda-1)+1}^{2^{n-2}\lambda}\left[A^{r,k}_{n,p}\frac{\partial\psi^{w}_{n,p,m}}{\partial x}+B^{r,k}_{n,p}\frac{\partial\psi^{w}_{n,p,m}}{\partial y}\right]$$
$$+\left[\mu_{r,k}\frac{\partial\psi^{w}_{r,k,m}}{\partial x}+\eta_{r,k}\frac{\partial\psi^{w}_{r,k,m}}{\partial y}+\frac{\sigma(x,y)}{\sqrt{1-\mu_m^2}}\psi^{w}_{r,k,m}\right]=0, \qquad (10.3)$$
$$r\geq 2,\ k=2^{r-2}(\lambda-1)+1\ldots 2^{r-2}\lambda,\ m\geq 1$$

with the constants defined in Chapter VI. The scalar flux is defined as

$$\phi(x,y)=\pi\sum_{\lambda=1}^{4}\sum_{n}\omega_n\psi^{s}_{2,\lambda,n}(x,y) \qquad (10.4)$$

and the system includes the boundary conditions

vaccum:	$\psi^{s}_{2,\lambda,m}(x,y)=0,\quad \psi^{w}_{r,k,m}(x,y)=0$
reflective:	$\psi^{s}_{2,\lambda,m}(x,y)=\psi^{s}_{2,\lambda',m'}(x,y),\quad \psi^{w}_{r,k,m}(x,y)=\psi^{w}_{r,k',m'}(x,y)$
incident:	$\psi^{s}_{2,\lambda,m}(x,y)=\tfrac{1}{2}f(x,y),\quad \psi^{w}_{r,k,m}(x,y)=0.$

Substituting the diamond difference approximation into this system, we find, for scaling coefficient λ,

$$\left[\frac{\sigma_{i,j}}{\sqrt{1-\mu_m^2}}+2\frac{|\mu_{2,\lambda}|}{\Delta x_{i,j}}+2\frac{|\eta_{2,\lambda}|}{\Delta y_{i,j}}\right]\psi^{s}_{\lambda,i,j,m}$$
$$+\sum_{r=2}^{N+1}\sum_{k=2^{r-2}(\lambda-1)+1}^{2^{r-2}\lambda}\left[2a\frac{\alpha_{r,k,\lambda}}{\Delta x_{i,j}}+2b\frac{\beta_{r,k,\lambda}}{\Delta y_{i,j}}\right]\psi^{w}_{r,k,i,j,m}=Q_{\lambda,i,j,m}, \qquad (10.5a)$$

where

$$Q_{\lambda,i,j,m} = \frac{\sigma_{s,i,j}}{8\pi\sqrt{1-\mu_m^2}}\phi_{i,j} + \frac{S_{ext,i,j}}{8\pi\sqrt{1-\mu_m^2}} + 2\frac{|\mu_{2,\lambda}|}{\Delta x_{i,j}}\psi^s_{\lambda,i\text{-inc},j,m} + 2\frac{|\eta_{2,\lambda}|}{\Delta y_{i,j}}\psi^s_{\lambda,i,j\text{-inc},m}$$
$$+\sum_{r=2}^{N+1}\sum_{k=1}^{2^{r-2}}\left[2a\frac{\alpha_{r,k,\lambda}}{\Delta x_{i,j}}\psi^w_{r,k,i\text{-inc},j,m} + 2b\frac{\beta_{r,k,\lambda}}{\Delta y_{i,j}}\psi^w_{r,k,i,j\text{-inc},m}\right], \tag{10.5b}$$

and

$$a = \begin{cases} 1, & \lambda = \{1,4\} \\ -1, & \lambda = \{2,3\} \end{cases} \qquad b = \begin{cases} 1, & \lambda = \{1,2\} \\ -1, & \lambda = \{3,4\} \end{cases}.$$

For wavelet coefficient r,k, in quadrant λ, the equation is

$$\left[\frac{\sigma_{i,j}}{\sqrt{1-\mu_m^2}} + 2\frac{|\mu_{r,k}|}{\Delta x_{i,j}} + 2\frac{|\eta_{r,k}|}{\Delta y_{i,j}}\right]\psi^w_{r,k,i,j,m} + \left[2a\frac{\alpha_{r,k,\lambda}}{\Delta x_{i,j}} + 2b\frac{\beta_{r,k,\lambda}}{\Delta y_{i,j}}\right]\psi^s_{\lambda,i,j,m}$$
$$+\sum_{\substack{n=2\\ n\neq r}}^{N+1}\sum_{p=2^{n-2}(\lambda-1)+1}^{2^{n-2}\lambda}\left[2a\frac{A^{r,k}_{n,p}}{\Delta x_{i,j}} + 2b\frac{B^{r,k}_{n,p}}{\Delta y_{i,j}}\right]\psi^w_{n,p,i,j,m} = Q_{r,k,\lambda,i,j,m}, \tag{10.6a}$$

where $\psi^w_{r,k,i\text{-inc},j,m}$ and $\psi^w_{r,k,i,j\text{-inc},m}$ are the incident wavelet function coefficients and

$$Q_{r,k,\lambda,i,j,m} = 2\frac{|\mu_{r,k}|}{\Delta x_{i,j}}\psi^w_{r,k,i\text{-inc},j,m} + 2\frac{|\eta_{r,k}|}{\Delta y_{i,j}}\psi^w_{r,k,i,j\text{-inc},m}$$
$$+2a\frac{\alpha_{r,k,\lambda}}{\Delta x_{i,j}}\psi^s_{\lambda,i\text{-inc},j,m} + 2b\frac{\beta_{r,k,\lambda}}{\Delta y_{i,j}}\psi^s_{\lambda,i,j\text{-inc},m} \tag{10.6b}$$
$$+\sum_{\substack{n=2\\ n\neq r}}^{N+1}\sum_{p=2^{n-2}(\lambda-1)+1}^{2^{n-2}\lambda}\left[2a\frac{A^{r,k}_{n,p}}{\Delta x_{i,j}}\psi^w_{n,p,i\text{-inc},j,m} + 2b\frac{B^{r,k}_{n,p}}{\Delta y_{i,j}}\psi^w_{n,p,i,j\text{-inc},m}\right].$$

The scalar flux is then

$$\phi_{i,j} = \pi\sum_{\lambda=1}^{4}\sum_{n}\omega_n\psi^s_{\lambda,i,j,n} \tag{10.7}$$

and the boundary conditions are

Table X–1: Reflecting azimuthal angles for CW-W_N method.

Face	Quadrant 1	Quadrant 2	Quadrant 3	Quadrant 4
Left	$\frac{1}{2}2^r-\lambda+1$	-	-	$\frac{3}{2}2^r-\lambda+1$
Right	-	$\frac{1}{2}2^r-\lambda+1$	$\frac{3}{2}2^r-\lambda+1$	-
Top	-	-	$2^r-\lambda+1$	$2^r-\lambda+1$
Bottom	$2^r-\lambda+1$	$2^r-\lambda+1$	-	-

$$\begin{aligned} &\text{vacuum:} && \psi^s_{\lambda,i,j,m}=0 \\ &\text{reflective:} && \psi^s_{\lambda,i,j,m}=\psi^s_{\lambda',i,j,m'} \\ &\text{incident:} && \psi^s_{\lambda,i,j,m}=\tfrac{1}{2}f_{i,j}. \end{aligned}$$

where *i* and *j* are located on the boundary. If we define the quadrants as

$$\begin{aligned} &\text{Quadrant 1:} && \mu>0,\ \eta>0 \\ &\text{Quadrant 2:} && \mu<0,\ \eta>0 \\ &\text{Quadrant 3:} && \mu<0,\ \eta<0 \\ &\text{Quadrant 4:} && \mu>0,\ \eta<0 \end{aligned}$$

then, for the reflecting boundary conditions, for the azimuthal angles, λ' are shown in Table X–1, and for the polar angles, $m'=M-m+1$. The boundary conditions for the wavelet coefficients are

$$\begin{aligned} &\text{vacuum:} && \psi^w_{r,k,i,j,m}=0 \\ &\text{reflective:} && \psi^w_{r,k,i,j,m}=\psi^w_{r,k',i,j,m'} \\ &\text{incident:} && \psi^w_{r,k,i,j,m}=0 \end{aligned}$$

Equations (10.1) and (10.2) can be written in matrix form as

$$\mathbf{M\psi}=\left(\mathbf{\Sigma}+\tfrac{2}{\Delta x_{i,j}}\mathbf{C_x}+\tfrac{2}{\Delta y_{i,j}}\mathbf{C_y}\right)\mathbf{\psi}=\mathbf{q} \tag{10.8}$$

where

$$\boldsymbol{\psi}=\begin{bmatrix}\psi^{s}_{\lambda,i,j,m}\\ \psi^{w}_{2,\lambda,i,j,m}\\ \psi^{w}_{3,2\lambda-1,i,j,m}\\ \vdots\\ \psi^{w}_{N+1,2^{N-1}\lambda,i,j,m}\end{bmatrix},\qquad \boldsymbol{\Sigma}=\begin{bmatrix}\frac{\sigma_{i,j}}{\sqrt{1-\mu_m^2}} &&&&\\ & \frac{\sigma_{i,j}}{\sqrt{1-\mu_m^2}} &&&\\ && \frac{\sigma_{i,j}}{\sqrt{1-\mu_m^2}} &&\\ &&& \ddots &\\ &&&& \frac{\sigma_{i,j}}{\sqrt{1-\mu_m^2}}\end{bmatrix},$$

$$\mathbf{C_x}=\begin{bmatrix}|\mu_{2,\lambda}| & a\alpha_{2,\lambda,\lambda} & a\alpha_{3,2\lambda-1,\lambda} & \cdots & a\alpha_{N+1,2^{N-1}\lambda,\lambda}\\ a\alpha_{2,\lambda,\lambda} & |\mu_{2,\lambda}| & aA^{2,\lambda}_{3,2\lambda-1} & \cdots & aA^{2,\lambda}_{N+1,2^{N-1}\lambda}\\ a\alpha_{3,2\lambda-1,\lambda} & aA^{2,\lambda}_{3,2\lambda-1} & |\mu_{3,2\lambda-1}| & \cdots & aA^{3,2\lambda-1}_{N+1,2^{N-1}\lambda}\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ a\alpha_{N+1,2^{N-1}\lambda,\lambda} & aA^{2,\lambda}_{N+1,2^{N-1}\lambda} & aA^{3,2\lambda-1}_{N+1,2^{N-1}\lambda} & \cdots & |\mu_{N+1,2^{N-1}\lambda}|\end{bmatrix},$$

$$\mathbf{C_y}=\begin{bmatrix}|\eta_{2,\lambda}| & b\beta_{2,\lambda,\lambda} & b\beta_{3,2\lambda-1,\lambda} & \cdots & b\beta_{N+1,2^{N-1}\lambda,\lambda}\\ b\beta_{2,\lambda,\lambda} & |\eta_{2,\lambda}| & bB^{2,\lambda}_{3,2\lambda-1} & \cdots & bB^{2,\lambda}_{N+1,2^{N-1}\lambda}\\ b\beta_{3,2\lambda-1,\lambda} & bB^{2,\lambda}_{3,2\lambda-1} & |\eta_{3,2\lambda-1}| & \cdots & bB^{3,2\lambda-1}_{N+1,2^{N-1}\lambda}\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ b\beta_{N+1,2^{N-1}\lambda,\lambda} & bB^{2,\lambda}_{N+1,2^{N-1}\lambda} & bB^{3,2\lambda-1}_{N+1,2^{N-1}\lambda} & \cdots & |\eta_{N+1,2^{N-1}\lambda}|\end{bmatrix}.$$

Also, we can calculate the source vector as

$$\mathbf{q}=\mathbf{s}+\tfrac{2}{\Delta x_{i,j}}\mathbf{C_x}\boldsymbol{\psi}_{i\text{-inc}}+\tfrac{2}{\Delta y_{i,j}}\mathbf{C_y}\boldsymbol{\psi}_{j\text{-inc}} \tag{10.9}$$

where

$$\mathbf{s}=\begin{bmatrix}\frac{\sigma_{s,i,j}}{8\pi\sqrt{1-\mu_m^2}}\phi_{i,j}+\frac{S_{ext,i,j}}{8\pi\sqrt{1-\mu_m^2}}\\ 0\\ 0\\ \vdots\\ 0\end{bmatrix},\qquad \boldsymbol{\psi}_{i\text{-inc}}=\begin{bmatrix}\psi^{s}_{\lambda,i\text{-inc},j,m}\\ \psi^{w}_{2,\lambda,i\text{-inc},j,m}\\ \psi^{w}_{3,2\lambda-1,i\text{-inc},j,m}\\ \vdots\\ \psi^{w}_{N+1,2^{N-1}\lambda,i\text{-inc},j,m}\end{bmatrix},\qquad \boldsymbol{\psi}_{j\text{-inc}}=\begin{bmatrix}\psi^{s}_{\lambda,i,j\text{-inc},m}\\ \psi^{w}_{2,\lambda,i,j\text{-inc},m}\\ \psi^{w}_{3,2\lambda-1,i,j\text{-inc},m}\\ \vdots\\ \psi^{w}_{N+1,2^{N-1}\lambda,i,j\text{-inc},m}\end{bmatrix}$$

The final consideration is convergence of the iteration. We will define the error vector as

$$e^{(k)} = \phi^{(k)} - \phi^{(k-1)}$$

Then, we will calculate an estimate to the spectral radius of the iteration as

$$\rho^{(k)} = \frac{\left\| e^{(k)} \right\|_2}{\left\| e^{(k-1)} \right\|_2}$$

Then, we will consider the method converged if

$$\left\| e^{(k)} \right\|_2 \leq \varepsilon \left(1 - \rho^{(k)} \right) \left\| \phi^{(k)} \right\|_2$$

The term $(1-\rho)$ will prevent false convergence.

Algorithm

Now, we will present an algorithm for implementing the CW-W_N method. The "linked list" data structure is used extensively in this program. Details about linked list implementation in Fortran 90 are given in Appendix D. The algorithm follows. The values after the descriptions are operation counts assuming an $N \times N$ spatial mesh, with M polar angles and an azimuthal resolution of r. The algorithm is shown in Figure X–1. In total, we have $O\left(N^2\right)$ operations for the input section, and $O\left(Mr2^{3r}N^2\right)$ operations on the thresholding iteration, with $O\left(M\,2^{3r}N^2\right)$ operations on all other source iterations. For our benchmark problems, this is about 40,000 operations for the input section, plus 146,800,640,000 operations on the thresholding iteration, and 20,971,520,000 operations per source iteration. This large number of operations is due to the program having to compute coupling to all other directions at each "discrete" direction (represented by a wavelet index), in addition to directly solving a matrix system for the wavelet coefficients in each cell. Because of this, the CW-W_N algorithm runs $2^{2r}-1$ times slower than discrete ordinates with no thresholding, and $(r-1)2^{2r}-1$ times slower than discrete ordinates for the thresholding iteration. Thus, in order to be competitive, wewould need to be able to threshold $(r-1)2^{2r}$ wavelet coefficients. Since there are only 2^r wavelet coefficients total per quadrant, this goal cannot be attained, and this algorithm cannot be faster than discrete ordinates. However, this algorithm will show that wavelets can be used to adapt the solution of the transport equation, although it is not the most efficient algorithm.

The input file is composed of regional inputs, so the problem simply has to be divided into regions. The code also has built-in material specifications, so that one only

```
1) Read input file and allocate memory, $O(N^2)$
2) Calculate polar angles and weights, $O(M^3)$
3) Begin source iteration, $k$
   a) Compute scattering source array, $O(N^2)$
   b) Initialize convergence variables, $O(N^2)$
   c) Loop over polar angles, $M$
      i) Loop over quadrant, 4
         (1) Set up sweep directions, $O(1)$
         (2) Construct y-boundary conditions, $O(2^{r-1}N)$
         (3) Loop over y-position index, $N$
            (a) Construct x-boundary conditions, $O(2^{r-1})$
            (b) Loop over x-position index, $N$
               (i)  Calculate scaling and coarsest wavelet coefficients from
                    incident fluxes, if thresholding, $O(2^r)$
               (ii) Loop over resolution level, if thresholding, $r$
```

```
                    1. Loop over wavelet index, $2^{r-2}$
                       a. Is $r$, $k$ a descendant of a wavelet in the list? , $O\left(2^{r-1}\right)$
                          i. Yes: Add wavelet to list, $O\left(2^{r-1}\right)$
                       b. Estimate wavelet coefficient using scaling function
                          and coarsest wavelet coefficients, $O\left(2^{2r}\right)$
                       c. Compute threshold criterion, $O\left(2^{r-1}\right)$
                       d. If wavelet coefficient estimate is less than threshold,
                          add wavelet to list, $O\left(2^{r-1}\right)$
                (iii) Set cell resolution level and count wavelet coefficients to be
                      calculated, $O\left(2^{r-1}\right)$
                (iv) Build $\mathbf{C}_x$, $\mathbf{C}_y$, and $\mathbf{\Sigma}$ matrices, $O\left(2^{2r}\right)$
                (v) Compute $\mathbf{M}$ matrix, $O\left(2^{r-1}\right)$
                (vi) Build $\mathbf{s}$, $\boldsymbol{\psi}_{i\text{-inc}}$, and $\boldsymbol{\psi}_{j\text{-inc}}$ vectors, $O\left(2^{r}\right)$
                (vii) Compute $\mathbf{q}$ vector, $O\left(2^{2r-2}\right)$
                (viii) Solve system $\mathbf{M}\boldsymbol{\psi} = \mathbf{q}$ for $\boldsymbol{\psi}$, $O\left(2^{3r-3}\right)$
                (ix) Save x-exiting coefficients, $O\left(2^{r+1}\right)$
                (x) Save y-exiting coefficients, $O\left(2^{r+1}\right)$
                (xi) Save x-reflecting conditions, $O\left(2^{r+1}\right)$
                (xii) Increment scalar flux, $O(1)$
                (xiii) Nullify lists, $O\left(2^{r-1}\right)$
             (c) Save y-reflecting conditions, $O\left(2^{r+1}N\right)$
          (4) Nullify y-exiting coefficient list, $O\left(2^{r+1}N\right)$
    d) Calculate error vectors and determine convergence, $O\left(N^2\right)$
4) Write to output files, $O\left(N^2\right)$
```

Figure X–1: Algorithm for implementing the CW-W_N method.

needs to read the number corresponding to a particular material. The available materials are:

1. Void ($\sigma = 0.0$, $\sigma_s = 0.0$)
2. Box1 Problem ($\sigma = 0.75$, $\sigma_s = 0.5$)
3. Box2 Problem ($\sigma = 0.375$, $\sigma_s = 0.25$)
4. Weak pure scatterer ($\sigma = 1.0$, $\sigma_s = 1.0$)
5. Weak absorber ($\sigma = 10.0$, $\sigma_s = 0.0$)
6. Strong absorber ($\sigma = 100.0$, $\sigma_s = 0.0$)
7. Pure scatterer ($\sigma = 100.0$, $\sigma_s = 100.0$)
8. Weak absorber ($\sigma = 1.0$, $\sigma_s = 0.0$)

Material indices (2) and (3) are used as described in Section 0. The source code for CW-W_N is in Appendix C, as well as an example input file. The identifier in the incident fluxes section of the input file specifies the type of boundary condition to be implemented (vac = vacuum, ref = reflecting, inc = incident). The boundary conditions are listed in the order left, right, top, and bottom. The incident flux values are listed to the right of the identifier. The iteration parameters are the maximum number of iterations and the convergence criterion. Finally, the angular resolution section contains the number of polar angles and the value of *r* in the azimuthal discretization. All codes described in this dissertation are written in Fortran 90.

CHAPTER XI

RESULTS

In this chapter, we will present the results using the CW-W_N method on the four benchmark problems first presented in Chapter IV. These results were obtained using the CW-W_N method algorithm, programmed in Fortran 90 using Digital Visual Fortran, version 6. The program was run on a Dell Dimension XPS with an Intel Pentium 4 processor with hyperthreading, clocked at 3.00 GHz, with 1 GB of memory. We will presents two errors in these results. The first is the L_2-norm of the relative errors between the scalar flux solution with no thresholding and the scalar flux solution of interest, i.e. if ϕ' is the scalar flux of interest and ϕ is the scalar flux with no thresholding, then the error presented is

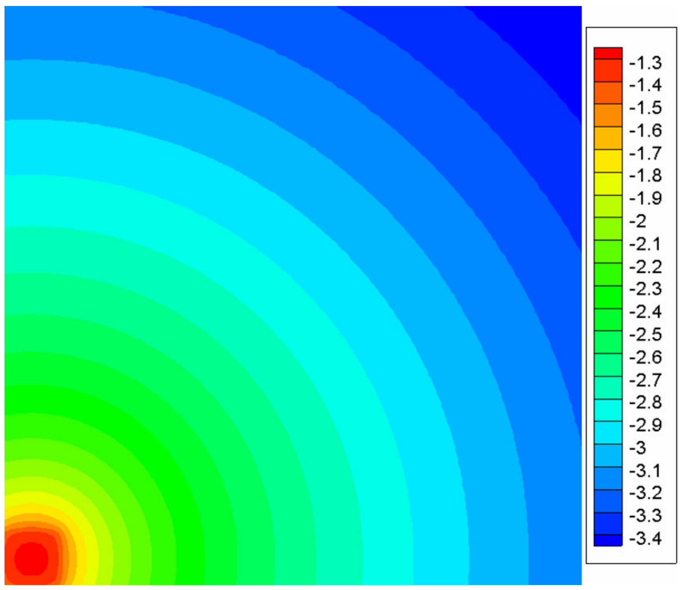


Figure XI–1: Benchmark solution of the test box problem.

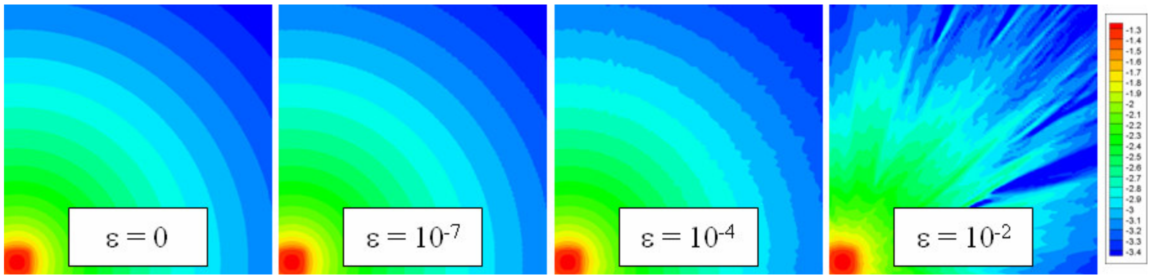


Figure XI–2: Solutions of the test box problem using CW-W_N.

$$\text{error} = \frac{\left\| \phi'(x,y) - \phi(x,y) \right\|_{L^2(x,y)}}{\left\| \phi(x,y) \right\|_{L^2(x,y)}}. \tag{11.1}$$

This error represents the overall accuracy of the method. The second value we will present is the root-mean-square (RMS) error. This error will give the thresholding's

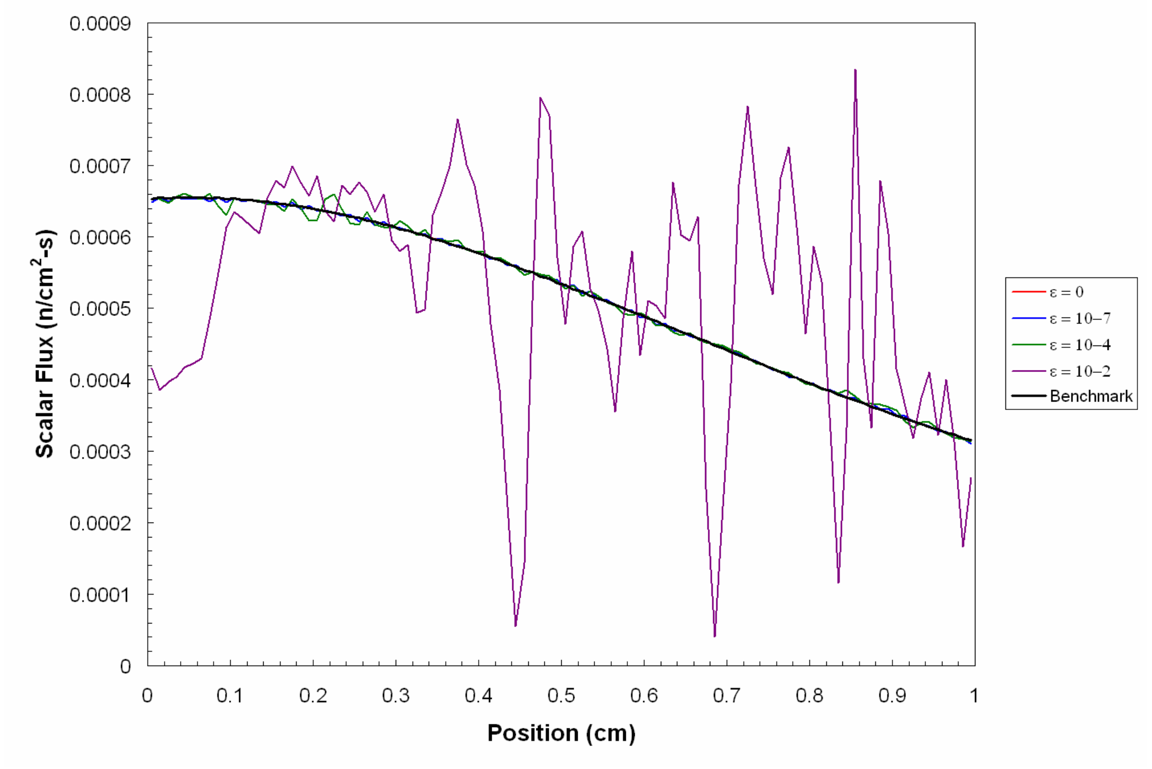


Figure XI–3: Solutions of the test box problem along top edge using CW-W_N.

effectiveness at dealing with ray effects, and is defined as

$$\mathrm{RMS} = \left[\overline{\frac{\phi'(x,y)-\phi(x,y)}{\phi(x,y)}^{2}} \right]^{1/2} , \tag{11.2}$$

where the bar represents the average over all the spatial domain.

The Box-in-Box Problems

We will consider three box-in-box type problems in this dissertation as benchmark problems. The latter two were taken from Morel.[2] The test box problem is shown in Figure IV–1. It consists of a square domain 1.0 cm in length on each side, with vacuum boundary conditions on all sides of the domain. The cross sections are constant throughout the domain, with $\sigma = 1.0$ cm^{-1}, and $\sigma_s = 0.0$ cm^{-1}. There is a source located in the bottom left corner of the domain that is square in shape with dimensions 0.1 cm on each side. This source is isotropic with an intensity of 1.0 particles per second.

Using the benchmark code S_N, a logarithmic contour plot of the scalar flux is shown in Figure XI–1, at resolution level R = 9 with a 100×100 cell spatial mesh. This was solved by computing a total of 20,480,000 unknowns. It took 18.58 seconds of CPU time.

Table XI–I: Results from the test box problem.

