ULTIMATE INTRINSIC SNR IN MAGNETIC RESONANCE IMAGING BY OPTIMIZING THE RF FIELD GENERATED BY INTERNAL COILS

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Imad Amin Abdel-Hafez

June 2000

ULTIMATE INTRINSIC SNR IN MAGNETIC RESONANCE IMAGING BY OPTIMIZING THE EM FIELD GENERATED BY INTERNAL COILS

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND

ELECTRONICS ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCES

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Imad Amin Abdel-Hafez

June 2000

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Ergin Atalar(Supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Ayhan Altıntaş (Co-supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. İrşadi Aksun

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Nevzat Gençer

Approved for the Institute of Engineering and Sciences:

Prof. Dr. Mehmet Baray
Director of Institute of Engineering and Sciences

ABSTRACT

ULTIMATE INTRINSIC SNR IN MAGNETIC RESONANCE IMAGING BY OPTIMIZING THE EM FIELD GENERATED BY INTERNAL COILS

Imad Amin Abdel-Hafez
M.S. in Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Ergin Atalar
June 2000

A method to find the ultimate intrinsic signal-to-noise ratio (ISNR) in a magnetic resonance imaging experiment is applied to a human body model. The method uses cylindrical wave expansion to represent an arbitrary electromagnetic field inside the body. This field is optimized to give the maximum possible ISNR for some point of interest from which the signal is received, and repeated for all points inside the body. Optimization is conducted by finding the set of coefficients associated with expansion modes that give the maximum ISNR. Application of this method enables the determination of the ultimate ISNR and the associated optimum electromagnetic field without the necessity of finding the receiving coil configuration needed to obtain the ultimate value of ISNR.

Results of this work can be used to examine the efficiency of already available commercial coils and how far they can be improved. Moreover, the solution can be used to determine the performance difference between internal and external Magnetic Resonance Imaging (MRI) coils. Finally, knowledge of the optimum electromagnetic field inside the human body can be used to find the coil configuration that can radiate this field by solving an inverse problem.

Keywords: Magnetic Resonance Imaging (MRI), wave equation, cylindrical wave representation, constraint optimization

ÖZET

MANYETİK REZONANS GÖRÜNTÜLEMEDEKİ NİHAİ İÇSEL SİNYAL/GÜRÜLTÜ ORANININ DAHİLİ BOBİNLERCE OLUŞTURULAN ELEKTROMANYETİK ALANIN OPTIMIZASYONU İLE BELİRLENMESİ

Imad Amin Abdel-Hafez

Elektrik ve Elektronik Mühendisli~~

gi Bölümü Yüksek Lisans

Tez Yöneticisi: Yrd. Doç. Dr. Ergin Atalar

Haziran 2000

Manyetik rezonans görüntüleme deneyindeki nihai içsel sinyal gürütlü oranı'nı (SGO) bulmak için kullanılan bir yönetem insan vücudu modeline uygulanmıştır. Bu yöntemde vücut içindeki rasgele elektromanyetik alanı ifade etmek için silindirik dalga açlımı kullanılmaktadır. Bu alan sinyalin geldiği alandaki bir noktanın mümkün olan maksimum içsel SGO'ını bulmak için optimize edilir ve bu işlem vücuttaki her nokta için tekrarlanır. Optimizasyon maksimum içsel SGO'ı veren açılım modlarıyla ilintili katsayı kümesini bularak yapılır. Bu yöntem nihai içsel SGO ve ilgili optimum elektromanyetik alanı alıcı bobin konfigürasyonunu bulma gereği olmaksızın belirlemeyi mümkün kılar.

Bu çalışmanın sonuçları halıhazırda mevcut olan ticari bobinlerin verimliliğini ve daha ne kadar geliştirilebileceklerini incelemede kullanılabilir. Ayrıca, önerilen çözüm iç ve dış Manyetik Rezonans Görüntüleme (MRG) bobinlerinin performans fakını belirlemede kullanılabilir. Son olarak, insan vücudundaki optimum elektromanyetik alanın bilgisi bir ters problemin çözümüyle bu alanı yapacak bobin konfigürasyonunu bulamada kullanılabilir.

Anahtar Kelimeler: Manyetik Rezonans Görüntüleme (MRG), dalga denklemi, silindrik dalga gösterimi, kısıtli eniyileme

ACKNOWLEDGMENTS

I would like to use this opportunity to express my deep gratitude to Dr. Ergin Atalar for his supervision, guidance, suggestions and encouragement throughout the development of this thesis.

I would like to thank Prof. Dr. Ayhan Altıntaş for his valuable co-supervision of this work and for the experience he made available for us. I would like also to thank Prof. Dr. İrşadi Aksun and Assoc. Prof. Dr. Nevzat Gençer for reading and commenting on the thesis.

Finally, I would like to thank my family and especially my parents for their continuous support along my studies.

Contents

List of Figures

List of Tables

To my Family ...

Chapter 1

INTRODUCTION

Magnetic Resonance Imaging, or MRI, is a widely used tomographic imaging technique in medicine for high-resolution imaging of internal body parts without a surgical operation [1].

In MRI, patient is placed inside a large cavity that produces a strong magnetic field. The magnetized body, immediately after application of a radiofrequency (RF) signal, responds with a weak RF signal. This phenomenon is known as Nuclear Magnetic Resonance. This weak RF signal is picked up with a receiver coil placed on the surface of the body.

The receiver coil not only picks up the signal, but also picks up the noise that distorts the signal. Small coils pick up small amount of noise, but they have to be placed close to the point of interest.

In MRI, for each point of interest in the body a different RF coil is optimum. Because of this, seperate coils for head, shoulder, neck, spine, heart, pelvis, arm

and legs are being developed.For point of interest inside the body, internal coils are being developed. Probes are placed into rectum to increase signal-to-noise SNR of the prostate images [12]. Probes placed in the esophogus for imaging esophogeal wall and aorta [13]. Some probes are being developed for imaging atherosclerotic plaques by placing the probes inside the blood vessels[10,14-22]. Although investigators are developing various internal and external coils, the performance of the coils could not be compared properly. Among the problems in the comparison of the external and internal coils are: i)many coil configurations exist for different clinical applications ii)performance of a given coil depends on the size of the body iii)tuning, matching and proper placement of the coil have their effect on coil performance iv)even if the two coils are placed simultaneously, mutual interaction results in degraded image quality.

In this work, ultimate intrinsic SNR (ISNR) of an internal coil is investigated. ISNR is a quantity that is independent of signal processing algorithms involved in imaging of body or any different parameters concerning the imaging device. It is a quantity that is determined by body geometry and physical characteristics only. This quantity can be used further to find the SNR of an image that is produced by a specific device by considering that device's own parameters and applying it to the ISNR. Results obtained from this work can be used for comparison with ultimate ISNR of an external coil. This comparison was not possible before because of the forementioned problems above. The approach followed here is to find the electromagnetic field that achieves this ISNR, regardless of the coil configuration that generates this field. An inverse problem of finding the appropriate coil for this field may be investigated as a further work.

Once the ultimate ISNR value is known for an internal coil, it can be compared to the ultimate ISNR value of the external coil. In addition, the ultimate ISNR can be used as a basis for the performance of the internal coil to test whether there is room for further improvement in their performance.

Chapter 2 of this thesis gives a brief background information on this problem. Chapter 3 describes the formulation and solution of the problem. Chapter 4 is devoted to the numerical methods used to solve this problem. Chapter 5 gives the simulation results. Finally, conclusion and future work are discussed in the last chapter. Throughout the thesis, an $e^{j\omega t}$ time variation is assumed and suppressed for the electromagnetic fields.

Chapter 2

BACKGROUND

In a magnetic resonance imaging (MRI) experiment a patient is laid horizontally over a table which enters longitudinally into a magnet cavity as seen in Fig. 2.1. This magnet exposes an extremely uniform (non-uniformity of 1 part per million) DC magnetic field B_0 to the body [1]. Application of this magnetic field results in magnetization in the human body. This magnetization can be used to collect information from inside the body. When an RF magnetic field signal is applied to the body, a phenomenon known as "Magnetic Resonance" results in a reflection which is received by a coil (antenna) that converts this electromagnetic signal to a voltage signal. For the magnetic resonance phenomenon to take place, the RF signal has to be at a specific frequency called the Larmor frequency. This frequency is related to the DC component of the applied magnetic field by the relation

$$\omega_0 = \gamma B_0 \tag{2.1}$$

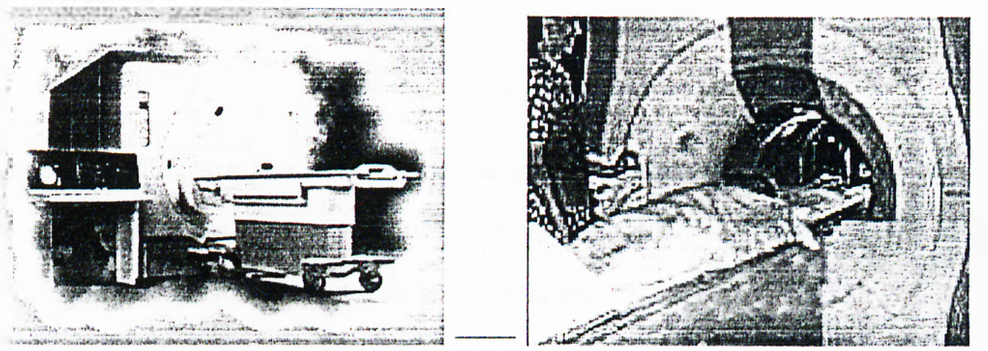

Figure 2.1: Two representative pictures of an MRI scanner.

where ω_0 is the Larmor frequency in radian/sec. γ is the gyromagnetic ratio, the value of which depends on the nuclei of interest. For example, the gyromagnetic ratio of proton is 2.68×10^8 rad/sec/Tesla. In the above equation, B_0 is the strength of the main magnetic field in Tesla. Most widely field strength is 1.5T, but the field strengths ranging from 0.2 to 4 Tesla is available for clinical practice.

The voltage signal induced on the coil is related to different parameters of the experiment and can be calculated using the reciprocity principle by the following formula [3]

$$v_s = \omega_0 \mu \| \vec{H} \cdot \vec{M}_0 \| \tag{2.2}$$

where v_s is the signal voltage, $\vec{M}_0$ is the total transverse nuclear magnetic moment in the sample, $\vec{H}$ is the magnetic field generated by the receiving coil at the point of interest r_0 when a unit input current is applied to the coil, μ is the magnetic permeability of the sample. The notation $\| \cdot \|$ is used for the absolute value of a quantity. In (2.2) $\vec{H}$ and $\vec{M}_0$ are complex vectors and their inner product is considered.

The magnetic field $\vec{H}$ is written in its general form as

$$\vec{H} = H_x\vec{a}_x + H_y\vec{a}_y \tag{2.3}$$

where $\vec{a}_x$ and $\vec{a}_y$ are the unit vectors along he x- and y-axes, respectively. $\vec{M}_0$ has components only in the x and y directions with the same amplitude in each. The magnetic moment $\vec{M}_0$ is written as

$$\vec{M}_0 = M_0(\vec{a}_x + ja_y). \tag{2.4}$$

If we define H_+ as

$$H_+ = \frac{H_x - jH_y}{\sqrt{2}} \tag{2.5}$$

then v_s can be written as

$$v_s = \sqrt{2}\omega_0\mu H_+ M_0 \tag{2.6}$$

H_+ is the right-hand polarized magnetic field at some point of interest.

Note that H_+ is the value of the right-hand polarized magnetic field at some point of interest r_0, and hence v_s changes from point to point. v_s is function of point of interest r_0.

The RMS noise voltage per one square root Hertz can be calculated as

$$v_N = \sqrt{4k_BTR} \tag{2.7}$$

where k_B is the Boltzmann constant, T is the sample temperature, and R is the real part of the input impedance seen from the input terminals of the coil.

Intrinsic signal-to-noise-ratio (ISNR) is one of the important parameters in a Magnetic Resonance Imaging experiment. ISNR is defined by the following formula [4]

$$ISNR = \psi = \frac{v_s}{v_N} \propto \frac{H_+}{\sqrt{R}} \tag{2.8}$$

Note that ISNR is function of point of interest r_0 since H_+ itself is function of r_0. Accordingly, optimum electromagnetic field that maximizes ISNR for some point of interest is not in general the same electromagnetic field that maximizes ISNR for different point of interest. Optimum electromagnetic field distribution is a function of the point of interest r_0. In an MRI experiment, the noise level is determined by the dissipative power losses in the system. There are different power dissipation mechanisms that cause power loss, including conductor loss, radiation loss, and body loss. Each loss mechanism contributes to the noise resistance in the equivalent circuit. For a properly designed system, the limiting loss mechanism (most significant noise source), should be the body loss. Other losses can be reduced to insignificant levels by the use of proper materials, carefully designed coil geometry and low noise electronic components. The ultimate value of the intrinsic SNR depends only on the body loss [2].

Much research has been conducted to design receiver coils that achieve ISNR better than pre-existing coils. However, the value of the maximum achievable ISNR was not known. Therefore it was not known how much room was available to improve pre-existing coils. A straightforward approach to SNR maximization was to design the receiver antenna with a number of unknown parameters, (height, radius, etc...), calculate the SNR parametrically, i.e. as function of these parameters, and then determine the optimum values for those parameters. However, in the process, one had to solve the associated electromagnetic field equations in terms of unknown parameters, which is a difficult task even for the simplest of shapes with simplified approximations [2].

In a paper by O. Ocali and E. Atalar a method to determine the maximum achievable ISNR using external coil in an MRI experiment is proposed [2]. An algorithm to find ultimate ISNR was developed in that paper. The method involves finding the electromagnetic field generated by the receiving coil when a current of 1amp is applied to its terminals, then optimizing this field so that it yields the maximum possible ISNR. Plane-wave spectral representation of electromagnetic fields was used to find the optimum electromagnetic field. In principle the idea was to represent an arbitrary electromagnetic field by a linear combination of plane waves, each plane wave has an associated weight. By finding these weights the electromagnetic field is determined and then the ultimate ISNR is calculated. Plane wave representation of electromagnetic fields is most appropriate for rectangular-shaped bodies. For other shapes different representations are more appropriate, Bessel functions for cylindrical shapes, or spherical harmonics for spheres [5].

Signal to noise ratio (SNR) is a quantity that is directly proportional to the right-hand polarized magnetic field intensity H_+ at some point of interest r_0, and inversely proportional to the square root of the body resistance (which is equal to the coil total resistance) R_{body}, (2.8) [4.] Increasing the right-hand polarized magnetic field at r_0 and decreasing body resistance are conflicting goals, and a compromise between these two quantities is consisered.

By compromising the power loss with the right-hand polarized field, Ocali and Atalar concluded that if an electromagnetic field is presented as a sum of finite number of plane-waves (which is a good approximation for the ideal case with infinite number of plane-waves) as in [5-7]

$$\vec{E}(r) = \sum_i a_i \vec{E}_i(r) \tag{2.9}$$

Each $\vec{E}_i$ is a plane-wave electric field (mode) oriented in a different direction. The set of modes is to cover the whole direction space (ideally infinite number of modes, practically finite number of modes.) $\vec{E}_i$ can be any complete set of orthogonal solutions of the wave equation, such as the cylindrical wave equations that will be used in this work. Then dissipative consumed power becomes

$$\begin{aligned} R_{body} &= \int_{body} \sigma \|\vec{E}\|^2 dv = \sigma \int_{body} \vec{E}^* \cdot \vec{E} dv \\ &= \sum_i \sum_j \sigma \int_{body} [a_i^* a_j \vec{E}_i^* \vec{E}_j] dv \end{aligned} \tag{2.10}$$

or

$$R_{body} = \sum_i \sum_j a_i^* a_j r_{i,j} = \mathbf{a}^H \mathbf{R} \mathbf{a} \tag{2.11}$$

where

$$r_{i,j} = \sigma \int_{body} \vec{E}_i^* \cdot \vec{E}_j dv \tag{2.12}$$

and $\mathbf{R}$ is the noise correlation matrix $[r_{i,j}]$. $(\cdot)^H$ stands for the Hermitian (conjugate transpose) of a matrix.

Without loss of generality, the electromagnetic field can be scaled to have the right hand polarized magnetic field component equal to 1, and then to minimize the quantity R_{body} under this scaling condition. The constraint can be written in matrix notation as

$$\mathbf{b}^T \mathbf{a} = 1 \tag{2.13}$$

where $\mathbf{b}^T = [H_{0+}, H_{1+}, ..., H_{n+}]$, a vector containing the right-hand polarized magnetic field of each mode of the plane waves, and $\mathbf{a}^T = [a_0, a_1, ..., a_n]$ is the weight vector. $(\cdot)^T$ stands for the transpose of a matrix.

At this point the problem of finding the maximum ISNR is reduced to optimization problem under constraint. It can be stated as follows

$$R_{min} = min \quad \mathbf{a}^H \mathbf{R} a \tag{2.14}$$

subject to

$$\mathbf{a}^T \mathbf{b} = 1$$

After some manipulations, then the optimum set of $\mathbf{a}$ that gives the optimum field which produces the ultimate ISNR is given by

$$\mathbf{a}_{opt} = \frac{\mathbf{R}^{-1}\mathbf{b}^*}{\mathbf{b}^T\mathbf{R}^{-1}\mathbf{b}^*} \tag{2.15}$$

Substituting (2.15) into (2.11) yields the minimum noise resistance value:

$$R_{min} = \mathbf{a}_{opt}^H \mathbf{R} \mathbf{a}_{opt} = \frac{1}{\mathbf{b}^T\mathbf{R}^{-1}\mathbf{b}^*} \tag{2.16}$$

from which the value of the ultimate ISNR can be determined as

$$\psi_{max} = \frac{\omega_0 \mu M_0}{\sqrt{2k_B T R_{min}}} \tag{2.17}$$

here the right-hand polarized field is scaled to unity in the numerator [2].

In this work, the above formulation developed by Ocali and Atalar, will be reformulated using cylindrical waves and applied to a cylinder with a small hole in the center to understand the limits of SNR in endo-rectal coils.

Chapter 3

THEORY

Human body is modeled by a cylinder of finite dimensions. Although human body is not perfectly cylindrical, modeling it by a cylinder gives a good approximation. That is the main reason cylindrical system of coordinates is chosen to represent the electromagnetic field propagating inside the body. At the axis center of the cylinder, there is an axial hole as indicated in Fig. 3.1 . The radius of the hole plays a significant role in determining the ultimate intrinsic signal-to-noise ratio, ISNR, this is because it allows placement of internal coils.

3.1 Solution of The Wave Equation

We will formulate the problem in terms of EM fields rather than coils. Starting with the most general form of an electromagnetic field that is subject only to

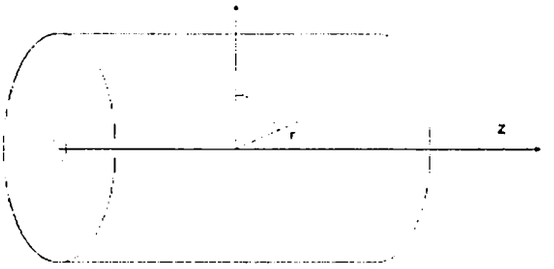

Figure 3.1: Human body modeled by a cylinder with an axial hole.

Maxwell's equations, the general solution will be written in an expanded form similar to expanding an arbitrary function in terms of Fourier integral or series.

The set of basis functions for this expansion is the solutions of the homogeneous wave equation in cylindrical coordinate system. Cylindrical system of coordinates is chosen because it is most suitable for human body shape and easiest to use to find the corresponding coil configuration used to generate the required field[5.]

Any electromagnetic field must satisfy Maxwell's equations. So to find the desired electromagnetic field Maxwell's equations are solved in cylindrical coordinates.