ε	Error	RMS	Unknowns	CPU Time (hr)
0	0.0001%	0.0006%	20,480,000	6.10
10^{-7}	0.110%	0.331%	4,483,469	1.21
10^{-4}	0.563%	1.27%	2,627,648	0.740
10^{-2}	5.98%	27.4%	466,423	0.141

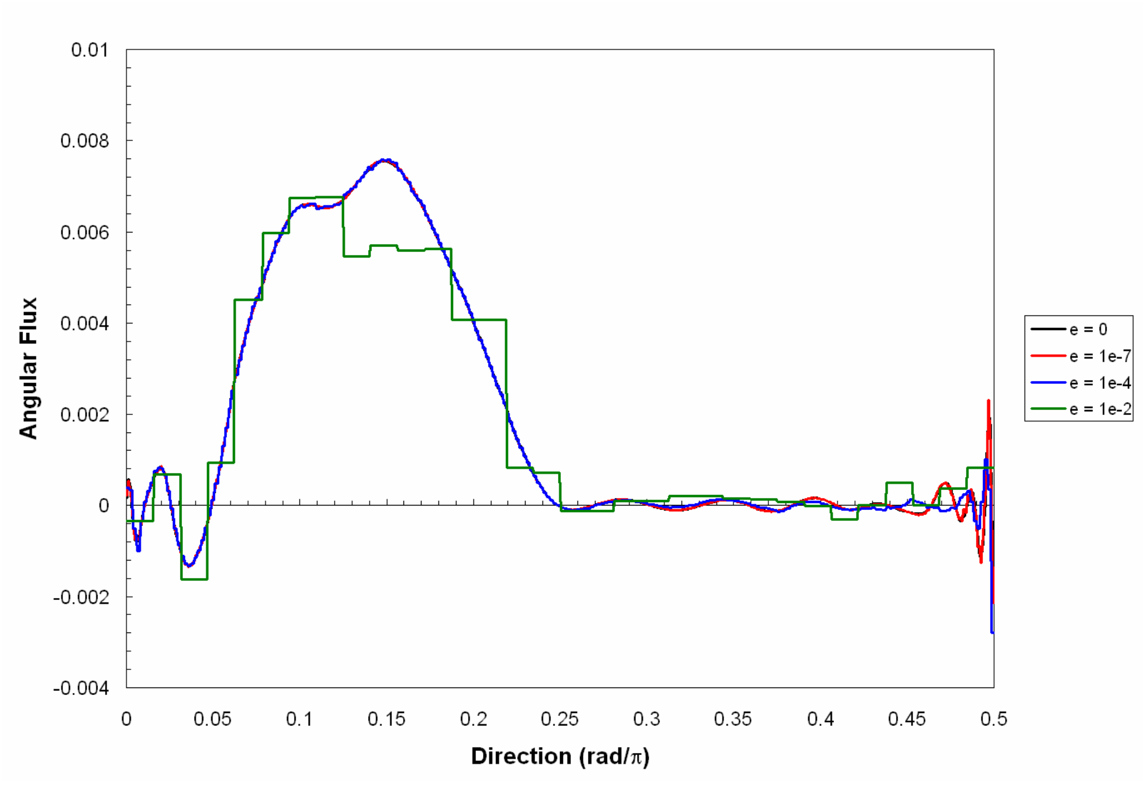


Figure IV–4: Angular flux at point (0.3 cm, 0.2 cm) in test box problem.

Solutions of this problem at resolution level R = 11 for several threshold fractions using CW-W_N are shown in Figure XI–2. Table XI–I shows the number of unknowns computed and the amount of CPU time used. Also shown are the relative and RMS errors when compared to the benchmark code. Figure XI–3 shows the scalar flux along the top edge of the domain. The solution at threshold fractions of zero and 10^{-7}

Table XI–II: Results from the first box-in-box problem.

ε	Error	RMS	Unknowns	Iterations	CPU Time (min)
0	0.00%	0.00%	1,280,000	31	26.6
10^{-7}	0.0102%	0.0206%	1,279,912	31	28.5
10^{-4}	0.460%	0.641%	1,223,681	31	24.6
10^{-2}	2.13%	3.19%	435,697	31	7.88

cannot be seen because they are so close to the benchmark solution.

The angular flux at a point located at coordinates (0.3 cm, 0.2 cm) is shown in Figure XI–4 at resolution level R = 11 for several threshold fractions. As the threshold fraction is increased, one can see how well the method approximates the shape of the peak given in Figure IV–2. One can also see, by looking at the plot corresponding to a threshold fraction of 10^{-2}, how some wavelet coefficients are thresholded in error, and how well the approximation of these coefficients is carried out.

It appears that the W_N method works well for this problem up to a threshold fraction of 10^{-4}. This thresholding level improves the number of unknowns computed by 87.2% and the computational time by 87.9%, with a relative error of 0.563% and an RMS error of 1.27%.

The next problem is shown in Figure IV–6, and will be referred to as the first box

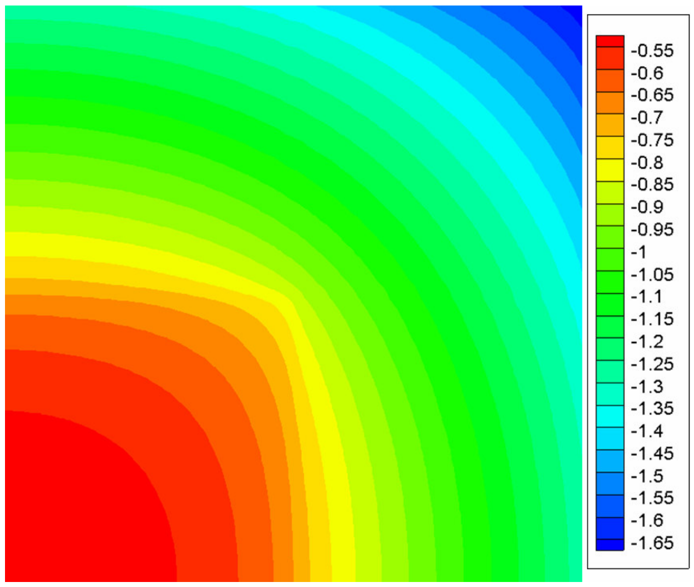


Figure XI–5: Benchmark solution of the first box-in-box problem.

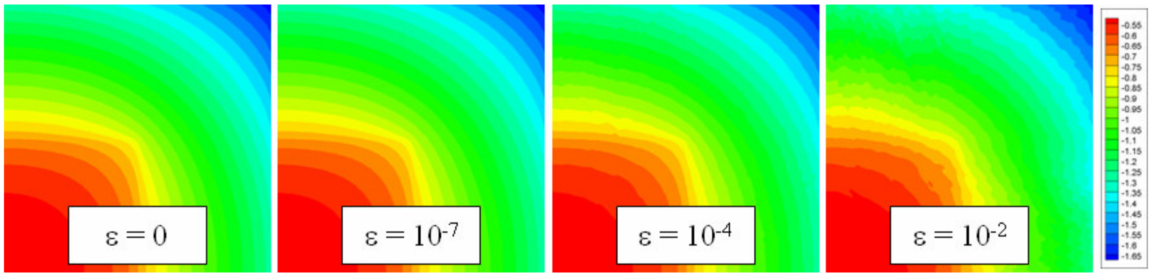


Figure XI–6: Solutions of the first box-in-box problem using CW-W_N.

problem. It resembles the classic ray-effect problem described by Lathrop[6] and has subsequently been used in many papers on ray-effect mitigation. It consists of a square domain 2.0 cm in length on each side, with reflective boundary conditions on the left and on the bottom of the domain, and vacuum boundary conditions on the top and right

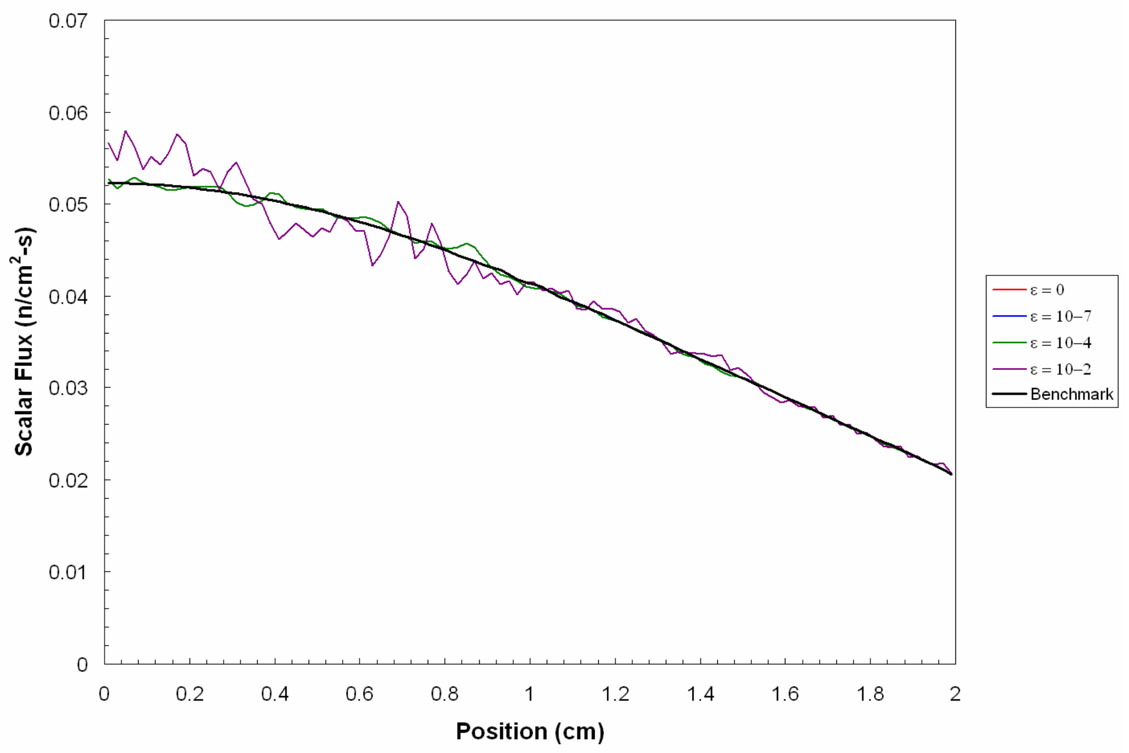


Figure XI–7: Solutions of the first box-in-box problem along top edge using CW-W_N.

sides. The cross sections are constant throughout the domain, with $\sigma = 0.75\ \text{cm}^{-1}$, $\sigma_s = 0.5\ \text{cm}^{-1}$, and isotropic scattering. There is a source located in the bottom left corner of the domain that is square in shape with dimensions 1.0 cm on each side. This source is isotropic with an intensity of 0.25 particles per second.

Using the benchmark code S_N, a logarithmic contour plot of the scalar flux is shown in Figure XI–5, at resolution level R = 5. This was solved in 31 iterations to a tolerance of 10^{-7}, computing 1,280,000 unknowns per iteration. It took 33.17 seconds of CPU time.

Solutions of this problem at resolution level R = 7 for several threshold fractions using CW-W_N are shown in Figure XI–6. Table XI–II shows the number of unknowns per iteration, the total number of iterations, and the amount of CPU time per iteration. Also shown are the relative and RMS errors when compared to the benchmark code.

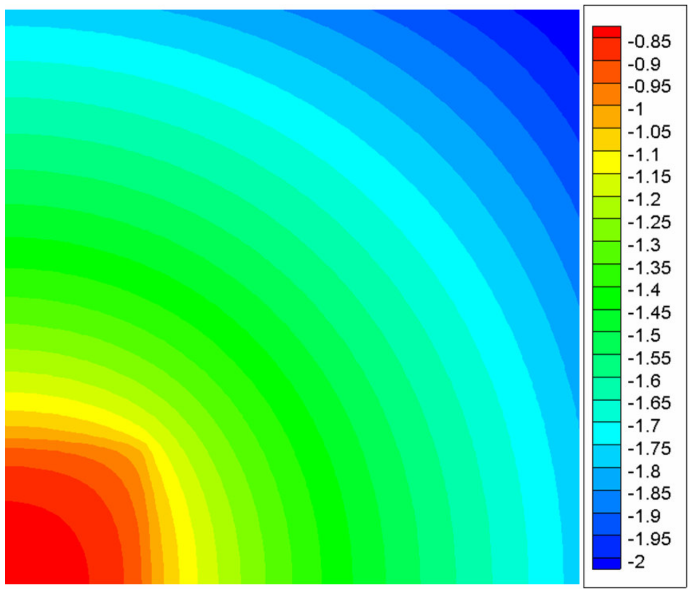


Figure XI–8: Benchmark solution of the second box-in-box problem.

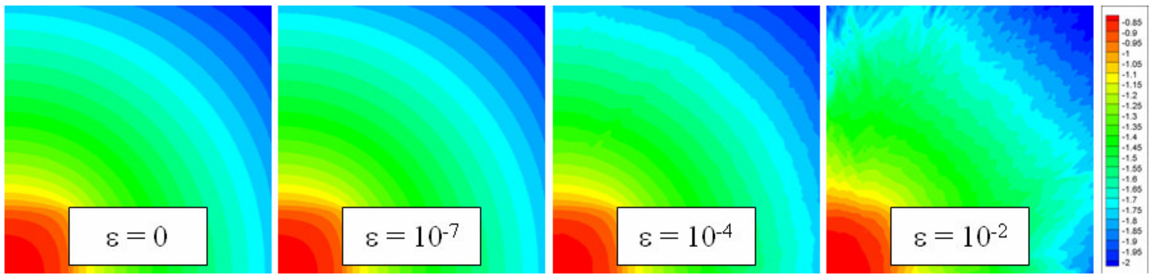


Figure XI–9: Solutions of the second box-in-box problem using CW-W_N.

Figure XI–7 shows the scalar flux along the top edge of the domain. The solution at threshold fractions of zero and 10^{-7} cannot be seen because they are so close to the benchmark solution.

It appears that the W_N method works well for this problem, even up to a threshold fraction of 10^{-2}. The maximum thresholding used improves the number of unknowns computed by 66.0% and the computational time by 70.4%, with a relative error of 2.13% and an RMS error of 3.19%.

The final box-in-box type problem is shown in Figure IV–9. It has been shown[2] that ray effects become more pronounced as a problem approaches a line source in a void. Thus, to approximate this problem, the second box-in-box problem has a smaller source spatially and a smaller total cross-section. The spatial domain is a square 2.0 cm in width. It has reflective boundary conditions on the left and bottom faces, and vacuum

Table XI–III: Results from the second box-in-box problem.

ε	Error	RMS	Unknowns	Iterations	CPU Time (hr)
0	0.0001%	0.0001%	5,120,000	24	4.60
10^{-7}	0.0238%	0.0515%	5,115,559	24	4.49
10^{-4}	0.609%	0.850%	3,206,465	24	2.38
10^{-2}	4.46%	7.55%	413,315	24	0.715

boundary conditions on the top and right faces. The cross-sections are constant throughout the domain with $\sigma = 0.375\ \text{cm}^{-1}$ and $\sigma_s = 0.25\ \text{cm}^{-1}$. The source is a square located in the bottom left corner, and it is 0.5 cm in width. The source is isotropic with an intensity of 0.25 particles/sec.

Using the benchmark code S_N, a logarithmic contour plot of the scalar flux is shown in Figure XI–8, at resolution level R = 7. This was solved in 23 iterations to a tolerance of 10^{-7}, computing 5,120,000 unknowns per iteration. It took 99.75 seconds of CPU time.

Solutions of this problem at resolution level R = 9 for several threshold fractions

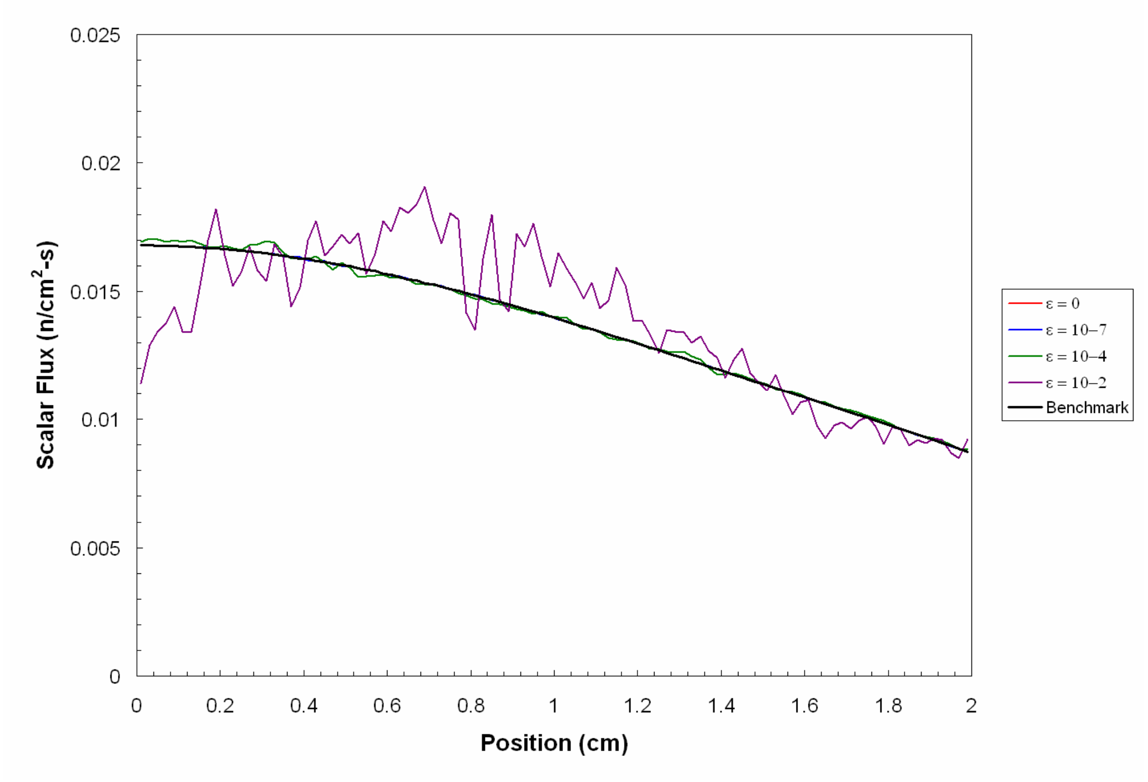


Figure XI–10: Solutions of the second box-in-box problem along top edge using CW-W_N.

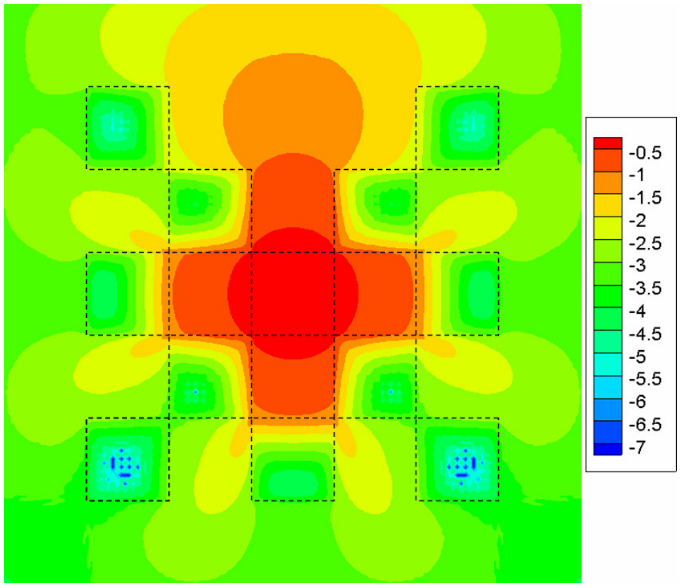


Figure XI–11: Benchmark solution of the lattice problem.

using CW-W_N are shown in Figure XI–9. Table XI–III shows the number of unknowns per iteration, the total number of iterations, and the amount of CPU time taken. Also shown are the relative and RMS errors when compared to the benchmark code. Figure XI–10 shows the scalar flux along the top edge of the domain. The solution at threshold fractions of zero and 10^{-7} cannot be seen because they are so close to the benchmark solution.

It appears that the W_N method works well for this problem up to a threshold

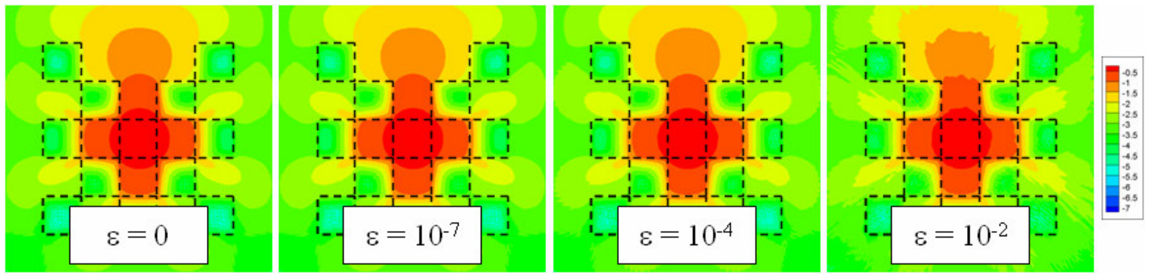


Figure XI–12: Solutions of the lattice problem using CW-W_N.

fraction of 10^{-4}. This threshold fraction improves the number of unknowns computed by 37.4% and the computational time by 48.3%, with a relative error of 0.609% and an RMS error of 0.850%.

The Brunner Problems

Two other, and more realistic, examples that demonstrate this phenomenon were created by Thomas Brunner[20] of Sandia National Laboratory. The first example, referred to as the “lattice problem,” is shown in Figure IV–12. The domain is square and the lattice is composed of squares of width 1.0 cm. The lattice is made up of squares (A) of pure absorbers, with $\sigma = 10\ \text{cm}^{-1}$. The rest of the domain (B) is purely scattering with

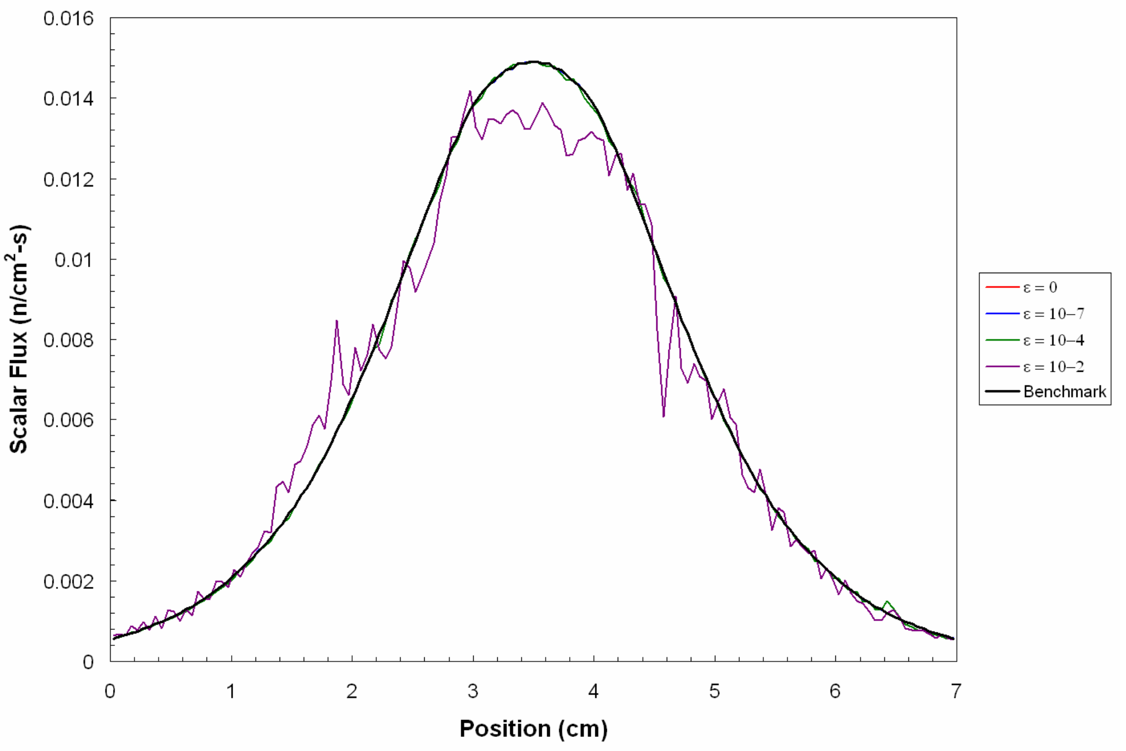


Figure XI–13: Solutions of the lattice problem along top edge using CW-W_N.

Table XI–IV: Results from the lattice problem.

ε	Error	RMS	Unknowns	Iterations	CPU Time (hr)
0	0.0042%	298%	5,017,600	39	3.77
10^{-7}	0.0069%	298%	5,013,032	39	3.77
10^{-4}	0.0863%	436%	4,691,299	39	2.90
10^{-2}	1.429%	2180%	1,832,146	39	1.16

$\sigma = \sigma_s = 1$ cm^{-1}. The source is in the center square of the lattice and is isotropic with an intensity of 1.0 particles/sec. The source square is also of material A.

Using the benchmark code S_N, a logarithmic contour plot of the scalar flux is shown in Figure XI–11, at resolution level R = 6. This was solved in 39 iterations to a tolerance of 10^{-7}, computing 5,017,600 unknowns per iteration. It took 167.88 seconds of CPU time.

Solutions of this problem at resolution level R = 8 for several threshold fractions using CW-W_N are shown in Figure XI–12. Table XI–IV shows the number of unknowns

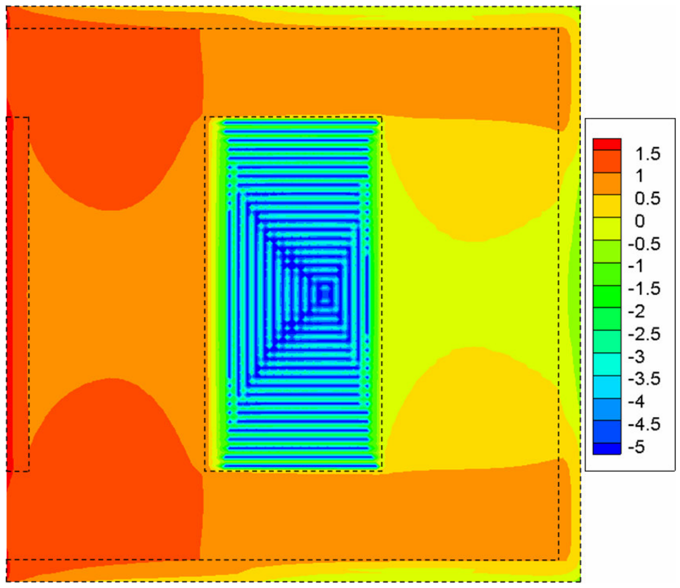


Figure XI–14: Benchmark solution of the Hohlraum problem.

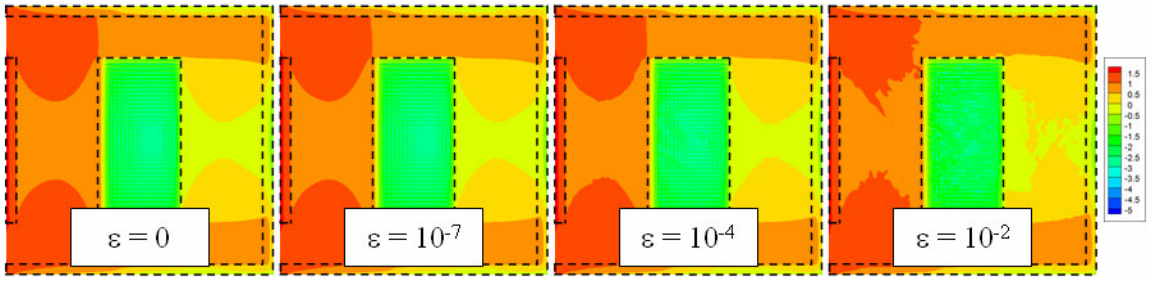


Figure XI–15: Solutions of the Hohlraum problem using CW-W_N.

per iteration, the total number of iterations, and the amount of CPU time per iteration. Also shown are the relative and RMS errors when compared to the benchmark code. Figure XI–13 shows the scalar flux along the top edge of the domain. The solution at threshold fractions of zero and 10^{-7} cannot be seen because they are so close to the benchmark solution.

It appears that the W_N method works well for this problem up to a threshold fraction of 10^{-4}. This thresholding level improves the number of unknowns computed by 6.50% and the computational time by 23.1%, with a relative error of 0.0863% and an RMS error of 436%. The RMS error is high due to the flux calculations inside the absorbers.

The second example, referred to as the "Hohlraum problem," is shown in Figure IV–15. The domain is square with width of 13 mm. The walls of the hohlraum are

Table XI–V: Results from the Hohlraum problem.

ε	Error	RMS	Unknowns	CPU Time (hr)
0	0.244%	2420%	8,652,800	23.3
10^{-7}	0.251%	2370%	8,548,112	21.7
10^{-4}	0.458%	2310%	7,113,066	9.03
10^{-2}	3.35%	4870%	2,434,625	3.74

Table XI–VI: Relative errors about the target for the Hohlraum problem.