The general solution to Maxwell's equations in cylindrical coordinates is given by [8]

$$E_z = (AJ_m(\beta_\rho \rho) + CY_m(\beta_\rho \rho))e^{jm\phi}e^{-j\beta_z z} \tag{3.1}$$

where β_ρ and β_z are the propagation constants in the ρ and z directions, respectively. m is the propagation constant in the ϕ direction, which must be

an integer to satisfy the physical requirement of periodicity with 2π period interval. A and C are arbitrary constants, $J_m(\cdot)$ and $Y_m(\cdot)$ are the *mth* order Bessel and Neumann functions, respectively. m can take positive and negative integer values, β_z and β_ρ are related by

$$\beta_z = \pm\sqrt{\beta^2 - \beta_\rho^2} \tag{3.2}$$

where β is the propagation constant in a lossy medium

$$\beta = \sqrt{-j\omega\mu[\sigma + j\omega\epsilon]}. \tag{3.3}$$

The other field components are given as

$$H_z = (BJ_m(\beta_\rho\rho) + DY_m(\beta_\rho\rho))e^{j(m\phi-\pi/2)}e^{-j\beta_z z} \tag{3.4}$$

$$\begin{aligned} E_\rho &= -\frac{j}{\beta_\rho^2}[\frac{m\omega\mu}{\rho}(BJ_m(\beta_\rho\rho) + DY_m(\beta_\rho\rho)) \\ &+ \beta_z(AJ_m(\beta_\rho\rho) + CY_m(\beta_\rho\rho))]e^{jm\phi}e^{-j\beta_z z} \end{aligned} \tag{3.5}$$

$$\begin{aligned} E_\phi &= \frac{j}{\beta_\rho^2}[\frac{m\beta_z}{\rho}(AJ_m(\beta_\rho\rho) + CY_m(\beta_\rho\rho)) \\ &+ \omega\mu(BJ_m(\beta_\rho\rho) + DY_m(\beta_\rho\rho))]e^{j(m\phi-\pi/2)}e^{-j\beta_z z} \end{aligned} \tag{3.6}$$

$$H_\rho = -\frac{1}{\omega\mu}[\beta_z E_\phi + \frac{jm}{\rho}(AJ_m(\beta_\rho\rho) + CY_m(\beta_\rho\rho))e^{j(m\phi-\pi/2)}e^{-j\beta_z z} \tag{3.7}$$

$$H_\phi = \frac{1}{\omega\mu}[\beta_z E_\rho - j(AJ'_m(\beta_\rho\rho) + CY'_m(\beta_\rho\rho))e^{jm\phi}e^{-j\beta_z z} \tag{3.8}$$

where

$$J'_m(\beta_\rho\rho) \equiv \frac{d}{d\rho}J_m(\beta_\rho\rho) = \frac{m}{\rho}J_m(\beta_\rho\rho) - \beta_\rho J_{m+1}(\beta_\rho\rho) \tag{3.9}$$

$$Y'_m(\beta_\rho\rho) \equiv \frac{d}{d\rho}Y_m(\beta_\rho\rho) = \frac{m}{\rho}Y_m(\beta_\rho\rho) - \beta_\rho Y_{m+1}(\beta_\rho\rho) \tag{3.10}$$

Overall electric and magnetic fields are

$$\vec{E}(\rho,\phi,z) = \vec{a}_\rho E_\rho(\rho,\phi,z) + \vec{a}_\phi E_\phi(\rho,\phi,z) + \vec{a}_z E_z(\rho,\phi,z) \tag{3.11}$$

$$\vec{H}(\rho,\phi,z) = \vec{a}_\rho H_\rho(\rho,\phi,z) + \vec{a}_\phi H_\phi(\rho,\phi,z) + \vec{a}_z H_z(\rho,\phi,z) \tag{3.12}$$

where $\vec{a}_\rho$, $\vec{a}_\phi$ and $\vec{a}_z$ are unit vectors in the ρ,ϕ and z directions, respectively. E_ρ, E_ϕ and E_z are the electric field components in the three directions and H_ρ, H_ϕ and H_z are the associated magnetic fields, each one of these six components is function of the three coordinate variables ρ,ϕ, and z.

3.2 EM Field Expansion in Terms of Cyiindrical Waves

One basis function of the solution expansion is called a mode. To determine a mode of the propagating wave, we fix two parameters, namely m and one of β_ρ or β_z, since fixing any one of them determines the other according to (3.2) . So expansion of the solution is in two dimensions, m which is discrete, and β_z which is continuous and complex. For β_z, an integral expansion-analog to Fourier transform-would be taken if an analytical solution were possible. Instead sample values of β_z,β_{zn}, are taken and discrete summation in terms of β_{zn} is used. Four arbitrary complex constants, A, B, C and D are associated with each mode.

So the electric field in the z direction, as an example, would be expanded as follows [8]

$$E_z = \sum_m \sum_n E_{zmn} \tag{3.13}$$

$$E_{zmn} = (A_{mn} J_m(\beta_{\rho n}\rho) + C_{mn} Y_m(\beta_{\rho n}\rho)) e^{jm\phi} e^{-j\beta_{zn} z} \tag{3.14}$$

Similarly the other components of the field are expanded as follows

$$H_z = \sum_m \sum_n H_{zmn} \tag{3.15}$$

$$H_{zmn} = (B_{mn}J_m(\beta_{\rho n}\rho) + D_{mn}Y_m(\beta_{\rho n}\rho))e^{j(m\phi-\pi/2)}e^{-j\beta_{zn}z} \quad (3.16)$$

$$E_\rho = \sum_m \sum_n E_{\rho mn} \quad (3.17)$$

$$E_{\rho mn} = -\frac{j}{\beta_{\rho n}^2}[\frac{m\omega\mu}{\rho}(B_{mn}J_m(\beta_{\rho n}\rho) + D_{mn}Y_m(\beta_{\rho n}\rho)) + \beta_{zn}(A_{mn}J_m(\beta_{\rho n}\rho) + C_{mn}Y_m(\beta_{\rho n}\rho))]e^{jm\phi}e^{-j\beta_{zn}z} \quad (3.18)$$

$$E_\phi = \sum_m \sum_n E_{\phi mn} \quad (3.19)$$

$$E_{\phi mn} = \frac{j}{\beta_{\rho n}^2}[\frac{m\beta_{zn}}{\rho}(A_{mn}J_m(\beta_{\rho n}\rho) + C_{mn}Y_m(\beta_{\rho n}\rho)) + \omega\mu(B_{mn}J_m(\beta_{\rho n}\rho) + D_{mn}Y_m(\beta_{\rho n}\rho))]e^{j(m\phi-\pi/2)}e^{-j\beta_{zn}z} \quad (3.20)$$

$$H_\rho = \sum_m \sum_n H_{\rho mn} \quad (3.21)$$

$$H_{\rho mn} = -\frac{1}{\omega\mu}[\beta_{zn}E_\phi + \frac{jm}{\rho}(A_{mn}J_m(\beta_{\rho n}\rho) + C_{mn}Y_m(\beta_{\rho n}\rho))e^{j(m\phi-\pi/2)}e^{-j\beta_z z} \quad (3.22)$$

$$H_\phi = \sum_m \sum_n H_{\phi mn} \quad (3.23)$$

$$H_{\phi mn} = \frac{1}{\omega\mu}[\beta_z E_{\rho n} - j(A_{mn}J'_m(\beta_{\rho n}\rho) + C_{mn}Y'_m(\beta_{\rho n}\rho))e^{jm\phi}e^{-j\beta_z z} \quad (3.24)$$

We need to calculate two quantities. 1) the consumed power, which is equal to the resistance of the coil R_{body} when 1-ampere current is assumed to be applied to the coil, 2) the right-hand polarized magnetic field H_+.

Dissipative consumed power is evaluated by

$$R_{body} = \int_{body} \sigma\|\vec{E}\|^2 dv = \sigma \int_{body} \vec{E}^* \cdot \vec{E} dv \quad (3.25)$$

$$= \sigma \int_{body} [E_z^* E_z + E_\rho^* E_\rho + E_\phi^* E_\phi] dv \quad (3.26)$$

Expanding over m only gives

$$R_{body} = \sigma \int_{body} [(\sum_m E_{zm})^* (\sum_l E_{zl}) + (\sum_m E_{\rho m})^* (\sum_l E_{\rho l})$$

$$+ \quad (\sum_m E_{\phi m})^*(\sum_l E_{\phi l})]dv \tag{3.27}$$

$$= \quad \sum_m \sum_l \sigma \int_{body} [E^*_{zm}E_{zl} + E^*_{\rho m}E_{\rho l} + E^*_{\phi m}E_{\phi l}]dv \tag{3.28}$$

In the above equation $E_{\rho m}, E_{\phi m} and E_{zm}$ (or similarly $E_{\rho l}, E_{\phi l} and E_{zl}$) are given by

$$E_{\rho m} = \sum_j E_{\rho mj} \tag{3.29}$$

$$E_{\phi m} = \sum_j E_{\phi mj} \tag{3.30}$$

$$E_{zm} = \sum_j E_{zmj} \tag{3.31}$$

$$E_{\rho n} = \sum_j E_{\rho lj} \tag{3.32}$$

$$E_{\phi n} = \sum_j E_{\phi lj} \tag{3.33}$$

$$E_{zn} = \sum_j E_{zlj} \tag{3.34}$$

Integration over a volume is a triple integration involving one integral with respect to ϕ. When substituting (3.14) ,(3.18) and (3.20) in (3.29) -(3.33) and then into (3.28) , each term of the integrands is multiplied by "$e^{j(m-l)\phi}$", and when integrated over $\phi = 0$ to $\phi = 2\pi$ it gives zero unless $m = l$. In other words, there is no consumed cross power between solutions of different (Bessel) order (m). Cross power consumption occurs only between solutions of the same order, regardless of values of β_z's.

Note that β is a complex constant. β_ρ and β_z are also complex variables under the constraint (3.2) . In order to get the most general form of an electromagnetic field we have to span the complex plane for β_ρ and β_z.

3.3 Construction of R matrix

Resistance can be written as

$$R_{body} = \sum_{m} \sum_{k,l} \sigma \int \vec{E}^{*}_{mk} \cdot \vec{E}_{ml} dv \tag{3.35}$$

where

$$\vec{E}_{mk} = E_{zmk}\vec{a}_z + E_{\rho mk}\vec{a}_\rho + E_{\phi mk}\vec{a}_\phi \tag{3.36}$$

$$\vec{E}_{ml} = E_{zml}\vec{a}_z + E_{\rho ml}\vec{a}_\rho + E_{\phi ml}\vec{a}_\phi \tag{3.37}$$

Note that the first summation is taken only over m since power consumption occurs only between electric fields of the same order, while the second summation is a double summation over two different indices k and l.

To be consistent with matrix notation, power is written as

$$R_{body} = \sum_{m,k} \sum_{n,l} \sigma \int_{body} \vec{E}^{*}_{mk} \cdot \vec{E}_{nl} dv \tag{3.38}$$

We define a matrix $\mathbf{R}$ called the noise correlation matrix to contain the elements $r_{mk,nl}$ given by

$$r_{mk,nl} = \sigma \int_{body} \vec{E}^{*}_{mk} \cdot \vec{E}_{nl} dv \tag{3.39}$$

In the above expression for $r_{mk,nl}$, if the constants $A_{mk}, A_{nl}, B_{mk}, B_{nl}, C_{mk}, C_{nl}, D_{mk}$ and D_{nl} are omitted, then when substituting (3.14) ,(3.18) and (3.20) in (3.38) we get

$$R_{body} = \sum_{mk} \sum_{nl} a^{*}_{mk} a_{nl} r_{mk,nl} \tag{3.40}$$

or in matrix notation

$$R_{body} = \mathbf{a}^{*}\mathbf{R}^{H}\mathbf{a} \tag{3.41}$$

where a_{mk} can take value on any of A_{mk}, B_{mk}, C_{mk} or D_{mk}, similarly a_{nl} can take on any value of A_{mk}, B_{mk}, C_{mk} or D_{mk}. In (3.41) , a is a vector containing a_{mk}'s. $(.)^H$ is the hermitian (conjugate transpose) notation. Equations (3.41) ,(3.40) and (3.38) show that $\mathbf{R}$ is positive definite.

The following table shows how $r_{mk,nl}$ is computed according to a_{mk} and a_{nl}. The left hand side of the table refers to x in

$$r_{mk,nl} = \sigma \int_{body} x dv = \sigma \int_{body} \vec{E}^*_{mk} \cdot \vec{E}_{nl} dv \tag{3.42}$$

equation(3.42) is exactly equation (3.39) .

$r_{mk,nl}$ is computed differently according to the coefficient by which it is multiplied. For example, if it is multiplied by A^*_{mk} and B_{ml} then $r_{mk,nl}$ is computed according to the the formula given in the fifth line in the table above.

Note that $\mathbf{R}$ is Hermitian(conjugate symmetric), so the coefficient multiplied by $a_m k^*$ and a_{nl}, take them to be B^*_{ml} and A_{mk} as an example, is the conjugate of the coefficient multiplied by $a_m k$ and a^*_{nl}, A^*_{mk} and B_{ml} in this case.

The other quantity we need to calculate is the right-hand polarized field $H_+(\rho_0)$ at the point of interest $p(\rho_0, \phi_0, z_0)$. By means of scaling we'll fix $H_+(\rho_0)$ at ρ_0 to be equal to unity, and proceed in minimizing the resistance, or equivalently the power loss.

Coefficients $a^*_{mk}a_{ml} =$	Integrand x
$A^*_{mk}A_{ml}$	$J^*_m(\beta_{\rho k}\rho)J_m(\beta_{\rho l}\rho)[1 + \frac{m^2\beta^*_{zk}\beta_{zl}}{\beta^{*2}_{\rho k}\beta^2_{\rho k}\rho^2}] + \frac{\beta^*_{zk}\beta_{zl}}{\beta^{*2}_{\rho k}\beta^2_{\rho k}}J^{*'}_m(\beta_{\rho k}\rho)J'_m(\beta_{\rho l}\rho)$
$B^*_{mk}B_{ml}$	$\frac{m^2\omega^2\mu^2}{\beta^{*2}_{\rho k}\beta^2_{\rho k}\rho^2}J^*_m(\beta_{\rho k}\rho)J_m(\beta_{\rho l}\rho) + \frac{\omega^2\mu^2}{\beta^{*2}_{\rho k}\beta^2_{\rho k}\rho^2}J^{*'}_m(\beta_{\rho k}\rho)J'_m(\beta_{\rho l}\rho)$
$C^*_{mk}C_{ml}$	$Y^*_m(\beta_{\rho k}\rho)Y_m(\beta_{\rho l}\rho)[1 + \frac{m^2\beta^*_{zk}\beta_{zl}}{\beta^{*2}_{\rho k}\beta^2_{\rho k}\rho^2}] + \frac{\beta^*_{zk}\beta_{zl}}{\beta^{*2}_{\rho k}\beta^2_{\rho k}}Y^{*'}_m(\beta_{\rho k}\rho)Y'_m(\beta_{\rho l}\rho)$
$D^*_{mk}D_{ml}$	$\frac{m^2\omega^2\mu^2}{\beta^{*2}_{\rho k}\beta^2_{\rho k}\rho^2}Y^*_m(\beta_{\rho k}\rho)Y_m(\beta_{\rho l}\rho) + \frac{\omega^2\mu^2}{\beta^{*2}_{\rho k}\beta^2_{\rho k}\rho^2}Y^{*2}_m(\beta_{\rho k}\rho)Y'_m(\beta_{\rho l}\rho)$
$A^*_{mk}B_{ml}$	$\frac{m\omega\mu\beta^*_{zk}}{\beta^{*2}_{\rho k}\beta^2_{\rho k}\rho}[J^{*'}_m(\beta_{\rho k}\rho)J_m(\beta_{\rho l}\rho) + J^*_m(\beta_{\rho k}\rho)J'_m(\beta_{\rho l}\rho)]$
$A^*_{mk}C_{ml}$	$J^*_m(\beta_{\rho k}\rho)Y_m(\beta_{\rho l}\rho)[1 + \frac{m^2\beta^*_{zk}\beta_{zl}}{\beta^{*2}_{\rho k}\beta^2_{\rho k}\rho^2}] + \frac{\beta^*_{zk}\beta_{zl}}{\beta^{*2}_{\rho k}\beta^2_{\rho k}}J^{*'}_m(\beta_{\rho k}\rho)Y'_m(\beta_{\rho l}\rho)$
$A^*_{mk}D_{ml}$	$\frac{m\omega\mu\beta^*_{zk}}{\beta^{*2}_{\rho k}\beta^2_{\rho k}\rho}[J^{*'}_m(\beta_{\rho k}\rho)Y_m(\beta_{\rho l}\rho) + J^*_m(\beta_{\rho k}\rho)Y'_m(\beta_{\rho l}\rho)]$
$B^*_{mk}C_{ml}$	$\frac{m\omega\mu\beta_{zl}}{\beta^{*2}_{\rho k}\beta^2_{\rho k}\rho}[J^*_m(\beta_{\rho k}\rho)Y'_m(\beta_{\rho l}\rho) + J^{*'}_m(\beta_{\rho k}\rho)Y_m(\beta_{\rho l}\rho)]$
$B^*_{mk}D_{ml}$	$\frac{m^2\omega^2\mu^2}{\beta^{*2}_{\rho k}\beta^2_{\rho k}\rho^2}J^*_m(\beta_{\rho k}\rho)Y_m(\beta_{\rho l}\rho) + \frac{\omega^2\mu^2}{\beta^{*2}_{\rho k}\beta^2_{\rho k}\rho^2}J^{*'}_m(\beta_{\rho k}\rho)Y'_m(\beta_{\rho l}\rho)$
$C^*_{mk}D_{ml}$	$\frac{m\omega\mu\beta^*_{zk}}{\beta^{*2}_{\rho k}\beta^2_{\rho k}\rho}[Y^{*'}_m(\beta_{\rho k}\rho)Y_m(\beta_{\rho l}\rho) + Y^*_m(\beta_{\rho k}\rho)Y'_m(\beta_{\rho l}\rho)]$

Table 3.1: Formulas used in computing **R** matrix elements.

Coefficient a_{mk}=	b_{mk}
A_{mk}	$[\frac{-m}{\omega\mu\rho_0}J_m(\beta_\rho\rho_0)(\frac{\beta_{zk}^2}{\beta_{\rho k}^2}+1) - \frac{1}{\omega\mu}J'_m(\beta_\rho\rho_0)(\frac{\beta_{zk}^2}{\beta_{\rho k}^2}+1)]e^{jm\phi_0}e^{-j\beta_{zk}z_0}/\sqrt{2}$
B_{mk}	$\frac{-\beta_{zk}^2}{\beta_{\rho k}^2}[\frac{m}{\rho_0}J_m(\beta_\rho\rho_0)+J'_m(\beta_\rho\rho_0)]$
C_{mk}	$[\frac{-m}{\omega\mu\rho_0}Y_m(\beta_\rho\rho_0)(\frac{\beta_{zk}^2}{\beta_{\rho k}^2}+1) - \frac{1}{\omega\mu}Y'_m(\beta_\rho\rho_0)(\frac{\beta_{zk}^2}{\beta_{\rho k}^2}+1)]e^{jm\phi_0}e^{-j\beta_{zk}z_0}/\sqrt{2}$
D_{mk}	$\frac{-\beta_{zk}^2}{\beta_{\rho k}^2}[\frac{m}{\rho_0}Y_m(\beta_\rho\rho_0)+Y'_m(\beta_\rho\rho_0)]$

Table 3.2: Formulas used in computing b vector elements.

Right-hand polarized magnetic field $H_+(\rho_0)$ at the point of interest equals the summation of individual mode fields H_{mk+}'s. So

$$H_+(\rho_0) = \sum_{mk} H_{mk+}(\rho_0) = \sum_{mk} a_{mk}b_{mk} \tag{3.43}$$

b_{mk} is the right hand polarized field produced by mode mk and associated with a_{mk} which is one of A_{mk}, B_{mk}, C_{mk} or D_{mk} constants.

The next equation (3.44) shows how b_mk's are computed.

$$H_{mk+}(\rho_0) = \frac{H_{\rho mk}(\rho_0) - jH_{\phi mk}(\rho_0)}{\sqrt{2}} \tag{3.44}$$

Substituting (3.16) ,(3.22) and (3.24) in (3.44) and (3.43) and rearranging terms we get the following table

Now the problem can be solved using (2.15) and (2.16) directly.

3.4 Practical Simplifications

In simulating the problem on the computer, **R** matrix is rearranged so that it becomes in a block diagonal form.

$$\begin{pmatrix} \mathbf{R}_0 & 0 & & 0 \\ 0 & \mathbf{R}_1 & & 0 \\ & & & \\ 0 & & 0 & \mathbf{R}_{r-1} \end{pmatrix}$$

$\mathbf{R}_0$,$\mathbf{R}_1$,...,$\mathbf{R}_{r-1}$ are square submatrices of the equal sizes. Block diagonalizing of $\mathbf{R}$ matrix is possible because cross power between different order modes is zero.

Due to this diagonalization we can write the following

$$\mathbf{b}^T\mathbf{R}^{-1}\mathbf{b}^* = \sum_{m=0}^{r-1} \mathbf{b}_m^T {\mathbf{R}_m}^{-1} \mathbf{b}_m^* \tag{3.45}$$

Each $\mathbf{R}_m$ is arranged such that the upper-left quarter of $\mathbf{R}_m$ corresponds to the power consumption of Bessel functions $J_m(.)$'s. While the lower-right quarter corresponds to the power consumption by Neumann functions $Y_m(.)$'s. The other two quarters correspond to the cross power between $J_m(.)$'s and $Y_m(.)$'s,i.e. elements containing $J_m(.)Y_m(.)$. As we will see later, when the point of interest is close to the circumference of the hole, R_{min} is dominantly determined by the lower-right quarter of $\mathbf{R}_m$. While when it is close to the outer circumference of the cylinder, R_{min} is dominated by the upper-left quarter of $\mathbf{R}_m$. This is due to the singularity of Neumann function $Y_m(.)$ at the origin,which gives much higher H_+ than that given by Bessel function $J_m(.)$ when the point of interest is close to the origin. When the point of interest is far from the origin H_+'s given by $J_m(.)$ or by $Y_m(.)$ are of moderate magnitudes, at the time singularity of $Y_m(.)$ results in very high power consumption, and hence the compromise between H_+ and power consumption is in favor of $J_m(.)$.

Negative values of m are taken into consideration. For negative m, $\mathbf{R}_{-m} = \mathbf{R}_m$. This is because [8]

$$J_{-m}(x) = (-1)^m J_m(x) \tag{3.46}$$

$$Y_{-m}(x) = (-1)^m Y_m(x) \tag{3.47}$$

and every term in $\mathbf{R}_m$ consists of exactly two of $J_m(.)$ or $Y_m(.)$ multiplied by each another, which results in $(-1)^{2m} = 1$. On the other hand $b_{-m} = (-1)^m b_m e^{-2jm\phi_0}$, since in each term of b_{-m}, instead of multiplying by $e^{jm\phi_0}$ we multiply by $e^{-jm\phi_0}$ and a phase shift of $(-1)^m e^{-2jm\phi_0}$ exists between b_m and b_{-m}.

Therefore

$$\mathbf{b}_{-m}^T \mathbf{R}_{-m}{}^{-1} \mathbf{b}_{-m}^* = (\mathbf{b}_m^T e^{-2jm\phi_0}) \mathbf{R}_m{}^{-1} (\mathbf{b}_m^* e^{2jm\phi_0}) = \mathbf{b}_m^T \mathbf{R}_m{}^{-1} \mathbf{b}_m^* \tag{3.48}$$

R_{min} is not function of the position angle of the point of interest, ϕ_0. This can be seen by looking at the entries of b_m and noting that the only dependence on ϕ_0 is in the form of $e^{jm\phi_0}$ and when taking Hermitian of b_m, $e^{jm\phi_0}$ becomes $e^{-jm\phi_0}$ and both eliminate each other in $b_m \mathbf{R}_m^{-1} b_m^H$.

If we define sensivity map to be the absolute value of the right hand polarized magnetic field as a function of position (ρ_0, ϕ_0, z) then it can be shown that map is symmetric around x-axis, i.e.

$$\|H_+(\rho_0, \phi_0, z_0)\| = \|H_+(\rho_0, -\phi_0, z_0)\| \tag{3.49}$$

Chapter 4

METHOD

In this chapter, practical implementation of the optimization process is described in detail. The optimization is performed using numerical calculations by computer. C and C++ are used to write a code to solve the problem (Appendix A). Main parts of this code are integration part that computes the $\mathbf{R}$ matrix elements by integrating the power loss density over the body, and linear system of algaibric equations solver which is used to find the optimum resistance after determing matrix $\mathbf{R}$ and right-hand polarization vector b.

4.1 Numerical Manipulations

4.1.1 Numerical Integrations

To find the elements of $\mathbf{R}$ matrix, we need to conduct triple integrations of functions as indicated in Table(3.1) in the previous chapter. Those functions are of

three variables, ρ,ϕ and z; and fortunately are seperable functions. Integrations over ϕ and z are easy since they involve exponential variation which can be integrated analytically. Closed forms of integrations for the third integration are only available under strict conditions on function orders and arguments, also they require infinite series calculations which make them more difficult to use. Instead, numerical integration is used. The integrations are of the form

$$\int_{x1}^{x2} V_m(c_1 x) W_n(c_2 x) dx \tag{4.1}$$

wher $V_m(\cdot)$ and $W_n(\cdot)$ are any of Bessel function $J_m(\cdot)$,$J_n(\cdot)$ or Neumann function $Y_m(\cdot)$,$Y_n(\cdot)$, and therefore we have four possible combinations for this integration. In the first attempt to compute the integrations, Trapizoidal rule was used. When running the program it appeared that such a method would take very long time to compute the integration with minimum acceptable accuracy. So instead of tha Trapizoidal rule, power series expansions of Bessel and Neumann functions are used. Although these expansions are infinite series expansions, they converge rapidly to the accurate value.

Power series expansions of Bessel functions are as follows [9].

$$J_m(x) = \sum_{n=0}^{\infty} \frac{(-1)^n (x/2)^{2n+m}}{n!(n+m)!} \tag{4.2}$$

while for for $Y_m(\cdot)$, when $m = 0$

$$Y_0(x) = \frac{2}{\pi} J_0(x)(log(x/2) + \gamma) - \frac{2}{\pi} \sum_{n=1}^{\infty} \frac{(-1)^n (x/2)^{2n}}{(n!)^2} \psi(n) \tag{4.3}$$

where

$$\psi(n) = \sum_{k=1}^{n} \frac{1}{k} \tag{4.4}$$

and γ is a constant equal to 0.577215 . When m is not zero, $Y_m(\cdot)$ takes the form

$$Y_m(x) = \frac{2}{\pi} J_m(x) log(x/2) - \frac{1}{\pi} \sum_{n=0}^{m-1} \frac{(m-n-1)!}{n!} (x/2)^{2n-m}$$
$$- \frac{1}{\pi} \sum_{k=0}^{\infty} \frac{(-1)^k (x/2)^{2k+m}}{k!(k+m)!} (\Gamma(k+m+1) + \Gamma(k+1)) \qquad (4.5)$$

where

$$\Gamma(k) \;=\; -\gamma + \psi(k-1) \qquad (4.6)$$

In the above relations x is a complex number, and m is an integer.

When substituted in (4.1) , power series expansions of Bessel and Neumann functions result in other power series which are very easy to integrate analytically term by term according to the simple rule

$$\int x^n \;=\; \frac{x^{n+1}}{n+1} \qquad (4.7)$$

Again when implementing the idea on computer code, the run time was still long, specially when the number of modes taken into consideration in expanding the electromagnetic field is high, given that we had to run the program large number of times to examine the results. So further simplification to save more running time is necessary. When looking at the expansions in general we can notice a general pattern of the form

$$\sum_{n=0}^{v} (ax)^n \quad or \quad \sum_{n} (ax)^n \frac{n!}{(n+m)!} \quad etc... \qquad (4.8)$$

This kind of series, when computed directly, takes $O(v^2)$ multiplications. However it can be rewritten as

$$1 + ax(1 + ax(1 + ax(...))) \; or \; 1 + \frac{1}{m!} ax(1 + \frac{2}{m+1} ax(1 + \frac{3}{m+2} ax(...))) \qquad (4.9)$$

When implemented this way, number of multiplications is redused to $O(v)$, which is a significant reduction knowing that multiplications take most of the run time in the optimization process.

4.1.2 Matrix Manipulations

After calculation of $\mathbf{R}$ matrix and b vector entries, we need to use them to find the optimum resistance and coefficients according to (2.15) and (2.16)

Traditional LU decomposition is used by elementary row operations and exchanges to solve the system $Ax = b$ instead of direct inversion. This part of the code takes short time to run after the matrix and the vector are determined.

4.2 Algorithm

Algorithm for a cylindrical body

In the simulation program, computation of the intrinsic signal-to-noise ratio, ISNR, and sensitivity map is conducted as indicated by the following algorithm

1. (Input)

I. Step size of sampling β_z in the real and the imaginary parts.

II. Inner and outer radii of the model cylinder in meters.

III.Number of Bessel modes to be used.

IV. Initial sample point of β_z.

V. Point of interest.

VI. Tolerance.

2. Set the parameters of the body, conductivity, electric and magnetic permeability, frequency etc...

3. Start with one mode from initial sample of β_z.

4. Compute $\mathbf{R}$ matrix entries according to Table 3.1.

5. Compute b vector entries according to Table 3.2.

6. Apply equations (2.15) and (2.16) to find minimum resistance and optimum coefficients.

7. Add one more mode by taking another sample of β_z.

8. If reasonble increase in ISNR (more than tolerance) retain the added sample, else discard it.

9. Repeat 3 to 8 until saturation in ISNR is reached.

10. Find sensitivity map and plot it as function of space.

The program takes as inputs dimensions of the body, i.e. outer radius, inner radius (of the hole), length, location of the point of interest, and interval lengths seperating samples of β_z in the real and imaginary directions and their initial values. Also upper limit on the order of Bessel and Neumann functins to be used and tolerance beyond which it is to ignore samples of β_z are taken as inputs. The program starts by finding the optimum ISNR with only few samples of β_z, say 3 samples, then it adds more three samples and calculates optimum ISNR with 6 samples. If the three added samples conribute significantly to ISNR, i.e. $\frac{ISNR_{new}}{ISNRold} > (1 + Tolerance)$ then it retains the added samples, else it discards them and goes forward to the next three samples, and so on until saturation on the value of ISNR is reached.

The program can be run for variety of parameters such as frequency, electric and magnetic properties of the body material, temperature etc...Upper limit on the order of Bessel and Neumann functions used is determined manually according to radial distance of the point of interest. The further the point is from the hole the more Bessel and Neumann orders are needed. When the point of interest is near the hole only zero order functions are enough and higher oreder ones do not contribute significantly to ISNR. Optimum interval lengths in sampling β_z is also determined manually. Right-hand polarized magnetic field vector b is calculated using the power series expansion as done in calculating matrix $\mathbf{R}$ entries. When running the program there is no need to find b_{-m} or $\mathbf{R}_{-m}$ nor their solutions as mentioned in chapter two, since they are either the same as b_{-m} and $\mathbf{R}_{-m}$ or multiplied by some complex constant.

Chapter 5

RESULTS

In this chapter we show results of simulating the electromagnetic field inside a given body geometry and the computation of the ultimate intrinsic signal-to-noise ratio (ISNR) resulting from optimizing this EM field. Then we investigate the behavior of this ultimate ISNR with varying parameters of the body. For our model we consider a cylinder with radius of 0.25 m, length of 1.0 m and a coaxial hole of radius .001 m. Electromagnetic parameters of the body include conductivity σ of 0.37 Siemens/m, relative electric permittivity ϵ_r of 77.7 and relative magnetic permeability μ_r of 1.0. The operating frequency is taken to be 63.9 MHz, this value is used in simulations since 1.5 Tesla is a widely used main magnetic field intensity which yields a resonance frequency of 63.9 MHz. Temperature is considered to be 310^o Kelvin.

Optimization of the electromagnetic (EM) field is performed for one point of interest r_0, and the EM field distribution is different from one point of interest

to another. Fig. 5.1 shows a sketch diagram of the location of point of interest r_0 as well as the dimensions of the body r_h and r_b. Note that point of interest can be anywhere inside the human body except the region inside the hole.

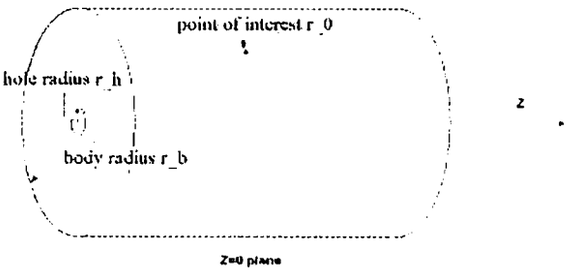

Figure 5.1: Point of interest r_0 location within body.

To simulate the EM field inside the human body C++ is used to write a program that accepts different inputs concerning body dimensions and other parameters and gives the ultimate ISNR. The simulation is about finding the weight vector by which the EM field is determined. The goal of the simulation is to determine the ultimate ISNR, to check whether results are much better than already achieved results, to study the behavior of SNR as body parameters changes, observe which parameter variations cause sharp changes in SNR and choose steady-state values for comparisons.

Computation time is one of the important factors in writing the simulation program. Most of the running time is used in numerical integrations. For one m order and 25 β_z samples used, the program takes about 15 minutes to compute the optimum weight vector and the ultimate ISNR. For two m orders the computation time doubles. This time could be achieved by using numerical

integration techniques described in chapter 4. Without using these techniques, it would be at least ten times longer to perform the computations.

Fig. 5.2 shows the simulation results for ISNR as function of the point of interest radial distance r_0 (while the other coordinates of r_0 are $z_0 = 0$ and $\phi_0 = 0$) . ISNR is at very high values for points of interest close to the hole. As moving away from the hole, ISNR decreases sharply with r_0 until it reaches a minimum, then it starts increasing at a low rate towards the cylinder surface.

To get a qualitative explanation of this behavior we recall that ISNR is a compromise between high $H_+(\rho_0)$ and low body resistance R_{body}. While $H_+(\rho_0)$ is computed at a fixed point r_0, R_{body} results from an integration over the whole volume. $H_+(\rho_0)$ formulas involve Neumann functions which possess singularities at the origin which make the ratio of $H_+(\rho_0)$ to R_{body} very high. Absolute value of Neumann functions then decays with arguement and the SNR value decreases accordingly.

ISNR is function of body dimensions. We expect that when body total volume increases, overall power loss increases and ISNR decreases; similarly the smaller the body volume is, the higher ISNR is achieved.

Fig. 5.3 shows ISNR as function of inner hole radius r_h when point of interest is at $r_0 = 5mm$. As shown, ISNR decreases with increasing hole radius. However, the rate of change is not very high. Fig. 5.4 describes ISNR behavior as function of body radius. Again as the radius increases, total volume occupied increases and the power loss gets higher which results in lower SNR. After some point, ISNR shows very little variation with the body radius and it almost saturates after $r_b = 10cm$. For SNR vs. r_0 plot and comparison

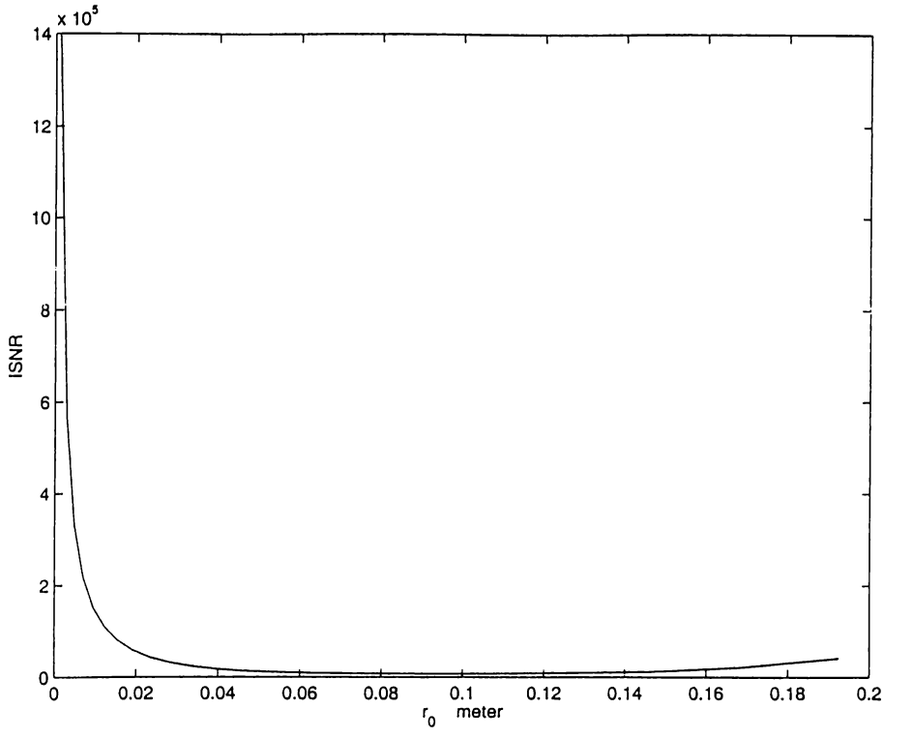

Figure 5.2: Intrinsic Signal-to-Noise Ratio vs. point of interest radial distance r_0.

purposes, r_b is taken to be 25cm, well beyond saturation. Typical value for cylinder length of 1m is chosen also to be beyond saturation as SNR changes with length.

Assuming a hole in the human body enables inserting internal coils inside the body so that clearer images are expected as a result of this internal coil. Also, using cylindrical wave modes to represent the EM field is more appropriate for cylindrical body. As a consequence, higher ISNR is expected than the one achived by plane wave representation [5] and when there is no hole inside the body. To compare the ultimate ISNR computed in this work simulations to that computed by Ocali and Atalar using external coil, or even to the ISNR of an internal loopless antenna developed by the same authors, we plot ISNR for

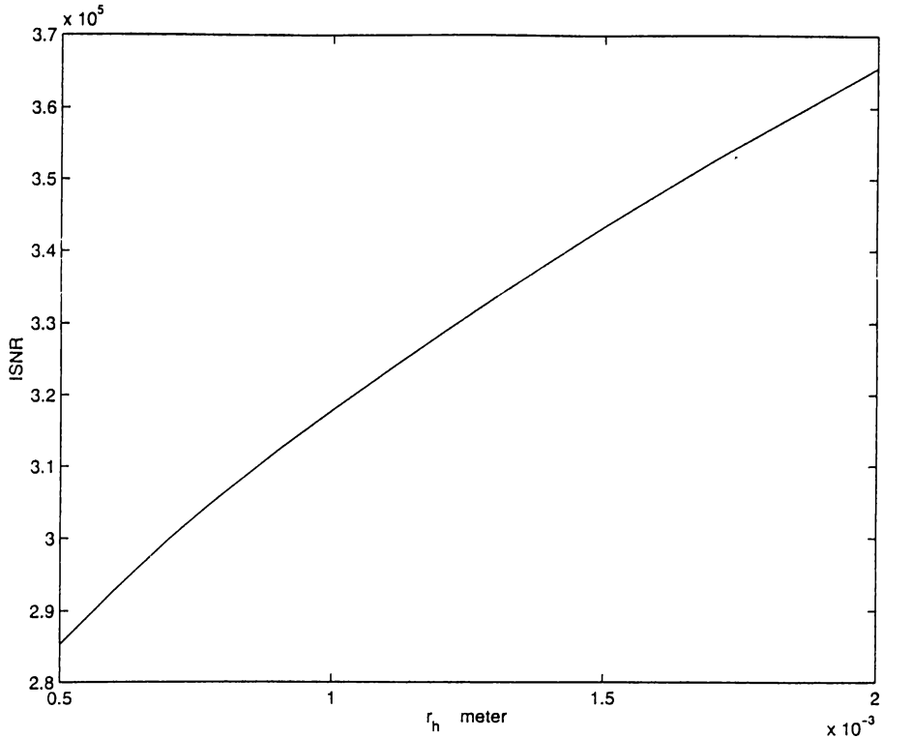

Figure 5.3: Intrinsic Signal-to-Noise Ratio vs. inner hole radius r_h.

each of these three cases on one log-scaled graph, Fig. 5.6. Loopless antenna is a simple-structured bipolar antenna, see Fig. 5.5, inserted through the hole inside the human body as described in [10] Results for ISNR as function of r_0 for loopless antenna can be found in [10] and is given by

$$ISNR(r_0) = \frac{4.72 \times 10^2}{r_0} \tag{5.1}$$

r_0 is in meters. As shown in Fig. 5.6, internal coil results in higher ISNR than the other two techniques for all points of interest along the radius.