ε	Error	RMS
0	0.776%	5.49%
10^{-7}	0.774%	5.48%
10^{-4}	0.861%	5.41%
10^{-2}	5.72%	8.33%

purely scattering, with $\sigma = \sigma_s = 100\ \text{cm}^{-1}$, and the target (the large, centrally located absorber) is a pure absorber with $\sigma = 100\ \text{cm}^{-1}$. The rest of the domain is a void. There is an isotropic incident flux with intensity 1.0 particles/sec along the entire length of the left wall.

Using the benchmark code S_N, a logarithmic contour plot of the scalar flux is

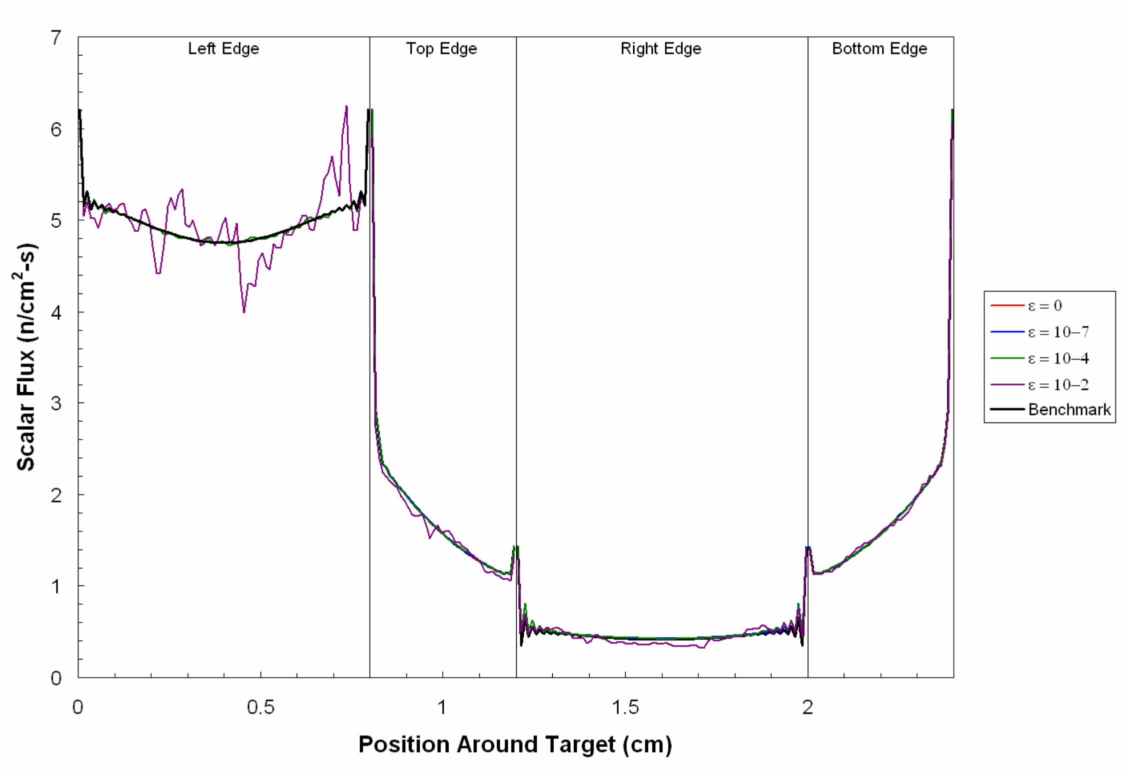


Figure XI–16: Solutions of the Hohlraum problem around the target using CW-W_N.

shown in Figure XI–14, at resolution level R = 9. This was solved to a tolerance of 10^{-7}, computing 34,611,200 unknowns. It took 40.23 seconds of CPU time.

Solutions of this problem at resolution level R = 9 for several threshold fractions using CW-W_N are shown in Figure X–15. Table XI–V shows the number of unknowns and the amount of CPU time used. Also shown are the relative and RMS errors when compared to the benchmark code. Figure XI–16 shows the scalar flux around the target area.

It appears that the W_N method may work well for this problem up to a threshold fraction of 10^{-4}. This thresholding level improves the number of unknowns computed by 17.8% and the computational time by 61.2%, with a relative error of 0.458% and an RMS error of 2310%. The RMS error is high due to the larege errors in the flux within the target. However, since these values are small, this error is misleading. Table XI–VI shows the relative errors around the perimeter of the target area for each resolution level. For a threshold fraction of 10^{-4}, the relative error is 0.861% and the RMS error is 5.41%.

CHAPTER XII

CONCLUSIONS

In this dissertation, we have shown that wavelet expansions of the angular flux may indeed be used to adaptively solve the neutral particle transport equation. We derived two methods in order to solve this problem.

The S_N-W_N method makes most sense from a computational sense. That is, if the S_N-W_N approach works, then solving the scaling coefficient and wavelet coefficient equations will be equally computationally costly as a standard discrete ordinates method. However, as presented in this dissertation, the S_N-W_N method does not work. The behavior of these equations resembles the behavior of the P_N equations. One of the iterative methods used to solved the P_N equations is the Krylov iteration. This iterative method takes the place of source iteration, and may be able to defeat the problems encountered in developing the S_N-W_N method. In addition, a more efficient thresholding method will need to be developed; one that does not require all wavelet coefficients to be thresholded in order to be computationally competitive.

Another way of possibly improving the performance of W_N methods in general would be to use the benchmark S_N code for the iterations before thresholding, transforming those coefficients into the scaling plus wavelet coefficients, then continuing with thresholding and convergence. This could result in a small improvement in the overall performance of the method. For a typical problem, one could expect a 1-10% decrease in the computational time needed for the S_N-W_N method, as written. However, if one could improve the S_N-W_N method in sucha a awy that it ran as quickly as the benchmark without thresholding, then the above argument would not hold, since the wavelet method would always be faster.

The CW-W_N method does work, and demonstrates how adaptivity using wavelets does indeed decrease the number of unknowns required to adequately solve a given problem to within a particular error. However, the computational costs in inverting the matrices are prohibitively high. The CW-W_N runs $(r-1)2^{2r}-1$ times slower than discrete ordinates. Thus, in order to be competitive with a transport sweep, we will need to threshold $(r-1)2^{2r}$ coefficients. Since there are only 2^r wavelet coefficients per quadrant, this is not possible, and the CW-W_N algorithm is only useful to examine the intricacies of using wavelets to adaptively solve the transport equation, and not as a production level method. The best solution would be to efficiently implement the S_N-W_N method.

The final consideration is that of determining the “best” threshold fraction to use for a given problem. For the problems we have presented, a trial-and-error method was used. This appears to be the only way of determining the “best” threshold fraction. We have not been able to find any other successful method of determining the threshold fraction value *a priori*.

So, in conclusion, our recommendation for future work would be to implement the S_N-W_N method with a Krylov iteration instead of a source iteration. We believe that this combination would be able to adequately resolve the solution while adapting significantly in angle. In addition, for 2-D problems, a higher order wavelet basis, such as a Chui-Wang or Daubechies basis, should be employed. For 3-D problems, wavelets on the sphere look promising. For more complicated scenarios, i.e. time and energy dependent problems, the only way of determining the best threshold fraction appears to be trial-and-error.

We believe that we understand more about the nature of wavelet behavior in transport problems, as well as the influence of wavelets on the convergence of source iteration. Thresholding did not change the number of iterations to convergence in source iteration. In short, more research is needed in the use of wavelets in solving the neutral particle transport equation, but these preliminary results are promising.

REFERENCES

1. Y. YOU, Wavelet-Based Methods for Numerical Solution of Differential Equations, Ph.D. Dissertation, Texas A&M University (1995).

2. J. MOREL, T. A. WAREING, R. LOWRIE, and D. PARSONS, "Analysis of Ray-Effect Mitigation Techniques," *Nucl. Sci. Eng.*, **144**, 1 (2003).

3. E. M. GELBARD, Discussion Panel on Outstanding Problems in Reactor Mathematics, *Trans. Am. Nucl. Soc.*, **8**, 231 (1965).

4. G. HANSEN, Personal Communication to K. D. Lathrop (1965).

5. J. DAVIS and S. KAPLAN, "Transport Synthesis," *Trans. Am. Nucl. Soc.*, **8**, 509 (1965).

6. K. D. LATHROP, "Ray Effects in Discrete Ordinates Equations," *Nucl. Sci. Eng.*, **32**, 357 (1968).

7. S. KAPLAN, "A New Derivation of Discrete Ordinate Approximations," *Nucl. Sci. Eng.*, **34**, 76 (1968).

8. L. B. MILLER and R. E. JARKA, "The Importance of Discrete Ordinate Ray Effects in Fast Reactor Design Studies," *Trans. Am. Nucl. Soc.*, **13**, 629 (1970).

9. M. NATELSON, "Variational Derivation of Discrete Ordinate-Like Approximations," *Nucl. Soc. Eng.*, **43**, 141 (1971).

10. K. D. LATHROP, "Remedies for Ray Effects," *Nucl. Sci. Eng.*, **45**, 255 (1971).

11. K. D. LATHROP, "Elimination of Ray Effects in Curved Geometries," *Trans. Am. Nucl. Soc.*, **15**, 272 (1972).

12. J. JUNG, H. CHIJIWA, K. KOBAYASHI, and H. NISHIHARA, "Discrete Ordinate Neutron Transport Equation Equivalent to P_1 Approximation," *Nucl. Sci. Eng.*, **49**, 1 (1972).

13. Wm. H. REED, "Spherical Harmonic Solutions of the Neutron Transport Equation from Discrete Ordinates Codes," *Nucl. Sci. Eng.*, **49**, 10 (1972).

14. Wm. E. MILLER, Jr., E. E. LEWIS, and E. C. ROSSOW, "Application of Phase-Space Finite-Elements to 2-Dimensional Neutron Transport Equation in X-Y Geometry," *Nucl. Sci. Eng.*, **52**, 12 (1973).

15. L. L. BRIGGS, W. F. MILLER, and E. E. LEWIS, "Ray-Effect Mitigation in Discrete Ordinate-Like Angular Finite Element Approximations in Neutron Transport," *Nucl. Sci. Eng.*, **57**, 205 (1975).

16. T. J. SEED, "Application of Walsh Functions to Neutron Transport Problems," *Nucl. Sci. Eng.*, **60**, 337 (1976).

17. W. F. MILLER and Wm. H. REED, "Ray-Effect Mitigation Methods for Two-Dimensional Neutron Transport Theory," *Nucl. Sci. Eng.*, **62**, 391 (1977).

18. J. K. FLETCHER, "The Solution of the Multigroup Neutron-Transport Equation Using Spherical-Harmonics," *Nucl. Sci. Eng.,* **84**, 33 (1983).

19. P. N. BROWN, B. CHANG, and U. R. HANEBUTTE, "Spherical Harmonic Solutions of the Boltzmann Transport Equation via Discrete Ordinates," *Prog. Nucl. Energy*, **39**, 263 (2001).

20. T. BRUNNER, *Forms of Approximate Radiation Transport*, Technical Report SAND2002-1778, Sandia National Laboratories (2002).

21. B. JAWERTH and W. SWELDENS, "An Overview of Wavelet Based Multiresolution Analyses," *SIAM Review*, **36**, 377 (1994).

22. J. O. STRÖMBERG, "A Modified Franklin System and Higher Order Spline Systems on R^n as Unconditional Bases for Hardy Spaces," in Conference on Harmonic Analysis in Honor of Antoni Zygmund, B. et al., ed., vol. II, Univ. of Chicago Press , Chicago, 1981, pp. 475-494.

23. I. DAUBECHIES, "Orthonormal Bases of Compactly Supported Wavelets," *Comm. Our Appl. Math.*, **41**, 909 (1998).

24. Y. MADAY, V. PERRIER, and J. C. RAVEL, "Dynamical Adaptivity Using Wavelet Basis for the Approximation of Partial Differential Equations," *C. R. Acad. Sci. Série I*, **312**, 405 (1991).

25. Y. MADAY and J. C. RAVEL, "Adaptivity Using Wavelets: Boundary Conditions and Multidimensions," *C. R. Acad. Sci. Série I*, **315**, 85 (1992).

26. M. HOLMSTROM, "Solving Hyperbolic PDEs Using Interpolating Wavelets," *SIAM J. Sci. Comp.*, **21**, 405 (1999).

27. A. HAAR, "Zur Theorie der Orthogonalen Funktionensysteme," *Math. Ann.*, **69**, 331 (1910).

28. I. DAUBECHIES, "Orthonormal Bases of Compactly Supported Wavelets," *Comm. Pure Appl. Math.*, **41**, 909 (1988).

29. J. P. ANTOINE and P. VANDERGHEYNST, “Wavelets on the 2-Sphere: A Group–Theoretical Approach,” *Appl. Comp. Harm. Anal.*, **7**, 262 (1999).

30. J. P. ANTOINE, L. DEMANET, L. JACQUES, and P. VANDERGHEYNST, “Wavelets on the Sphere: Implementation and Approximations,” *Appl. Comp. Harm. Anal.*, **13**, 177 (2002).

APPENDIX A

The following is the source code for S_N:

```
Program S_N

Implicit None

Real :: te, ta(2), etime

! Region Declarations

Integer :: X_Regs, Y_Regs, index

! Declare source and material index arrays

Real*8, Allocatable :: Source(:,:)
Integer, Allocatable :: Mat_Regs(:,:)

! Declare spatial grid arrays

Real*8, Allocatable :: X_Cells_Regs(:,:), Y_Cells_Regs(:,:), Del_X_Regs(:,:)
Real*8, Allocatable :: Del_Y_Regs(:,:)

Integer :: X_Cells, Y_Cells

! Declare cell-wise arrays

Real*8, Allocatable :: Sigma_Tot(:,:), Sigma_Scat(:,:), Delta_X(:,:), Delta_Y(:,:)
Real*8, Allocatable :: Psi_Old(:,:,:,:), Psi_Left(:,:,:), Psi_Right(:,:,:)
Real*8, Allocatable :: Psi_Top(:,:,:), Psi_Bottom(:,:,:), Phi(:,:), Phi_Old(:,:), Q(:,:)
Real*8, Allocatable :: Psi_y_Inc(:), Fixed(:,:), Phi_Old_2(:,:)

Integer, Allocatable :: Material(:,:), Region_X(:,:), Region_Y(:,:), X_Left(:,:)
Integer, Allocatable :: X_Right(:,:), Y_Top(:,:), Y_Bottom(:,:)

Real*8 :: Psi_x_Inc, Norm_1, Norm_2, Norm_3, Sum_1, Sum_2, Rho, Psi, Min_Phi

! Declare boundary conditions

Real*8, Allocatable :: Bound_Regs(:,:,:), Boundary(:,:,:)

! Declare iteration parameters

Integer :: Maxit, t, Cnt, Cnt_Tot
Real*8 :: Eps, Err

! Declare angular arrays

Integer :: Res, Ang_Res, Quad, Polar, pol
Real*8, Allocatable :: Theta(:), mu(:), eta(:), xi(:), w(:)

! Declare other variables

Integer :: i, j, k, l, m, azi_left, azi_right, azi_top, azi_bottom, pol_opp
Integer :: I_Start, I_Finish, I_Step, J_Start, J_Finish, J_Step
Real*8 :: Pi, x, y
Logical :: scat, detail

Character(10) :: in_file
Character(3) :: bc_left, bc_right, bc_top, bc_bottom
Character(2) :: out_res
```

```
Pi = 4d0*datan(1d0)

!*****************************************************************************************
!
!   Input Section
!
!*****************************************************************************************

Write (*,*) "Creating Problem Domain..."

detail = .true.

Write (*,*) "Name of input file:"
Read (*,*) in_file

Open (Unit = 10, File = in_file, Status="old")

! Read number of regions in each direction

Read (10,*)
Read (10,*)
Read (10,*) X_Regs, Y_Regs

! Define source strength and cross sections for each material and region

Allocate (Source(X_Regs,Y_Regs), Mat_Regs(X_Regs,Y_Regs))

Read (10,*)
Read (10,*)
Read (10,*)
Do i = Y_Regs,1,-1
   Read (10,*) Source(1:X_Regs,i)
Enddo

Read (10,*)
Read (10,*)
Read (10,*)
Do i = Y_Regs,1,-1
   Read (10,*) Mat_Regs(1:X_Regs,i)
Enddo

! Read number of cells in each region in each direction

Allocate (X_Cells_Regs(X_Regs,Y_Regs), Y_Cells_Regs(X_Regs,Y_Regs))
Allocate (Del_X_Regs(X_Regs,Y_Regs), Del_Y_Regs(X_Regs,Y_Regs))

Read (10,*)
Read (10,*)
Read (10,*)
Do i = Y_Regs,1,-1
   Read (10,*) X_Cells_Regs(1:X_Regs,i)
Enddo

Read (10,*)
Read (10,*)
Read (10,*)
Do i = Y_Regs,1,-1
   Read (10,*) Y_Cells_Regs(1:X_Regs,i)
Enddo

Read (10,*)
Read (10,*)
Read (10,*)
Do i = Y_Regs,1,-1
   Read (10,*) Del_X_Regs(1:X_Regs,i)
Enddo
```

```
Read (10,*)
Read (10,*)
Read (10,*)
Do i = Y_Regs,1,-1
    Read (10,*) Del_Y_Regs(1:X_Regs,i)
Enddo

! Allocate cell-wise arrays

X_Cells = 0
Do i = 1,X_Regs
    X_Cells = X_Cells + X_Cells_Regs(i,1)
Enddo

Y_Cells = 0
Do i = 1,Y_Regs
    Y_Cells = Y_Cells + Y_Cells_Regs(1,i)
Enddo

Allocate (Sigma_Tot(X_Cells,Y_Cells), Sigma_Scat(X_Cells,Y_Cells))
Allocate (Delta_X(X_Cells,Y_Cells), Delta_Y(X_Cells,Y_Cells), Fixed(X_Cells,Y_Cells))
Allocate (Phi(X_Cells,Y_Cells), Phi_Old(X_Cells,Y_Cells), Phi_Old_2(X_Cells,Y_Cells))
Allocate (Q(X_Cells,Y_Cells), Material(X_Cells,Y_Cells), Psi_y_Inc(X_Cells))
Allocate (Region_X(X_Cells,Y_Cells), Region_Y(X_Cells,Y_Cells), X_Left(X_Regs,Y_Regs))
Allocate (X_Right(X_Regs,Y_Regs), Y_Top(X_Regs,Y_Regs), Y_Bottom(X_Regs,Y_Regs))

! Define region indices

Do i = 1,X_Regs
    Do j = 1,Y_Regs

        X_Left(i,j) = 0.
        Do k = 1,i-1
            X_Left(i,j) = X_Left(i,j) + X_Cells_Regs(k,j)
        Enddo

        Y_Bottom(i,j) = 0.
        Do k = 1,j-1
            Y_Bottom(i,j) = Y_Bottom(i,j) + Y_Cells_Regs(i,k)
        Enddo

        X_Right(i,j) = X_Left(i,j) + X_Cells_Regs(i,j)

        Y_Top(i,j) = Y_Bottom(i,j) + Y_Cells_Regs(i,j)

        Do k = 1,X_Cells
            If ((k.gt.X_Left(i,j)).and.(k.le.X_Right(i,j))) Region_X(k,1:Y_Cells) = i
        Enddo

        Do k = 1,Y_Cells
            If ((k.gt.Y_Bottom(i,j)).and.(k.le.Y_Top(i,j))) Region_Y(1:X_Cells,k) = j
        Enddo

    Enddo
Enddo

Do i = 1,X_Regs
    Do j = 1,Y_Regs

    Do k = 1,X_Cells
        If ((k.gt.X_Left(i,j)).and.(k.le.X_Right(i,j))) Then
            Do l = 1,Y_Cells
                Region_X(k,l) = i
            Enddo
        Endif
    Enddo
    Do k=1,Y_Cells
        If ((k.gt.Y_Bottom(i,j)).and.(k.le.Y_Top(i,j))) Then
            Do l=1,X_Cells
                Region_Y(l,k) = j
            Enddo
        Endif
    Enddo

    Enddo
Enddo

! Read boundary conditions

Allocate (Bound_Regs(4,X_Regs,Y_Regs), Boundary(4,X_Cells,Y_Cells))

Bound_Regs = 0d0
Boundary = 0d0

Read (10,*)
Read (10,*)
Read (10,*)
Read (10,*) bc_left, Bound_Regs(1,1,1:Y_Regs)
Read (10,*) bc_right, Bound_Regs(2,X_Regs,1:Y_Regs)
Read (10,*) bc_top, Bound_Regs(3,1:X_Regs,1)
Read (10,*) bc_bottom, Bound_Regs(4,1:X_Regs,Y_Regs)

If (bc_left.eq."vac") Bound_Regs(1,:,:) = 0d0
If (bc_right.eq."vac") Bound_Regs(2,:,:) = 0d0
If (bc_top.eq."vac") Bound_Regs(3,:,:) = 0d0
If (bc_bottom.eq."vac") Bound_Regs(4,:,:) = 0d0

Do i = 1,X_Cells
    Do j = 1,Y_Cells
        Do k = 1,4
            Boundary(k,i,j) = Bound_Regs(k,Region_X(i,j),Region_Y(i,j))
        Enddo
    Enddo
Enddo

! Read iteration parameters

Read (10,*)
Read (10,*)
Read (10,*)
Read (10,*) Maxit, Eps

! Define cross sections for materials

Do i = 1,X_Cells
    Do j = 1,Y_Cells
        Fixed(i,j) = Source(Region_X(i,j),Region_Y(i,j))
        Material(i,j) = Mat_Regs(Region_X(i,j),Region_Y(i,j))
        Delta_X(i,j) = Del_X_Regs(Region_X(i,j),Region_Y(i,j)) &
            /X_Cells_Regs(Region_X(i,j),Region_Y(i,j))
        Delta_Y(i,j) = Del_Y_Regs(Region_X(i,j),Region_Y(i,j)) &
            /Y_Cells_Regs(Region_X(i,j),Region_Y(i,j))
    Enddo
Enddo

Do i = 1,X_Cells
    Do j = 1,Y_Cells
        If (Material(i,j).eq.1) Then
            Sigma_Tot(i,j) = 0d0
            Sigma_Scat(i,j) = 0d0
        Elseif (Material(i,j).eq.2) Then
            Sigma_Tot(i,j) = 0.75d0
            Sigma_Scat(i,j) = 0.5d0
        Elseif (Material(i,j).eq.3) Then
            Sigma_Tot(i,j) = 0.375d0
            Sigma_Scat(i,j) = 0.25d0
        Elseif (Material(i,j).eq.4) Then
            Sigma_Tot(i,j) = 1d0
            Sigma_Scat(i,j) = 1d0
        Elseif (Material(i,j).eq.5) Then
            Sigma_Tot(i,j) = 10d0
            Sigma_Scat(i,j) = 0d0
        Elseif (Material(i,j).eq.6) Then
            Sigma_Tot(i,j) = 100d0
            Sigma_Scat(i,j) = 0d0
        Elseif (Material(i,j).eq.7) Then
            Sigma_Tot(i,j) = 100d0
            Sigma_Scat(i,j) = 100d0
        Elseif (Material(i,j).eq.8) Then
            Sigma_Tot(i,j) = 1d0
            Sigma_Scat(i,j) = 0d0
        Endif
    Enddo
Enddo

! Read number of angular cells to consider

Read (10,*)
Read (10,*)
Read (10,*)
Read (10,*) Polar, Res

Write (out_res,'(i2)') Res

Ang_Res = 2d0**(Res+2)

Close (10)

Open (unit=20, &
    file="convergence "//trim(in_file)//" (R="//trim(adjustl(out_res))//").txt")
Write(20,'("File:  ",a10,"  R = ",a2)') in_file, out_res
Write(20,*)
Write(*,'("File:  ",a10,"  R = ",a2)') in_file, out_res

Allocate (xi(Polar), w(Polar))

Call Gauleg(Polar,xi,w)

!*****************************************************************************************
!
!   Computational Section
!
!*****************************************************************************************

Allocate (Theta(Ang_Res+1), mu(Ang_Res), eta(Ang_Res))
Allocate (Psi_Old(Polar,Ang_Res,X_Cells,Y_Cells), Psi_Left(Polar,Ang_Res,Y_Cells))
Allocate (Psi_Right(Polar,Ang_Res,Y_Cells), Psi_Top(Polar,Ang_Res,X_Cells))
Allocate (Psi_Bottom(Polar,Ang_Res,X_Cells))

! Create list of angular points and mu/eta values

Do i = 1,Ang_Res
    mu(i) = (2d0**(Res+1))/Pi*(dsin(2d0**(-1d0-dble(Res))*Pi*i) &
        -dsin(2d0**(-1d0-dble(Res))*Pi*(i-1)))
    eta(i) = (2d0**(Res+1))/Pi*(-dcos(2d0**(-1d0-dble(Res))*Pi*i) &
        +dcos(2d0**(-1d0-dble(Res))*Pi*(i-1)))
Enddo

! Begin Source Iteration
```

```
Write (*,*) "Beginning Source Iteration..."
Write (20,*) "Beginning Source Iteration..."

If (maxval(Sigma_Scat).gt.0d0) scat = .true.
If (scat) detail = .false.

If (not(detail).or.scat) Then
    Write (*,*)
    Write (*,'(" Iter    Radius         Error           Norm        Time    Unknowns")')
    Write (*,'(" ----   --------    ------------    ------------    ----    --------")')
    Write (20,*)
    Write (20,'(" Iter    Radius         Error           Norm        Time    Unknowns")')
    Write (20,'(" ----   --------    ------------    ------------    ----    --------")')
Endif

Phi(:,:) = 0d0

Psi_Old = 0d0
Psi_Left = 0d0
Psi_Right = 0d0
Psi_Top = 0d0
Psi_Bottom = 0d0

Phi_Old = 0d0
Phi = 0d0

Cnt_Tot = 0

Do k = 1,Maxit

    Cnt = 0

    Q = (Fixed + Sigma_Scat*Phi)/(4d0*Pi)*(2d0**(-1d0-dble(Res)/2d0))

    Phi_Old_2 = Phi_Old
    Phi_Old = Phi
    Phi = 0d0
    Err = 0d0

    ! Begin Polar Iteration

    Do pol=1,Polar

        If (detail) Write(*,'(t4,"Polar angle: ",f15.12,"    Weight: ",f14.12)') &
            xi(pol), w(pol)
        If (detail) Write(20,'(t4,"Polar angle: ",f15.12,"    Weight: ",f14.12)') &
            xi(pol), w(pol)

        If (trim(in_file).eq."test_box") &
            Open (unit=99, file="thresh (R="//trim(adjustl(out_res))//").txt")
        Do m = 1,Ang_Res

            If ((mu(m).ge.0.).and.(eta(m).ge.0.)) Quad = 1
            If ((mu(m).lt.0.).and.(eta(m).ge.0.)) Quad = 2
            If ((mu(m).lt.0.).and.(eta(m).lt.0.)) Quad = 3
            If ((mu(m).ge.0.).and.(eta(m).lt.0.)) Quad = 4

            te = etime(ta)
            If (detail.and.m.eq.1) &
                Write(*,'(t6,"Quadrant: ",i2,", Time = ",i5)') 1, nint(te)
            If (detail.and.m.eq.Ang_Res/4+1) &
                Write(*,'(t6,"Quadrant: ",i2,", Time = ",i5)') 2, nint(te)
            If (detail.and.m.eq.Ang_Res/2+1) &
                Write(*,'(t6,"Quadrant: ",i2,", Time = ",i5)') 3, nint(te)
            If (detail.and.m.eq.3*Ang_Res/4+1) &
                Write(*,'(t6,"Quadrant: ",i2,", Time = ",i5)') 4, nint(te)
            If (detail.and.m.eq.1) &
                Write(20,'(t6,"Quadrant: ",i2,", Time = ",i5)') 1, nint(te)
            If (detail.and.m.eq.Ang_Res/4+1) &
                Write(20,'(t6,"Quadrant: ",i2,", Time = ",i5)') 2, nint(te)
            If (detail.and.m.eq.Ang_Res/2+1) &
                Write(20,'(t6,"Quadrant: ",i2,", Time = ",i5)') 3, nint(te)
            If (detail.and.m.eq.3*Ang_Res/4+1) &
                Write(20,'(t6,"Quadrant: ",i2,", Time = ",i5)') 4, nint(te)

            If (Quad.eq.1) Then
                azi_left = Ang_Res/2.-m+1
                azi_bottom = Ang_Res-m+1
            Elseif (Quad.eq.2) Then
                azi_right = Ang_Res-m+1
                azi_bottom = Ang_Res-m+1
            Elseif (Quad.eq.3) Then
                azi_right = 3./2.*Ang_Res-m+1
                azi_top = Ang_Res-m+1
            Elseif (Quad.eq.4) Then
                azi_left = 3./2.*Ang_Res-m+1
                azi_top = Ang_Res-m+1
            Endif

            pol_opp = Polar-pol+1

            If (Quad.eq.1) Then
                I_Start = 1; I_Finish = X_Cells; I_Step =1
                J_Start = 1; J_Finish = Y_Cells; J_Step =1
            Elseif (Quad.eq.2) Then
                I_Start = X_Cells; I_Finish = 1; I_Step = -1
                J_Start = 1; J_Finish = Y_Cells; J_Step =1
            Elseif (Quad.eq.3) Then
                I_Start = X_Cells; I_Finish = 1; I_Step = -1
                J_Start = Y_Cells; J_Finish = 1; J_Step = -1
            Elseif (Quad.eq.4) Then
                I_Start = 1; I_Finish = X_Cells; I_Step =1
                J_Start = Y_Cells; J_Finish = 1; J_Step = -1
            Endif

            If (bc_bottom.eq."ref") Then
                If ((Quad.eq.1).or.(Quad.eq.2)) &
                    Psi_y_Inc(1:X_Cells) = Psi_Bottom(pol_opp,azi_bottom,1:X_Cells)
            Else
                If ((Quad.eq.1).or.(Quad.eq.2)) &
                    Psi_y_Inc(1:X_Cells) = Boundary(4,1:X_Cells,Y_Cells) &
                        *2d0**(1d0-dble(Res)/2d0)
            Endif
            If (bc_top.eq."ref") Then
                If ((Quad.eq.3).or.(Quad.eq.4)) &
                    Psi_y_Inc(1:X_Cells) = Psi_Top(pol_opp,azi_top,1:X_Cells)
            Else
                If ((Quad.eq.3).or.(Quad.eq.4)) &
                    Psi_y_Inc(1:X_Cells) = Boundary(3,1:X_Cells,1) &
                        *2d0**(1d0-dble(Res)/2d0)
            Endif

            Do j = J_Start,J_Finish,J_Step

                If (bc_left.eq."ref") Then
                    If ((Quad.eq.1).or.(Quad.eq.4)) &
                        Psi_x_Inc = Psi_Left(pol_opp,azi_left,j)
                Else
                    If ((Quad.eq.1).or.(Quad.eq.4)) &
                        Psi_x_Inc = Boundary(1,1,j)*2d0**(1d0-dble(Res)/2d0)
                Endif
                If (bc_right.eq."ref") Then
                    If ((Quad.eq.2).or.(Quad.eq.3)) &
                        Psi_x_Inc = Psi_Right(pol_opp,azi_right,j)
                Else
                    If ((Quad.eq.2).or.(Quad.eq.3)) &
                        Psi_x_Inc = Boundary(2,X_Cells,j)*2d0**(1d0-dble(Res)/2d0)
                Endif

                Do i = I_Start,I_Finish,I_Step

                    Psi = (2d0*Dabs(mu(m))/Delta_X(i,j)*Psi_x_Inc &
                        +2d0*Dabs(eta(m))/Delta_Y(i,j)*Psi_y_Inc(i) &
                        +Q(i,j)/dsqrt(1d0-xi(pol)**2d0)) &
                        /(Sigma_Tot(i,j)/dsqrt(1d0-xi(pol)**2d0)+2d0*Dabs(mu(m)) &
                        /Delta_X(i,j)+2d0*Dabs(eta(m))/Delta_Y(i,j))
                    Psi_x_Inc = 2d0*Psi - Psi_x_Inc
                    Psi_y_Inc(i) = 2d0*Psi - Psi_y_Inc(i)
                    Cnt = Cnt + 1


                    If (trim(in_file).eq."test_box".and.i.eq.30.and.j.eq.20) &
                        Write (99,*) Psi

                    If (Quad.eq.1) Then
                        If (i.eq.X_Cells) Psi_Right(pol,m,j) = Psi_x_Inc
                    Elseif (Quad.eq.2) Then
                        If (i.eq.1) Psi_Left(pol,m,j) = Psi_x_Inc
                    Elseif (Quad.eq.3) Then
                        If (i.eq.1) Psi_Left(pol,m,j) = Psi_x_Inc
                    Elseif (Quad.eq.4) Then
                        If (i.eq.X_Cells) Psi_Right(pol,m,j) = Psi_x_Inc
                    Endif

                    Psi_Old(pol,m,i,j) = Psi
                    Phi(i,j) = Phi(i,j) + (2d0**(-dble(Res)/2d0))*Pi*Psi*w(pol)

                Enddo

                If (Quad.eq.1) Then
                    If (j.eq.Y_Cells) Psi_Top(pol,m,:) = Psi_y_Inc(:)
                Elseif (Quad.eq.2) Then
                    If (j.eq.Y_Cells) Psi_Top(pol,m,:) = Psi_y_Inc(:)
                Elseif (Quad.eq.3) Then
                    If (j.eq.1) Psi_Bottom(pol,m,:) = Psi_y_Inc(:)
                Elseif (Quad.eq.4) Then
                    If (j.eq.1) Psi_Bottom(pol,m,:) = Psi_y_Inc(:)
                Endif

            Enddo

        Enddo
        If (trim(in_file).eq."test_box") Close (99)
    Enddo

    If (not(scat)) exit

    Sum_1 = 0d0
    Do j = 1,Y_Cells
        Sum_2 = 0d0
        Do i = 1,X_Cells
            Sum_2 = Sum_2 + (Phi(i,j)-Phi_Old(i,j))
        Enddo
        Sum_1 = Sum_1 + Sum_2**2d0
    Enddo
    Norm_1 = dsqrt(Sum_1/dble(X_Cells))

    Sum_1 = 0d0
    Do j = 1,Y_Cells
        Sum_2 = 0d0
        Do i = 1,X_Cells
```

```
            Sum_2 = Sum_2 + (Phi_Old(i,j)-Phi_Old_2(i,j))
        Enddo
        Sum_1 = Sum_1 + Sum_2**2d0
    Enddo
    Norm_2 = dsqrt(Sum_1/dble(X_Cells))

    Sum_1 = 0d0
    Do j = 1,Y_Cells
        Sum_2 = 0d0
        Do i = 1,X_Cells
            Sum_2 = Sum_2 + Phi(i,j)
        Enddo
        Sum_1 = Sum_1 + Sum_2**2d0
    Enddo
    Norm_3 = dsqrt(Sum_1/dble(X_Cells))

    Rho = 0d0
    If (Norm_2.ne.0d0) Rho = Norm_1/Norm_2

    te = etime(ta)

    If (detail) Write(*,*)
"************************************************************************"
    If (Norm_3.ne.0d0) Write (*,'(i5,f11.6,2es16.6,i8,i12)') &
        k, Rho, Norm_1/((1d0-Rho)*Norm_3), Norm_3, int(te), Cnt
    If (detail) Write(*,*)
"************************************************************************"
    If (detail) Write(20,*)
"************************************************************************"
    If (Norm_3.ne.0d0) Write (20,'(i5,f11.6,2es16.6,i8,i12)') &
        k, Rho, Norm_1/((1d0-Rho)*Norm_3), Norm_3, int(te), Cnt
    If (detail) Write(20,*)
"************************************************************************"
    If (k.ge.2.and.(Norm_1.le.Eps*(1d0-Rho)*Norm_3)) Exit

    If (k.eq.MaxIt) Write (*,*) "The source iteration did not converge!"
    If (k.eq.MaxIt) Write (20,*) "The source iteration did not converge!"

    Cnt_Tot = Cnt_Tot + Cnt

    Continue

Enddo

20 Continue

!*****************************************************************************************
!
!   Output Section
!
!*****************************************************************************************

Write (*,*) "Writing Output Files..."
Write (20,*) "Writing Output Files..."

Open (unit=10, file="top "//trim(in_file)//" (R="//trim(adjustl(out_res))//").txt")

x = 0.
Do i = 1,X_Cells
    If (i.eq.1) Then
        x = Delta_X(1,1)/2.
    Else
        x = x + (Delta_X(i-1,1)+Delta_X(i,1))/2.
    Endif
    Write (10,'(3es16.6)') x, Phi(i,Y_Cells)
Enddo
```

```
Close (10)

!***********************************************************************************

Open (unit=10, file="output "//trim(in_file)//" (R="//trim(adjustl(out_res))//").dat")

Write(10,*) 'Variables = "X", "Y", "Scalar Flux"'
Write(10,*) 'Zone i=',Y_Cells,', j=',X_Cells,', F=point'

Do i=1,X_Cells
    If (i.eq.1) x = Delta_X(1,1)/2d0
    If (i.ne.1) x = x + (Delta_X(i-1,1)+Delta_X(i,1))/2d0
    Do j=1,Y_Cells
        If (j.eq.1) y = Delta_Y(i,1)/2d0
        If (j.ne.1) y = y + (Delta_Y(i,j-1)+Delta_Y(i,j))/2d0
        Write (10,'(3es16.6)') x, y, Phi(i,j)
    Enddo
Enddo

Close (10)

!***********************************************************************************

Open (unit=10, file="log "//trim(in_file)//" (R="//trim(adjustl(out_res))//").dat")

Write(10,*) 'Variables = "X", "Y", "Scalar Flux"'
Write(10,*) 'Zone i=',Y_Cells,', j=',X_Cells,', F=point'

Min_Phi = maxval(Phi)
Do i=1,X_Cells
    Do j=1,Y_Cells
        If (Phi(i,j).lt.Min_Phi.and.Phi(i,j).gt.0d0) Min_Phi = Phi(i,j)
    Enddo
Enddo

Do i=1,X_Cells
    If (i.eq.1) x = Delta_X(1,1)/2d0
    If (i.ne.1) x = x + (Delta_X(i-1,1)+Delta_X(i,1))/2d0
    Do j=1,Y_Cells
        If (j.eq.1) y = Delta_Y(i,1)/2d0
        If (j.ne.1) y = y + (Delta_Y(i,j-1)+Delta_Y(i,j))/2d0
        If (Phi(i,j).le.0d0) Then
            Write (10,'(3es16.6)') x, y, dlog10(Min_Phi)
        Else
            Write (10,'(3es16.6)') x, y, dlog10(Phi(i,j))
        Endif
    Enddo
Enddo

Close (10)

!***********************************************************************************

If (trim(in_file).eq."hohlraum") Then

Open (unit=10, file="target "//trim(in_file)//" (R="//trim(adjustl(out_res))//").txt")

Do i=1,X_Cells
    If (i.eq.1) x = Delta_X(1,1)/2d0
    If (i.ne.1) x = x + (Delta_X(i-1,1)+Delta_X(i,1))/2d0
    Do j=1,Y_Cells
        If (j.eq.1) y = Delta_Y(i,1)/2d0
        If (j.ne.1) y = y + (Delta_Y(i,j-1)+Delta_Y(i,j))/2d0
        If ((x.ge.0.45.and.x.le.0.45+Delta_X(i,j)).and.(y.ge.0.25.and.y.le.1.05)) Then
            Write (10,'(3es16.6)') x, y, Phi(i,j)
        Endif
    Enddo
```

```
Enddo

Do i=1,X_Cells
    If (i.eq.1) x = Delta_X(1,1)/2d0
    If (i.ne.1) x = x + (Delta_X(i-1,1)+Delta_X(i,1))/2d0
    Do j=1,Y_Cells
        If (j.eq.1) y = Delta_Y(i,1)/2d0
        If (j.ne.1) y = y + (Delta_Y(i,j-1)+Delta_Y(i,j))/2d0
        If ((y.ge.1.05-Delta_Y(i,j).and.y.le.1.05).and.(x.ge.0.45.and.x.le.0.85)) &
            Write (10,'(3es16.6)') x, y, Phi(i,j)
    Enddo
Enddo

Do i=1,X_Cells
    If (i.eq.1) x = Delta_X(1,1)/2d0
    If (i.ne.1) x = x + (Delta_X(i-1,1)+Delta_X(i,1))/2d0
    y = 0d0
    Do j=1,Y_Cells
        y = y + Delta_Y(i,j)
    Enddo
    Do j=Y_Cells,1,-1
        If (j.eq.Y_Cells) y = y - Delta_Y(i,Y_Cells)/2d0
        If (j.ne.Y_Cells) y = y - (Delta_Y(i,j)+Delta_Y(i,j+1))/2d0
        If ((x.ge.0.85-Delta_X(i,j).and.x.le.0.85).and.(y.ge.0.25.and.y.le.1.05)) &
            Write (10,'(3es16.6)') x, y, Phi(i,j)
    Enddo
Enddo

x = 0d0
Do i=1,X_Cells
    x = x + Delta_X(i,1)
Enddo

Do i=X_Cells,1,-1
    If (i.eq.X_Cells) x = x - Delta_X(X_Cells,X_Cells)/2d0
    If (i.ne.X_Cells) x = x - (Delta_X(i,1)+Delta_X(i+1,1))/2d0
    Do j=1,Y_Cells
        If (j.eq.1) y = Delta_Y(i,1)/2d0
        If (j.ne.1) y = y + (Delta_Y(i,j-1)+Delta_Y(i,j))/2d0
        If ((y.ge.0.25.and.y.le.0.25+Delta_Y(i,j)).and.(x.ge.0.45.and.x.le.0.85)) &
            Write (10,'(3es16.6)') x, y, Phi(i,j)
    Enddo
Enddo

Close (10)

Endif

!*****************************************************************************************

te = etime(ta)

Write (*,'("The program took ",i4," iterations and ran in ",f8.2," seconds.")') k, te
Write (*,'("There were a total of ",i10, " unknowns calculated.")') Cnt_Tot
Write (20,'("The program took ",i4," iterations and ran in ",f8.2," seconds.")') k, te
Write (20,'("There were a total of ",i10, " unknowns calculated.")') Cnt_Tot

Close (20)

!*****************************************************************************************

End Program S_N
```