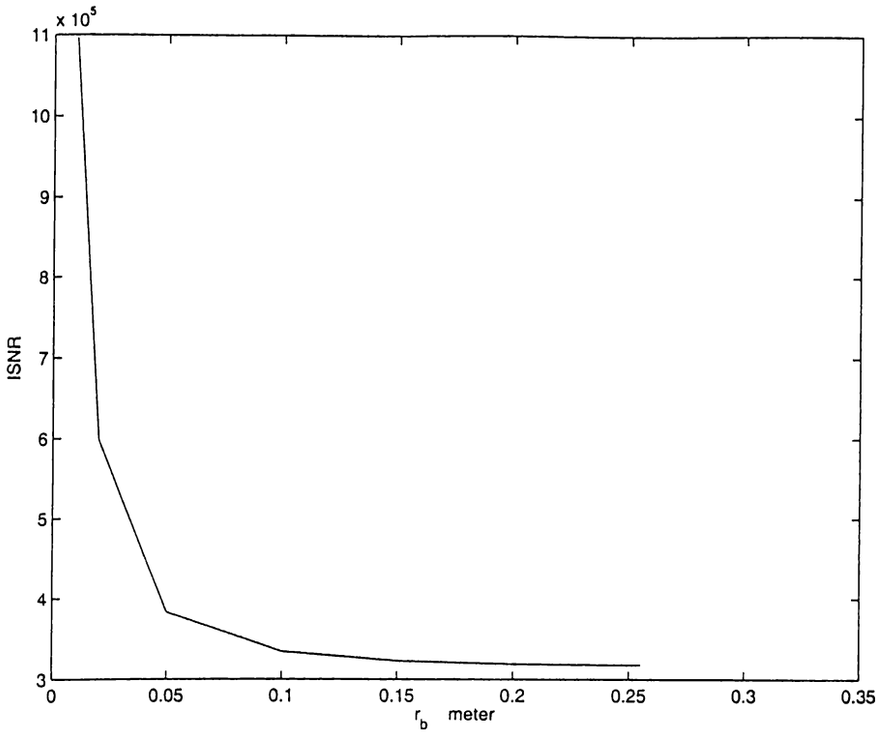

Figure 5.4: Intrinsic Signal-to-Noise Ratio vs. body radius r_b.

In the previous comparison, 500 plane-wave modes are used in plane-wave representation. While for cylindrical wave representation case only 25 β_z samples are used. This is a numerical example that shows cylindrical wave representation is much more efficient for the cylindrical geometry.

In Fig. 5.6 [10], while we expect the plane-wave and cylindrical wave results to become closer to each other as point of interest approches the cylinder surface (at which both use external coils only,) we note an increasing deviation after some point. This deviation is due to the inconsistency between plane-wave structure and cylindrical body geometry which results in less ISNR than expected from plane-wave simulation.

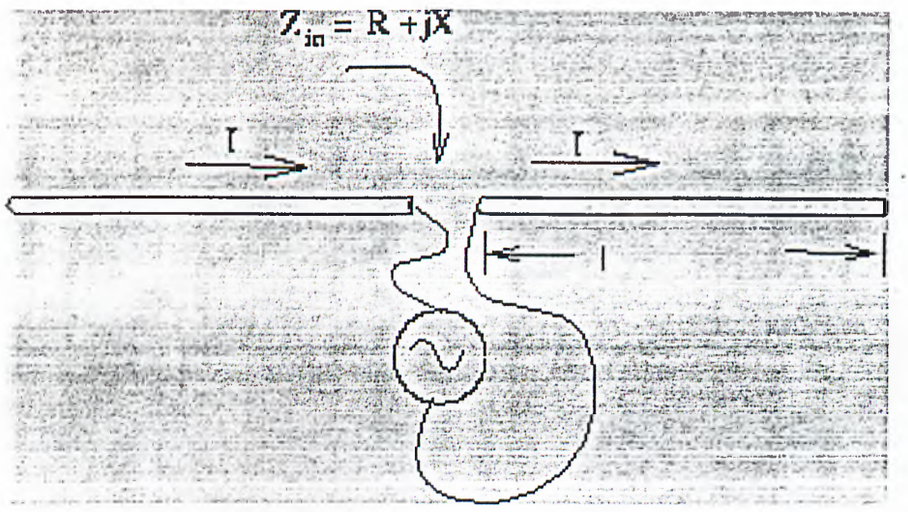

Figure 5.5: Basic structure of a loopless antenna.

Next we show several maps of the right-hand polarized magnetic field $H_+(\vec{r})$ as function of space within the body when the point of interest r_0 is just next to the hole, in the midway between the hole and the cylinder surface, and far from the hole. $\phi = 0$ is considered to be at the vertical line in cross sectional maps, while longitudinal maps range from $z = -.5m$ to $z = .5m$ and from $r = -r_b$ to $r = r_b$. For all these maps $\phi_0 = 0$ and $z_0 = 0$. These maps are produced by the same program used in simulating the EM field after computing the optimum weights. For each point in the space, H_+ is computed and a proportional value of brightness is put on a grey-scale image. This map can be interpreted as relative ISNR as function of space. Note that for each map, the electromagnetic field is optimized for one point of interest r_0.

As shown in the following maps, when the point of interest is close to the hole, high H_+ signal (bright region) is concentrated around the hole, which reflects the effect of using internal coil, Fig. 5.7. Bright region can be interpreted as shadow of coil, shadow around the hole indicates an internal coil

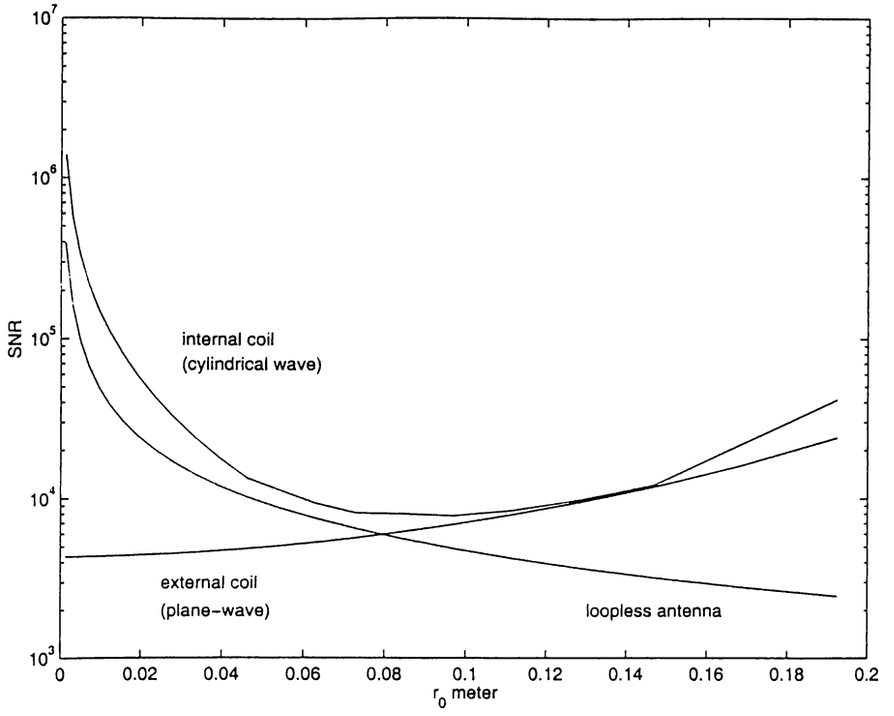

Figure 5.6: Intrinsic Signal-to-Noise Ratio vs. r_0 for internal coil (cylindrical wave), external coil (plane-wave) and loopless antenna.

(source.) On the other hand, low H_+ signal at the cylinder outer surface shows the absence of external coil since they would be far from the point of interest.

Internal coil is placed somewhere inside the hole as indicated by Fig. 5.8.

When the point of interest is far from the hole, we see high H_+ signal shadow of coil) near the outer surface, which means an external coil needs to be used to achieve the optimum electromagnetic field Fig. 5.9. While low H_+ signal around the hole shows that no internal coil is placed for this optimization.

External coil placement can be anywhere outside the body according to r_0 location. Fig. 5.10 shows how an external coil can be placed.

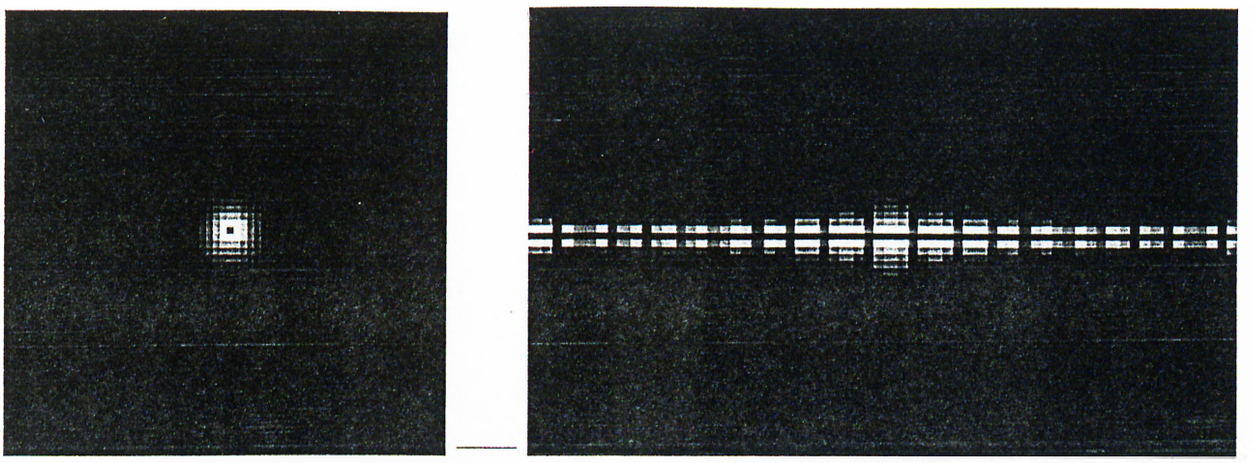

Figure 5.7: Perspective map of H_+ for point of interest $r_0(.0012, 0, 0)$.

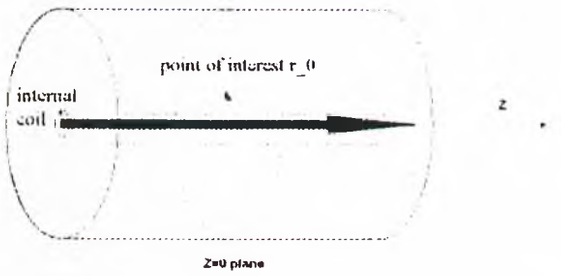

Figure 5.8: Internal coil placement inside the hole.

Both internal and external coils are placed when the point of interest is in the midway along the radius as indicated by coil shadows shown in Figure (Fig. 5.11). Both coils try to contribute to a higher H_+ while keeping as low resistance as possible.

Another observation is that when the point of interest is near the hole, only the *0th* order Bessel and Neumann functions are sufficient for the optimization

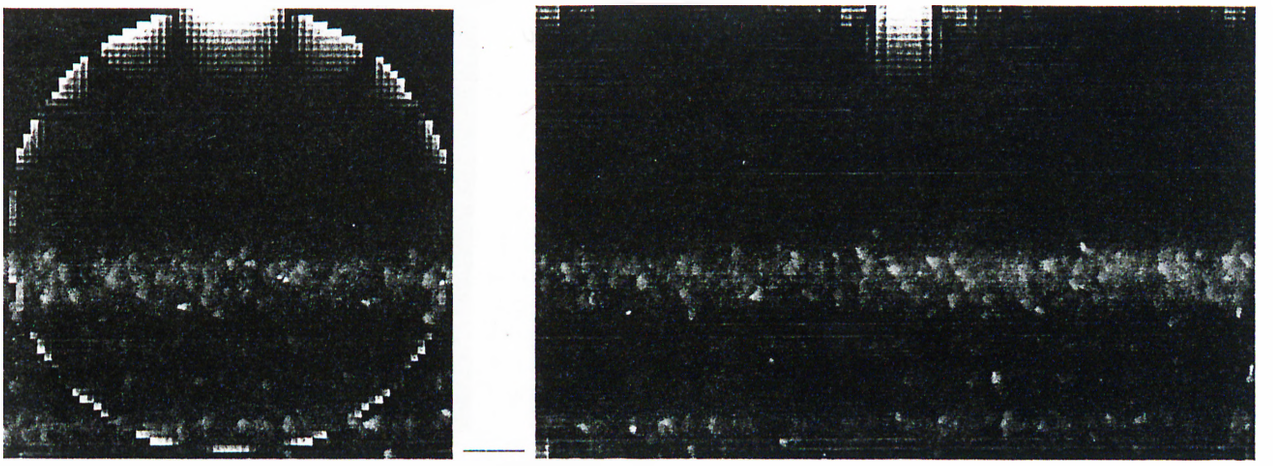

Figure 5.9: Perspective map of H_+ for point of interest $r_0(.1924, 0, 0)$.

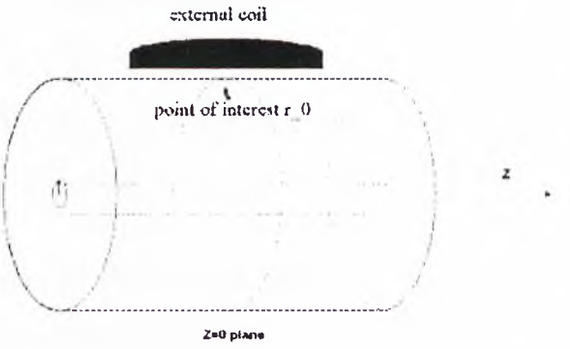

Figure 5.10: External coil placement outside the body.

(only functions of order *0* contribute to the ISNR significantly, higher order functions are marginal in their contributions), and so no angular variation is noticed, fig(Fig. 5.7). This is due to the nature of the angular dependency of the field modes used which are of the form $e^{jm\phi}$, m is the Bessel order. As r_0 moves away from the hole radius r_h, more Bessel (and Neumann) modes (higher m) contribute significantly to ISNR, and hence the angular variation is noted in the electromagnetic field distribution as shown in Fig. 5.11,Fig. 5.9.

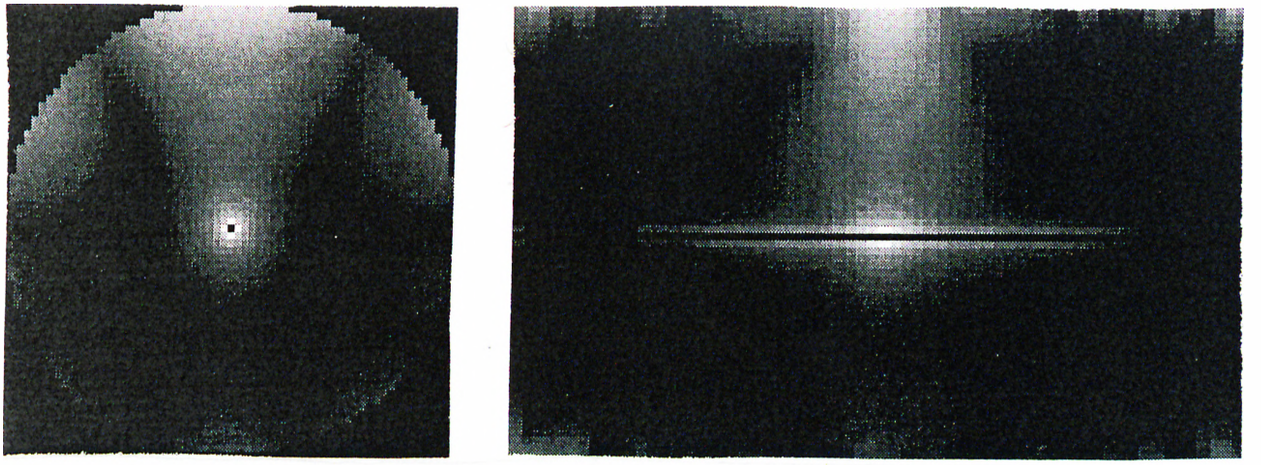

Figure 5.11: Perspective map of H_+ for point of interest $r_0(.0969, 0, 0)$.

Some information about coil configuration may be deduced from the Bessel order m used in the field representation. For example *0th* order cylindrical waves can be excited by a dipole antenna (i.e loopless antenna), 1st order waves may be excited by a loop antenna, 2nd order waves may be wxcited by quadruple antenna etc.

By examining several $\mathbf{R}$ matrices and associated $\mathbf{b}$ vectors in the quantity $1/(\mathbf{b}^T\mathbf{R}\mathbf{b}^*)$, it can be noticed that when r_0 is close to hole, Neumann functions are dominant in determining ISNR due to high values of H_+ they produce at points close to the origin while Bessel functions are of marginal contribution to SNR. On the other hand, when the point of interest is far from the hole, Bessel functions are dominant in determining the ISNR.

Looking at the longitudinal section maps we see radial and longitudinal dependencies of ISNR,(Fig. 5.7,Fig. 5.11,Fig. 5.9). In all cases, however, signal is maximum at the center (point of interest is at z=0) and decaying toward the edges. The spread of bright values of these maps gives an indication of the

coil length as r_0 changes its position.

For all the results given in this chapter, 25 β_z samples are taken to represent the field modes. These samples are taken to be pure real between the values of -60 and 60 and with sampling step size of $5m^{-1}$. This value of sampling step size is taken so that it satisfies the Nyquist rate criteria. Recalling that we need to represent the electromagnetic field in an interval L=1m along the z-axis, then minimum sampling rate in the frequency domain shoud be chosen greater than $2\pi/L$. By trial and error the value of $5m^{-1}$ is found to be optimum in terms of ISNR maximization and it is close to the theoritical value of 2π. Samples of β_z are taken to be pure real since the cylinder length is considered to be practically infinite. This leads to symmetry in the field amplitude along the z-axis, and only phase variations are allowed to exist under this assumption.

Number of β_z samples is restricted to 25 to avoid numerical errors since the $\mathbf{R}$ matrix becomes more singular as its size gets larger. However sampling of β_z is chosen such that more samples would not contribute significantly to the ISNR.

Chapter 6

CONCLUSION

In this thesis, ultimate intrinsic signal-to-noise ratio (ISNR) of a magnetic resonance imaging experiment using internal coils is studied. In the study, cylindrical wave expansion is used to represent the EM fields generated inside the human body. In cylindrical wave expansion, an arbitrary EM field is represented by a linear combination of cylindrical waves. Reciprocity principle is used to determine the signal voltage induced by some magnetized point inside the body in the receiving coil. By means of scaling the signal voltage is kept to be exactly unity, and ultimate SNR is determined by minimizing the noise level which is directly proportional to the square root of the coil resistance. Computer simulations are performed to find ultimate ISNR values and EM fields distributions for a human body model, the model was taken to be a cylinder containing an axial hole for the internal coil to inserted in. In the simulations, coil resistance, which is taken to be equal to body power loss resistance only, is computed and minimized by finding the optimum weights of the cylindrical

waves used in the EM expansion. Each point of the body , when imaged, results in a different ultimate SNR since the value of signal voltage induced by its magnetization is kept to be equal to 1.

Sinulations are performed for both internal coils and external coils. For external coils plane-wave expansion is used. Results of simulations show that internal coil performance is better than internal coil performance in terms of ultimate ISNR. Ultimate ISNR of internal coils is much higher than (more than 10 times) that of external coils when point of interest is close to (within 2 cm from) the body hole. However ultimate ISNR of internal coils and external coils are almost equal when poit of interest is more than 10cm from the hole. For an object of 25cm radius, internal coil starts to contribute to ISNR when point of interest is 10cm away from it.

Maps of right-hand polarized magnetic field inside human body give indications about coils used in imaging, length of the coil, its proper placement etc... These maps show that when point of interest is further than 14cm from the hole no internal coil is needed to produce images with ultimate ISNR.

Results of this work can be used to find whether there is still room for enhancement in the performance of any coil alredy used in MRI, and how further one can improve it. Loopless design [10] is used as an example coil to be tested. By compairing the ultimate ISNR and the ISNR of a loopless coil, it can be seen that there is much room for improveming loopless coil performance when point of interest is more than 10cm away from the hole. When point of interest is between 4cm and 8cm from the hole loopless coil performs very close to optimum and there is no need for further improvement. Some improvement may be achieved when point of is less than 4cm from the hole.

This study gives a way to find the ultimate intrinsic SNR in MRI experiments and the distribution of the electromagnetic field generated by the required coil inside the human body. However the coil needed to produce the desired electromagnetic field can take several configurations. None of these configurations is known although many attempts to design high-SNR coils have been made. Results of this study give primary information about optimum coil configuration. The modes used in optimization as well as their associated weights give a thorough understanding of the way optimum coil may be designed and how its final shape may look like. An inverse problem starting from the optimum electromagnetic field desired to the coil configuration that produces this field has to be solved in order to find the optimum coil design. Depending on the point of interest location, it is expected that optimum coil design is different accordingly. When more than one design is under consideration, feasibility of coil as well as patient comfort must be considered.

Some practical considerations may result in deviation from the results obtained in this study. Other sources of noise, beside the body resistance, can degrade the ultimate ISNR. Human body is not completely cylindrical and the hole assumed to be at the center of the body may not be exactly at the assumed location in real life application. Perfect uniformity of electromagnetic characteristics is another ideal assumption that may result in deviations from the computed ISNR.

APPENDIX A

Computer program to find optimum ISNR value and mode coefficients for a given body parameters

```
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>
//#include <complexnew.h>
#include "rasterfile.h"
#define RAND_MAX 2000000000
#define complex __complex__  double
#define  EULER 0.57721566490153286060651 2 /* Euler constant */
#define  D 1000 /*dimension of arrays*/
#define  STEP   500
#define  pi      3.14159269
#define  gam     0.577215
int pp,nosam,jum;
double ssnr;
complex abn;
complex conj(complex x)
{return(~x);}
```

```
 double red,imd;
complex dinit;
double real(complex x)
{return (__real__ x);}
double imag(complex x)
{return (__imag__ x);}
complex j,kmag;
double rmin,rmax,ss,drealmax,dimagmax,ss1,ss2;
int nofss,nofssreal,nofssimag;
complex exp(complex x)
{
return(exp(real(x))*(cos(imag(x))+1.0fi*sin(imag(x))));
}
complex expp(double x)
{
return((cos(x)+1.0fi*sin(x)));
}
complex log(complex x)
{
 double mag,xxo,yyo;
 double  arg;
xxo=real(x);yyo=imag(x);mag=sqrt(real(x)*real(x)+imag(x)*imag(x));
if(xxo!=0)
{if(xxo>.0) {arg=atan(yyo/xxo);}
else {if (yyo<0.0)arg=atan(yyo/xxo)-pi;else arg=atan(yyo/xxo)+pi;}}
else {if(yyo>.0) {arg=pi/2;} else {arg=-pi/2;}}
return(log(mag)+1.0fi*arg);
}
complex pow(complex x,int n)
{
if(x==0.0) return(0.0);
else if(n==0) return(1.0);
else if(n<0) return(1.0/pow(x,-n));
else return(x*pow(x,n-1));
}
complex sqrt(complex x)
{
 double mag,arg,xxo,yyo;
complex i;
xxo=real(x);yyo=imag(x);mag=sqrt(real(x)*real(x)+imag(x)*imag(x));
if(xxo!=0)
{if(xxo>.0) {arg=atan(yyo/xxo);}
else {if (yyo<0.0)arg=atan(yyo/xxo)-pi;else arg=atan(yyo/xxo)+pi;}}
else {if(yyo>.0) {arg=pi/2;} else {arg=-pi/2;}}
i=1.0fi;
return(sqrt(mag)*exp(i*arg/2.0));
}
 double accYY;int acc; double abso(complex xx)
{
 return hypot(__real__(xx),__imag__(xx));
}
int nore,noim,nofmodes,acc1;
 double fact( double n)
{
```

```
if (n<1e-3) return (1);
else return(n*fact(n-1));
}
 double gamma( double x)
{
return(fact(x-1));
}
complex B;complex bessel( double v,complex x)
{
 double k;
complex del,sum;
sum=k=0;
del=1;
 double fa;
while (abso(del)>1e-6)
{fa=1/fact(k);del=(pow(-1,k)*pow((x/2.),2*k+v)*fa/gamma(k+v+1));sum=sum+del;k=k+1;}
B=sum;return(sum);}
 double suin( double n)
{int k;double m,sum;
sum=0;m=1;
for(k=1;k<n+1;k++)
{sum=sum+1/m;m=m+1;}
return(sum);}
 double digm( double n)
{return(-gam+suin(n-1));}
complex besselY( double v,complex x)
 {complex sum,del;double k;
sum=k=0;del=1;
 double fa;
 if(v==0)
{while (abso(del)>1e-6)
{k=k+1;fa=1/fact(k);del=pow(-1,k)*pow(x/2.0,2*k)*fa*suin(k)*fa;sum=sum+del;}
return(2/pi*bessel(v,x)*(log(x/2.0)+gam)-2/pi*sum);
}
else
{complex sum1,sum2,del2;double k1,k2;
int i;k1=0;sum1=0;
 double fa;
for(i=0;i<v;i++)
{fa=1/fact(k1);sum1=sum1+fact(v-k1-1)*fa*pow(x/2.0,2*k1-v);k1=k1+1;}
k2=0;sum2=0;del2=1;
 double fav;
while (abso(del2)>1e-6)
{fa=1/fact(k2);fav=1/fact(k2+v);del2=pow(-1,k2)*pow(x/2.0,2*k2+v)*fa*(digm(k2+v+1)+digm(k2+1))*fav;sum2=sum2+del2;k2=k2+1;}
return(2/pi*bessel(v,x)*log(x/2.0)-1/pi*sum1-1/pi*sum2);}}
complex integ(complex qi,complex ql, double l, double mu, double nu, double x1, double x2)
{
int m,k,n,cnd,cndk;
complex f,s1,s2,qm,qk,a,g;
 double x[2],fnu,fmu,ym,yk,ni,np,nk;
complex y[2];
x[0]=x1;x[1]=x2;fnu=fact(nu);fmu=fact(mu);qm=ql*ql/4.0;qk=conj(qi)*conj(qi)/4.0;
if(l!=-1 || mu!=0 || nu!=0 ) cnd=1; else cnd=0;
for(n=1;n>=0;n--)
```

```
{ym=yk=x[n]*x[n];
s2=0.0;
for(k=acc;k>=0;k--)
  {
  if(cnd==1 || k!=0) cndk=1; else cndk=0;
  nk=2*k+l+mu+nu+1;s1=1.0;
  for(m=acc;m>0;m--)
      {
nl=nk+2.*m;np=nl-2.;
if(cndk==1 || m!=1)
       { f=-1*qm*ym/m/(m+nu)*np/nl;s1=s1*f+1; }
else {s1=s1*(-1*qm*ym/2.);}
      }
if (cndk==1)
a=s1/fnu/(2*k+l+mu+nu+1);
else a=s1+log(x[n]);
if(k!=0)
g=-1*qk*yk/k/(k+mu);
else g=1/fmu*pow(x[n],l+mu+nu+1)*pow(conj(qi)/2.,mu)*pow(ql/2.,nu);
s2=(a+s2)*g;
   }
y[n]=s2;
 }
return(y[1]-y[0]); }
 double chos( double k, double m)
{return(fact(k)/fact(m)/fact(k-m));}
complex integJY(complex qi,complex ql, double l, double mu, double nu, double x1, double x2)
{complex h,e,c,b,sum,g,a,del,max3;
 double suinnow,suinprev,lnx;
int i,k,lll,m,n,cnd,cndk;
complex f,s1,s2,qm,qk,r;
 double fnu,fmu,ym,yk,ii,kk,ll,mm;double x[2];
complex y[2];
x[0]=x1;x[1]=x2;fnu=fact(nu);fmu=fact(mu);qk=conj(qi*qi)/4.;qm=ql*ql/4.0;
if(l!=-1 || mu!=0 || nu!=0 ) cnd=1; else cnd=0;
if(nu==0)
{
for(n=1;n>=0;n--)
{
ym=x[n]*x[n];lnx=log(x[n]);s2=0.0;
for(k=acc;k>=0;k--)
 {
if(cnd==1 || k!=0) cndk=1;else cndk=0;
  s1=1.0;
  for(m=acc;m>0;m--)
      {
      if(cndk==1 || m!=1)
       { f=-1*qm*ym/m/(m+nu)*(2*(k+m-1)+l+mu+nu+1.)/(2*(k+m)+l+mu+nu+1);s1=s1*f+1; }
else {s1=s1*(-1*qm*ym/2.);}
      }
if (cndk==1)
a=s1/fnu/(2*k+l+mu+nu+1);
else a=s1+lnx;
yk=x[n]*x[n];
```

```
if(k!=0)
g=-1*qk*yk/k/(k+mu);
else g=1/fmu*pow(x[n],1+mu+nu+1)*pow(conj(qi)/2.,mu)*pow(ql/2.,nu);s2=(a+s2)*g;}
y[n]=s2;}
r=2./pi*log(ql/2.0)*(y[1]-y[0]);
for(n=1;n>=0;n--)
{ym=x[n]*x[n];lnx=log(x[n]);s2=0.0;
for(k=acc;k>=0;k--)
 {
if(cnd==1 || k!=0) cndk=1;else cndk=0;
  s1=1.0;
  for(m=acc;m>0;m--)
      {
        if(cndk==1 || m!=1)
       { f=-1*qm*ym/m/(m+nu)*(2*(k+m-1)+1+mu+nu+1.)/(2*(k+m)+1+mu+nu+1);
f=f*(lnx-1/(2*(k+m)+1+mu+nu+1))/(lnx-1./(2*(k+m-1)+1+mu+nu+1.));
        s1=s1*f+1; }
else {s1=s1*(-1*qm*ym/2.)*(lnx-.5);}
      }
if (cndk==1)
a=s1/fnu/(2*k+1+mu+nu+1)*(lnx-1/(2*k+mu+nu+1+1.));
else a=s1+pow(lnx,2.)/2.;yk=x[n]*x[n];
if(k!=0)
g=-1*qk*yk/k/(k+mu);
else g=1/fmu*pow(x[n],1+mu+nu+1)*pow(conj(qi)/2.,mu)*pow(ql/2.,nu);
s2=(a+s2)*g;   }
y[n]=s2;}
r=r+2./pi*(y[1]-y[0]);
for(n=1;n>=0;n--)
{
s2=0.0;
for(k=acc;k>=0;k--)
 {if(cnd==1 || k!=0) cndk=1;else cndk=0;
  s1=1.0;
  for(m=acc;m>0;m--)
      {ym=x[n]*x[n];
if(cndk==1 || m!=1)
       { f=-1*qm*ym/m/(m+nu)*(2*(k+m-1)+1+mu+nu+1.)/(2*(k+m)+1+mu+nu+1);
        s1=s1*f+1; }
else {s1=s1*(-1*qm*ym/2.);}
      }
if (cndk==1)
{a=s1/fnu/(2*k+1+mu+nu+1);}
else {a=s1+log(x[n]);}
yk=x[n]*x[n];
if(k!=0)
g=-1*qk*yk/k/(k+mu);
else g=1/fmu*pow(x[n],1+mu+nu+1)*pow(conj(qi)/2.,mu)*pow(ql/2.,nu);
s2=(a+s2)*g; }
y[n]=s2;}
r=r+2./pi*gam*(y[1]-y[0]);for(n=1;n>=0;n--)
{s2=0.0;
for(k=acc;k>=0;k--)
 {
```

```
  s1=1.0;mm=acc;suinnow=suin(mm);
  for(m=acc;m>1;m--)
      {
suinprev=suinnow-1/mm;ym=x[n]*x[n];f=-ym*qm/m/m*suinnow/suinprev/(2*(k+m)+mu+nu+l+1)*(2*(k+m-1)+mu+nu+l+1);suinnow=suinprev;s1=s1*f+1.;mm-=1.;}
a=-1*s1*ym*qm/(2*k+2+l+mu+nu+1.);yk=x[n]*x[n];
if(k!=0)
g=-1*1*qk*yk/k/(k+mu);
else g=1/fmu*pow(x[n],l+mu+nu+1)*pow(conj(qi)/2.,mu)*pow(ql/2.,nu);
s2=(a+s2)*g;   }
y[n]=s2;}
r=r-2./pi*(y[1]-y[0]);
return(r);}
else
{
for(n=1;n>=0;n--)
{
ym=x[n]*x[n];lnx=log(x[n]);s2=0.0;
for(k=acc;k>=0;k--)
 {
if(cnd==1 || k!=0) cndk=1;else cndk=0;
  s1=1.0;
  for(m=acc;m>0;m--)
      {
        if(cndk==1 || m!=1)
       { f=-1*1*qm*ym/m/(m+nu)*(2*(k+m-1)+l+mu+nu+1.)/(2*(k+m)+l+mu+nu+1);s1=s1*f+1; }
else {s1=s1*(-1*qm*ym/2.);}
      }
if (cndk==1)
a=s1/fnu/(2*k+l+mu+nu+1);
else a=s1+lnx;
yk=x[n]*x[n];
if(k!=0)
g=-1*1*qk*yk/k/(k+mu);
else g=1/fmu*pow(x[n],l+mu+nu+1)*pow(conj(qi)/2.,mu)*pow(ql/2.,nu);
s2=(a+s2)*g;   }
y[n]=s2;}
r=2./pi*log(ql/2.0)*(y[1]-y[0]);
for(n=1;n>=0;n--)
{ym=x[n]*x[n];lnx=log(x[n]);s2=0.0;
for(k=acc;k>=0;k--)
 {
if(cnd==1 || k!=0) cndk=1;else cndk=0;
  s1=1.0;
  for(m=acc;m>0;m--)
      {
        if(cndk==1 || m!=1)

       { f=-1*1*qm*ym/m/(m+nu)*(2*(k+m-1)+l+mu+nu+1.)/(2*(k+m)+l+mu+nu+1);
f=f*(lnx-1/(2*(k+m)+l+mu+nu+1))/(lnx-1./(2*(k+m-1)+l+mu+nu+1.));s1=s1*f+1; }
else {s1=s1*(-1*qm*ym/2.)*(lnx-.5);}}
if (cndk==1)
a=s1/fnu/(2*k+l+mu+nu+1)*(lnx-1/(2*k+mu+nu+l+1.));
else a=s1+pow(lnx,2.)/2.;
yk=x[n]*x[n];
```

```
if(k!=0)
g=-1*1*qk*yk/k/(k+mu);
else g=1/fmu*pow(x[n],1+mu+nu+1)*pow(conj(qi)/2.,mu)*pow(ql/2.,nu);
s2=(a+s2)*g;   }
y[n]=s2;}
r=r+2./pi*(y[1]-y[0]);
for(n=1;n>=0;n--)
{
ym=x[n]*x[n];lnx=log(x[n]);s2=0.0;
for(k=acc;k>=0;k--)
 {
if(cnd==1 || k!=0) cndk=1;else cndk=0;
  s1=1.0;
  for(m=acc;m>0;m--)
      {
        if(cndk==1 || m!=1)
       { f=-1*1*qm*ym/m/(m+nu)*(2*(k+m-1)+1+mu+nu+1.)/(2*(k+m)+1+mu+nu+1);
f=f*(digm(nu+m+1.)+digm(m+1.))/(digm(nu+m)+digm(m));s1=s1*f+1; }
else {s1=s1*(-1*qm*ym/2.);}
      }
if (cndk==1)
a=s1/fnu/(2*k+1+mu+nu+1)*(digm(nu+1.)+digm(1.));
else a=s1*2*digm(2.)+lnx;
yk=x[n]*x[n];
if(k!=0)
g=-1*1*qk*yk/k/(k+mu);
else g=1/fmu*pow(x[n],1+mu+nu+1)*pow(conj(qi)/2.,mu)*pow(ql/2.,nu);
s2=(a+s2)*g;   }
y[n]=s2;}
r=r-1./pi*(y[1]-y[0]);
for(n=1;n>=0;n--)
{
e=0.0;
double fa,famu;
kk=0.0;
  for(k=0;k<acc;k++)
   {
fa=1/fact(kk);famu=1/fact(kk+mu);h=pow(-1,kk)*pow(conj(qi/2.),kk+mu)*fa*pow(conj(qi/2.),kk)*famu;mm=0.0;
    for(m=0;m<=nu-1;m++)
{
g=fact(nu-mm-1.)/fact(mm)*pow(ql/2.,2*mm-nu);
if((2*(kk+mm)+mu-nu+1+1.)!=0)
e=e+h*g*pow(x[n],2*(kk+mm)+mu-nu+1+1.)/(2*(kk+mm)+mu-nu+1+1.);
else
e=e+g*h*(log(x[n]));mm+=1.;}
kk+=1.;}
y[n]=e;}
r=r-1./pi*(y[1]-y[0]);return(r);}}
complex integYY(complex qi,complex ql, double l, double mu, double nu, double x1, double x2)
{
int m,k,n;
complex f,s1,s2,qm,qk,a,g,r,e,h;
 double mn1,nnp,lnx,lnx2,suinnow,suinnow2,suinprev,suinprev2;
 double x[2],fnu,fmu,ym,yk,n1,np,mm,kk;
```

```
complex y[2];
x[0]=x1;x[1]=x2;fnu=fact(nu);fmu=fact(mu);
if(mu==0 && nu==0)
{
r=0.0;
for(n=1;n>=0;n--)
{
ym=yk=x[n]*x[n];qm=ql*ql/4.0;lnx=log(x[n]);lnx2=lnx*lnx;s2=0.0;
for(k=acc;k>=0;k--)
 {
  s1=1.0;
  for(m=acc;m>0;m--)
      {
n1=2.*(k+m)+l+1.;np=n1-2.;nn1=1./n1;nnp=1./np;
if(l!=-1 || k!=0 || m!=1)
       { f=-1*1.*qm*ym/m/m*(nn1*(lnx2+nn1*(-2.*lnx+nn1*2.)))/(nnp*(lnx2+nnp*(-2.*lnx+nnp*2.)));
        s1=s1*f+1.; }
else s1=s1*(-1.*qm*ym)*(.5*(lnx2+.5*(-2.*lnx+1.)));}
n1=2.*k+1.+l;nn1=1./n1;
if (l!=-1 || k!=0)
a=s1*(nn1*(lnx2+nn1*(-2.*lnx+nn1*2.)));
else a=s1+lnx2*lnx/3.;
qk=conj(qi*qi)/4.0;
if(k!=0)
g=-1*1.*qk*yk/k/k;
else g=1.*pow(x[n],l+1.);
s2=(a+s2)*g;   }
y[n]=s2;}
r=4./pi/pi*(y[1]-y[0]);
for(n=1;n>=0;n--)
{
ym=yk=x[n]*x[n];qm=ql*ql/4.0;lnx=log(x[n]);s2=0.0;
for(k=acc;k>=0;k--)
 {s1=1.0;
  for(m=acc;m>0;m--)
     {n1=2.*(k+m)+l+1.;
np=n1-2.;
if(l!=-1 || k!=0 || m!=1)
       {f=-1.*qm*ym/m/m*np/n1;
f=f*(lnx-1./n1)/(lnx-1./np);
        s1=s1*f+1.; }
else {s1=s1*(-1.*qm*ym/2.)*(lnx-.5);}}
if (l!=-1 || k!=0)
a=s1/(2.*k+l+1.)*(lnx-1./(2.*k+l+1.));
else a=s1+pow(lnx,2.)/2.;qk=conj(qi)*conj(qi)/4.0;
if(k!=0)
g=-1.*qk*yk/k/k;
else g=pow(x[n],l+1.);
s2=(a+s2)*g;   }
y[n]=s2;}