The following is the source code for the subroutine gauleg:

```
Subroutine gauleg(n,x,w)
```

```
  Implicit None

  Integer :: NP,N,i,j,m,NQ
  Parameter (NP=130,NQ=5)
  Real*8 :: X(NP),W(NP),EPS,p1,p2,p3,pp,xl,xm,z,z1, pi
  Parameter (EPS=3.d-14)

  pi = 4d0*datan(1d0)

  m=(n+1)/2
  xm=0d0
  xl=1d0
  Do i=1,m
     z=dcos(pi*(dble(i)-.25d0)/(dble(n)+.5d0))
1    continue
     p1=1d0
     p2=0d0
     Do j=1,n
        p3=p2
        p2=p1
        p1=((2.d0*dble(j)-1.d0)*z*p2-(dble(j)-1.d0)*p3)/dble(j)
     Enddo
     pp=n*(z*p1-p2)/(z*z-1.d0)
     z1=z
     z=z1-p1/pp
     If (dabs(z-z1).gt.EPS)goto 1
     x(i)=xm-xl*z
     x(n+1-i)=xm+xl*z
     w(i)=2.d0*xl/((1.d0-z*z)*pp*pp)
     w(n+1-i)=w(i)
  Enddo

End Subroutine
```

The following is an input file for the first box-in-box problem:

```
Regions:

2 2

Source:

0    0
0.25 0

Material Index:

2 2
2 2

Cells Per Region (x):

50 50
50 50

Cells per Region (y):

50 50
50 50

Width of Region:

1.0 1.0
1.0 1.0
```

```
Height of Region:

1.0 1.0
1.0 1.0

Incident Fluxes (id, left, right, top, bottom):

ref 0 0
vac 0 0
vac 0 0
ref 0 0

Iteration Parameters:

1000000 1e-7

Angular Resolution:

1 7
```