r=r+(8.*gam/pi/pi+4./pi/pi*log(conj(qi)*ql/4.))*(y[1]-y[0]);
for(n=1;n>=0;n--)
{
ym=yk=x[n]*x[n];qm=ql*ql/4.0;s2=0.0;
```

```
for(k=acc;k>=0;k--)
 {s1=1.0;
  for(m=acc;m>0;m--)
      {
n1=2.*(k+m)+l+1.;np=n1-2.;
if(l!=-1 || k!=0 || m!=1)
       {f=-1.*qm*ym/m/m*np/n1;s1=s1*f+1.; }
else {s1=s1*(-1.*qm*ym/2.);}
      }
if (l!=-1 || k!=0)
a=s1/(2.*k+l+1.);
else {a=s1+log(x[n]);}
qk=conj(qi)*conj(qi)/4.0;
if(k!=0)
g=-1.*qk*yk/k/k;
else g=pow(x[n],l+1.);
s2=(a+s2)*g;   }
y[n]=s2;}
r=r+(log(conj(qi/2.))*log(ql/2.)+gam*log(conj(qi)*ql/4.)+gam*gam)*4./pi/pi*(y[1]-y[0]);
for(n=1;n>=0;n--)
{ym=yk=x[n]*x[n];
qm=ql*ql/4.0;s2=0.0;
for(k=acc;k>=0;k--)
 {
  s1=1.0;mm=acc;suinnow=suin(mm);
  for(m=acc;m>1;m--)
      {suinprev=suinnow-1./mm;
n1=2.*(k+m)+l+1.;np=n1-2.;f=-1*ym*qm/m/m*suinnow/suinprev/n1*np;suinnow=suinprev;s1=s1*f+1.;mm-=1.;}
a=-1*s1*ym*qm/(2.*k+2.+l+1.);qk=conj(qi)*conj(qi)/4.0;
if(k!=0)
g=-1.*qk*yk/k/k;
else g=pow(x[n],l+1.);
s2=(a+s2)*g;   }
y[n]=s2;}
r=r-4./pi/pi*(gam+log(conj(qi/2.)))*(y[1]-y[0]);
for(n=1;n>=0;n--)
{ym=yk=x[n]*x[n];qm=conj(qi*qi)/4.0;s2=0.0;
for(k=acc;k>=0;k--)
 {
  s1=1.0;mm=acc;suinnow=suin(mm);
  for(m=acc;m>1;m--)
      {
suinprev=suinnow-1./mm;n1=2.*(k+m)+l+1.;np=n1-2.;f=-ym*qm/m/m*suinnow/suinprev/n1*np;suinnow=suinprev;s1=s1*f+1.;mm-=1.;}
a=-1*s1*ym*qm/(2.*k+2.+l+1.);qk=ql*ql/4.0;
if(k!=0)
g=-1.*qk*yk/k/k;
else g=pow(x[n],l+1.);
s2=(a+s2)*g;   }
y[n]=s2;}
r=r-4./pi/pi*(gam+log(ql/2.))*(y[1]-y[0]);
for(n=1;n>=0;n--)
{qm=ql*ql/4.0;ym=yk=x[n]*x[n];lnx=log(x[n]);s2=0.0;
for(k=acc;k>=0;k--)
 {s1=1.0;mm=acc;suinnow=suin(mm);
```

```
  for(m=acc;m>1;m--)
      {ni=2.*(k+m)+l+1.;np=ni-2.;suinprev=suinnow-1./mm;
if(l!=-1 || k!=0 || m!=1)
       { f=-1.*qm*ym/m/m*np/ni*suinnow/suinprev;f=f*(lnx-1./ni)/(lnx-1./np);si=si*f+1.; }
else si=si*(-1.*qm*ym/2.)*(lnx-.5);
suinnow=suinprev;mm-=1.;}
a=-1*si*ym*qm/(2.*k+l+2.+1.)*(lnx-1./(2.*k+l+2.+1.));qk=conj(qi)*conj(qi)/4.0;
if(k!=0)
g=-1.*qk*yk/k/k;
else g=pow(x[n],l+1.);
s2=(a+s2)*g;   }
y[n]=s2;}
r=r-4./pi/pi*(y[1]-y[0]);
for(n=1;n>=0;n--)
{qm=conj(qi*qi)/4.0;ym=yk=x[n]*x[n];lnx=log(x[n]);s2=0.0;
for(k=acc;k>=0;k--)
 {
  si=1.0;mm=acc;suinnow=suin(mm);
  for(m=acc;m>1;m--)
      {ni=2.*(k+m)+l+1.;np=ni-2.;suinprev=suinnow-1./mm;
if(l!=-1 || k!=0 || m!=1)
       { f=-1.*qm*ym/m/m*np/ni*suinnow/suinprev;f=f*(lnx-1./ni)/(lnx-1./np);si=si*f+1.; }
else si=si*(-1.*qm*ym/2.)*(lnx-.5);suinnow=suinprev;mm-=1.;}
a=-1*si*ym*qm/(2.*k+2.+l+1.)*(lnx-1./(2.*k+2.+l+1.));qk=ql*ql/4.0;
if(k!=0)
g=-1.*qk*yk/k/k;
else g=pow(x[n],l+1.);
s2=(a+s2)*g;   }
y[n]=s2;}
r=r-4./pi/pi*(y[1]-y[0]);
for(n=1;n>=0;n--)
{ym=yk=x[n]*x[n];
qm=ql*ql/4.0;s2=0.0;kk=acc;suinnow2=suin(kk);
for(k=acc;k>0;k--)
 {suinprev2=suinnow2-1./kk;si=1.0;mm=acc;suinnow=suin(mm);
  for(m=acc;m>1;m--)
      {ni=2.*(k+m)+l+1.;np=ni-2.;suinprev=suinnow-1./mm;f=-ym*qm/m/m*suinnow/suinprev/ni*np;suinnow=suinprev;si=si*f+1.;mm-=1.;}
a=-1*si*ym*qm/(2.*k+2.+l+1.);qk=conj(qi)*conj(qi)/4.0;g=-1.*qk*yk/k/k*suinnow2/suinprev2;suinnow2=suinprev2;
if (k!=1)
s2=(a+s2)*g;
else
s2=(a+s2)*(-1.)*qk*yk*pow(x[n],l+1.);
kk-=1.;  }
y[n]=s2;}
r=r+4./pi/pi*(y[1]-y[0]);return(r);}
else if (mu==0 && nu!=0)
{r=0.0;
for(n=1;n>=0;n--)
{
ym=yk=x[n]*x[n];qm=ql*ql/4.0;lnx=log(x[n]);lnx2=lnx*lnx;s2=0.0;
for(k=acc;k>=0;k--)
 {si=1.0;
  for(m=acc;m>0;m--)
      {ni=2.*(k+m)+l+nu+1.;np=ni-2.;nni=1./ni;nnp=1./np;f=-1.*qm*ym/m/(m+nu)*(nni*(lnx2+nni*(-2.*lnx+nni*2.)))/(nnp*(lnx2+nnp*(-2.*lnx+nnp*2.)));si*
```

```
n1=2.*k+1.+l+nu;nn1=1./n1;a=s1/fnu*(nn1*(lnx2+nn1*(-2.*lnx+nn1*2.)));qk=conj(qi*qi)/4.0;
if(k!=0)
g=-1.*qk*yk/k/k;
else g=1.*pow(x[n],l+1.+nu)*pow(ql/2.,nu);
s2=(a+s2)*g;   }
y[n]=s2;}
r=4./pi/pi*(y[1]-y[0]);
for(n=1;n>=0;n--)
{
ym=yk=x[n]*x[n];qm=qi*ql/4.0;lnx=log(x[n]);s2=0.0;
for(k=acc;k>=0;k--)
 {s1=1.0;
  for(m=acc;m>0;m--)
      {n1=2.*(k+m)+l+nu+1.;np=n1-2.;f=-1.*qm*ym/m/(m+nu)*np/n1;f=f*(lnx-1./n1)/(lnx-1./np);s1=s1*f+1.; }
a=s1/fnu/(2.*k+l+1.+nu)*(lnx-1./(2.*k+l+nu+1.));qk=conj(qi)*conj(qi)/4.0;
if(k!=0)
g=-1.*qk*yk/k/k;
else g=pow(x[n],l+1.+nu)*pow(ql/2.,nu);
s2=(a+s2)*g; }
y[n]=s2;}
r=r+4./pi/pi*(log(conj(qi)*ql/4.)+gam)*(y[1]-y[0]);
for(n=1;n>=0;n--)
{ym=yk=x[n]*x[n];qm=ql*ql/4.0;s2=0.0;
for(k=acc;k>=0;k--)
 {s1=1.0;
  for(m=acc;m>0;m--)
      {n1=2.*(k+m)+l+nu+1.;np=n1-2.;f=-1.*qm*ym/m/(m+nu)*np/n1;s1=s1*f+1.; }
a=s1/fnu/(2.*k+l+nu+1.);qk=conj(qi)*conj(qi)/4.0;
if(k!=0)
g=-1.*qk*yk/k/k;
else g=pow(x[n],l+nu+1.)*pow(ql/2.,nu);
s2=(a+s2)*g;}
y[n]=s2;}
r=r+(log(conj(qi/2.))*log(ql/2.)+gam*log(ql/2.))*4./pi/pi*(y[1]-y[0]);
for(n=1;n>=0;n--)
{e=0.0;kk=0.0;double fr;
  for(k=0;k<acc;k++)
   {fr=1/fact(kk);h=pow(-1,kk)*pow(conj(qi/2.),kk)*fr*pow(conj(qi/2.),kk)*fr;mm=0.0;
for(m=0;m<=nu-1;m++)
{
g=fact(nu-mm-1.)/fact(mm)*pow(ql/2.,2*mm-nu);
if((2*(kk+mm)-nu+l+1.)!=0)
e=e+h*g*pow(x[n],2*(kk+mm)-nu+l+1.)/(2*(kk+mm)-nu+l+1.);
else
e=e+g*h*(log(x[n]));mm+=1.;}
kk+=1.;}
y[n]=e;}
r=r-2.*gam/pi/pi*(y[1]-y[0]);
for(n=1;n>=0;n--)
{ym=yk=x[n]*x[n];qm=ql*ql/4.;lnx=log(x[n]);s2=0.0;
for(k=acc;k>=0;k--)
 {s1=1.0;
  for(m=acc;m>0;m--)
      {
```

```
        f=-1*qm*ym/m/(m+nu)*(2*(k+m-1)+l+nu+1.)/(2*(k+m)+l+nu+1);f=f*(digm(nu+m+1.)+digm(m+1.))/(digm(nu+m)+digm(m));s1=s1*f+1; }
a=s1/fnu/(2*k+l+nu+1)*(digm(nu+1.)+digm(1.));qk=conj(qi)*conj(qi)/4.0;yk=x[n]*x[n];
if(k!=0)
g=-1*qk*yk/k/(k);
else g=pow(x[n],l+nu+1)*pow(ql/2.,nu);
s2=(a+s2)*g;   }
y[n]=s2;}r=r-2.*gam/pi/pi*(y[1]-y[0]);
for(n=1;n>=0;n--)
{ym=yk=x[n]*x[n];qm=conj(qi*qi)/4.0;s2=0.0;
for(k=acc;k>=0;k--)
 {s1=1.0;mm=acc;suinnow=suin(mm);
  for(m=acc;m>1;m--)
      {suinprev=suinnow-1./mm;n1=2.*(k+m)+l+nu+1.;np=n1-2.;f=-ym*qm/m/m*suinnow/suinprev/n1*np;suinnow=suinprev;s1=s1*f+1.;mm-=1.;}
a=-1*s1*ym*qm/(2.*k+2.+l+nu+1.);qk=ql*ql/4.0;
if(k!=0)
g=-1.*qk*yk/k/(k+nu);
else g=pow(x[n],l+1.+nu)*pow(ql/2.,nu);
s2=(a+s2)*g;   }
y[n]=s2;}
r=r-4./pi/pi*log(ql/2.)*(y[1]-y[0]);
for(n=1;n>=0;n--)
{qm=conj(qi*qi)/4.0;ym=yk=x[n]*x[n];lnx=log(x[n]);s2=0.0;
for(k=acc;k>=0;k--)
 {s1=1.0;mm=acc;suinnow=suin(mm);
  for(m=acc;m>1;m--)
      {
n1=2.*(k+m)+l+nu+1.;np=n1-2.;suinprev=suinnow-1./mm;
       { f=-1.*qm*ym/m/m*np/n1*suinnow/suinprev;f=f*(lnx-1./n1)/(lnx-1./np);s1=s1*f+1.; }
suinnow=suinprev;mm-=1.;}
a=-1*s1*ym*qm/(2.*k+l+nu+2.+1.)*(lnx-1./(2.*k+l+nu+2.+1.));qk=ql*ql/4.0;
if(k!=0)
g=-1.*qk*yk/k/(k+nu);
else g=pow(x[n],l+nu+1.)*pow(ql/2.,nu);
s2=(a+s2)*g;   }
y[n]=s2;}
r=r-4./pi/pi*(y[1]-y[0]);
for(n=1;n>=0;n--)
{e=0.0;kk=0.0;
 double fre;
  for(k=0;k<acc;k++)
   {fre=1/fact(kk);h=pow(-1,kk)*pow(conj(qi/2.),kk)*fre*pow(conj(qi/2.),kk)*fre;
mm=0.0;
    for(m=0;m<=nu-1;m++)
{g=fact(nu-mm-1.)/fact(mm)*pow(ql/2.,2*mm-nu);
if((2*(kk+mm)-nu+l+1.)!=0)
e=e+h*g*pow(x[n],2*(kk+mm)-nu+l+1.)/(2*(kk+mm)-nu+l+1.);
else
e=e+g*h*(log(x[n]));mm+=1.;}
kk+=1.;}
y[n]=e;}
r=r-2./pi/pi*log(conj(qi/2.))*(y[1]-y[0]);
for(n=1;n>=0;n--)
{lnx=log(x[n]);e=0.0;kk=0.0;
 double fre;
```

```
  for(k=0;k<acc;k++)
   {fre=1/fact(kk);h=pow(-1,kk)*pow(conj(qi/2.),kk)*fre*pow(conj(qi/2.),kk)*fre;
mm=0.0;
    for(m=0;m<=nu-1;m++)
{n1=2*(kk+mm)-nu+l+1.;g=fact(nu-mm-1.)/fact(mm)*pow(ql/2.,2*mm-nu);if((2*(kk+mm)-nu+l+1.)!=0)e=e+h*g*pow(x[n],n1)/n1*(lnx-1/n1);
else
e=e+g*h*(lnx*lnx/2.);mm+=1.;}
kk+=1.;}
y[n]=e;}
r=r-2./pi/pi*(y[1]-y[0]);
for(n=1;n>=0;n--)
{ym=yk=x[n]*x[n];qm=ql*ql/4.;lnx=log(x[n]);s2=0.0;
for(k=acc;k>=0;k--)
 {s1=1.0;
  for(m=acc;m>0;m--)
      {f=-1*qm*ym/m/(m+nu)*(2*(k+m-1)+l+mu+nu+1.)/(2*(k+m)+l+mu+nu+1);f=f*(digm(nu+m+1.)+digm(m+1.))/(digm(nu+m)+digm(m));s1=s1*f+1; }
a=s1/fnu/(2*k+l+mu+nu+1)*(digm(nu+1.)+digm(1.));qk=conj(qi)*conj(qi)/4.0;yk=x[n]*x[n];
if(k!=0)
g=-1*qk*yk/k/(k+mu);
else g=1/fmu*pow(x[n],l+mu+nu+1)*pow(conj(qi)/2.,mu)*pow(ql/2.,nu);
s2=(a+s2)*g;   }
y[n]=s2;}
r=r-2./pi/pi*log(conj(qi/2.))*(y[1]-y[0]);
for(n=1;n>=0;n--)
{
ym=yk=x[n]*x[n];qm=ql*ql/4.;lnx=log(x[n]);s2=0.0;
for(k=acc;k>=0;k--)
 {s1=1.0;
  for(m=acc;m>0;m--)
      {n1=2*(k+m)+l+mu+nu+1;np=n1-2.;f=-1*qm*ym/m/(m+nu)*np/n1*(lnx-1./n1)/(lnx-1./np);f=f*(digm(nu+m+1.)+digm(m+1.))/(digm(nu+m)+digm(m));s1=s1*f+1
a=s1/fnu/(2*k+l+mu+nu+1)*(digm(nu+1.)+digm(1.))*(lnx-1/(2*k+l+mu+nu+1));
qk=conj(qi)*conj(qi)/4.0;yk=x[n]*x[n];
if(k!=0)
g=-1*qk*yk/k/(k+mu);
else g=1/fmu*pow(x[n],l+mu+nu+1)*pow(conj(qi)/2.,mu)*pow(ql/2.,nu);
s2=(a+s2)*g;   }
y[n]=s2;}
r=r-2./pi/pi*(y[1]-y[0]);
for(n=1;n>=0;n--)
{e=0.0;kk=0.0;double rfe;
for(k=0;k<acc;k++)
   {rfe=1/fact(kk);h=pow(-1.,kk)*pow(conj(qi/2.),kk)*rfe*suin(kk)*pow(conj(qi/2.),kk)*rfe;mm=0.0;
complex er;
er=pow(conj(qi/2.),2*kk);er=pow(fact(kk),2);er=suin(kk);
    for(m=0;m<=nu-1;m++)
{g=fact(nu-mm-1.)/fact(mm)*pow(ql/2.,2*mm-nu);
if((2*(kk+mm)-nu+l+1.)!=0)
e=e+h*g*pow(x[n],2*(kk+mm)-nu+l+1.)/(2*(kk+mm)-nu+l+1.);
else
e=e+g*h*(log(x[n]));mm+=1.;}
kk+=1.;}
y[n]=e;}
r=r+2./pi/pi*(y[1]-y[0]);
for(n=1;n>=0;n--)
```

```
{yk=ym=x[n]*x[n];qm=ql*ql/4.;lnx=log(x[n]);s2=0.0;kk=acc;suinnow=suin(kk);
for(k=acc;k>0;k--)
 {s1=1.0;
  for(m=acc;m>0;m--)
      {f=-1*qm*ym/m/(m+nu)*(2*(k+m-1)+l+nu+1.)/(2*(k+m)+l+nu+1);f=f*(digm(nu+m+1.)+digm(m+1.))/(digm(nu+m)+digm(m));s1=s1*f+1; }
a=s1/fnu/(2*k+l+nu+1)*(digm(nu+1.)+digm(1.));qk=conj(qi)*conj(qi)/4.0;yk=x[n]*x[n];suinprev=suinnov-1/kk;
if(k!=1)
g=-1*qk*yk/k/k*suinnow/suinprev;
else g=-pow(x[n],l+nu+1)*pow(ql/2.,nu)*qk*yk;
s2=(a+s2)*g;suinnow=suinprev;kk-=1.;   }
y[n]=s2;}
r=r+2./pi/pi*(y[1]-y[0]);
return(r);
}
else if (mu!=0 && nu==0)
return(conj(integYY(ql,qi,l,nu,mu,x1,x2)));
else
{
r=0.0;
for(n=1;n>=0;n--)
{
ym=yk=x[n]*x[n];qm=ql*ql/4.0;lnx=log(x[n]);lnx2=lnx*lnx;s2=0.0;
for(k=acc;k>=0;k--)
 {s1=1.0;
  for(m=acc;m>0;m--)
      {n1=2.*(k+m)+l+nu+mu+1.;np=n1-2.;nn1=1./n1;nnp=1./np;f=-1.*qm*ym/m/(m+nu)*(nn1*(lnx2+nn1*(-2.*lnx+nn1*2.)))/(nnp*(lnx2+nnp*(-2.*lnx+nn
n1=2.*k+1.+l+nu+mu;nn1=1./n1;a=s1/fnu*(nn1*(lnx2+nn1*(-2.*lnx+nn1*2.)));qk=conj(qi*qi)/4.0;
if(k!=0)
g=-1.*qk*yk/k/(k+mu);
else g=1./fmu*pow(x[n],l+1.+nu+mu)*pow(ql/2.,nu)*pow(conj(qi/2.),mu);
s2=(a+s2)*g;   }
y[n]=s2;}
r=4./pi/pi*(y[1]-y[0]);
for(n=1;n>=0;n--)
{ym=yk=x[n]*x[n];qm=ql*ql/4.0;lnx=log(x[n]);s2=0.0;
for(k=acc;k>=0;k--)
 {s1=1.0;
  for(m=acc;m>0;m--)
      {n1=2.*(k+m)+l+nu+1.+mu;np=n1-2.; f=-1.*qm*ym/m/(m+nu)*np/n1;f=f*(lnx-1./n1)/(lnx-1./np);s1=s1*f+1.; }
a=s1/fnu/(2.*k+l+1.+nu+mu)*(lnx-1./(2.*k+l+nu+mu+1.));qk=conj(qi)*conj(qi)/4.0;
if(k!=0)
g=-1.*qk*yk/k/(k+mu);
else g=1./fmu*pow(x[n],l+1.+nu+mu)*pow(ql/2.,nu)*pow(conj(qi)/2.,mu);
s2=(a+s2)*g;   }
y[n]=s2;}
r=r+4./pi/pi*log(conj(qi)*ql/4.)*(y[1]-y[0]);
for(n=1;n>=0;n--)
{ym=yk=x[n]*x[n];qm=ql*ql/4.0;s2=0.0;
for(k=acc;k>=0;k--)
 {s1=1.0;
  for(m=acc;m>0;m--)
      {
n1=2.*(k+m)+l+nu+mu+1.;np=n1-2.;f=-1.*qm*ym/m/(m+nu)*np/n1;s1=s1*f+1.; }
a=s1/fnu/(2.*k+l+nu+mu+1.);qk=conj(qi)*conj(qi)/4.0;
```

```
if(k!=0)
g=-1.*qk*yk/k/(k+mu);
else g=1./fmu*pow(x[n],1+nu+mu+1.)*pow(ql/2.,nu)*pow(conj(qi)/2.,mu);
s2=(a+s2)*g;   }
y[n]=s2;}
r=r+log(conj(qi/2.))*log(ql/2.)*4./pi/pi*(y[1]-y[0]);
for(n=1;n>=0;n--)
{lnx=log(x[n]);
e=0.0;kk=0.0;
 double gre,gremu;
  for(k=0;k<acc;k++)
   {gre=1/fact(kk);gremu=1/fact(kk+mu);
    h=pow(-1,kk)*pow(conj(qi/2.),kk+mu)*gre*pow(conj(qi/2.),kk)*gremu;mm=0.0;
    for(m=0;m<=nu-1;m++)
{n1=2*(kk+mm)-nu+mu+1+1.;g=fact(nu-mm-1.)/fact(mm)*pow(ql/2.,2*mm-nu);
if((2*(kk+mm)-nu+1+mu+1.)!=0)
e=e+h*g*pow(x[n],n1)/n1*(lnx-1/n1);
else
e=e+g*h*(lnx*lnx/2.);mm+=1.;}
kk+=1.;}
y[n]=e;}
r=r-2./pi/pi*(y[1]-y[0]);
for(n=1;n>=0;n--)
{e=0.0;kk=0.0;
 double fa,famu;
  for(k=0;k<acc;k++)
   {
fa=1/fact(kk);famu=1/fact(kk+mu);h=pow(-1,kk)*pow(conj(qi/2.),kk+mu)*fa*pow(conj(qi/2.),kk)*famu;mm=0.0;
    for(m=0;m<=nu-1;m++)
{
g=fact(nu-mm-1.)/fact(mm)*pow(ql/2.,2*mm-nu);
if((2*(kk+mm)-nu+1+mu+1.)!=0)
e=e+h*g*pow(x[n],2*(kk+mm)-nu+mu+1+1.)/(2*(kk+mm)-nu+mu+1+1.);
else
e=e+g*h*(log(x[n]));mm+=1.;}
kk+=1.;}
y[n]=e;}
r=r-2.*log(conj(qi/2.))/pi/pi*(y[1]-y[0]);
for(n=1;n>=0;n--)
{lnx=log(x[n]);
e=0.0;kk=0.0;
 double fa,fanu;
  for(k=0;k<acc;k++)