APPENDIX B

The following is the source code for S_N-W_N:

```
Program SNWN

  Implicit None

  Real :: te, ta(2), etime

  ! Region Declarations

  Integer :: X_Regs, Y_Regs, index

  ! Declare source and material index arrays

  Real*8, Allocatable :: Source(:,:)
  Integer, Allocatable :: Mat_Regs(:,:)

  ! Declare spatial grid arrays

  Real*8, Allocatable :: X_Cells_Regs(:,:), Y_Cells_Regs(:,:), Del_X_Regs(:,:)
  Real*8, Allocatable :: Del_Y_Regs(:,:), tar(:), tar_old(:), tar_old_2(:)

  Integer :: X_Cells, Y_Cells

  ! Declare cell-wise arrays

  Real*8, Allocatable :: Sigma_Tot(:,:), Sigma_Scat(:,:), Delta_X(:,:), Delta_Y(:,:)
  Real*8, Allocatable :: Psi_Old(:,:,:,:), Psi_Left(:,:,:), Psi_Right(:,:,:)
  Real*8, Allocatable :: Psi_Top(:,:,:), Psi_Bottom(:,:,:), Phi(:,:), Phi_Old(:,:)
  Real*8, Allocatable :: Q(:,:), Psi_y_Inc(:), Fixed(:,:), Phi_Old_2(:,:)
  Integer, Allocatable :: Material(:,:), Region_X(:,:), Region_Y(:,:), X_Left(:,:)
  Integer, Allocatable :: X_Right(:,:), Y_Top(:,:), Y_Bottom(:,:)
  Real*8, Allocatable :: Diff_Wave(:,:,:), Diff_Scaling(:,:,:), Diff_x_s(:,:,:)
  Real*8, Allocatable :: Diff_y_s(:,:,:), Diff_x_w(:,:,:,:), Diff_y_w(:,:,:,:)
  Real*8 :: Psi_x_Inc, Norm_1, Norm_2, Norm_3, Sum_1, Sum_2, Rho, maxtar, Norm_Phi_Old
  Real*8 :: Norm_Phi, Err_Old, Err_Sum, Err_Avg, Err(10)

  ! Declare boundary conditions

  Real*8, Allocatable :: Bound_Regs(:,:,:), Boundary(:,:,:)

  ! Declare iteration parameters

  Integer :: Maxit, t
  Real*8 :: Eps, W_Eps

  ! Declare angular variables

  Integer :: MaxRes, Quad, Polar, pol, R, lambda, n, p, pmin, pmax
  Real*8, Allocatable :: Theta(:), xi(:), w(:)
  Real*8, External :: mu, eta, A, B, alpha, beta

  ! Declare other variables

  Integer :: i, j, k, l, m, azi_left, azi_right, azi_top, azi_bottom, pol_opp
  Integer :: I_Start, I_Finish, I_Step, J_Start, J_Finish, J_Step
  Real*8 :: Pi, x, y, Psi, Psi_W

  Character(20) :: in_file
  Character(3) :: bc_left, bc_right, bc_top, bc_bottom
```

```
! Declare and create list

Type w_list
   Integer :: r, k
   Type(w_list), Pointer :: Next
End Type w_list

Type array_of_list
   Type(w_list), Pointer :: Head, Ptr, Tail
End Type array_of_list

Type(array_of_list), Allocatable :: List(:,:,:)

Real*8 :: Ptr_l, Ptr_r, rk_l, rk_r, Thresh
Logical :: inlist

Pi = 4d0*datan(1d0)

!***************************************************************************************
!
!  Input Section
!
!***************************************************************************************

Write (*,*) "Name of input file:"
Read (*,*) in_file

Open (Unit = 10, File = in_file)

! Read number of regions in each direction

Read (10,*)
Read (10,*)
Read (10,*) X_Regs, Y_Regs

! Define source strength and cross sections for each material and region

Allocate (Source(X_Regs,Y_Regs), Mat_Regs(X_Regs,Y_Regs))

Read (10,*)
Read (10,*)
Read (10,*)
Do i = Y_Regs,1,-1
   Read (10,*) Source(1:X_Regs,i)
Enddo

Read (10,*)
Read (10,*)
Read (10,*)
Do i = Y_Regs,1,-1
   Read (10,*) Mat_Regs(1:X_Regs,i)
Enddo

! Read number of cells in each region in each direction

Allocate (X_Cells_Regs(X_Regs,Y_Regs), Y_Cells_Regs(X_Regs,Y_Regs))
Allocate (Del_X_Regs(X_Regs,Y_Regs), Del_Y_Regs(X_Regs,Y_Regs))

Read (10,*)
Read (10,*)
Read (10,*)
Do i = Y_Regs,1,-1
   Read (10,*) X_Cells_Regs(1:X_Regs,i)
Enddo

Read (10,*)
Read (10,*)
Read (10,*)
Do i = Y_Regs,1,-1
   Read (10,*) Y_Cells_Regs(1:X_Regs,i)
Enddo

Read (10,*)
Read (10,*)
Read (10,*)
Do i = Y_Regs,1,-1
   Read (10,*) Del_X_Regs(1:X_Regs,i)
Enddo

Read (10,*)
Read (10,*)
Read (10,*)
Do i = Y_Regs,1,-1
   Read (10,*) Del_Y_Regs(1:X_Regs,i)
Enddo

! Allocate cell-wise arrays

X_Cells = 0
Do i = 1,X_Regs
   X_Cells = X_Cells + X_Cells_Regs(i,1)
Enddo

Y_Cells = 0
Do i = 1,Y_Regs
   Y_Cells = Y_Cells + Y_Cells_Regs(1,i)
Enddo

Allocate (Sigma_Tot(X_Cells,Y_Cells), Sigma_Scat(X_Cells,Y_Cells))
Allocate (Delta_X(X_Cells,Y_Cells), Delta_Y(X_Cells,Y_Cells), Fixed(X_Cells,Y_Cells))
Allocate (Phi(X_Cells,Y_Cells), Phi_Old(X_Cells,Y_Cells), Phi_Old_2(X_Cells,Y_Cells))
Allocate (Q(X_Cells,Y_Cells), Material(X_Cells,Y_Cells), Psi_y_Inc(X_Cells))
Allocate (Region_X(X_Cells,Y_Cells), Region_Y(X_Cells,Y_Cells), X_Left(X_Regs,Y_Regs))
Allocate (X_Right(X_Regs,Y_Regs), Y_Top(X_Regs,Y_Regs), Y_Bottom(X_Regs,Y_Regs))
Allocate (Diff_Wave(4,X_Cells,Y_Cells), Diff_Scaling(4,X_Cells,Y_Cells))
Allocate (Diff_x_s(4,X_Cells,Y_Cells), Diff_y_s(4,X_Cells,Y_Cells))

! Define region indices

Do i = 1,X_Regs
   Do j = 1,Y_Regs

      X_Left(i,j) = 0.
      Do k = 1,i-1
         X_Left(i,j) = X_Left(i,j) + X_Cells_Regs(k,j)
      Enddo

      Y_Bottom(i,j) = 0.
      Do k = 1,j-1
         Y_Bottom(i,j) = Y_Bottom(i,j) + Y_Cells_Regs(i,k)
      Enddo

      X_Right(i,j) = X_Left(i,j) + X_Cells_Regs(i,j)

      Y_Top(i,j) = Y_Bottom(i,j) + Y_Cells_Regs(i,j)

      Do k = 1,X_Cells
         If ((k.gt.X_Left(i,j)).and.(k.le.X_Right(i,j))) Region_X(k,1:Y_Cells) = i
      Enddo

      Do k = 1,Y_Cells
         If ((k.gt.Y_Bottom(i,j)).and.(k.le.Y_Top(i,j))) Region_Y(1:X_Cells,k) = j
      Enddo
   Enddo
Enddo

Do i = 1,X_Regs
   Do j = 1,Y_Regs

      Do k = 1,X_Cells
         If ((k.gt.X_Left(i,j)).and.(k.le.X_Right(i,j))) Then
            Do l = 1,Y_Cells
               Region_X(k,l) = i
            Enddo
         Endif
      Enddo

      Do k=1,Y_Cells
         If ((k.gt.Y_Bottom(i,j)).and.(k.le.Y_Top(i,j))) Then
            Do l=1,X_Cells
               Region_Y(l,k) = j
            Enddo
         Endif
      Enddo

   Enddo
Enddo

! Read boundary conditions

Allocate (Bound_Regs(4,X_Regs,Y_Regs), Boundary(4,X_Cells,Y_Cells))

Bound_Regs = 0d0
Boundary = 0d0

Read (10,*)
Read (10,*)
Read (10,*)
Read (10,*) bc_left, Bound_Regs(1,1,1:Y_Regs)
Read (10,*) bc_right, Bound_Regs(2,X_Regs,1:Y_Regs)
Read (10,*) bc_top, Bound_Regs(3,1:X_Regs,1)
Read (10,*) bc_bottom, Bound_Regs(4,1:X_Regs,Y_Regs)

If (bc_left.eq."vac") Bound_Regs(1,:,:) = 0d0
If (bc_right.eq."vac") Bound_Regs(2,:,:) = 0d0
If (bc_top.eq."vac") Bound_Regs(3,:,:) = 0d0
If (bc_bottom.eq."vac") Bound_Regs(4,:,:) = 0d0

Do i = 1,X_Cells
   Do j = 1,Y_Cells
      Do k = 1,4
         Boundary(k,i,j) = Bound_Regs(k,Region_X(i,j),Region_Y(i,j))
      Enddo
   Enddo
Enddo

! Read iteration parameters

Read (10,*)
Read (10,*)
Read (10,*)
Read (10,*) Maxit, Eps, W_Eps

! Define cross sections for materials

Do i = 1,X_Cells
   Do j = 1,Y_Cells
      Fixed(i,j) = Source(Region_X(i,j),Region_Y(i,j))
      Material(i,j) = Mat_Regs(Region_X(i,j),Region_Y(i,j))
      Delta_X(i,j) = Del_X_Regs(Region_X(i,j),Region_Y(i,j))/ &
```

```
         X_Cells_Regs(Region_X(i,j),Region_Y(i,j))
      Delta_Y(i,j) = Del_Y_Regs(Region_X(i,j),Region_Y(i,j))/ &
         Y_Cells_Regs(Region_X(i,j),Region_Y(i,j))
   Enddo
Enddo

Do i = 1,X_Cells
   Do j = 1,Y_Cells
      If (Material(i,j).eq.1) Then
         Sigma_Tot(i,j) = 0d0
         Sigma_Scat(i,j) = 0d0
      Elseif (Material(i,j).eq.2) Then
         Sigma_Tot(i,j) = 0.75d0
         Sigma_Scat(i,j) = 0.5d0
      Elseif (Material(i,j).eq.3) Then
         Sigma_Tot(i,j) = 1d0
         Sigma_Scat(i,j) = 0d0
      Elseif (Material(i,j).eq.4) Then
         Sigma_Tot(i,j) = 100d0
         Sigma_Scat(i,j) = 0d0
      Elseif (Material(i,j).eq.5) Then
         Sigma_Tot(i,j) = 1d0
         Sigma_Scat(i,j) = 0d0
      Elseif (Material(i,j).eq.6) Then
         Sigma_Tot(i,j) = 1d0
         Sigma_Scat(i,j) = .9d0
      Elseif (Material(i,j).eq.7) Then
         Sigma_Tot(i,j) = 1d0
         Sigma_Scat(i,j) = .99d0
      Elseif (Material(i,j).eq.8) Then
         Sigma_Tot(i,j) = 1d0
         Sigma_Scat(i,j) = 1d0
      Endif
   Enddo
Enddo

! Read number of angular cells to consider

Read (10,*)
Read (10,*)
Read (10,*)
Read (10,*) Polar, MaxRes

Allocate (Diff_x_w(MaxRes+1,2**(MaxRes+1),X_Cells,Y_Cells))
Allocate (Diff_y_w(MaxRes+1,2**(MaxRes+1),X_Cells,Y_Cells))

Close (10)

! Allocate polar arrays and initialize variables

Allocate (Psi_Old(Polar,4,X_Cells,Y_Cells), xi(Polar), w(Polar))
Allocate (Psi_Left(Polar,4,Y_Cells), Psi_Right(Polar,4,Y_Cells))
Allocate (Psi_Top(Polar,4,X_Cells), Psi_Bottom(Polar,4,X_Cells))

Call Gauleg(Polar,xi,w)

Diff_Wave = 0d0
Diff_Scaling = 0d0
Diff_x_s = 0d0
Diff_y_s = 0d0
Diff_x_w = 0d0
Diff_y_w = 0d0
Err = 0d0

Allocate (List(4,X_Cells,Y_Cells))
Do m=1,4
   Do i=1,X_Cells
```

```
      Do j=1,Y_Cells
         Nullify(List(m,i,j)%Head, List(m,i,j)%Tail, List(m,i,j)%Ptr)
      Enddo
   Enddo
Enddo

! Begin Source Iteration

Open (Unit=99, File="convergence.txt")

Do k=1,MaxIt

   ! Save old data and initialize new data

   Norm_Phi_Old = Norm_Phi
   Phi_Old_2 = Phi_Old
   Phi_Old = Phi
   Phi = 0d0

   ! Loop over polar angles

   Do pol=1,Polar

      !********************************************************************************
      !
      !   Calculate Scaling Function Coefficients
      !
      !********************************************************************************

      ! Loop over azimuthal angles

      Do m=1,4

         ! Determine reflecting angles

         If (m.eq.1) Then
            azi_left = 3-m
            azi_bottom = 5-m
         Elseif (m.eq.2) Then
            azi_right = 3-m
            azi_bottom = 3-m
         Elseif (m.eq.3) Then
            azi_right = 7-m
            azi_top = 2-m
         Elseif (m.eq.4) Then
            azi_left = 7-m
            azi_top = 3-m
         Endif

         pol_opp = Polar-pol+1

         ! Calculate source

         Q = Sigma_Scat/(8d0*Pi)*Phi_Old + Fixed/(8d0*Pi) - Diff_Scaling(m,:,:)

         ! Set-up sweep directions

         Select Case (m)
            Case (1)
               I_Start = 1;       I_Finish = X_Cells;  I_Step =  1;
               J_Start = 1;       J_Finish = Y_Cells;  J_Step =  1;
            Case (2)
               I_Start = X_Cells; I_Finish = 1;        I_Step = -1;
               J_Start = 1;       J_Finish = Y_Cells;  J_Step =  1;
            Case (3)
               I_Start = X_Cells; I_Finish = 1;       I_Step = -1;
               J_Start = Y_Cells; J_Finish = 1;       J_Step = -1;
            Case (4)
               I_Start = 1;       I_Finish = X_Cells; I_Step =  1;
               J_Start = Y_Cells; J_Finish = 1;       J_Step = -1;
         End Select

         ! Construct "top" and "bottom" boundary conditions

         If (bc_bottom.eq."ref") Then
            If ((m.eq.1).or.(m.eq.2)) Psi_y_Inc(1:X_Cells) = &
               Psi_Bottom(pol_opp,azi_bottom,1:X_Cells)
         Else
            If ((m.eq.1).or.(m.eq.2)) Psi_y_Inc(1:X_Cells) = &
               Boundary(4,1:X_Cells,Y_Cells)
         Endif
         If (bc_top.eq."ref") Then
            If ((m.eq.3).or.(m.eq.4)) Psi_y_Inc(1:X_Cells) = &
               Psi_Top(pol_opp,azi_top,1:X_Cells)
         Else
            If ((m.eq.3).or.(m.eq.4)) Psi_y_Inc(1:X_Cells) = &
               Boundary(3,1:X_Cells,1)
         Endif

         ! Begin vertical sweep

         Do j=J_Start,J_Finish,J_Step

            ! Construct "left" and "right" boundary conditions

            If (bc_left.eq."ref") Then
               If ((m.eq.1).or.(m.eq.4)) Psi_x_Inc = Psi_Left(pol_opp,azi_left,j)
            Else
               If ((m.eq.1).or.(m.eq.4)) Psi_x_Inc = Boundary(1,1,j)
            Endif
            If (bc_right.eq."ref") Then
               If ((m.eq.2).or.(m.eq.3)) Psi_x_Inc = Psi_Right(pol_opp,azi_right,j)
            Else
               If ((m.eq.2).or.(m.eq.3)) Psi_x_Inc = Boundary(2,X_Cells,j)
            Endif

            ! Begin horizontal sweep

            Do i=I_Start,I_Finish,I_Step

               ! Calculate angular flux

               Psi = dmax1((2d0*Dabs(mu(2,m))/Delta_X(i,j)*Psi_x_Inc+ &
                  2d0*Dabs(eta(2,m))/Delta_Y(i,j)*Psi_y_Inc(i)+ &
                  Q(i,j)/dsqrt(1d0-xi(pol)**2d0))/(Sigma_Tot(i,j)/ &
                  dsqrt(1d0-xi(pol)**2d0)+ 2d0*Dabs(mu(2,m))/Delta_X(i,j)+ &
                  2d0*Dabs(eta(2,m))/Delta_Y(i,j)),0d0)

               ! Calculate spatial angular flux derivatives

               If (m.eq.1.or.m.eq.4) Diff_x_s(m,i,j)=2d0*(Psi-Psi_x_Inc)/Delta_X(i,j)
               If (m.eq.2.or.m.eq.3) Diff_x_s(m,i,j)=2d0*(Psi_x_Inc-Psi)/Delta_X(i,j)
               If (m.eq.1.or.m.eq.2) Diff_y_s(m,i,j)=2d0*(Psi-Psi_y_Inc(i))/ &
                  Delta_Y(i,j)
               If (m.eq.3.or.m.eq.4) Diff_y_s(m,i,j)=2d0*(Psi_y_Inc(i)-Psi)/ &
                  Delta_Y(i,j)

               ! Calculate cell exiting fluxes

               Psi_x_Inc = 2d0*Psi-Psi_x_Inc
               Psi_y_Inc(i) = 2d0*Psi-Psi_y_Inc(i)

               ! Save angular flux and sum scalar flux
```

```
         Psi_Old(pol,m,i,j) = Psi
         Phi(i,j) = Phi(i,j) + Pi*Psi*w(pol)

         ! Save "left" and "right" boundary reflective fluxes

         If (m.eq.1) Then
            If (i.eq.X_Cells) Psi_Right(pol,m,j) = Psi_x_Inc
         Elseif (m.eq.2) Then
            If (i.eq.1) Psi_Left(pol,m,j) = Psi_x_Inc
         Elseif (m.eq.3) Then
            If (i.eq.1) Psi_Left(pol,m,j) = Psi_x_Inc
         Elseif (m.eq.4) Then
            If (i.eq.X_Cells) Psi_Right(pol,m,j) = Psi_x_Inc
         Endif

      Enddo    ! Horizontal sweep

      ! Save "top" and "bottom" boundary reflective fluxes

      If (m.eq.1) Then
         If (j.eq.Y_Cells) Psi_Top(pol,m,:) = Psi_y_Inc(:)
      Elseif (m.eq.2) Then
         If (j.eq.Y_Cells) Psi_Top(pol,m,:) = Psi_y_Inc(:)
      Elseif (m.eq.3) Then
         If (j.eq.1) Psi_Bottom(pol,m,:) = Psi_y_Inc(:)
      Elseif (m.eq.4) Then
         If (j.eq.1) Psi_Bottom(pol,m,:) = Psi_y_Inc(:)
      Endif

   Enddo    ! Vertical sweep

Enddo    ! Azimuthal loop

! Initialize wavelet coupling data

Diff_Scaling = 0d0

!*******************************************************************************
!
!   Calculate Wavelet Function Coefficients
!
!*******************************************************************************

! Loop over resolution level

Do R=2,MaxRes+1

   ! Loop over wavelet index

   Do lambda=1,2**R

      ! Determine quadrant of interest

      If (mu(R,lambda).gt.0.and.eta(R,lambda).gt.0) Quad = 1
      If (mu(R,lambda).lt.0.and.eta(R,lambda).gt.0) Quad = 2
      If (mu(R,lambda).lt.0.and.eta(R,lambda).lt.0) Quad = 3
      If (mu(R,lambda).gt.0.and.eta(R,lambda).lt.0) Quad = 4

      ! Compute coupling of all other wavelets to wavelet (R,lambda)

      Diff_Wave = 0d0
      Do n=2,MaxRes+1
         If (n.eq.R) Cycle
         pmin = 2**(n-2)*(Quad-1)+1
         pmax = 2**(n-2)*Quad
         Do p=pmin,pmax
            Diff_Wave(Quad,:,:) = Diff_Wave(Quad,:,:) + A(R,lambda,n,p)* &
               Diff_x_w(n,p,:,:) + B(R,lambda,n,p)*Diff_y_w(n,p,:,:)
         Enddo
      Enddo

      ! Calculate source

      Q = -alpha(R,lambda)*Diff_x_s(Quad,:,:) - beta(R,lambda)* &
         Diff_y_s(Quad,:,:) - Diff_Wave(Quad,:,:)

      ! Set-up sweep directions

      Select Case (Quad)
         Case (1)
            I_Start=1;       I_Finish=X_Cells; I_Step= 1;
            J_Start=1;       J_Finish=Y_Cells; J_Step= 1;
         Case (2)
            I_Start=X_Cells; I_Finish=1;       I_Step=-1;
            J_Start=1;       J_Finish=Y_Cells; J_Step= 1;
         Case (3)
            I_Start=X_Cells; I_Finish=1;       I_Step=-1;
            J_Start=Y_Cells; J_Finish=1;       J_Step=-1;
         Case (4)
            I_Start=1;       I_Finish=X_Cells; I_Step= 1;
            J_Start=Y_Cells; J_Finish=1;       J_Step=-1;
      End Select

      ! Construct "top" and "bottom" boundary conditions

      Psi_y_Inc(:) = 0d0

      ! Begin vertical sweep

      Do j=J_Start,J_Finish,J_Step

         ! Construct "left" and "right" boundary conditions

         Psi_x_Inc = 0d0

         ! Begin horizontal sweep

         Do i=I_Start,I_Finish,I_Step

            ! Determine if wavelet (R,lambda) is descendant of wavelet in the
            ! list and calculate angular flux

            If (Associated(List(Quad,i,j)%Head)) Then
               List(Quad,i,j)%Ptr => List(Quad,i,j)%Head
               Do
                  If (List(Quad,i,j)%Ptr%r.lt.R) Then
                     Ptr_l = 2d0**(-List(Quad,i,j)%Ptr%r)*Pi* &
                        (List(Quad,i,j)%Ptr%k-1)
                     Ptr_r = 2d0**(-List(Quad,i,j)%Ptr%r)*Pi*List(Quad,i,j)%Ptr%k
                     rk_l = 2d0**(-R)*Pi*(lambda-1)
                     rk_r = 2d0**(-R)*Pi*lambda
                     If (rk_l.ge.Ptr_l.and.rk_r.le.Ptr_r) Then
                        Allocate(List(Quad,i,j)%Tail%Next)
                        List(Quad,i,j)%Tail => List(Quad,i,j)%Tail%Next
                        Nullify(List(Quad,i,j)%Tail%Next)
                        List(Quad,i,j)%Tail%r = R
                        List(Quad,i,j)%Tail%k = lambda
                        Psi_W = 0d0
                        Exit
                     Else
                        If (Associated(List(Quad,i,j)%Ptr%Next)) Then
                           List(Quad,i,j)%Ptr => List(Quad,i,j)%Ptr%Next
                           Cycle
                        Endif
                        Psi_W = (2d0*Dabs(mu(R,lambda))/Delta_X(i,j)*Psi_x_Inc+ &
                           2d0*Dabs(eta(R,lambda))/Delta_Y(i,j)*Psi_y_Inc(i)+ &
                           Q(i,j)/dsqrt(1d0-xi(pol)**2d0))/(Sigma_Tot(i,j)/ &
                           dsqrt(1d0-xi(pol)**2d0)+ 2d0*Dabs(mu(R,lambda))/ &
                           Delta_X(i,j)+ 2d0*Dabs(eta(R,lambda))/Delta_Y(i,j))
                        Exit
                     Endif
                  Else
                     If (Associated(List(Quad,i,j)%Ptr%Next)) Then
                        List(Quad,i,j)%Ptr => List(Quad,i,j)%Ptr%Next
                        Cycle
                     Else
                        Exit
                     Endif
                  Endif
               Enddo
            Else
               Psi_W = (2d0*Dabs(mu(R,lambda))/Delta_X(i,j)*Psi_x_Inc+ &
                  2d0*Dabs(eta(R,lambda))/Delta_Y(i,j)*Psi_y_Inc(i)+Q(i,j)/ &
                  dsqrt(1d0-xi(pol)**2d0))/(Sigma_Tot(i,j)/ &
                  dsqrt(1d0-xi(pol)**2d0)+ 2d0*Dabs(mu(R,lambda))/Delta_X(i,j)+ &
                  2d0*Dabs(eta(R,lambda))/Delta_Y(i,j))
            Endif

            ! Determine if wavelet (R,lambda) is in the list

            If (Associated(List(Quad,i,j)%Head)) Then
               List(Quad,i,j)%Ptr => List(Quad,i,j)%Head
               inlist = .false.
               Do
                  If (List(Quad,i,j)%Ptr%r.eq.R.and.List(Quad,i,j)%Ptr% &
                     k.eq.lambda) Then
                     inlist = .true.
                     Exit
                  Endif
                  List(Quad,i,j)%Ptr => List(Quad,i,j)%Ptr%Next
                  If (.not.Associated(List(Quad,i,j)%Ptr)) Exit
               Enddo
            Else
               inlist = .false.
            Endif

            ! Determine if wavelet (R,lambda) should be thresholded

            If (.not.inlist) Then
               Thresh = 2d0**(1d0-R/2d0)*W_Eps*Psi_Old(pol,Quad,i,j)
               If (dabs(Psi_W).lt.Thresh) Then
                  If (Not(Associated(List(Quad,i,j)%Head))) Then
                     Allocate(List(Quad,i,j)%Head)
                     List(Quad,i,j)%Ptr => List(Quad,i,j)%Head
                     List(Quad,i,j)%Tail => List(Quad,i,j)%Ptr
                     Nullify(List(Quad,i,j)%Ptr%Next)
                     List(Quad,i,j)%Ptr%r = R
                     List(Quad,i,j)%Ptr%k = lambda
                     Psi_W = 0d0
                  Else
                     Allocate(List(Quad,i,j)%Tail%Next)
                     List(Quad,i,j)%Tail => List(Quad,i,j)%Tail%Next
                     Nullify(List(Quad,i,j)%Tail%Next)
                     List(Quad,i,j)%Tail%r = R
                     List(Quad,i,j)%Tail%k = lambda
                     Psi_W = 0d0
                  Endif
               Endif
            Else
               Psi_W = 0d0
            Endif
```

```
               ! Calculate spatial angular flux derivatives

               If (Quad.eq.1.or.Quad.eq.4) Diff_x_w(R,lambda,i,j)= &
                  2d0*(Psi_W-Psi_x_Inc)/Delta_X(i,j)
               If (Quad.eq.2.or.Quad.eq.3) Diff_x_w(R,lambda,i,j)= &
                  2d0*(Psi_x_Inc-Psi_W)/Delta_X(i,j)
               If (Quad.eq.1.or.Quad.eq.2) Diff_y_w(R,lambda,i,j)= &
                  2d0*(Psi_W-Psi_y_Inc(i))/Delta_Y(i,j)
               If (Quad.eq.3.or.Quad.eq.4) Diff_y_w(R,lambda,i,j)= &
                  2d0*(Psi_y_Inc(i)-Psi_W)/Delta_Y(i,j)

               ! Calculate cell exiting fluxes

               Psi_x_Inc = 2d0*Psi_W-Psi_x_Inc
               Psi_y_Inc(i) = 2d0*Psi_W-Psi_y_Inc(i)

               ! Save wavelet contribution to angular flux

               Diff_Scaling(Quad,i,j) = Diff_Scaling(Quad,i,j) + alpha(R,lambda)* &
                  Diff_x_w(R,lambda,i,j) + beta(R,lambda)*Diff_y_w(R,lambda,i,j)

            Enddo    ! Horizontal sweep
         Enddo    ! Vertical sweep

         Continue

      Enddo    ! Wavelet index loop

   Enddo    ! Resolution level loop

   Continue

Enddo    ! Polar loop

!*********************************************************************************
!
!   Calculate Norms and Check for Convergence
!
!*********************************************************************************

If (k.gt.3) Then

   Sum_1 = 0d0
   Do j = 1,Y_Cells
      Sum_2 = 0d0
      Do i = 1,X_Cells
         Sum_2 = Sum_2 + (Phi(i,j)-Phi_Old(i,j))
      Enddo
      Sum_1 = Sum_1 + Sum_2**2d0
   Enddo
   Norm_1 = dsqrt(Sum_1/dble(X_Cells))

   Sum_1 = 0d0
   Do j = 1,Y_Cells
      Sum_2 = 0d0
      Do i = 1,X_Cells
         Sum_2 = Sum_2 + (Phi_Old(i,j)-Phi_Old_2(i,j))
      Enddo
      Sum_1 = Sum_1 + Sum_2**2d0
   Enddo
   Norm_2 = dsqrt(Sum_1/dble(X_Cells))

   Sum_1 = 0d0
   Do j = 1,Y_Cells
      Sum_2 = 0d0
      Do i = 1,X_Cells
```

```
            Sum_2 = Sum_2 + Phi(i,j)
         Enddo
         Sum_1 = Sum_1 + Sum_2**2d0
      Enddo
      Norm_3 = dsqrt(Sum_1/dble(X_Cells))

      Rho = 0d0
      If (Norm_2.ne.0d0) Rho = Norm_1/Norm_2

      te = etime(ta)

      If (Norm_3.ne.0d0) Write (*,'(i8,f11.6,2es16.6,f8.0)') k, Rho, &
         Norm_1/(Dabs(1d0-Rho)*Norm_3), Norm_3, te
      If (Norm_3.ne.0d0) Write (99,'(i8,f11.6,2es16.6,f8.0)') k, Rho, &
         Norm_1/(Dabs(1d0-Rho)*Norm_3), Norm_3, te
      If (Norm_1.le.Eps*(1d0-Rho)*Norm_3.or.te.gt.43200) Exit

   Endif

Enddo    ! Source iteration loop

If (k.ge.MaxIt) Write (*,*) "******** The Iteration Did Not Converge ********"
If (k.ge.MaxIt) Write (99,*) "******** The Iteration Did Not Converge ********"

Close (99)

!************************************************************************************
!
!   Write data to output files
!
!************************************************************************************

Open (unit=10, file="output.dat")

Write(10,*) 'Variables = "X", "Y", "Scalar Flux"'
Write(10,*) 'Zone i=',Y_Cells,', j=',X_Cells,', F=point'

Do i=1,X_Cells
   If (i.eq.1) x = Delta_X(1,1)/2d0
   If (i.ne.1) x = x + (Delta_X(i-1,1)+Delta_X(i,1))/2d0
   Do j=1,Y_Cells
      If (j.eq.1) y = Delta_Y(1,i)/2d0
      If (j.ne.1) y = y + (Delta_Y(i,j-1)+Delta_Y(i,j))/2d0
      If (phi(i,j).ne.0d0) Then
         Write (10,'(3es16.6)') x, y, dlog(dabs(phi(i,j)))
      Else
         Write (10,'(3es16.6)') x, y, -20d0
      Endif
   Enddo
Enddo

Close (10)

!************************************************************************************

Open (unit=20, file="diagonal.txt")

Do i=1,X_Cells
   If (i.eq.1) x = Delta_X(1,1)/2d0
   If (i.ne.1) x = x + (Delta_X(i-1,1)+Delta_X(i,1))/2d0
   Write (20,'(2es16.6)') x, dabs(phi(int(0.6825*i)+1,i))
Enddo

Close (20)

!************************************************************************************
```

```
End Program SNWN
```

The following is the source code for the subroutine gauleg, which computes the Gauss-Legendre quadrature points and weights:

```
Subroutine gauleg(n,x,w)

  Implicit None

  Integer :: NP,N,i,j,m,NQ
  Parameter (NP=130,NQ=5)
  Real*8 :: X(NP),W(NP),EPS,p1,p2,p3,pp,xl,xm,z,z1, pi
  Parameter (EPS=3.d-14)

  pi = 4d0*datan(1d0)

  m=(n+1)/2
  xm=0d0
  xl=1d0
  Do i=1,m
     z=dcos(pi*(dble(i)-.25d0)/(dble(n)+.5d0))
1    continue
     p1=1d0
     p2=0d0
     Do j=1,n
        p3=p2
        p2=p1
        p1=((2.d0*dble(j)-1.d0)*z*p2-(dble(j)-1.d0)*p3)/dble(j)
     Enddo
     pp=n*(z*p1-p2)/(z*z-1.d0)
     z1=z
     z=z1-p1/pp
     If (dabs(z-z1).gt.EPS)goto 1
     x(i)=xm-xl*z
     x(n+1-i)=xm+xl*z
     w(i)=2.d0*xl/((1.d0-z*z)*pp*pp)
     w(n+1-i)=w(i)
  Enddo

End Subroutine
```

The following is the source code for the function mu:

```
Real*8 Function mu(r,k)

  Implicit None

  Real*8 :: pi
  Integer :: r, k

  pi = 4d0*datan(1d0)
  mu = 2d0**r/(2d0*pi)*(dsin(2d0**(1-r)*pi*k)-dsin(2d0**(1-r)*pi*(k-1)))

End Function mu
```

The following is the source code for the subroutine eta:

```
Real*8 Function eta(r,k)
```

```
  Implicit None

  Real*8 :: pi
  Integer :: r, k

  pi = 4d0*datan(1d0)
  eta = 2d0**r/(2d0*pi)*(-dcos(2d0**(1-r)*pi*k)+dcos(2d0**(1-r)*pi*(k-1)))

End Function eta
```

The following is the source code for the subroutine alpha:

```
Real*8 Function alpha(r,k)

  Implicit None

  Real*8 :: pi
  Integer :: r, k

  pi = 4d0*datan(1d0)
  alpha = 2d0**(r/2d0)/pi*(-dsin(2d0**(1-r)*pi*k)+2d0*dsin(2d0**(1-r)*pi*(k-.5d0)) &
     -dsin(2d0**(1-r)*pi*(k-1d0)))

End Function alpha
```

The following is the source code for the subroutine beta:

```
Real*8 Function beta(r,k)

  Implicit None

  Real*8 :: pi
  Integer :: r, k

  pi = 4d0*datan(1d0)
  beta = 2d0**(r/2d0)/pi*(dcos(2d0**(1-r)*pi*k)-2d0*dcos(2d0**(1-r)*pi*(k-.5d0)) &
     +dcos(2d0**(1-r)*pi*(k-1d0)))

End Function beta
```

The following is the source code for the subroutine A:

```
Real*8 Function A(r,k,n,p)

  Implicit None

  Real*8 :: pi, u_l, u_m, u_r, l_l !w, s_l, s_m, s_r
  Integer :: r, k, n, p, ures, uind, lres, lind

  pi = 4d0*datan(1d0)

  If (n.gt.r) Then
     ures = r; uind = k; lres = n; lind = p;
  Else
     ures = n; uind = p; lres = r; lind = k;
  Endif

  u_l = 2d0**(-ures)*pi*(uind-1d0)
  u_m = 2d0**(-ures)*pi*(uind-.5d0)
  u_r = 2d0**(-ures)*pi*uind
  l_l = 2d0**(-lres)*pi*(lind-1d0)
  If (l_l.ge.u_l.and.l_l.lt.u_m) Then
     A = (2d0**(ures/2d0+lres/2d0))/(2d0*pi)*(-dsin(2d0**(1-lres)*pi*lind) &
        +2d0*dsin(2d0**(1-lres)*pi*(lind-0.5))-dsin(2d0**(1-lres)*pi*(lind-1)))
  Elseif (l_l.ge.u_m.and.l_l.lt.u_r) Then
     A = -(2d0**(ures/2d0+lres/2d0))/(2d0*pi)*(-dsin(2d0**(1-lres)*pi*lind) &
        +2d0*dsin(2d0**(1-lres)*pi*(lind-0.5))-dsin(2d0**(1-lres)*pi*(lind-1)))
  Else
     A = 0d0
  Endif

End Function A
```

The following is the source code for the subroutine B:

```
Real*8 Function B(r,k,n,p)

  Implicit None

  Real*8 :: pi, u_l, u_m, u_r, l_l !w, s_l, s_m, s_r
  Integer :: r, k, n, p, ures, uind, lres, lind

  pi = 4d0*datan(1d0)

  If (n.gt.r) Then
     ures = r; uind = k; lres = n; lind = p;
  Else
     ures = n; uind = p; lres = r; lind = k;
  Endif

  u_l = 2d0**(-ures)*pi*(uind-1d0)
  u_m = 2d0**(-ures)*pi*(uind-.5d0)
  u_r = 2d0**(-ures)*pi*uind
  l_l = 2d0**(-lres)*pi*(lind-1d0)

  If (l_l.ge.u_l.and.l_l.lt.u_m) Then
     B = (2d0**(ures/2d0+lres/2d0))/(2d0*pi)*(dcos(2d0**(1-lres)*pi*lind) &
        -2d0*dcos(2d0**(1-lres)*pi*(lind-0.5))+dcos(2d0**(1-lres)*pi*(lind-1)))
  Elseif (l_l.ge.u_m.and.l_l.lt.u_r) Then
     B = -(2d0**(ures/2d0+lres/2d0))/(2d0*pi)*(dcos(2d0**(1-lres)*pi*lind) &
        -2d0*dcos(2d0**(1-lres)*pi*(lind-0.5))+dcos(2d0**(1-lres)*pi*(lind-1)))
  Else
     B = 0d0
  Endif

End Function B
```

The following is an input file for the first box-in-box problem:

```
Regions:

2 2

Source:

0 0
0.25 0

Material Index:

2 2
2 2
```

```
Cells Per Region (x):

20 20
20 20

Cells per Region (y):

20 20
20 20

Width of Region:

1.0 1.0
1.0 1.0

Height of Region:

1.0 1.0
1.0 1.0

Incident Fluxes (id, left, right, top, bottom):

vac 0 0
vac 0 0
vac 0 0
vac 0 0

Iteration Parameters:

100 1d-7 1d-2

Angular Resolution:

1 4
```

APPENDIX C

The following is the source code for CW-W_N:

```
Program CWWN

Implicit None

Real :: te, ta(2), etime

! Region Declarations

Integer :: X_Regs, Y_Regs, index

! Declare source and material index arrays

Real*8, Allocatable :: Source(:,:)
Integer, Allocatable :: Mat_Regs(:,:)

! Declare spatial grid arrays

Real*8, Allocatable :: X_Cells_Regs(:,:), Y_Cells_Regs(:,:), Del_X_Regs(:,:)
Real*8, Allocatable :: Del_Y_Regs(:,:), tar(:), tar_old(:), tar_old_2(:)

Integer :: X_Cells, Y_Cells

! Declare cell-wise arrays

Real*8, Allocatable :: Sigma_Tot(:,:), Sigma_Scat(:,:), Delta_X(:,:), Delta_Y(:,:)
Real*8, Allocatable :: Fixed(:,:), Q(:,:), Material(:,:), Region_X(:,:), Region_Y(:,:)
Real*8, Allocatable :: X_Left(:,:), X_Right(:,:), Y_Top(:,:), Y_Bottom(:,:)
Real*8 :: Norm_1, Norm_2, Norm_3, Sum_1, Sum_2, Rho, maxtar

! Declare boundary conditions

Real*8, Allocatable :: Bound_Regs_Left(:), Bound_Regs_Right(:), Bound_Regs_Top(:)
Real*8, Allocatable :: Bound_Regs_Bottom(:), Boundary_Left(:), Boundary_Right(:)
Real*8, Allocatable :: Boundary_Top(:), Boundary_Bottom(:), Bound_X(:), Bound_Y(:)
Character(3) :: BC_Left, BC_Right, BC_Top, BC_Bottom

! Declare iteration parameters

Integer :: MaxIt, t
Real*8 :: Eps, W_Eps

! Declare angular variables

Integer :: MaxRes, Polar, pol, r, k, n, p, pmin, pmax
Real*8, Allocatable :: Xi(:), Omega(:)
Real*8, External :: mu, eta, alpha, beta, A, B

! Declare angular flux variables

Real*8, Allocatable :: Psi(:), Sig(:,:), C_x(:,:), C_y(:,:), Mat(:,:), Svec(:)
Real*8, Allocatable :: C_x2(:,:), C_y2(:,:), Psi_Save(:,:,:,:), Qvec(:), Psi_x(:)
Real*8, Allocatable :: Psi_y(:), Psi_x_Inc2(:), Psi_y_Inc2(:), Phi(:,:), Phi_Old(:,:)
Real*8, Allocatable :: Phi_Old_2(:,:), Psi_y_Inc(:), Psi_x_Inc(:), Psi_x_Inc_Old(:)
Real*8, Allocatable :: Psi_Temp(:)

! Declare other variables

Integer :: i, j, l, m, Azi_Left, Azi_Right, Azi_Top, Azi_Bottom, Pol_Opp
Integer :: I_Start, I_Finish, I_Step, J_Start, J_Finish, J_Step
Integer :: Res, ind1, ind2, iter, Res_x, Sout, Count, Cnt_Tot, Count1
Integer, Allocatable :: Res_y(:), Res_Left_1(:), Res_Left_2(:), Res_Right_1(:)
Integer, Allocatable :: Res_Right_2(:), Res_Top_1(:), Res_Top_2(:), Res_Bottom_1(:)
Integer, Allocatable :: Res_Bottom_2(:), Res_Psi(:,:), mat_count(:,:,:)

Real*8 :: Pi, x, y, Min_Phi, Psi_Norm
Real*8 :: ptr_l, ptr_r, rk_l, rk_r, Psi_Est, Thresh, theta_l, theta_r, x_neg, y_neg
Logical :: scat, detail
Character(10) :: in_file
Character(2) :: out_string, out_res, out_m

! Declare and create lists

Type w_list
    Integer :: r, k
    Type(w_list), Pointer :: next
End Type

Type array_of_w_list
    Type(w_list), Pointer :: Head, Tail, Ptr
End Type array_of_w_list

Type(array_of_w_list), Allocatable :: List(:,:,:)

Type psi_y_list
    Real*8 :: value
    Type(psi_y_list), Pointer :: next
End Type

Type array_of_psi_y
    Type(psi_y_list), Pointer :: Head, Ptr, Tail
End Type

Type(array_of_psi_y), Allocatable :: Psi_y_Inc_Next(:), Psi_y_Inc_Next_Old(:)
Type(array_of_psi_y), Allocatable :: Psi_Left_1(:), Psi_Left_2(:), Psi_Right_1(:)
Type(array_of_psi_y), Allocatable :: Psi_Right_2(:), Psi_Top_1(:), Psi_Top_2(:)
Type(array_of_psi_y), Allocatable :: Psi_Bottom_1(:), Psi_Bottom_2(:)

Pi = 4d0*datan(1d0)

!*****************************************************************************************
!
!  Input Section
!
!*****************************************************************************************

Write (*,*) "Creating Problem Domain..."

detail = .true.

Write (*,*) "Name of input file:"
Read (*,*) in_file

Open (Unit = 10, File = in_file, Status="old")

! Read number of regions in each direction

Read (10,*)
Read (10,*)
Read (10,*) X_Regs, Y_Regs

! Define source strength and cross sections for each material and region

Allocate (Source(X_Regs,Y_Regs), Mat_Regs(X_Regs,Y_Regs))

Read (10,*)
Read (10,*)
Read (10,*)
Do i = Y_Regs,1,-1
   Read (10,*) Source(1:X_Regs,i)
Enddo

Read (10,*)
Read (10,*)
Read (10,*)
Do i = Y_Regs,1,-1
   Read (10,*) Mat_Regs(1:X_Regs,i)
Enddo

! Read number of cells in each region in each direction

Allocate (X_Cells_Regs(X_Regs,Y_Regs), Y_Cells_Regs(X_Regs,Y_Regs))
Allocate (Del_X_Regs(X_Regs,Y_Regs), Del_Y_Regs(X_Regs,Y_Regs))

Read (10,*)
Read (10,*)
Read (10,*)
Do i = Y_Regs,1,-1
   Read (10,*) X_Cells_Regs(1:X_Regs,i)
Enddo

Read (10,*)
Read (10,*)
Read (10,*)
Do i = Y_Regs,1,-1
   Read (10,*) Y_Cells_Regs(1:X_Regs,i)
Enddo

Read (10,*)
Read (10,*)
Read (10,*)
Do i = Y_Regs,1,-1
   Read (10,*) Del_X_Regs(1:X_Regs,i)
Enddo

Read (10,*)
Read (10,*)
Read (10,*)
Do i = Y_Regs,1,-1
   Read (10,*) Del_Y_Regs(1:X_Regs,i)
Enddo

! Allocate cell-wise arrays

X_Cells = 0
Do i = 1,X_Regs
   X_Cells = X_Cells + X_Cells_Regs(i,1)
Enddo

Y_Cells = 0
Do i = 1,Y_Regs
   Y_Cells = Y_Cells + Y_Cells_Regs(1,i)
Enddo

Allocate (Sigma_Tot(X_Cells,Y_Cells), Sigma_Scat(X_Cells,Y_Cells))
Allocate (Delta_X(X_Cells,Y_Cells), Delta_Y(X_Cells,Y_Cells))
Allocate (Psi_Save(2,4,X_Cells,Y_Cells), Fixed(X_Cells,Y_Cells), Phi(X_Cells,Y_Cells))
Allocate (Phi_Old(X_Cells,Y_Cells), Phi_Old_2(X_Cells,Y_Cells), Q(X_Cells,Y_Cells))
Allocate (Material(X_Cells,Y_Cells), Psi_y_Inc_Next(X_Cells))
Allocate (Psi_y_Inc_Next_Old(X_Cells), Res_y(X_Cells), Region_X(X_Cells,Y_Cells))
Allocate (Region_Y(X_Cells,Y_Cells), X_Left(X_Regs,Y_Regs), X_Right(X_Regs,Y_Regs))
Allocate (Y_Top(X_Regs,Y_Regs), Y_Bottom(X_Regs,Y_Regs), Psi_Left_1(Y_Cells))
Allocate (Psi_Left_2(Y_Cells), Psi_Right_1(Y_Cells), Psi_Right_2(Y_Cells))
Allocate (Psi_Top_1(X_Cells), Psi_Top_2(X_Cells), Psi_Bottom_1(X_Cells))
Allocate (Psi_Bottom_2(X_Cells), Res_Left_1(Y_Cells), Res_Left_2(Y_Cells))
Allocate (Res_Right_1(Y_Cells), Res_Right_2(Y_Cells), Res_Top_1(X_Cells))
Allocate (Res_Top_2(X_Cells), Res_Bottom_1(X_Cells), Res_Bottom_2(X_Cells))
Allocate (List(4,X_Cells,Y_Cells), Res_Psi(X_Cells,Y_Cells))
Allocate (mat_count(4,X_Cells,Y_Cells))

Do m=1,4
    Do i=1,X_Cells
        Do j=1,Y_Cells
            Nullify(List(m,i,j)%Head)
            Nullify(List(m,i,j)%Ptr)
            Nullify(List(m,i,j)%Tail)
        Enddo
    Enddo
Enddo

! Define region indices

Do i = 1,X_Regs
    Do j = 1,Y_Regs

        X_Left(i,j) = 0.
        Do k = 1,i-1
            X_Left(i,j) = X_Left(i,j) + X_Cells_Regs(k,j)
        Enddo

        Y_Bottom(i,j) = 0.
        Do k = 1,j-1
            Y_Bottom(i,j) = Y_Bottom(i,j) + Y_Cells_Regs(i,k)
        Enddo

        X_Right(i,j) = X_Left(i,j) + X_Cells_Regs(i,j)

        Y_Top(i,j) = Y_Bottom(i,j) + Y_Cells_Regs(i,j)

        Do k = 1,X_Cells
            If ((k.gt.X_Left(i,j)).and.(k.le.X_Right(i,j))) Region_X(k,1:Y_Cells) = i
        Enddo

        Do k = 1,Y_Cells
            If ((k.gt.Y_Bottom(i,j)).and.(k.le.Y_Top(i,j))) Region_Y(1:X_Cells,k) = j
        Enddo

    Enddo
Enddo

Do i = 1,X_Regs
    Do j = 1,Y_Regs

        Do k = 1,X_Cells
            If ((k.gt.X_Left(i,j)).and.(k.le.X_Right(i,j))) Then
                Do l = 1,Y_Cells
                    Region_X(k,l) = i
                Enddo
            Endif
        Enddo

        Do k=1,Y_Cells
            If ((k.gt.Y_Bottom(i,j)).and.(k.le.Y_Top(i,j))) Then
                Do l=1,X_Cells
                    Region_Y(l,k) = j
                Enddo
            Endif
        Enddo

    Enddo
Enddo
```

```
! Read boundary conditions

Allocate (Bound_Regs_Left(Y_Regs), Bound_Regs_Right(Y_Regs), Bound_Regs_Top(X_Regs))
Allocate (Bound_Regs_Bottom(X_Regs), Boundary_Left(Y_Cells), Boundary_Right(Y_Cells))
Allocate (Boundary_Top(Y_Cells), Boundary_Bottom(Y_Cells), Bound_X(Y_Cells))
Allocate (Bound_Y(X_Cells))

Bound_Regs_Left = 0d0
Bound_Regs_Right = 0d0
Bound_Regs_Top = 0d0
Bound_Regs_Bottom = 0d0
Boundary_Left = 0d0
Boundary_Right = 0d0
Boundary_Top = 0d0
Boundary_Bottom = 0d0

Read (10,*)
Read (10,*)
Read (10,*)
Read (10,*) BC_Left, Bound_Regs_Left(1:Y_Regs)
Read (10,*) BC_Right, Bound_Regs_Right(1:Y_Regs)
Read (10,*) BC_Top, Bound_Regs_Top(1:X_Regs)
Read (10,*) BC_Bottom, Bound_Regs_Bottom(1:X_Regs)

If (BC_Left.eq."vac") Bound_Regs_Left = 0d0
If (BC_Right.eq."vac") Bound_Regs_Right = 0d0
If (BC_Top.eq."vac") Bound_Regs_Top = 0d0
If (BC_Bottom.eq."vac") Bound_Regs_Bottom = 0d0

Do i = 1,X_Cells
    Do j = 1,Y_Cells
        Boundary_Left(j) = Bound_Regs_Left(Region_Y(i,j))*2d0
        Boundary_Right(j) = Bound_Regs_Right(Region_Y(i,j))*2d0
        Boundary_Top(i) = Bound_Regs_Top(Region_X(i,j))*2d0
        Boundary_Bottom(i) = Bound_Regs_Bottom(Region_X(i,j))*2d0
    Enddo
Enddo

! Read iteration parameters

Read (10,*)
Read (10,*)
Read (10,*)
Read (10,*) Maxit, Eps, W_Eps

If (W_Eps.eq.0d0) Then
    Write (out_string,'(i2)') 0
Else
    Write (out_string,'(i2)') Nint(-dlog10(W_Eps))
Endif

! Define cross sections for materials

Do i = 1,X_Cells
    Do j = 1,Y_Cells
        Fixed(i,j) = Source(Region_X(i,j),Region_Y(i,j))
        Material(i,j) = Mat_Regs(Region_X(i,j),Region_Y(i,j))
        Delta_X(i,j) = Del_X_Regs(Region_X(i,j),Region_Y(i,j)) &
            /X_Cells_Regs(Region_X(i,j),Region_Y(i,j))
        Delta_Y(i,j) = Del_Y_Regs(Region_X(i,j),Region_Y(i,j)) &
            /Y_Cells_Regs(Region_X(i,j),Region_Y(i,j))
    Enddo
Enddo

Do i = 1,X_Cells
    Do j = 1,Y_Cells
```

```
        If (Material(i,j).eq.1) Then
            Sigma_Tot(i,j) = 0d0
            Sigma_Scat(i,j) = 0d0
        Elseif (Material(i,j).eq.2) Then
            Sigma_Tot(i,j) = 0.75d0
            Sigma_Scat(i,j) = 0.5d0
        Elseif (Material(i,j).eq.3) Then
            Sigma_Tot(i,j) = 0.375d0
            Sigma_Scat(i,j) = 0.25d0
        Elseif (Material(i,j).eq.4) Then
            Sigma_Tot(i,j) = 1d0
            Sigma_Scat(i,j) = 1d0
        Elseif (Material(i,j).eq.5) Then
            Sigma_Tot(i,j) = 10d0
            Sigma_Scat(i,j) = 0d0
        Elseif (Material(i,j).eq.6) Then
            Sigma_Tot(i,j) = 100d0
            Sigma_Scat(i,j) = 0d0
        Elseif (Material(i,j).eq.7) Then
            Sigma_Tot(i,j) = 100d0
            Sigma_Scat(i,j) = 100d0
        Elseif (Material(i,j).eq.8) Then
            Sigma_Tot(i,j) = 1d0
            Sigma_Scat(i,j) = 0d0
        Endif
    Enddo
Enddo

! Read number of angular cells to consider

Read (10,*)
Read (10,*)
Read (10,*)
Read (10,*) Polar, MaxRes

Write (out_res,'(i2)') MaxRes

MaxRes = MaxRes + 1

Close (10)

Open (unit=20, file="convergence "//trim(in_file)//" (R="//trim(adjustl(out_res))//",
    eps="//trim(adjustl(out_string))//").txt")
Write(20,'("File:  ",a10,"  R = ",a2,", eps = ",a2)') in_file, out_res, out_string
Write(20,*)
Write(*,'("File:  ",a10,"  R = ",a2,", eps = ",a2)') in_file, out_res, out_string

! Allocate polar arrays and initialize variables

Allocate (Xi(Polar), Omega(Polar))

Call Gauleg(Polar,Xi,Omega)

!*****************************************************************************************
!
!   Begin source iteration
!
!*****************************************************************************************

Write (*,*) "Beginning Source Iteration..."
Write (20,*) "Beginning Source Iteration..."

If (maxval(Sigma_Scat).gt.0d0) scat = .true.
If (scat) detail = .false.

If (not(detail).or.scat) Then
    Write (*,*)
    Write (*,'(" Iter    Radius         Error          Norm        Time    Unknowns")')
    Write (*,'(" ----   --------    ------------    ------------    ----    --------")')
    Write (20,*)
    Write (20,'(" Iter    Radius         Error          Norm        Time    Unknowns")')
    Write (20,'(" ----   --------    ------------    ------------    ----    --------")')
Endif

Cnt_Tot = 0
Psi_Save = 0d0

Do iter=1,MaxIt

    Count = 0

    ! Compute scattering source array

    Q = Sigma_Scat/(8d0*Pi)*Phi + Fixed/(8d0*Pi)

    ! Initialize convergence variables

    Phi_Old_2 = Phi_Old
    Phi_Old = Phi
    Phi = 0d0

    !   Loop over polar angle

    Do pol=1,Polar

        If (detail) Write(*,'(t4,"Polar angle: ",f15.12,"    Weight: ",f14.12)') &
            Xi(pol), Omega(pol)
        If (detail) Write(20,'(t4,"Polar angle: ",f15.12,"    Weight: ",f14.12)') &
            Xi(pol), Omega(pol)

        !   Loop over quadrant

        Do m=1,4

            te = etime(ta)
            If (detail) Write(*,'(t6,"Quadrant: ",i2,", Time = ",i5)') m, nint(te)
            If (detail) Write(20,'(t6,"Quadrant: ",i2,", Time = ",i5)') m, nint(te)

            ! Set up sweep directions

            Select Case (m)
                Case (1)
                    I_Start = 1;       I_Finish = X_Cells;    I_Step =  1;
                    J_Start = 1;       J_Finish = Y_Cells;    J_Step =  1;
                    Bound_X = Boundary_Left;              Bound_Y = Boundary_Bottom
                    x_neg = 1d0;       y_neg = 1d0
                Case (2)
                    I_Start = X_Cells; I_Finish = 1;      I_Step = -1;
                    J_Start = 1;       J_Finish = Y_Cells;    J_Step =  1;
                    Bound_X = Boundary_Right;             Bound_Y = Boundary_Bottom
                    x_neg = -1d0;      y_neg = 1d0
                Case (3)
                    I_Start = X_Cells; I_Finish = 1;      I_Step = -1;
                    J_Start = Y_Cells; J_Finish = 1;      J_Step = -1;
                    Bound_X = Boundary_Right;             Bound_Y = Boundary_Top
                    x_neg = -1d0;      y_neg = -1d0
                Case (4)
                    I_Start = 1;       I_Finish = X_Cells;    I_Step =  1;
                    J_Start = Y_Cells; J_Finish = 1;      J_Step = -1;
                    Bound_X = Boundary_Left;              Bound_Y = Boundary_Top
                    x_neg = 1d0;       y_neg = -1d0
            End Select

            ! Construct y-boundary conditions
```

```
If ((bc_bottom.eq."ref").and.(iter.gt.1)) Then
    If (m.eq.1) Then
        Do i=1,X_Cells
            Allocate (Psi_y_Inc_Next(i)%Head)
            Psi_Bottom_1(i)%ptr => Psi_Bottom_1(i)%Head
            Psi_y_Inc_Next(i)%ptr => Psi_y_Inc_Next(i)%Head
            Nullify (Psi_y_Inc_Next(i)%ptr%next)
            Psi_y_Inc_Next(i)%Head%value = Psi_Bottom_1(i)%Head%value
            Do
                If (Associated(Psi_Bottom_1(i)%ptr%next)) Then
                    Psi_Bottom_1(i)%ptr => Psi_Bottom_1(i)%ptr%next
                    Allocate (Psi_y_Inc_Next(i)%ptr%next)
                    Psi_y_Inc_Next(i)%ptr => Psi_y_Inc_Next(i)%ptr%next
                    Nullify (Psi_y_Inc_Next(i)%ptr%next)
                    Psi_y_Inc_Next(i)%ptr%value = Psi_Bottom_1(i)%ptr%value
                    Cycle
                Else
                    Exit
                Endif
            Enddo
            Res_y(i) = Res_Bottom_1(i)
        Enddo
    Endif
    If (m.eq.2) Then
        Do i=1,X_Cells
            Allocate (Psi_y_Inc_Next(i)%Head)
            Psi_Bottom_2(i)%ptr => Psi_Bottom_2(i)%Head
            Psi_y_Inc_Next(i)%ptr => Psi_y_Inc_Next(i)%Head
            Nullify (Psi_y_Inc_Next(i)%ptr%next)
            Psi_y_Inc_Next(i)%Head%value = Psi_Bottom_2(i)%Head%value
            Do
                If (Associated(Psi_Bottom_2(i)%ptr%next)) Then
                    Psi_Bottom_2(i)%ptr => Psi_Bottom_2(i)%ptr%next
                    Allocate (Psi_y_Inc_Next(i)%ptr%next)
                    Psi_y_Inc_Next(i)%ptr => Psi_y_Inc_Next(i)%ptr%next
                    Nullify (Psi_y_Inc_Next(i)%ptr%next)
                    Psi_y_Inc_Next(i)%ptr%value = Psi_Bottom_2(i)%ptr%value
                    Cycle
                Else
                    Exit
                Endif
            Enddo
            Res_y(i) = Res_Bottom_2(i)
        Enddo
    Endif
Else
    If ((m.eq.1).or.(m.eq.2)) Then
        Do i=1,X_Cells
            Allocate (Psi_y_Inc_Next(i)%Head)
            Nullify (Psi_y_Inc_Next(i)%Head%next)
            Psi_y_Inc_Next(i)%Head%value = Bound_Y(i)
            Res_y(i) = 2
        Enddo
    Endif
Endif

If ((bc_top.eq."ref").and.(iter.gt.1)) Then
    If (m.eq.3) Then
        Do i=1,X_Cells
            Allocate (Psi_y_Inc_Next(i)%Head)
            Psi_Top_1(i)%ptr => Psi_Top_1(i)%Head
            Psi_y_Inc_Next(i)%ptr => Psi_y_Inc_Next(i)%Head
            Nullify (Psi_y_Inc_Next(i)%ptr%next)
            Psi_y_Inc_Next(i)%Head%value = Psi_Top_1(i)%Head%value
            Do
                If (Associated(Psi_Top_1(i)%ptr%next)) Then
```

```
                    Psi_Top_1(i)%ptr => Psi_Top_1(i)%ptr%next
                    Allocate (Psi_y_Inc_Next(i)%ptr%next)
                    Psi_y_Inc_Next(i)%ptr => Psi_y_Inc_Next(i)%ptr%next
                    Nullify (Psi_y_Inc_Next(i)%ptr%next)
                    Psi_y_Inc_Next(i)%ptr%value = Psi_Top_1(i)%ptr%value
                    Cycle
                Else
                    Exit
                Endif
            Enddo
            Res_y(i) = Res_Top_1(i)
        Enddo
    Endif
    If (m.eq.4) Then
        Do i=1,X_Cells
            Allocate (Psi_y_Inc_Next(i)%Head)
            Psi_Top_2(i)%ptr => Psi_Top_2(i)%Head
            Psi_y_Inc_Next(i)%ptr => Psi_y_Inc_Next(i)%Head
            Nullify (Psi_y_Inc_Next(i)%ptr%next)
            Psi_y_Inc_Next(i)%Head%value = Psi_Top_2(i)%Head%value
            Do
                If (Associated(Psi_Top_2(i)%ptr%next)) Then
                    Psi_Top_2(i)%ptr => Psi_Top_2(i)%ptr%next
                    Allocate (Psi_y_Inc_Next(i)%ptr%next)
                    Psi_y_Inc_Next(i)%ptr => Psi_y_Inc_Next(i)%ptr%next
                    Nullify (Psi_y_Inc_Next(i)%ptr%next)
                    Psi_y_Inc_Next(i)%ptr%value = Psi_Top_2(i)%ptr%value
                    Cycle
                Else
                    Exit
                Endif
            Enddo
            Res_y(i) = Res_Top_2(i)
        Enddo
    Endif
Else
    If ((m.eq.3).or.(m.eq.4)) Then
        Do i=1,X_Cells
            Allocate (Psi_y_Inc_Next(i)%Head)
            Nullify (Psi_y_Inc_Next(i)%Head%next)
            Psi_y_Inc_Next(i)%Head%value = Bound_Y(i)
            Res_y(i) = 2
        Enddo
    Endif
Endif

! Loop over y-position index

Do j=J_Start,J_Finish,J_Step

!Write(*,'(3x,f5.2)') 100d0*dble(j)/440d0

    ! Construct x-boundary conditions

    If ((bc_left.eq."ref").and.(iter.gt.1)) Then
        If (m.eq.1) Then
            Res_x = Res_Left_1(j)
            Allocate(Psi_x_Inc(2**(Res_x-1)))
            Psi_x_Inc = 0d0
            Psi_Left_1(j)%Ptr => Psi_Left_1(j)%Head
            Do n=1,2**(Res_x-1)
                Psi_x_Inc(n) = Psi_Left_1(j)%Ptr%value
                If (Associated(Psi_Left_1(j)%Ptr%next)) Then
                    Psi_Left_1(j)%Ptr => Psi_Left_1(j)%Ptr%next
                Else
                    Exit
                Endif
            Enddo
        Endif
        If (m.eq.4) Then
            Res_x = Res_Left_2(j)
            Allocate(Psi_x_Inc(2**(Res_x-1)))
            Psi_x_Inc = 0d0
            Psi_Left_2(j)%Ptr => Psi_Left_2(j)%Head
            Do n=1,2**(Res_x-1)
                Psi_x_Inc(n) = Psi_Left_2(j)%Ptr%value
                If (Associated(Psi_Left_2(j)%Ptr%next)) Then
                    Psi_Left_2(j)%Ptr => Psi_Left_2(j)%Ptr%next
                Else
                    Exit
                Endif
            Enddo
        Endif
    Else
        If ((m.eq.1).or.(m.eq.4)) Then
            Res_x = 2
            Allocate(Psi_x_Inc(2**(Res_x-1)))
            Psi_x_Inc = 0d0
            Psi_x_Inc(1) = Bound_X(j)
        Endif
    Endif

    If ((bc_right.eq."ref").and.(iter.gt.1)) Then
        If (m.eq.2) Then
            Res_x = Res_Right_1(j)
            Allocate(Psi_x_Inc(2**(Res_x-1)))
            Psi_x_Inc = 0d0
            Psi_Right_1(j)%Ptr => Psi_Right_1(j)%Head
            Do n=1,2**(Res_x-1)
                Psi_x_Inc(n) = Psi_Right_1(j)%Ptr%value
                If (Associated(Psi_Right_1(j)%Ptr%next)) Then
                    Psi_Right_1(j)%Ptr => Psi_Right_1(j)%Ptr%next
                Else
                    Exit
                Endif
            Enddo
        Endif
        If (m.eq.3) Then
            Res_x = Res_Right_2(j)
            Allocate(Psi_x_Inc(2**(Res_x-1)))
            Psi_x_Inc = 0d0
            Psi_Right_2(j)%Ptr => Psi_Right_2(j)%Head
            Do n=1,2**(Res_x-1)
                Psi_x_Inc(n) = Psi_Right_2(j)%Ptr%value
                If (Associated(Psi_Right_2(j)%Ptr%next)) Then
                    Psi_Right_2(j)%Ptr => Psi_Right_2(j)%Ptr%next
                Else
                    Exit
                Endif
            Enddo
        Endif
    Else
        If ((m.eq.2).or.(m.eq.3)) Then
            Res_x = 2
            Allocate(Psi_x_Inc(2**(Res_x-1)))
            Psi_x_Inc = 0d0
            Psi_x_Inc(1) = Bound_X(j)
        Endif
    Endif

    ! Loop over x-position index

    Do i=I_Start,I_Finish,I_Step
```

```
! Construct y-incident vector from list

Allocate(Psi_y_Inc(2**(Res_y(i)-1)))
Psi_y_Inc = 0d0
Psi_y_Inc_Next(i)%Ptr => Psi_y_Inc_Next(i)%Head
Do n=1,2**(Res_y(i)-1)
    Psi_y_Inc(n) = Psi_y_Inc_Next(i)%Ptr%value
    If (Associated(Psi_y_Inc_Next(i)%Ptr%next)) Then
        Psi_y_Inc_Next(i)%Ptr => Psi_y_Inc_Next(i)%Ptr%next
    Else
        Exit
    Endif
Enddo

! Threshold after two scattering iterations, if there is scattering

If ((scat.and.iter.eq.4).or.(.not.scat)) Then

    Allocate(Sig(2,2), C_x(2,2), C_y(2,2), Qvec(2), Mat(2,2), Svec(2))
    Allocate(Psi_x(2), Psi_y(2), Psi(2), C_x2(2,2**(Res_x-1)))
    Allocate(C_y2(2,2**(Res_y(i)-1)))

    !*******************************************************************
    !
    !   Calculate scaling and coarsest wavelet coefficients from
    !       incident fluxes
    !
    !*******************************************************************

    C_x = 0d0
    C_x(1,1) = mu(2,m)
    C_x(1,2) = alpha(2,m)
    C_x(2,1) = C_x(1,2)
    C_x(2,2) = mu(2,m)

    C_y = 0d0
    C_y(1,1) = eta(2,m)
    C_y(1,2) = beta(2,m)
    C_y(2,1) = C_y(1,2)
    C_y(2,2) = eta(2,m)

    Sig = 0d0
    Do n=1,2
        Sig(n,n) = Sigma_Tot(i,j)/dsqrt(1d0-Xi(pol)**2d0)
    Enddo

    Mat = Sig + x_neg*2d0*C_x/Delta_X(i,j) + &
        y_neg*2d0*C_y/Delta_Y(i,j)

    Svec = 0d0
    Svec(1) = Q(i,j)/dsqrt(1d0-Xi(pol)**2d0)

    C_x2 = 0d0
    C_x2(1,1) = mu(2,m)
    Do r=2,Res_x
        Do k=2**(r-2)*(m-1)+1,2**(r-2)*m
            ind1 = 2**(r-2)*(2-m)+k
            C_x2(1,ind1) = alpha(r,k)
        Enddo
    Enddo
    C_x2(2,1) = C_x2(1,2)
    C_x2(2,2) = mu(2,m)
    Do n=3,Res_x
        pmin = 2**(n-2)*(m-1)+1
        pmax = 2**(n-2)*m
        Do p=pmin,pmax
            ind2 = 2**(n-2)*(2-m)+p
            C_x2(2,ind2) = A(2,m,n,p)
        Enddo
    Enddo

    C_y2 = 0d0
    C_y2(1,1) = eta(2,m)
    Do r=2,Res_y(i)
        Do k=2**(r-2)*(m-1)+1,2**(r-2)*m
            ind1 = 2**(r-2)*(2-m)+k
            C_y2(1,ind1) = beta(r,k)
        Enddo
    Enddo
    C_y2(2,1) = C_y2(1,2)
    C_y2(2,2) = eta(2,m)
    Do n=3,Res_y(i)
        pmin = 2**(n-2)*(m-1)+1
        pmax = 2**(n-2)*m
        Do p=pmin,pmax
            ind2 = 2**(n-2)*(2-m)+p
            C_y2(2,ind2) = B(2,m,n,p)
        Enddo
    Enddo

    Call mvmult(C_x2,Psi_x_Inc,Psi_x,2,2**(Res_x-1))
    Call mvmult(C_y2,Psi_y_Inc,Psi_y,2,2**(Res_y(i)-1))

    Svec = Svec + x_neg*2d0*Psi_x/Delta_X(i,j) + &
        y_neg*2d0*Psi_y/Delta_Y(i,j)

    Call CG(Mat,Svec,Psi,2,Count1)

    Psi_Save(:,m,i,j) = Psi(:)

    Psi_Norm = 0d0
    Do r=1,2
        Do k=1,4
            Psi_Norm = Psi_Norm + Psi_Save(r,k,i,j)**2d0
        Enddo
    Enddo
    Psi_Norm = dsqrt(2d0*Pi*Psi_Norm)

    !*****************************************************************
    !
    !   Use scaling and coarsest wavelet coefficients to threshold
    !       estimate of flux
    !
    !*****************************************************************

    r = 3
    Do While (r.le.MaxRes)
        Do k=2**(r-2)*(m-1)+1,2**(r-2)*m
            If (Associated(List(m,i,j)%Head)) Then
                If (inlist(List(m,i,j)%Head,r,k)) Cycle
            Endif
            Do

                ! If bounded by wavelet in List, add wavelet to List

                If (Associated(List(m,i,j)%Head)) Then
                    List(m,i,j)%Ptr => List(m,i,j)%Head
                    ptr_l = 2d0**(-List(m,i,j)%Ptr%r)*Pi &
                       *(List(m,i,j)%Ptr%k-1)
                    ptr_r = 2d0**(-List(m,i,j)%Ptr%r)*Pi &
                       *List(m,i,j)%Ptr%k
                    rk_l = 2d0**(-r)*Pi*(k-1)
                    rk_r = 2d0**(-r)*Pi*k
                    If (rk_l.ge.ptr_l.and.rk_r.le.ptr_r) Then
                        Allocate(List(m,i,j)%Tail%next)
                        List(m,i,j)%Tail => List(m,i,j)%Tail%next
                        Nullify(List(m,i,j)%Tail%next)
                        List(m,i,j)%Tail%r = r
                        List(m,i,j)%Tail%k = k
                        Exit
                    Endif
                Endif

                ! Estimate Psi from incident coefficients

                ind1 = 2**(r-2)*(2-m)+k
                Psi_Est = 2d0*x_neg*alpha(r,k)/Delta_X(i,j) &
                    *Psi_x_Inc(1) + 2d0*y_neg*beta(r,k)/Delta_Y(i,j) &
                    *Psi_y_Inc(1)
                Do n=2,Res_x
                    If (n.eq.r) Then
                        Psi_Est = Psi_Est + 2d0*mu(r,k)/Delta_X(i,j) &
                           *Psi_x_Inc(ind1)
                        Cycle
                    Endif
                    pmin = 2**(n-2)*(m-1)+1
                    pmax = 2**(n-2)*m
                    Do p=pmin,pmax
                        ind2 = 2**(n-2)*(2-m)+p
                        Psi_Est = Psi_Est + 2d0*x_neg*A(r,k,n,p) &
                           /Delta_X(i,j)*Psi_x_Inc(ind2)
                    Enddo
                Enddo
                Do n=2,Res_y(i)
                    If (n.eq.r) Then
                        Psi_Est = Psi_Est + 2d0*eta(r,k)/Delta_Y(i,j) &
                           *Psi_y_Inc(ind1)
                        Cycle
                    Endif
                    pmin = 2**(n-2)*(m-1)+1
                    pmax = 2**(n-2)*m
                    Do p=pmin,pmax
                        ind2 = 2**(n-2)*(2-m)+p
                        Psi_Est = Psi_Est + 2d0*y_neg*B(r,k,n,p) &
                           /Delta_Y(i,j)*Psi_y_Inc(ind2)
                    Enddo
                Enddo
                Psi_Est = Psi_Est/(2d0*mu(r,k)/Delta_X(i,j) &
                    +2d0*eta(r,k)/Delta_Y(i,j)+Sigma_Tot(i,j)/ &
                    dsqrt(1d0-Xi(pol)**2d0))

                ! Threshold based on Psi estimate

                Thresh = W_Eps/(2*Pi)*Psi_Norm
                If (dabs(Psi_Est).lt.Thresh) Then
                    If (not(associated(List(m,i,j)%Head))) Then
                        Allocate(List(m,i,j)%Head)
                        List(m,i,j)%Ptr => List(m,i,j)%Head
                        List(m,i,j)%Tail => List(m,i,j)%Ptr
                        Nullify(List(m,i,j)%Ptr%next)
                        List(m,i,j)%Ptr%r = r
                        List(m,i,j)%Ptr%k = k
                    Else
                        Allocate(List(m,i,j)%Tail%next)
                        List(m,i,j)%Tail => List(m,i,j)%Tail%next