   {fa=1/fact(kk);fanu=1/fact(kk+nu);h=pow(-1,kk)*pow(ql/2.,kk+nu)*fa*pow(ql/2.,kk)*fanu;mm=0.0;
    for(m=0;m<=mu-1;m++)
{n1=2*(kk+mm)-mu+nu+1+1.;g=fact(mu-mm-1.)/fact(mm)*pow(conj(qi)/2.,2*mm-mu);
if((2*(kk+mm)-mu+1+nu+1.)!=0)
e=e+h*g*pow(x[n],n1)/n1*(lnx-1/n1);
else
e=e+g*h*(lnx*lnx/2.);mm+=1.;}
kk+=1.;}
y[n]=e;}
r=r-2./pi/pi*(y[1]-y[0]);
for(n=1;n>=0;n--)
```

```
{e=0.0;kk=0.0;
 double fa,fanu;
  for(k=0;k<acc;k++)
   {fa=1/fact(kk);fanu=1/fact(kk+nu);h=pow(-1,kk)*pow(ql/2.,kk+nu)*fa*pow(ql/2.,kk)*fanu;mm=0.0;
    for(m=0;m<=mu-1;m++)
{g=fact(mu-mm-1.)/fact(mm)*pow(conj(qi)/2.,2*mm-mu);
if((2*(kk+mm)-mu+1+nu+1.)!=0)
e=e+h*g*pow(x[n],2*(kk+mm)-mu+nu+1+1.)/(2*(kk+mm)-mu+nu+1+1.);
else
e=e+g*h*(log(x[n]));mm+=1.;}
kk+=1.;}
y[n]=e;}
r=r-2.*log(ql/2.)/pi/pi*(y[1]-y[0]);
for(n=1;n>=0;n--)
{
ym=yk=x[n]*x[n];qm=ql*ql/4.;lnx=log(x[n]);s2=0.0;
for(k=acc;k>=0;k--)
 {s1=1.0;
  for(m=acc;m>0;m--)
      {n1=2*(k+m)+1+mu+nu+1;np=n1-2.;f=-1*qm*ym/m/(m+nu)*np/n1*(lnx-1./n1)/(lnx-1./np);f=f*(digm(nu+m+1.)+digm(m+1.))/(digm(nu+m)+digm(m));s1=s1*f+1
a=s1/fnu/(2*k+1+mu+nu+1)*(digm(nu+1.)+digm(1.))*(lnx-1/(2*k+1+mu+nu+1));qk=conj(qi)*conj(qi)/4.0;
yk=x[n]*x[n];
if(k!=0)
g=-1*qk*yk/k/(k+mu);
else g=1/fmu*pow(x[n],1+mu+nu+1)*pow(conj(qi)/2.,mu)*pow(ql/2.,nu);
s2=(a+s2)*g;   }
y[n]=s2;}
r=r-2./pi/pi*(y[1]-y[0]);
for(n=1;n>=0;n--)
{ym=yk=x[n]*x[n];qm=ql*ql/4.;lnx=log(x[n]);s2=0.0;
for(k=acc;k>=0;k--)
 {s1=1.0;
  for(m=acc;m>0;m--)
{f=-1*qm*ym/m/(m+nu)*(2*(k+m-1)+1+mu+nu+1.)/(2*(k+m)+1+mu+nu+1);f=f*(digm(nu+m+1.)+digm(m+1.))/(digm(nu+m)+digm(m));s1=s1*f+1; }
a=s1/fnu/(2*k+1+mu+nu+1)*(digm(nu+1.)+digm(1.));qk=conj(qi)*conj(qi)/4.0;yk=x[n]*x[n];
if(k!=0)
g=-1*qk*yk/k/(k+mu);
else g=1/fmu*pow(x[n],1+mu+nu+1)*pow(conj(qi)/2.,mu)*pow(ql/2.,nu);
s2=(a+s2)*g;   }
y[n]=s2;}
r=r-2./pi/pi*log(conj(qi/2.))*(y[1]-y[0]);
for(n=1;n>=0;n--)
{
ym=yk=x[n]*x[n];qm=conj(qi*qi)/4.;lnx=log(x[n]);s2=0.0;
for(k=acc;k>=0;k--)
 {s1=1.0;
  for(m=acc;m>0;m--)
      {n1=2*(k+m)+1+mu+nu+1;np=n1-2.;f=-1*qm*ym/m/(m+mu)*np/n1*(lnx-1./n1)/(lnx-1./np);f=f*(digm(mu+m+1.)+digm(m+1.))/(digm(mu+m)+digm(m));s1=s1*f+1
a=s1/fmu/(2*k+1+mu+nu+1)*(digm(mu+1.)+digm(1.))*(lnx-1/(2*k+1+mu+nu+1));qk=ql*ql/4.0;yk=x[n]*x[n];
if(k!=0)
g=-1*qk*yk/k/(k+nu);
else g=1/fnu*pow(x[n],1+mu+nu+1)*pow(conj(qi)/2.,mu)*pow(ql/2.,nu);
s2=(a+s2)*g;   }
y[n]=s2;}
```

```
r=r-2./pi/pi*(y[1]-y[0]);
for(n=1;n>=0;n--)
{ym=yk=x[n]*x[n];qm=conj(qi*qi)/4.;lnx=log(x[n]);s2=0.0;
for(k=acc;k>=0;k--)
 {s1=1.0;
  for(m=acc;m>0;m--)
      {f=-1*qm*ym/m/(m+mu)*(2*(k+m-1)+1+mu+nu+1.)/(2*(k+m)+1+mu+nu+1);f=f*(digm(mu+m+1.)+digm(m+1.))/(digm(mu+m)+digm(m));s1=s1*f+1; }
a=s1/fmu/(2*k+1+mu+nu+1)*(digm(mu+1.)+digm(1.));qk=ql*ql/4.0;yk=x[n]*x[n];
if(k!=0)
g=-1*qk*yk/k/(k+nu);
else g=1/fnu*pow(x[n],1+mu+nu+1)*pow(conj(qi)/2.,mu)*pow(ql/2.,nu);
s2=(a+s2)*g;   }
y[n]=s2;}
r=r-2./pi/pi*log(ql/2.)*(y[1]-y[0]);
for(n=1;n>=0;n--)
{e=0.0;kk=0.0;
 double fa,fanu;
  for(k=0;k<acc;k++)
   {fa=1/fact(kk);fanu=1/fact(kk+nu);h=pow(-1,kk)*pow(ql/2.,kk+nu)*fa*pow(ql/2.,kk)*fanu;h=h*(digm(kk+nu+1)+digm(kk+1));mm=0.0;
    for(m=0;m<=mu-1;m++)
{
g=fact(mu-mm-1.)/fact(mm)*pow(conj(qi)/2.,2*mm-mu);
if((2*(kk+mm)-mu+1+nu+1.)!=0)
e=e+h*g*pow(x[n],2*(kk+mm)-mu+nu+1+1.)/(2*(kk+mm)-mu+nu+1+1.);
else
e=e+g*h*(log(x[n]));mm+=1.;}
kk+=1.;}
y[n]=e;}
r=r+1./pi/pi*(y[1]-y[0]);
for(n=1;n>=0;n--)
{e=0.0;kk=0.0;
 double fa,famu;
  for(k=0;k<acc;k++)
   {fa=1/fact(kk);famu=1/fact(kk+mu);h=pow(-1,kk)*pow(conj(qi)/2.,kk+mu)*fa*pow(conj(qi)/2.,kk)*famu;h=h*(digm(kk+mu+1)+digm(kk+1));
mm=0.0;
    for(m=0;m<=nu-1;m++)
{
g=fact(nu-mm-1.)/fact(mm)*pow(ql/2.,2*mm-nu);
if((2*(kk+mm)-nu+1+mu+1.)!=0)
e=e+h*g*pow(x[n],2*(kk+mm)-nu+mu+1+1.)/(2*(kk+mm)-nu+mu+1+1.);
else
e=e+g*h*(log(x[n]));
mm+=1.;}
kk+=1.;}
y[n]=e;}
r=r+1./pi/pi*(y[1]-y[0]);

for(n=1;n>=0;n--)
{e=0.0;kk=0.0;
  for(k=0;k<=mu-1;k++)
   {h=fact(mu-kk-1.)/fact(kk)*pow(conj(qi/2.),kk-mu)*pow(conj(qi/2.),kk);
mm=0.0;
    for(m=0;m<=nu-1;m++)
{g=fact(nu-mm-1.)/fact(mm)*pow(ql/2.,2*mm-nu);
```

```
if((2*(kk+mm)-nu+l-mu+1.)!=0)
e=e+h*g*pow(x[n],2*(kk+mm)-nu-mu+l+1.)/(2*(kk+mm)-nu-mu+l+1.);
else
e=e+g*h*(log(x[n]));mm+=1.;}
kk+=1.;}
y[n]=e;}
r=r+1./pi/pi*(y[1]-y[0]);
for(n=1;n>=0;n--)
{ym=yk=x[n]*x[n];qm=ql*ql/4.;lnx=log(x[n]);s2=0.0;
for(k=acc;k>=0;k--)
 {s1=1.0;
  for(m=acc;m>0;m--)
      {f=-1*qm*ym/m/(m+nu)*(2*(k+m-1)+l+mu+nu+1.)/(2*(k+m)+l+mu+nu+1);f=f*(digm(nu+m+1.)+digm(m+1.))/(digm(nu+m)+digm(m));s1=s1*f+1; }
a=s1/fnu/(2*k+l+mu+nu+1)*(digm(nu+1.)+digm(1.));qk=conj(qi)*conj(qi)/4.0;yk=x[n]*x[n];
if(k!=0)
g=-1*1*qk*yk/k/(k+mu)*(digm(mu+k+1.)+digm(k+1.))/(digm(mu+k)+digm(k));
else g=1/fmu*pow(x[n],l+mu+nu+1)*pow(conj(qi)/2.,mu)*pow(ql/2.,nu)*(digm(mu+1)+digm(1));
s2=(a+s2)*g;   }
y[n]=s2;}
r=r+1./pi/pi*(y[1]-y[0]);
return(r);}}
void checksize(int n,int m)
     {if( n <1 || m <1)
       {cout<<"matrix size of ("<<n<<","<<m<<") is not appropriate \n";exit(1);}
class matrix
      {
public:
      int *nuser;
      int n,m;
      complex **rowptr;
      complex *elem;

      matrix(int k,int l) ;


inline matrix(matrix &in)
      {n=in.n;m=in.m;elem=in.elem;rowptr=in.rowptr;nuser=in.nuser;
 (*nuser)++;  // number of users is increased by one}
inline ~matrix()
      {(*nuser)--;
if((*nuser)==0 )
{delete[]elem;
   delete[] rowptr;
   delete nuser ;}}

inline complex *operator[](int k)
       {return(rowptr[k]);}
void operator=(matrix &xx);
void operator<(matrix &xx);
friend matrix operator*(matrix &in,complex sc);
friend matrix operator*(complex sc,matrix &in);
complex & operator()(int k,int l);
complex & matrix::operator()(int k);};
complex& matrix::operator()(int k,int l)
```

```
        {
      if(k<1 || k>n || l<1 || l>m)
{
cout<<"matrix index out of range \n";
exit(1);}return(*(rowptr[k-1]+l-1));}
complex & matrix::operator()(int k)
         {if(n!=1 &&  m!=1)
{cout<<"this is matrix not a vector\n";exit(1);}
return(*(elem+k-1));}
matrix::matrix(int k=1,int l=1)
      {
n=k;
m=l;
checksize(n,m);
        nuser=new int;(*nuser)=1;elem=new complex [n*m];int i,j;j=n*m;complex *zz;

for(zz=elem,i=0;i<j;i++)*zz++ = 0;
rowptr= new complex* [n];
if(elem==NULL || rowptr ==NULL)
        {
           cout<<" sorry!! cannot allocate memory for this matrix \n";
   exit(1);
}
 for(i=0;i<n;i++)rowptr[i]= elem+i*m;}
void matrix::operator<(matrix &xx)
    {if((*nuser)==1)
        {delete[] elem ;delete[] rowptr ;delete nuser ;}
        else (*nuser)-- ;n=xx.n;m=xx.m;nuser=xx.nuser;elem=xx.elem;rowptr=xx.rowptr;(*nuser)++;}
 void matrix::operator=(matrix &xx)
     {int i,k;
       register complex *t1,*t2;
if((*nuser)==1)
      {if(n==xx.n && m==xx.m)
  {k=n*m;
    for(i=0,t1=elem,t2=xx.elem;i<k;i++)*t1++ = *t2++;}
  else
  {delete[] elem;delete[] rowptr;n=xx.n;m=xx.m;elem= new complex [n*m];
rowptr= new complex* [n];
    if(elem==NULL || rowptr ==NULL)
    {
cout<<" sorry!! cannot allocate memory for this matrix \n";
exit(1);
            }
    for(i=0;i<n;i++)rowptr[i]= elem+i*m;
    k=n*m;
    for(i=0,t1=elem,t2=xx.elem;i<k;i++)*t1++ = *t2++;}}
      else
      {(*nuser)--;n=xx.n;m=xx.m;elem= new complex [n*m];nuser=new int;(*nuser)=1;rowptr= new complex* [n];
    if(elem==NULL || rowptr ==NULL)
    {
cout<<" sorry!! cannot allocate memory for this matrix \n";
exit(1);}
    for(i=0;i<n;i++)rowptr[i]= elem+i*m;
    k=n*m;
```

```
    for(i=0,t1=elem,t2=xx.elem;i<k;i++)*t1++ = *t2++;}
     }
matrix operator*(matrix &in,complex sc) // in*sc
       {
 int n,m,k;
 register int i;
 register complex *t1,*t2;
matrix result(n=in.n,m=in.m);
k=n*m;
 for(t1=in.elem,t2=result.elem,i=0;i<k;i++)
 *t2++= sc* *t1++;
 return(result);
       }
matrix operator*(complex sc,matrix &in) // sc*in
       {
 int n,m,k;
 register int i;
 register complex *t1,*t2;
 matrix result(n=in.n,m=in.m);

 k=n*m;
 for(t1=in.elem,t2=result.elem,i=0;i<k;i++)
  *t2++= sc* *t1++;
 return(result);
       }
matrix operator+(matrix &a,matrix &b) // a+b
       {
 int n,m,k;
 if((n=a.n) != b.n || (m=a.m)!=b.m)
        {
     cout<<" in addition matrix sizes must be the same\n";
            exit(1);
         }
 k= n*m;


 register int i;

 register complex *t1,*t2,*t3;
matrix result(n,m);
 for(i=0,t1=a.elem,t2=b.elem,t3=result.elem;i<k;i++)
    *t3++ = *t2++ + *t1++;
return(result);}
matrix operator-(matrix &a,matrix &b) // a-b
       {
 int n,m,k;
 if((n=a.n) != b.n || (m=a.m)!=b.m)
{cout<<" in subtraction matrix sizes must be the same\n";exit(1);}
 k= n*m;
 register int i;
register complex *t1,*t2,*t3;
matrix result(n,m);
 for(i=0,t1=a.elem,t2=b.elem,t3=result.elem;i<k;i++)
    *t3++ = *t1++ - *t2++;return(result);}
```

```
matrix operator*(matrix &a,matrix &b) // a*b
       {
 if(a.m != b.n)
 {cout<<"incompatible matrix size in multiplication"<<a.m<<b.n<<"\n";exit(1);}
int i,j,k,n,m,l;
complex sum;
 register complex *t1,*t2;
 n=a.n;m=b.m;l=a.m;
matrix res(n,m);
for(j=0;j<m;j++)
 {for(i=0;i<n;i++)
   {sum=0;t1=a[i];t2=b.elem+j;
     for(k=0;k<l;k++,t2+=m)sum+= *t1++ * *t2;res[i][j]=sum;}}
return(res);}
complex *xtt2;
void rxchg(matrix &in,int k,int l)
     {
int i,m;
complex xx;
if(in.m==1)
{xx=in[k][0];in[k][0]=in[l][0];in[l][0]=xx;return;}
complex *t1,*t2,*t3;
t2=xtt2;
m=in.m;
for(i=0,t1=in[k],t3=t2;i<m;i++)
   *t3++ = *t1++;
for(i=0,t1=in[k],t3=in[l];i<m;i++)
  *t1++ = *t3++;
for(i=0,t1=t2,t3=in[l];i<m;i++)
  *t3++ = *t1++;}
void prxchg(matrix &in,int k,int l)
     {
int i,m;
complex *t1,*t2,*t3;
t2=xtt2;m=in.m;
for(i=k,t1=in[k]+k,t3=t2;i<m;i++)
   *t3++ = *t1++;
for(i=k,t1=in[k]+k,t3=in[l]+k;i<m;i++)
  *t1++ = *t3++;
for(i=k,t1=t2,t3=in[l]+k;i<m;i++)
  *t3++ = *t1++;}
void rowop(matrix &in,int l,int k,complex sc)// l= l-sc*k
     {int i,m;complex *t1,*t2;m=in.m;
for(i=0,t2=in[k],t1=in[l];i<m;i++)
  *t1++ -= sc * *t2++;}
void prowop(matrix &in,int l,int k,complex sc)// l= l-sc*k
     {int i,m;register complex *t1,*t2;m=in.m;
for(i=k,t2=in[k]+k,t1=in[l]+k;i<m;i++)
  *t1++ -= sc * *t2++;}
int maxincol(matrix &in,matrix &y, int k)
    {
       int i,j,n;
        double mmax,t;mmax=0;n=in.n;j=k;
       for(i=k;i<n;i++)
```

```
       {if((t=abso(in[i][k]  ))>mmax)
 {mmax=t;
   j=i;}}
       return(j);}

void donothing()
     {}
void (*erroraction)()=donothing;
matrix solve(matrix &a,matrix &b) // solves ax=b;
       {int i,j,k,n,m;
  complex s,*t1,*t2;
if((n=(a.n)) != a.m)
          {cout<<"matrix must be square to invert\n";
            exit(1);}
  if(a.n != b.n)
          {cout<<" number of rows must be the same for solving";exit(1);}
matrix lu,y;
int mm= a.m>b.m?a.m:b.m;
xtt2= new complex [mm];
  lu=a;y=b;
for(i=0;i< n-1;i++)
  {k=maxincol(lu,y,i);

// the number 0.9 below represents a compromise
// between speed and precision. As you decrease
// that number speed increases but precision may
// not be very good. if you increase this number
// towards 1.0 you get a very high precision
// but program will run slightly slower.
    { prxchg(lu,i,k);rxchg(y,i,k);}
    if(abso(lu[i][i])<1e-20)
            {(*erroraction)(); }
    for(j=i+1;j<n;j++)
    {s=lu[j][i]/lu[i][i];prowop(lu,j,i,s);rowop(y,j,i,s);}}
for(i=n-1;i>=0;i--)
  {m=y.m;
     for(k=0;k<m;k++)
     {
       for(s=0,j=i+1,t1=lu[i]+i+1,t2=y[i+1]+k;
      j<n;j++,t2+=m)
s+= *t1++ * *t2;y[i][k]= (y[i][k]-s)/lu[i][i];}}delete [] xtt2;return(y);}
complex det(matrix a)
       {int i,j,k,n,m,ccnt=0;complex s,*t1,*t2;matrix b;b=matrix(a.n,1);int mm= a.m>b.m?a.m:b.m;xtt2= new complex [mm];
          for(i=0;i<a.n;i++)b[i][0]=1;
if((n=(a.n)) != a.m)
          {cout<<"matrix must be square to invert\n";exit(1);}
  if(a.n != b.n)
          {cout<<" number of rows must be the same for solving";exit(1);}
matrix lu,y;
lu=a;y=b;
for(i=0;i< n-1;i++)
  {k=maxincol(lu,y,i);

// the number 0.9 below represents a compromise
```

```
// between speed and precision. As you decrease