                        Nullify(List(m,i,j)%Tail%next)
                        List(m,i,j)%Tail%r = r
                        List(m,i,j)%Tail%k = k
                    Endif
                    Exit
                Else
```

```
                    Exit
                Endif

            Enddo
        Enddo

        ind1 = 0
        List(m,i,j)%Ptr => List(m,i,j)%Head
        Do
            If (Associated(List(m,i,j)%Ptr)) Then
                If (List(m,i,j)%Ptr%r.eq.r) ind1 = ind1 + 1
                List(m,i,j)%Ptr => List(m,i,j)%Ptr%next
            Else
                Exit
            Endif
        Enddo

        If (ind1.eq.2**(r-2)) Exit
        r = r + 1

    Enddo

    !*****************************************************************

    Deallocate(C_x, C_y, Sig, Mat, Qvec, Psi_x, Psi_y, Svec, Psi)
    Deallocate(C_x2, C_y2)

Endif

! Set resolution level and count wavelet coefficients

ind2 = 0
If (.not.Associated(List(m,i,j)%Head)) Then
    Res = MaxRes
    Sout = 2**(Res-1)
Else
    List(m,i,j)%Ptr => List(m,i,j)%Head
    Do
        ind2 = ind2 + 1
        If (Associated(List(m,i,j)%Ptr%next)) Then
            List(m,i,j)%Ptr => List(m,i,j)%Ptr%next
        Else
            Exit
        Endif
    Enddo
    Res = List(m,i,j)%Ptr%r-1
    Sout = 2**(Res-1)-(ind2-2**(Res-1))
Endif

Allocate(C_x(Sout,Sout), C_y(Sout,Sout), Sig(Sout,Sout))
Allocate(Mat(Sout,Sout), Qvec(Sout), Psi_x(Sout), Psi_y(Sout))
Allocate(Svec(Sout), Psi_x_Inc2(Sout), Psi_y_Inc2(Sout), Psi(Sout))

!*********************************************************************
!
!   Use List to build matrix for solving all coefficients to MaxRes
!
!*********************************************************************

!   Build C_x matrix

C_x = 0d0
C_x(1,1) = mu(2,m)
ind1 = 1
Do r=2,Res
    Do k=2**(r-2)*(m-1)+1,2**(r-2)*m
        If (Associated(List(m,i,j)%Head)) Then
```

```
            If (inlist(List(m,i,j)%Head,r,k)) Cycle
        Endif
        ind1 = ind1 + 1
        C_x(1,ind1) = alpha(r,k)
        C_x(ind1,1) = C_x(1,ind1)
    Enddo
Enddo
ind1 = 2
Do r=2,Res
    Do k=2**(r-2)*(m-1)+1,2**(r-2)*m
        If (Associated(List(m,i,j)%Head)) Then
            If (inlist(List(m,i,j)%Head,r,k)) Cycle
        Endif
        C_x(ind1,ind1) = mu(r,k)
        If (Associated(List(m,i,j)%Head)) Then
            ind2 = 0
            List(m,i,j)%Ptr => List(m,i,j)%Head
            Do
                If (List(m,i,j)%Ptr%r.eq.r) Then
                    If (List(m,i,j)%Ptr%k.gt.k) ind2 = ind2 + 1
                Endif
                If (Associated(List(m,i,j)%Ptr%next)) Then
                    List(m,i,j)%Ptr => List(m,i,j)%Ptr%next
                Else
                    Exit
                Endif
            Enddo
            ind2 = ind1+1+m*2**(r-2)-k-ind2
        Else
            ind2 = ind1+1+m*2**(r-2)-k
        Endif
        Do n=2,Res
            If (n.le.r) Cycle
            pmin = 2**(n-2)*(m-1)+1
            pmax = 2**(n-2)*m
            Do p=pmin,pmax
                If (Associated(List(m,i,j)%Head)) Then
                    If (inlist(List(m,i,j)%Head,n,p)) Cycle
                Endif
                C_x(ind1,ind2) = A(r,k,n,p)
                C_x(ind2,ind1) = C_x(ind1,ind2)
                ind2 = ind2 + 1
            Enddo
        Enddo
        ind1 = ind1 + 1
    Enddo
Enddo

!   Build C_y matrix

C_y = 0d0
C_y(1,1) = eta(2,m)
ind1 = 1
Do r=2,Res
    Do k=2**(r-2)*(m-1)+1,2**(r-2)*m
        If (Associated(List(m,i,j)%Head)) Then
            If (inlist(List(m,i,j)%Head,r,k)) Cycle
        Endif
        ind1 = ind1 + 1
        C_y(1,ind1) = beta(r,k)
        C_y(ind1,1) = C_y(1,ind1)
    Enddo
Enddo
ind1 = 2
Do r=2,Res
    Do k=2**(r-2)*(m-1)+1,2**(r-2)*m
        If (Associated(List(m,i,j)%Head)) Then
```

```
            If (inlist(List(m,i,j)%Head,r,k)) Cycle
        Endif
        C_y(ind1,ind1) = eta(r,k)
        If (Associated(List(m,i,j)%Head)) Then
            ind2 = 0
            List(m,i,j)%Ptr => List(m,i,j)%Head
            Do
                If (List(m,i,j)%Ptr%r.eq.r) Then
                    If (List(m,i,j)%Ptr%k.gt.k) ind2 = ind2 + 1
                Endif
                If (Associated(List(m,i,j)%Ptr%next)) Then
                    List(m,i,j)%Ptr => List(m,i,j)%Ptr%next
                Else
                    Exit
                Endif
            Enddo
            ind2 = ind1+1+m*2**(r-2)-k-ind2
        Else
            ind2 = ind1+1+m*2**(r-2)-k
        Endif
        Do n=2,Res
            If (n.le.r) Cycle
            pmin = 2**(n-2)*(m-1)+1
            pmax = 2**(n-2)*m
            Do p=pmin,pmax
                If (Associated(List(m,i,j)%Head)) Then
                    If (inlist(List(m,i,j)%Head,n,p)) Cycle
                Endif
                C_y(ind1,ind2) = B(r,k,n,p)
                C_y(ind2,ind1) = C_y(ind1,ind2)
                ind2 = ind2 + 1
            Enddo
        Enddo
        ind1 = ind1 + 1
    Enddo
Enddo

!   Build Sig matrix

Sig = 0d0
Do n=1,Sout
    Sig(n,n) = Sigma_Tot(i,j)/dsqrt(1d0-Xi(pol)**2d0)
Enddo

Mat = Sig + x_neg*2d0*C_x/Delta_X(i,j) + y_neg*2d0*C_y/Delta_Y(i,j)

!   Build Qvec(i,j)/dsqrt(1d0-Xi(pol)**2d0) vector

Qvec = 0d0
Qvec(1) = Q(i,j)/dsqrt(1d0-Xi(pol)**2d0)

Psi_x_Inc2 = 0d0
Psi_x_Inc2(1) = Psi_x_Inc(1)
ind1 = 1
ind2 = 1
Do n=2,min(Res,Res_x)
    pmin = 2**(n-2)*(m-1)+1
    pmax = 2**(n-2)*m
    Do p=pmin,pmax
        ind2 = ind2 + 1
        If (Associated(List(m,i,j)%Head)) Then
            If (inlist(List(m,i,j)%Head,n,p)) Cycle
        Endif
        ind1 = ind1 + 1
        Psi_x_Inc2(ind1) = Psi_x_Inc(ind2)
    Enddo
Enddo
```

```
Psi_y_Inc2 = 0d0
Psi_y_Inc2(1) = Psi_y_Inc(1)
ind1 = 1
ind2 = 1
Do n=2,min(Res,Res_y(i))
    pmin = 2**(n-2)*(m-1)+1
    pmax = 2**(n-2)*m
    Do p=pmin,pmax
        ind2 = ind2 + 1
        If (Associated(List(m,i,j)%Head)) Then
            If (inlist(List(m,i,j)%Head,n,p)) Cycle
        Endif
        ind1 = ind1 + 1
        Psi_y_Inc2(ind1) = Psi_y_Inc(ind2)
    Enddo
Enddo

Call mvmult(C_x,Psi_x_Inc2,Psi_x,Sout,Sout)
Call mvmult(C_y,Psi_y_Inc2,Psi_y,Sout,Sout)

Qvec = Qvec + x_neg*2d0*Psi_x/Delta_X(i,j) + &
    y_neg*2d0*Psi_y/Delta_Y(i,j)

! Solve M*psi=q

If (m.eq.2) Then
    Continue
Endif

Call CG(Mat,Qvec,Psi,Sout,Count)

mat_count(m,i,j) = mat_count(m,i,j) + count

Psi(1) = dmax1(Psi(1),0d0)

!*********************************************************************

! Save old x-exiting coefficients

Allocate (Psi_x_Inc_Old(2**(Res_x-1)))

Psi_x_Inc_Old = Psi_x_Inc

! Save old y-exiting coefficients

Allocate (Psi_y_Inc_Next_Old(i)%Head)
Psi_y_Inc_Next(i)%ptr => Psi_y_Inc_Next(i)%Head
Psi_y_Inc_Next_Old(i)%ptr => Psi_y_Inc_Next_Old(i)%Head
Nullify (Psi_y_Inc_Next_Old(i)%ptr%next)
Psi_y_Inc_Next_Old(i)%Head%value = Psi_y_Inc_Next(i)%Head%value
Do
    If (Associated(Psi_y_Inc_Next(i)%ptr%next)) Then
        Psi_y_Inc_Next(i)%ptr => Psi_y_Inc_Next(i)%ptr%next
        Allocate (Psi_y_Inc_Next_Old(i)%ptr%next)
        Psi_y_Inc_Next_Old(i)%ptr => Psi_y_Inc_Next_Old(i)%ptr%next
        Nullify (Psi_y_Inc_Next_Old(i)%ptr%next)
        Psi_y_Inc_Next_Old(i)%ptr%value = Psi_y_Inc_Next(i)%ptr%value
        Cycle
    Else
        Exit
    Endif
Enddo

! Increment scalar flux

Phi(i,j) = Phi(i,j) + Pi*Omega(pol)*Psi(1)
```

```
! Save diamond x-exiting coefficients

Deallocate (Psi_y_Inc, Psi_x_Inc, Sig, C_x, C_y, Qvec, Mat, Svec)
Deallocate (Psi_x, Psi_y, Psi_x_Inc2, Psi_y_Inc2)

If (Associated(Psi_y_Inc_Next(i)%Head)) Then
    Do
        If (Associated(Psi_y_Inc_Next(i)%Head%next)) Then
            Psi_y_Inc_Next(i)%Ptr => Psi_y_Inc_Next(i)%Head
            Psi_y_Inc_Next(i)%Head => Psi_y_Inc_Next(i)%Head%next
            Deallocate (Psi_y_Inc_Next(i)%Ptr)
            Nullify (Psi_y_Inc_Next(i)%Ptr)
            Cycle
        Else
            Deallocate (Psi_y_Inc_Next(i)%Head)
            Nullify (Psi_y_Inc_Next(i)%Head)
            Exit
        Endif
    Enddo
Endif

Allocate (Psi_x_Inc(2**(Res-1)))

Psi_x_Inc(1) = 2d0*Psi(1) - Psi_x_Inc_Old(1)
ind1 = 2
ind2 = 2
Do r=2,Res
    Do k=2**(r-2)*(m-1)+1,2**(r-2)*m
        If (Associated(List(m,i,j)%Head)) Then
            If (inlist(List(m,i,j)%Head,r,k)) Then
                If (ind1.le.2**(Res_x-1)) Psi_x_Inc(ind1) = &
                    -Psi_x_Inc_Old(ind1)
                If (ind1.gt.2**(Res_x-1)) Psi_x_Inc(ind1) = 0d0
                ind1 = ind1 + 1
                Cycle
            Else
                If (ind1.le.2**(Res_x-1)) Psi_x_Inc(ind1) = &
                    2d0*Psi(ind2) - Psi_x_Inc_Old(ind1)
                If (ind1.gt.2**(Res_x-1)) Psi_x_Inc(ind1) = &
                    2d0*Psi(ind2)
                ind1 = ind1 + 1
                ind2 = ind2 + 1
                Cycle
            Endif
        Else
            If (ind1.le.2**(Res_x-1)) Psi_x_Inc(ind1) = &
                2d0*Psi(ind2) - Psi_x_Inc_Old(ind1)
            If (ind1.gt.2**(Res_x-1)) Psi_x_Inc(ind1) = 2d0*Psi(ind2)
            ind1 = ind1 + 1
            ind2 = ind2 + 1
            Cycle
        Endif
    Enddo
Enddo

! Save x-exiting boundary coefficients

If (m.eq.1) Then
    If (i.eq.X_Cells) Then
        If (Associated(Psi_Right_1(j)%Head)) &
            Nullify(Psi_Right_1(j)%Head)
        Allocate(Psi_Right_1(j)%Head)
        Nullify(Psi_Right_1(j)%Head%Next)
        Psi_Right_1(j)%Ptr => Psi_Right_1(j)%Head
        Psi_Right_1(j)%Ptr%Value = Psi_x_Inc(1)
        Do r=2,Res
```

```
            Do k=2**(r-1),2**(r-2)+1,-1
                Allocate(Psi_Right_1(j)%Ptr%Next)
                Psi_Right_1(j)%Ptr => Psi_Right_1(j)%Ptr%Next
                Nullify(Psi_Right_1(j)%Ptr%Next)
                Psi_Right_1(j)%Ptr%Value = -Psi_x_Inc(k)
            Enddo
        Enddo
        Res_Right_1(j) = Res
    Endif
Elseif (m.eq.2) Then
    If (i.eq.1) Then
        If (Associated(Psi_Left_1(j)%Head)) &
            Nullify(Psi_Left_1(j)%Head)
        Allocate(Psi_Left_1(j)%Head)
        Nullify(Psi_Left_1(j)%Head%Next)
        Psi_Left_1(j)%Ptr => Psi_Left_1(j)%Head
        Psi_Left_1(j)%Ptr%Value = Psi_x_Inc(1)
        Do r=2,Res
            Do k=2**(r-1),2**(r-2)+1,-1
                Allocate(Psi_Left_1(j)%Ptr%Next)
                Psi_Left_1(j)%Ptr => Psi_Left_1(j)%Ptr%Next
                Nullify(Psi_Left_1(j)%Ptr%Next)
                Psi_Left_1(j)%Ptr%Value = -Psi_x_Inc(k)
            Enddo
        Enddo
        Res_Left_1(j) = Res
    Endif
Elseif (m.eq.3) Then
    If (i.eq.1) Then
        If (Associated(Psi_Left_2(j)%Head)) &
            Nullify(Psi_Left_2(j)%Head)
        Allocate(Psi_Left_2(j)%Head)
        Nullify(Psi_Left_2(j)%Head%Next)
        Psi_Left_2(j)%Ptr => Psi_Left_2(j)%Head
        Psi_Left_2(j)%Ptr%Value = Psi_x_Inc(1)
        Do r=2,Res
            Do k=2**(r-1),2**(r-2)+1,-1
                Allocate(Psi_Left_2(j)%Ptr%Next)
                Psi_Left_2(j)%Ptr => Psi_Left_2(j)%Ptr%Next
                Nullify(Psi_Left_2(j)%Ptr%Next)
                Psi_Left_2(j)%Ptr%Value = -Psi_x_Inc(k)
            Enddo
        Enddo
        Res_Left_2(j) = Res
    Endif
Elseif (m.eq.4) Then
    If (i.eq.X_Cells) Then
        If (Associated(Psi_Right_2(j)%Head)) &
            Nullify(Psi_Right_2(j)%Head)
        Allocate(Psi_Right_2(j)%Head)
        Nullify(Psi_Right_2(j)%Head%Next)
        Psi_Right_2(j)%Ptr => Psi_Right_2(j)%Head
        Psi_Right_2(j)%Ptr%Value = Psi_x_Inc(1)
        Do r=2,Res
            Do k=2**(r-1),2**(r-2)+1,-1
                Allocate(Psi_Right_2(j)%Ptr%Next)
                Psi_Right_2(j)%Ptr => Psi_Right_2(j)%Ptr%Next
                Nullify(Psi_Right_2(j)%Ptr%Next)
                Psi_Right_2(j)%Ptr%Value = -Psi_x_Inc(k)
            Enddo
        Enddo
        Res_Right_2(j) = Res
    Endif
Endif

! Save diamond y-exiting coefficients
```

```
Allocate (Psi_y_Inc_Next(i)%Head)
Nullify (Psi_y_Inc_Next(i)%Head%next)
Psi_y_Inc_Next(i)%Ptr => Psi_y_Inc_Next(i)%Head
Psi_y_Inc_Next_Old(i)%Ptr => Psi_y_Inc_Next_Old(i)%Head
Psi_y_Inc_Next(i)%Tail => Psi_y_Inc_Next(i)%Ptr
Psi_y_Inc_Next(i)%Ptr%value = 2d0*Psi(1) - &
    Psi_y_Inc_Next_Old(i)%Ptr%value
ind2 = 2
Do r=2,Res
    Do k=2**(r-2)*(m-1)+1,2**(r-2)*m
        If (Associated(List(m,i,j)%Head)) Then
            If (inlist(List(m,i,j)%Head,r,k)) Then
                Allocate (Psi_y_Inc_Next(i)%Ptr%next)
                Psi_y_Inc_Next(i)%Ptr => Psi_y_Inc_Next(i)%Ptr%next
                Nullify (Psi_y_Inc_Next(i)%Ptr%next)
                If (Associated(Psi_y_Inc_Next_Old(i)%Ptr%next)) Then
                    Psi_y_Inc_Next_Old(i)%Ptr => &
                        Psi_y_Inc_Next_Old(i)%Ptr%next
                Else
                    Psi_y_Inc_Next_Old(i)%Ptr%value = 0d0
                Endif
                Psi_y_Inc_Next(i)%Ptr%value = &
                    -Psi_y_Inc_Next_Old(i)%Ptr%value
                Cycle
            Else
                Allocate (Psi_y_Inc_Next(i)%Ptr%next)
                Psi_y_Inc_Next(i)%Ptr => Psi_y_Inc_Next(i)%Ptr%next
                Nullify (Psi_y_Inc_Next(i)%Ptr%next)
                If (Associated(Psi_y_Inc_Next_Old(i)%Ptr%next)) Then
                    Psi_y_Inc_Next_Old(i)%Ptr => &
                        Psi_y_Inc_Next_Old(i)%Ptr%next
                Else
                    Psi_y_Inc_Next_Old(i)%Ptr%value = 0d0
                Endif
                Psi_y_Inc_Next(i)%Ptr%value = &
                    2d0*Psi(ind2) - Psi_y_Inc_Next_Old(i)%Ptr%value
                ind2 = ind2 + 1
                Cycle
            Endif
        Else
            Allocate (Psi_y_Inc_Next(i)%Ptr%next)
            Psi_y_Inc_Next(i)%Ptr => Psi_y_Inc_Next(i)%Ptr%next
            Nullify (Psi_y_Inc_Next(i)%Ptr%next)
            If (Associated(Psi_y_Inc_Next_Old(i)%Ptr%next)) Then
                Psi_y_Inc_Next_Old(i)%Ptr => &
                    Psi_y_Inc_Next_Old(i)%Ptr%next
            Else
                Psi_y_Inc_Next_Old(i)%Ptr%value = 0d0
            Endif
            Psi_y_Inc_Next(i)%Ptr%value = &
                2d0*Psi(ind2) - Psi_y_Inc_Next_Old(i)%Ptr%value
            ind2 = ind2 + 1
            Cycle
        Endif
    Enddo
Enddo

!**********************************************************************

If (trim(in_file).eq."test_box".and.i.eq.25.and.j.eq.15.and.m.eq.1) &
    Then
    Open (unit=99, file="thresh "//trim(in_file)//" &
        (R="//trim(adjustl(out_res))//", &
        eps="//trim(adjustl(out_string))//").txt")
    Do r=1,Size(Psi_x_Inc)
        Write (99,*) Psi_x_Inc(r)
    Enddo
```

```
            Close (99)
        Endif

        !**********************************************************************

        Deallocate(Psi)
        Res_x = Res
        Res_y(i) = Res

        ! Nullify old x-incident coefficients

        Deallocate (Psi_x_Inc_Old)

        ! Nullify old y-incident coefficients

        If (Associated(Psi_y_Inc_Next_Old(i)%Head)) Then
            Do
                If (Associated(Psi_y_Inc_Next_Old(i)%Head%next)) Then
                    Psi_y_Inc_Next_Old(i)%Ptr => Psi_y_Inc_Next_Old(i)%Head
                    Psi_y_Inc_Next_Old(i)%Head => &
                        Psi_y_Inc_Next_Old(i)%Head%next
                    Deallocate (Psi_y_Inc_Next_Old(i)%Ptr)
                    Nullify (Psi_y_Inc_Next_Old(i)%Ptr)
                    Cycle
                Else
                    Deallocate (Psi_y_Inc_Next_Old(i)%Head)
                    Nullify (Psi_y_Inc_Next_Old(i)%Head)
                    Exit
                Endif
            Enddo
        Endif

    Enddo

    Deallocate(Psi_x_Inc)

    ! Save y-exiting boundary coefficients

    Do i=1,X_Cells
        Allocate(Psi_Temp(2**(Res_y(i)-1)))
        Psi_Temp = 0d0
        Psi_y_Inc_Next(i)%Ptr => Psi_y_Inc_Next(i)%Head
        Do n=1,2**(Res_y(i)-1)
            Psi_Temp(n) = Psi_y_Inc_Next(i)%Ptr%value
            If (Associated(Psi_y_Inc_Next(i)%Ptr%next)) Then
                Psi_y_Inc_Next(i)%Ptr => Psi_y_Inc_Next(i)%Ptr%next
            Else
                Exit
            Endif
        Enddo
        If (m.eq.1) Then
            If (j.eq.Y_Cells) Then
                If (Associated(Psi_Top_2(i)%Head)) Nullify(Psi_Top_2(i)%Head)
                Allocate(Psi_Top_2(i)%Head)
                Nullify(Psi_Top_2(i)%Head%Next)
                Psi_Top_2(i)%Ptr => Psi_Top_2(i)%Head
                Psi_Top_2(i)%Ptr%Value = Psi_Temp(1)
                Do r=2,Res_y(i)
                    Do k=2**(r-1),2**(r-2)+1,-1
                        Allocate(Psi_Top_2(i)%Ptr%Next)
                        Psi_Top_2(i)%Ptr => Psi_Top_2(i)%Ptr%Next
                        Nullify(Psi_Top_2(i)%Ptr%Next)
                        Psi_Top_2(i)%Ptr%Value = -Psi_Temp(k)
                    Enddo
                Enddo
                Res_Top_2(i) = Res_y(i)
            Endif
        Elseif (m.eq.2) Then
            If (j.eq.Y_Cells) Then
                If (Associated(Psi_Top_1(i)%Head)) Nullify(Psi_Top_1(i)%Head)
                Allocate(Psi_Top_1(i)%Head)
                Nullify(Psi_Top_1(i)%Head%Next)
                Psi_Top_1(i)%Ptr => Psi_Top_1(i)%Head
                Psi_Top_1(i)%Ptr%Value = Psi_Temp(1)
                Do r=2,Res_y(i)
                    Do k=2**(r-1),2**(r-2)+1,-1
                        Allocate(Psi_Top_1(i)%Ptr%Next)
                        Psi_Top_1(i)%Ptr => Psi_Top_1(i)%Ptr%Next
                        Nullify(Psi_Top_1(i)%Ptr%Next)
                        Psi_Top_1(i)%Ptr%Value = -Psi_Temp(k)
                    Enddo
                Enddo
                Res_Top_1(i) = Res_y(i)
            Endif
        Elseif (m.eq.3) Then
            If (j.eq.1) Then
                If (Associated(Psi_Bottom_2(i)%Head)) &
                    Nullify(Psi_Bottom_2(i)%Head)
                Allocate(Psi_Bottom_2(i)%Head)
                Nullify(Psi_Bottom_2(i)%Head%Next)
                Psi_Bottom_2(i)%Ptr => Psi_Bottom_2(i)%Head
                Psi_Bottom_2(i)%Ptr%Value = Psi_Temp(1)
                Do r=2,Res_y(i)
                    Do k=2**(r-1),2**(r-2)+1,-1
                        Allocate(Psi_Bottom_2(i)%Ptr%Next)
                        Psi_Bottom_2(i)%Ptr => Psi_Bottom_2(i)%Ptr%Next
                        Nullify(Psi_Bottom_2(i)%Ptr%Next)
                        Psi_Bottom_2(i)%Ptr%Value = -Psi_Temp(k)
                    Enddo
                Enddo
                Res_Bottom_2(i) = Res_y(i)
            Endif
        Elseif (m.eq.4) Then
            If (j.eq.1) Then
                If (Associated(Psi_Bottom_1(i)%Head)) &
                    Nullify(Psi_Bottom_1(i)%Head)
                Allocate(Psi_Bottom_1(i)%Head)
                Nullify(Psi_Bottom_1(i)%Head%Next)
                Psi_Bottom_1(i)%Ptr => Psi_Bottom_1(i)%Head
                Psi_Bottom_1(i)%Ptr%Value = Psi_Temp(1)
                Do r=2,Res_y(i)
                    Do k=2**(r-1),2**(r-2)+1,-1
                        Allocate(Psi_Bottom_1(i)%Ptr%Next)
                        Psi_Bottom_1(i)%Ptr => Psi_Bottom_1(i)%Ptr%Next
                        Nullify(Psi_Bottom_1(i)%Ptr%Next)
                        Psi_Bottom_1(i)%Ptr%Value = -Psi_Temp(k)
                    Enddo
                Enddo
                Res_Bottom_1(i) = Res_y(i)
            Endif
        Endif
        Deallocate(Psi_Temp)
    Enddo

Enddo

! Nullify y-exiting coefficient list

Do i=1,X_Cells
    If (Associated(Psi_y_Inc_Next(i)%Head)) Then
        Do
            If (Associated(Psi_y_Inc_Next(i)%Head%next)) Then
                Psi_y_Inc_Next(i)%Ptr => Psi_y_Inc_Next(i)%Head
                Psi_y_Inc_Next(i)%Head => Psi_y_Inc_Next(i)%Head%next
```

```
                        Deallocate (Psi_y_Inc_Next(i)%Ptr)
                        Nullify (Psi_y_Inc_Next(i)%Ptr)
                        Cycle
                    Else
                        Deallocate (Psi_y_Inc_Next(i)%Head)
                        Nullify (Psi_y_Inc_Next(i)%Head)
                        Exit
                    Endif
                Enddo
            Endif
        Enddo

    Enddo

Enddo

!************************************************************************************
!
!   Calculate error vectors and determine convergence
!
!************************************************************************************

Cnt_Tot = Cnt_Tot + Count

If (not(scat)) exit

Sum_1 = 0d0
Do j=1,Y_Cells
    Sum_2 = 0d0
    Do i=1,X_Cells
        Sum_2 = Sum_2 + (Phi(i,j)-Phi_Old(i,j))
    Enddo
    Sum_1 = Sum_1 + Sum_2**2d0
Enddo
Norm_1 = dsqrt(Sum_1/dble(X_Cells))

Sum_1 = 0d0
Do j = 1,Y_Cells
    Sum_2 = 0d0
    Do i = 1,X_Cells
        Sum_2 = Sum_2 + (Phi_Old(i,j)-Phi_Old_2(i,j))
    Enddo
    Sum_1 = Sum_1 + Sum_2**2d0
Enddo
Norm_2 = dsqrt(Sum_1/dble(X_Cells))

Sum_1 = 0d0
Do j = 1,Y_Cells
    Sum_2 = 0d0
    Do i = 1,X_Cells
        Sum_2 = Sum_2 + Phi(i,j)
    Enddo
    Sum_1 = Sum_1 + Sum_2**2d0
Enddo
Norm_3 = dsqrt(Sum_1/dble(X_Cells))

Rho = 0d0
If (Norm_2.ne.0d0) Rho = Norm_1/Norm_2

te = etime(ta)

If (detail) Write(*,*) &
    "***********************************************************************"
If (Norm_3.ne.0d0) Write (*,'(i5,f11.6,2es16.6,i8,i12)') &
    iter, Rho, Norm_1/((1d0-Rho)*Norm_3), Norm_3, int(te), Count
If (detail) Write(*,*) &
    "***********************************************************************"
    If (detail) Write(20,*) &
        "*********************************************************************"
    If (Norm_3.ne.0d0) Write (20,'(i5,f11.6,2es16.6,i8,i12)') &
        iter, Rho, Norm_1/((1d0-Rho)*Norm_3), Norm_3, int(te), Count
    If (detail) Write(20,*) &
        "*********************************************************************"
    If (iter.ge.2.and.(Norm_1.le.Eps*(1d0-Rho)*Norm_3)) Exit

    If (iter.eq.MaxIt) Write (*,*) "The source iteration did not converge!"
    If (iter.eq.MaxIt) Write (20,*) "The source iteration did not converge!"


Enddo

!*************************************************************************************
!
!   Write output files
!
!*************************************************************************************

Write (*,*) "Writing Output Files..."
Write (20,*) "Writing Output Files..."

Open (unit=10, file="output "//trim(in_file)//" (R="//trim(adjustl(out_res))//", &
    eps="//trim(adjustl(out_string))//").dat")

Write(10,*) 'Variables = "X", "Y", "Scalar Flux"'
Write(10,*) 'Zone i=',Y_Cells,', j=',X_Cells,', F=point'

Do i=1,X_Cells
    If (i.eq.1) x = Delta_X(1,1)/2d0
    If (i.ne.1) x = x + (Delta_X(i-1,1)+Delta_X(i,1))/2d0
    Do j=1,Y_Cells
        If (j.eq.1) y = Delta_Y(1,i)/2d0
        If (j.ne.1) y = y + (Delta_Y(i,j-1)+Delta_Y(i,j))/2d0
        Write (10,'(3es16.6)') x, y, Phi(i,j)
    Enddo
Enddo

Close (10)

!*************************************************************************************

Open (unit=10, file="log "//trim(in_file)//" (R="//trim(adjustl(out_res))//", &
    eps="//trim(adjustl(out_string))//").dat")

Write(10,*) 'Variables = "X", "Y", "Scalar Flux"'
Write(10,*) 'Zone i=',Y_Cells,', j=',X_Cells,', F=point'

Min_Phi = maxval(Phi)
Do i=1,X_Cells
    Do j=1,Y_Cells
        If (Phi(i,j).lt.Min_Phi.and.Phi(i,j).gt.0d0) Min_Phi = Phi(i,j)
    Enddo
Enddo

Do i=1,X_Cells
    If (i.eq.1) x = Delta_X(1,1)/2d0
    If (i.ne.1) x = x + (Delta_X(i-1,1)+Delta_X(i,1))/2d0
    Do j=1,Y_Cells
        If (j.eq.1) y = Delta_Y(i,1)/2d0
        If (j.ne.1) y = y + (Delta_Y(i,j-1)+Delta_Y(i,j))/2d0
        If (Phi(i,j).le.0d0) Then
            Write (10,'(3es16.6)') x, y, dlog10(Min_Phi)
        Else
            Write (10,'(3es16.6)') x, y, dlog10(Phi(i,j))
        Endif
    Enddo
Enddo

Close (10)

!**************************************************************************************************

If (trim(in_file).eq."hohlraum") Then

Open (unit=10, file="target "//trim(in_file)//" (R="//trim(adjustl(out_res))//", &
    eps="//trim(adjustl(out_string))//").txt")

Do i=1,X_Cells
    If (i.eq.1) x = Delta_X(1,1)/2d0
    If (i.ne.1) x = x + (Delta_X(i-1,1)+Delta_X(i,1))/2d0
    Do j=1,Y_Cells
        If (j.eq.1) y = Delta_Y(i,1)/2d0
        If (j.ne.1) y = y + (Delta_Y(i,j-1)+Delta_Y(i,j))/2d0
        If ((x.ge.0.45.and.x.le.0.45+Delta_X(i,j)).and.(y.ge.0.25.and.y.le.1.05)) &
            Write (10,'(3es16.6)') x, y, Phi(i,j)
    Enddo
Enddo

Do i=1,X_Cells
    If (i.eq.1) x = Delta_X(1,1)/2d0
    If (i.ne.1) x = x + (Delta_X(i-1,1)+Delta_X(i,1))/2d0
    Do j=1,Y_Cells
        If (j.eq.1) y = Delta_Y(i,1)/2d0
        If (j.ne.1) y = y + (Delta_Y(i,j-1)+Delta_Y(i,j))/2d0
        If ((y.ge.1.05-Delta_Y(i,j).and.y.le.1.05).and.(x.ge.0.45.and.x.le.0.85)) &
            Write (10,'(3es16.6)') x, y, Phi(i,j)
    Enddo
Enddo

Do i=1,X_Cells
    If (i.eq.1) x = Delta_X(1,1)/2d0
    If (i.ne.1) x = x + (Delta_X(i-1,1)+Delta_X(i,1))/2d0
    y = 0d0
    Do j=1,Y_Cells
        y = y + Delta_Y(i,j)
    Enddo
    Do j=Y_Cells,1,-1
        If (j.eq.Y_Cells) y = y - Delta_Y(i,Y_Cells)/2d0
        If (j.ne.Y_Cells) y = y - (Delta_Y(i,j)+Delta_Y(i,j+1))/2d0
        If ((x.ge.0.85-Delta_X(i,j).and.x.le.0.85).and.(y.ge.0.25.and.y.le.1.05)) &
            Write (10,'(3es16.6)') x, y, Phi(i,j)
    Enddo
Enddo

x = 0d0
Do i=1,X_Cells
    x = x + Delta_X(i,1)
Enddo

Do i=X_Cells,1,-1
    If (i.eq.X_Cells) x = x - Delta_X(X_Cells,X_Cells)/2d0
    If (i.ne.X_Cells) x = x - (Delta_X(i,1)+Delta_X(i+1,1))/2d0
    Do j=1,Y_Cells
        If (j.eq.1) y = Delta_Y(i,1)/2d0
        If (j.ne.1) y = y + (Delta_Y(i,j-1)+Delta_Y(i,j))/2d0
        If ((y.ge.0.25.and.y.le.0.25+Delta_Y(i,j)).and.(x.ge.0.45.and.x.le.0.85)) &
            Write (10,'(3es16.6)') x, y, Phi(i,j)
    Enddo
Enddo

Close (10)
```

```
Endif

!*************************************************************************************

If (trim(in_file).ne."hohlraum") Then

Open (unit=10, file="top "//trim(in_file)//" (R="//trim(adjustl(out_res))//", &
    eps="//trim(adjustl(out_string))//").txt")

x = 0.
Do i = 1,X_Cells
    If (i.eq.1) Then
        x = Delta_X(1,1)/2.
    Else
        x = x + (Delta_X(i-1,1)+Delta_X(i,1))/2.
    Endif
    Write (10,'(3es16.6)') x, Phi(i,Y_Cells)
Enddo

Close (10)

Endif

!*************************************************************************************

te = etime(ta)

Write (*,'("The program took ",i4," iterations and ran in ",f8.2," seconds.")') iter, te
Write (*,'("There were a total of ",i10, " unknowns calculated.")') Cnt_Tot
Write (20,'("The program took ",i4," iterations and ran in ",f8.2," seconds.")') iter, te
Write (20,'("There were a total of ",i10, " unknowns calculated.")') Cnt_Tot

Close (20)

Contains

!*****************************************************************

Logical Function inlist(List,r,k)

    Implicit None

    Integer, Intent(in) :: r, k

    Type(w_list), Pointer :: List, Ptr

    Ptr => List
    inlist = .false.
    Do
        If (Ptr%r.eq.r.and.Ptr%k.eq.k) Then
            inlist = .true.
            Exit
        Endif
        Ptr => Ptr%next
        If (not(associated(Ptr))) Exit
    Enddo

End Function inlist

!*****************************************************************

End Program CWWN
```

The following is the source code for the subroutine gauleg, which computes the Gauss-Legendre quadrature points and weights:

```
Subroutine Gauleg(n,x,w)

  Implicit None

  Integer :: NP,N,i,j,m,NQ
  Parameter (NP=130,NQ=5)
  Real*8 :: X(NP),W(NP),EPS,p1,p2,p3,pp,xl,xm,z,z1, pi
  Parameter (EPS=3.d-14)

  pi = 4d0*datan(1d0)

  m=(n+1)/2
  xm=0d0
  xl=1d0
  Do i=1,m
     z=dcos(pi*(dble(i)-.25d0)/(dble(n)+.5d0))
1    continue
     p1=1d0
     p2=0d0
     Do j=1,n
        p3=p2
        p2=p1
        p1=((2.d0*dble(j)-1.d0)*z*p2-(dble(j)-1.d0)*p3)/dble(j)
     Enddo
     pp=n*(z*p1-p2)/(z*z-1d0)
     z1=z
     z=z1-p1/pp
     If (dabs(z-z1).gt.EPS) goto 1
     x(i)=xm-xl*z
     x(n+1-i)=xm+xl*z
     w(i)=2.d0*xl/((1d0-z*z)*pp*pp)
     w(n+1-i)=w(i)
  Enddo

End Subroutine Gauleg
```

The following is the source code for the function mu:

```
Real*8 Function mu(r,k)

  Implicit None

  Real*8 :: pi
  Integer, Intent(in) :: r, k

  pi = 4d0*datan(1d0)
  mu = 2d0**dble(r)/(2d0*pi)*(dsin(2d0**dble(1-r)*pi*dble(k))- &
     dsin(2d0**dble(1-r)*pi*dble(k-1)))

End Function mu
```

The following is the source code for the function eta:

```
Real*8 Function eta(r,k)

  Implicit None
  Real*8 :: pi
  Integer, Intent(in) :: r, k

  pi = 4d0*datan(1d0)
  eta = 2d0**dble(r)/(2d0*pi)*(-dcos(2d0**dble(1-r)*pi*dble(k))+ &
     dcos(2d0**dble(1-r)*pi*dble(k-1)))

End Function eta
```

The following is the source code for the function alpha:

```
Real*8 Function alpha(r,k)

  Implicit None

  Real*8 :: pi
  Integer, Intent(in) :: r, k

  pi = 4d0*datan(1d0)
  alpha = 2d0**(dble(r)/2d0)/pi*(-dsin(2d0**dble(1-r)*pi*dble(k))+ &
     2d0*dsin(2d0**dble(1-r)*pi*(dble(k)-0.5d0))-dsin(2d0**dble(1-r)*pi*dble(k-1)))

End Function alpha
```

The following is the source code for the function beta:

```
Real*8 Function beta(r,k)

  Implicit None

  Real*8 :: pi
  Integer, Intent(in) :: r, k

  pi = 4d0*datan(1d0)
  beta = 2d0**(dble(r)/2d0)/pi*(dcos(2d0**dble(1-r)*pi*dble(k))- &
     2d0*dcos(2d0**dble(1-r)*pi*(dble(k)-0.5d0))+dcos(2d0**dble(1-r)*pi*dble(k-1)))

End Function beta
```

The following is the source code for the function A:

```
Real*8 Function A(r,k,n,p)

  Implicit None

  Real*8 :: pi, u_l, u_m, u_r, l_l
  Integer :: r, k, n, p, ures, uind, lres, lind

  pi = 4d0*datan(1d0)

  If (n.gt.r) Then
     ures = r; uind = k; lres = n; lind = p;
  Else
     ures = n; uind = p; lres = r; lind = k;
  Endif

  u_l = 2d0**(-ures)*pi*(uind-1d0)
  u_m = 2d0**(-ures)*pi*(uind-.5d0)
  u_r = 2d0**(-ures)*pi*uind
  l_l = 2d0**(-lres)*pi*(lind-1d0)

  If (l_l.ge.u_l.and.l_l.lt.u_m) Then
     A = (2d0**(ures/2d0+lres/2d0))/(2d0*pi)*(-dsin(2d0**(1-lres)*pi*lind)+ &
        2d0*dsin(2d0**(1-lres)*pi*(lind-0.5))-dsin(2d0**(1-lres)*pi*(lind-1)))
  Elseif (l_l.ge.u_m.and.l_l.lt.u_r) Then
     A = -(2d0**(ures/2d0+lres/2d0))/(2d0*pi)*(-dsin(2d0**(1-lres)*pi*lind)+ &
        2d0*dsin(2d0**(1-lres)*pi*(lind-0.5))-dsin(2d0**(1-lres)*pi*(lind-1)))
  Else
     A = 0d0
  Endif

End Function A
```

The following is the source code for the function B:

```
Real*8 Function B(r,k,n,p)

  Implicit None

  Real*8 :: pi, u_l, u_m, u_r, l_l
  Integer :: r, k, n, p, ures, uind, lres, lind

  pi = 4d0*datan(1d0)

  If (n.gt.r) Then
     ures = r; uind = k; lres = n; lind = p;
  Else
     ures = n; uind = p; lres = r; lind = k;
  Endif

  u_l = 2d0**(-ures)*pi*(uind-1d0)
  u_m = 2d0**(-ures)*pi*(uind-.5d0)
  u_r = 2d0**(-ures)*pi*uind
  l_l = 2d0**(-lres)*pi*(lind-1d0)

  If (l_l.ge.u_l.and.l_l.lt.u_m) Then
     B = (2d0**(ures/2d0+lres/2d0))/(2d0*pi)*(dcos(2d0**(1-lres)*pi*lind)- &
        2d0*dcos(2d0**(1-lres)*pi*(lind-0.5))+dcos(2d0**(1-lres)*pi*(lind-1)))
  Elseif (l_l.ge.u_m.and.l_l.lt.u_r) Then
     B = -(2d0**(ures/2d0+lres/2d0))/(2d0*pi)*(dcos(2d0**(1-lres)*pi*lind)- &
        2d0*dcos(2d0**(1-lres)*pi*(lind-0.5))+dcos(2d0**(1-lres)*pi*(lind-1)))
  Else
     B = 0d0
  Endif

End Function B
```

The following is the source code for the subroutine mvmult, which multiplies a vector and a matrix:

```
Subroutine mvmult(A,b,c,n,m)

  Implicit None

  Integer :: n, m, i, j
  Real*8 :: A(n,m), b(m), c(n)

  Do i=1,n
     c(i) = 0d0
     Do j=1,m
        c(i) = c(i) + A(i,j)*b(j)
     Enddo
  Enddo

End Subroutine
```

The following is the source code for the subroutine Dot, which computes the Euclidean inner product of two vectors:

```
Subroutine dot(a,b,c,n)

  Implicit None

  Integer :: n, i
  Real*8 :: a(n), b(n), c

  c = 0d0
  Do i=1,n
     c = c + a(i)*b(i)
  Enddo

End Subroutine
```

The following is the source code for the subroutine CG, which uses the conjugate gradient method to solve a linear system:

```
Subroutine CG(A,b,x,n,cnt)

  Implicit None

  Integer :: n, i, j, k, cnt
  Real*8 :: A(n,n), b(n), x(n), mult(n), r(n), v(n), c, z(n), t, d, c1, c2

  cnt = cnt + n

  x = 1d0
  Call mvmult(A,x,mult,n,n)
  r = b - mult
  v = r
  Call dot(r,r,c,n)
  Do k=1,1000000
     Call dot(v,v,c1,n)
     If (dsqrt(c1).lt.1d-7) Exit
     Call mvmult(A,v,z,n,n)
     Call dot(v,z,c2,n)
     t = c/c2
     x = x + t*v
     r = r - t*z
     Call dot(r,r,d,n)
     If (d.lt.1d-7) Exit
     v = r + (d/c)*v
     c = d
  Enddo

End Subroutine CG
```

The following is an input file for the first box-in-box problem:

```
Regions:

2 2

Source:

0 0
0.25 0

Material Index:

2 2
2 2

Cells Per Region (x):

20 20
20 20

Cells per Region (y):

20 20
20 20

Width of Region:

1.0 1.0
1.0 1.0

Height of Region:

1.0 1.0
1.0 1.0

Incident Fluxes (id, left, right, top, bottom):

vac 0 0
vac 0 0
vac 0 0
vac 0 0

Iteration Parameters:

100 1d-7 1d-2

Angular Resolution:

1 4
```

APPENDIX D

In this appendix, we will discuss the use of linked lists in Fortran 90. A linked list consists of a user-defined data type that consists of data and a pointer to a memory location of the same data type. For instance, if one were to define the data type "data_type," storing an integer, in Fortran 90, the declaration would be:

```
Type data_type
   Integer :: Data
End Type data_type

Type(data_type) :: List
```

Thus, the variable "List%Data" stores an integer. If we wish to create a list of integer values, the declaration would be:

```
Type data_list
   Integer :: Data
   Type(data_list), Pointer :: Next
End Type data_list

Type(data_list), Pointer :: List, Ptr
```

Using this declaration, the data is stored in "List%Data," and the next element in the list is "List%Next." The pointer "Ptr" is the currently indicated element in the list. Since "List" is a pointer, however, it must be allocated before it can be used. The following code will create a list of five elements:

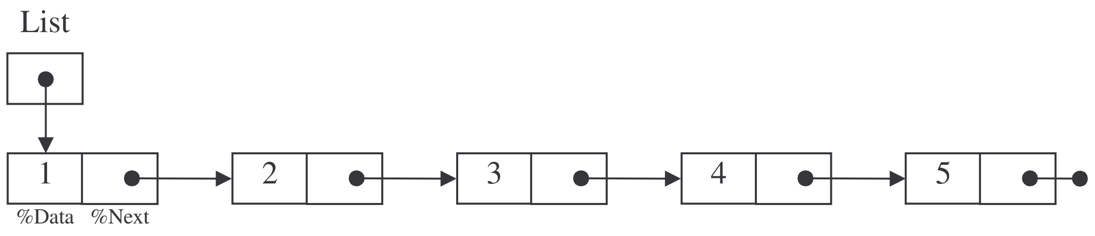
```
Allocate(List)
Nullify(List%Next)
Ptr => List
Do i=1,4
   Ptr%Data = i
   Allocate(Ptr%Next)
   Ptr => Ptr%Next
   Nullify(Ptr%Next)
Enddo
Ptr%Data = 5
```

Now, the list "List" consists of the following data:

```
List%Data = 1
List%Next%Data = 2
```

```
List%Next%Next%Data = 3
List%Next%Next%Next%Data = 4
List%Next%Next%Next%Next%Data = 5
```

This data structure can be better seen graphically as:



Thus, in essence, a linked list may be used as an array with variable length. Since each element is allocated separately, it has no set length and its size may be increased or decreased as one sees necessary. This feature is useful when, for instance, one wishes to store wavelet indices but does not know how many will ultimately need to be stored. Care must be taken when deleting a list, however. Simply nullifying the pointer "List" would only make the list inaccessible to the user, but the data would remain in the memory. Thus, to delete the list, one must nullify the pointers in the order that they were created. First, we will nullify "List," then "List%Next," then "List%Next%Next," and so on. A routine to do this would be:

```
Do
   If (Associated(List%Next)) Then
      Ptr => List
      List => List%Next
      Nullify(Ptr)
      Cycle
   Else
      Nullify(List)
      Exit
   Endif
Enddo
```

Another application of linked lists is for the array that saves the scaling and wavelet function coefficients in the y-direction for the sweep routine. This array has a set number of columns (the number of cells in the x-direction), but differs by column in the number of rows, since each cell may use a different number of wavelets to resolve the angular flux in that cell. Graphically, this can be seen as:

1	2	3	4	5	6	7	8	9	10
-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-			
	-	-	-	-	-	-			
	-	-							
	-	-							
	-	-							

This structure could be implemented by creating an array of lists. The declaration for this structure would be:

```
Type data_list
   Integer :: Data
   Type(data_list), Pointer :: Next
End Type data_list

Type list_array
   Type(data_list), Pointer :: Head, Ptr
End Type list_array

Type(list_array), Allocatable :: List(:)
```

To create the above array, we would first allocate the memory for the array, then fill each column as desired. For instance:

```
Allocate(List(10))

Do i=1,10
   Allocate(List(i)%Head)
   Nullify(List(i)%Head%Next)
   List(i)%Ptr => List(i)%Head
   Do j=1,2*i
      List(i)%Ptr%Data = j
      Allocate(List(i)%Ptr%Next)
      List(i)%Ptr => List(i)%Ptr%Next
      Nullify(List(i)%Ptr%Next)
   Enddo
   List(i)%Ptr%Data = 2*i+1
Enddo
```

The resulting array is

1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2
	3	3	3	3	3	3	3	3	3
	4	4	4	4	4	4	4	4	4
		5	5	5	5	5	5	5	5
		6	6	6	6	6	6	6	6
			7	7	7	7	7	7	7
			8	8	8	8	8	8	8
				9	9	9	9	9	9
				10	10	10	10	10	10
					11	11	11	11	11
					12	12	12	12	12
						13	13	13	13
						14	14	14	14
							15	15	15
							16	16	16
								17	17
								18	18
									19
									20

To delete this array, we must nullify each list column by column:

```
Do i=1,10
   Do
      If (Associated(List(i)%Head%Next)) Then
         List(i)%Ptr => List(i)%Head
         List(i)%Head => List(i)%Head%Next
         Nullify(List(i)%Ptr)
         Cycle
      Else
         Nullify(List(i)%Head)
         Exit
      Endif
   Enddo
Enddo
```

In the codes presented in this dissertation, several applications of linked lists are employed. In the S_N-W_N code, a list is used to store the indices of wavelets that have been thresholded in each cell in each quadrant. Thus, it is of type "List(Quad,i,j)%Head." In the CW-W_N code, a simple list is employed within each cell to store the wavelet indices that have been thresholded. Also, two arrays of lists, of the type “List(i)%Head” are used to store the y-incident and y-exit fluxes in the diamond difference approximation.

VITA

Name:	Aaron Michael Watson
Address:	149A Eastwood Drive, Clifton Park, New York 12065
Email Address:	aa.watson@hotmail.com
Education:	B.S., Nuclear Engineering, Texas A&M University, 2001
	M.S., Nuclear Engineering, Texas A&M University, 2002