// that number speed increases but precision may
// not be very good. if you increase this number
// towards 1.0 you get a very high precision
// but program will run slightly slower.
if(abso(lu[i][i]/lu[k][i])<1.0)  //.9 instead of 1.0
    {prxchg(lu,i,k);rxchg(y,i,k);ccnt++;}
    if(abso(lu[i][i])<1e-50)
            {(*erroraction)();}
    for(j=i+1;j<n;j++)
    {s=lu[j][i]/lu[i][i];prowop(lu,j,i,s);rowop(y,j,i,s);}}
        complex rer;
         rer = ccnt%2==1? -1 : 1;
for(i=0;i<n;i++)rer*= lu[i][i];delete [] xtt2;return(rer);}
matrix eye(int k);
matrix inv(matrix in)
       {return(solve(in,eye(in.n)));}
matrix trps(matrix &in)
       {int n,m;matrix temp(n=in.m,m=in.n);
  complex *t1,*t2;
  int i,j;
for(i=0;i<n;i++)
  for(j=0,t1=temp[i],t2=in.elem+i;j<m;j++,t2+=n)
   *t1++ =conj(*t2);return(temp);}
matrix  inv2(matrix &in)
        {
    int n,m;
  matrix temp(n=in.m,m=in.n);
   double totabs;
  complex *t1,*t2;
  int i,j;
  for(i=0;i<m;i++)
  {for(j=0,t1=in[i],totabs=0;j<n;j++)totabs+=abso(*t1++);
     for(j=0,t1=in[i],t2=temp.elem+i;j<n;j++,t2+=m)
     *t2 =conj(*t1++)/totabs;}
  return(temp);}
matrix operator!(matrix &in)
       {return(trps(in));}
matrix operator/(matrix &in1,matrix &in2)
       {return(in1*inv(in2));}
matrix operator|(matrix &in1,matrix &in2)
       {return(solve(in1,in2));}
matrix operator/(matrix &in1,complex bolen)
       {return(in1*((complex)1.0/bolen));}
matrix operator/(matrix &in1, double bolen)
       {return(in1*((complex)1.0/bolen));}
void print(matrix &m) // prints the matrix
     {
       cout<<"\n jmatrix size is "<< m.n <<" by "<< m.m<<"\n";
int i,j;
       for(i=0;i<m.n;i++)
       {cout<<"\n";for(j=0;j<m.m;j++)cout<<real(m[i][j])<<"+i"<<"\n";}
       printf("\n");}
matrix eye(int k) // matlab's eye ( k by k Identity )
```

```
       {matrix result(k,k);
 int i;
for(i=0;i<k;i++)result[i][i]=1;
 return(result);}
matrix  ones(int k,int l=1) // k by l matrix whose entries are all 1.0
{
  int i,j;
  matrix result(k,l);
for(i=0;i<k;i++)for(j=0;j<l;j++)result[i][j]=1;return(result);}
matrix  zeros(int k, int l)
{return(matrix(k,l));}
matrix random(int n,int m) // n by m random matrix (uniform betw. 0 and 1.xxx
       {
 matrix temp(n,m);
 int i,j;
for(i=0;i<n;i++)for(j=0;j<m;j++)
 temp[i][j]=rand()*1.0/RAND_MAX;
 return(temp);}
matrix leastsq(matrix &a,matrix &b)
       {matrix c;c=a;return((!c*c)|(!c*b));}
void myerrorhandler()
     {cout<<" matrix is close to singular then";}
matrix optimize(matrix &R, matrix &b)
       {matrix xnm,xn,yn,del,mel,hel,zz;
          complex t1,t2,t3,t4;
           hel=(b*(!b));t1=hel[0][0];xn=(!b)/t1;yn=xn;t1=((!xn)*xn)[0][0];zz= R*xn;hel= xn;xn=zz-xn*((!xn)*zz)/t1;xnm=hel;zz=R*xn;yn= yn-xn*((!zz)*y
          t1= ((!xn)*xn)[0][0];
for(int i=0;i<18;i++)
          {hel= xn;xn=zz-xn*((!xn)*zz)/t1-xnm*((!xnm)*zz)/t2;xnm=hel;zz=R*xn;
          yn= yn-xn*((!zz)*yn)/((((!zz)*xn)[0][0]);t2=t1;t1= ((!xn)*xn)[0][0];}
          return(yn);}
          inline complex   absq(matrix &a)
       {return(((!a)*a)[0][0]);}
inline complex  dot(matrix &a,matrix &b)
       {return(((!a)*b)[0][0]);}
matrix optimize2(matrix &R, matrix &b)
       {matrix xn[350],hel,mel,zn;complex al[350],bet[350];int i;
xn[0]= (!b)/(b*!b);zn= R*xn[0];al[0]= absq(xn[0]);bet[0]= dot(xn[0],zn);
xn[1]= zn - bet[0]/al[0]*xn[0];zn= R*xn[1];al[1]=absq(xn[1]);bet[1]=dot(xn[1],zn);
for(i=2;i<350;i++)
          {xn[i]=zn-bet[i-1]/al[i-1]*xn[i-1]-dot(xn[i-2],zn)/al[i-2]*xn[i-2];zn=R*xn[i];al[i]=absq(xn[i]);bet[i]=dot(xn[i],zn);
       if(abso(bet[i])<1e-70)break;}
cout<< "the algorithm stopped at "<<i<<" th step \n";
matrix RP(i+1,i+1);
 matrix bp(1,i+1);
  int m;
          for( m=0;m<i+1;m++)
              RP[m][m]=bet[m];

          for( m=0;m<i;m++)
          {
             RP[m+1][m]=al[m+1];
             RP[m][m+1]=conj(al[m+1]);
          }
```

```
          bp[0][0]=1;

          hel=solve(RP,!bp);
          hel= hel/(bp*hel);

          matrix result(R.n,1);

          for(m=0;m<i+1;m++)result=result+xn[m]*(hel[m][0]);
          return(result);



       }



matrix  findnext(matrix &R, matrix &b, matrix &x)
        {
            matrix temp;

            temp=R*x;
            temp= temp-(!b)/(b*!b)*(b*temp);
            return(temp);
        }


 matrix betterize(matrix &R, matrix &b, matrix &x)
       {
          matrix xn[350],hel,mel,zn;
          complex al[350],bet[350];
  int i;


          xn[0]= x/(b*x);
          zn= findnext(R,b,xn[0]);
  al[0]= absq(xn[0]);
          bet[0]= dot(xn[0],zn);


          xn[1]= zn - bet[0]/al[0]*xn[0];
          zn= findnext(R,b,xn[1]);
          al[1]=absq(xn[1]);
          bet[1]=dot(xn[1],zn);

          for(i=2;i<350;i++)
          {
             xn[i]=zn-bet[i-1]/al[i-1]*xn[i-1]
         -dot(xn[i-2],zn)/al[i-2]*xn[i-2];
        zn=findnext(R,b,xn[i]);
            al[i]=absq(xn[i]);
           bet[i]=dot(xn[i],zn);
            if(abso(bet[i])<1e-20)break;
          }
```

```
        cout<< "the algorithm stopped at "<<i<<" th step \n";

        matrix RP(i+1,i+1);
        matrix bp(1,i+1);
 int m;
        for( m=0;m<i+1;m++)
            RP[m][m]=bet[m];

        for( m=0;m<i;m++)
        {
           RP[m+1][m]=al[m+1];
           RP[m][m+1]=conj(al[m+1]);
        }

        bp[0][0]=1;

        hel=solve(RP,!bp);
        hel= hel/(bp*hel);

        matrix result(R.n,1);

        for(m=0;m<i+1;m++)result=result+xn[m]*(hel[m][0]);
        x=result;



      }


matrix   solve3(matrix &R, matrix &b)
        {
             matrix t,tt,di(b.m,b.m);
             int i;


             for(i=0;i<di.n;i++)
             di[i][i]= b[0][i]*conj(b[0][i]); //*1e-15;

             t< !R;
             tt< solve(R*t+di,!b);
             return(t*tt);
          }



matrix solve4(matrix &R, matrix &b)
       {
          matrix x,y;
          int i;

          x=random(R.n,6);
          for(i=0;i<200;i++)
          {
```

```
            x=R*x/((!x)+x);

          }
          y<matrix(R.n,i);
          for(i=0;i<R.n;i++)y[i][0]=x[i][4];
         // cout<<(!x)*x<<"\n";

          return(y/((!b)*y));
        }


//#define ss1 0.5
//#define ss2 0.5


//  #define nofmodes 2



matrix *d;
matrix *dnew;

matrix   *R,*R0,*R1,*R2,*R3,*R4,*R5,*R6,*R7,*R8,*R9,*R10,*R11,*R12,*R13,*R14,*R15,*R16,*R17,*R18,*R19,*R20,*R21,*R22,*R23;
matrix   *b;
matrix   E0(1,9);
matrix   H0(1,9);
matrix   *gains;
matrix   *tp;
matrix   kvec(9,9);
matrix   *gfin;



 double   xi,yi,zi;
 double  e0,er,mu,sigma,f,w,l,rn,Ry,al,r0,phi0,z0;
 double  M,kmax,kB,T,deltaf;

 double drand()
      {
         return(rand()*1.0/0x7fffffff);
       }


void setparams()
    {

        j=1.0fi;

        //l=1.0 ;
//physical dimensions  of the torso
        //Ry=rmax;
        al=1.376;
 z0=0.0;
```

```
        T=310;                          // degrees kelvin
        kB = 1.38e-23;                  // Boltzmann constant
        deltaf=0.06*4*2;              //eff  pbw  for  16 KHz rec HBW, 2NEX
        er=77.7;                    //tissue  relative epsilon apprx
        mu=4*pi*1e-7;
        e0= 1/36.0/pi*1e-9;
        f=63.9e6;  // *********       beware frequency has increased dramatically
        w=2*pi*f;
        sigma = 0.37;             // apprx avg tissue conductivity
        xi=0.0;yi=0.05;zi=0.0;       // point of interest
       M=3e-9*1.5*0.156*0.156*0.15*f/63.9e6;  // 8cm fov 512x512 1.5mm, 1.5 T
        M=3e-9*1.5*f/63.9e6;
        // propagating mode wavevector

        kmag=sqrt((complex)-1*j*w*mu*(sigma + j*w*er*e0));
cout<<real(kmag)<<"+i"<<imag(kmag)<<"\n";

        // number of modes should go approximately as
        //  m^4 where m is 1/size of smallest structure
        // or in the frequency domain the highest spatial frequency
        // that is considered is m*2pi/l

        kmax=sqrt(sqrt(nofmodes))*2;


        srand(1235);
      }

double tis[5000];

double  findc(int n)
       {
          double beta,sum,tt,c;

          beta=sqrt(2*pi*n);
          sum=0;
          tt=0;
          int i;
          for(i=0;i<20000;i++,tt+= pi/20000.0)
          sum+=sqrt(1+beta*beta*sin(tt)*sin(tt));
          sum*=pi/20000.0;

          c=sum/n;

          tis[0]=0;
          for(i=0;i<n;i++)
           tis[i+1]=tis[i]+c/sqrt(1+beta*beta*sin(tis[i])
                          *sin(tis[i]));
          c=tis[n]/pi;

          for(i=0;i<n+1;i++)tis[i]/=c;

            return(beta);
```

```
        }



matrix map(double pr0,double pphi0,double pz0)
{

int i,kk1,lll;
int bs,bs4,mnbl,col,modbl,bbl,cbl,dbl;
complex ans;
bs=nosam;
bs4=4*bs;
mnbl=4*nofmodes*nosam;
obl=nofmodes*bs;
cbl=2*nofmodes*bs;
dbl=3*nofmodes*bs;
//cout<<"BBB"<<"\n";

matrix *bb;
matrix *bm;
bm=new matrix(1,4*(2*nofmodes-1)*bs);
bb=new matrix(1,4*(2*nofmodes)*bs);
complex q1;
double d1;
        //cout<<"finding the magnetic field and the B matrix \n";
//cout<<"NOSAM="<<nosam<<"\n";
for(i=0;i<nofmodes;i++)
{
  modbl=i*bs;
 for (kk1=0;kk1<nosam;kk1++)
   {

col=kk1;
d1=(*d)[0][kk1];
q1=sqrt(pow(kmag,2)-pow(d1,2));


ans=-1/w/mu*(d1/q1/q1*i*d1/pr0+i/pr0)*bessel(i,q1*pr0)*expp(-i*d1*pz0);




(*bb)[0][modbl+col]=ans*exp(j*i*pphi0);
if (i!=0) (*bb)[0][modbl+col+mnbl]=ans*exp(-i*j*i*pphi0);


ans=1/w/mu*
(  i/pr0*bessel(i,q1*pr0)-q1*bessel(i+1,q1*pr0))*
  ( d1*( -i*j/q1/q1*d1 )-j )*expp(-i*d1*pz0);



(*bb)[0][modbl+col]=
```

```
(
 (*bb)[0][modbl+col]-j*ans*exp(j*i*pphi0))/sqrt(2);

if (i!=0)  (*bb)[0][modbl+col+mnbl]=( (*bb)[0][modbl+col+mnbl]-j*ans*exp(-1*j*i*pphi0))/sqrt(2);

///


ans=(
 -1*d1/q1/q1*(  i/pr0*bessel(i,q1*pr0)-q1*bessel(i+1,q1*pr0 )  )-j*
 ( d1*(-1*j)/q1/q1*i/pr0*bessel(i,q1*pr0)
 )
)*expp(-1*d1*pz0)/sqrt(2);




(*bb)[0][modbl+(nofmodes*bs)+col]=ans*exp(j*i*pphi0);

if(i!=0)  (*bb)[0][modbl+4*nofmodes*bs+col+(nofmodes)*bs]=ans*exp(-1*j*i*pphi0);

///


ans=-1/w/mu*(d1/q1/q1*i*d1/pr0+i/pr0)*besselY(i,q1*pr0)*expp(-1*d1*pz0);



(*bb)[0][modbl+col+cbl]=ans*exp(j*i*pphi0);
if (i!=0) (*bb)[0][modbl+col+mnbl+cbl]=ans*exp(-1*j*i*pphi0);


ans=1/w/mu*
(  i/pr0*besselY(i,q1*pr0)-q1*besselY(i+1,q1*pr0))*
  ( d1*( -1*j/q1/q1*d1 )-j )*expp(-1*d1*pz0);



(*bb)[0][modbl+col+cbl]=
(
 (*bb)[0][modbl+col+cbl]-1*j*ans*exp(j*i*pphi0))/sqrt(2);

if (i!=0)  (*bb)[0][modbl+col+mnbl+cbl]=( (*bb)[0][modbl+col+mnbl+cbl]-1*j*ans*exp(-1*j*i*pphi0))/sqrt(2);

///


ans=(
 -1*d1/q1/q1*(  i/pr0*besselY(i,q1*pr0)-q1*besselY(i+1,q1*pr0 )  )-1*j*
 ( d1*(-1*j)/q1/q1*i/pr0*besselY(i,q1*pr0)
 )
)*expp(-1*d1*pz0)/sqrt(2);
```

```
(*bb)[0][modbl+dbl+col]=ans*exp(j*i*pphi0);

if(i!=0)  (*bb)[0][modbl+4*nofmodes*bs+col+3*(nofmodes)*bs]=ans*exp(-1*j*i*pphi0);




}
//  cout<<i);
}

        //  cout<<"B matrix computed \n";
// cout<<(*b));
int aq,ab,ac;
for (aq=0;aq<nofmodes;aq++)
 { for (ab=0;ab<bs;ab++)
  { for (ac=0;ac<4;ac++)
   (*bm)[0][aq*4*bs+ac*bs+ab]=(*bb)[0][ac*nofmodes*bs+aq*bs+ab];
   }
  }


 for (aq=0;aq<nofmodes-1;aq++)
 { for (ab=0;ab<bs;ab++)
  { for (ac=0;ac<4;ac++)

(*bm)[0][aq*4*bs+ac*bs+ab+mnbl]=(*bb)[0][ac*nofmodes*bs+(aq+1)*bs+ab+mnbl];


}
  }



return((*bm));

}


void findh()
      {
       int i,kk1,ll1;
int bs,bs4,mnbl,col,modbl,bbl,cbl,dbl;
complex ans;
bs=nosam;
bs4=4*bs;
mnbl=4*nofmodes*nosam;
bbl=nofmodes*bs;
cbl=2*nofmodes*bs;
dbl=3*nofmodes*bs;
//cout<<"BBB"<<"\n";

matrix *bb;
```

```
bb=new matrix(1,4*(2*nofmodes)*bs);
complex q1;
double d1;
        //cout<<"finding the magnetic field and the B matrix \n";

for(i=0;i<nofmodes;i++)
{
  modbl=i*bs;
 for (kk1=0;kk1<nosam;kk1++)
   {

col=kk1;
d1=(*d)[0][kk1];
q1=sqrt(pow(kmag,2)-pow(d1,2));



ans=-1/w/mu*(d1/q1/q1*i*d1/r0+i/r0)*bessel(i,q1*r0)*expp(-1*d1*z0);







(*bb)[0][modbl+col]=ans*exp(j*i*phi0);
if (i!=0) (*bb)[0][modbl+col+mnbl]=ans*exp(-1*j*i*phi0);


ans=1/w/mu*
(  i/r0*bessel(i,q1*r0)-q1*bessel(i+1,q1*r0))*
  ( d1*( -1*j/q1/q1*d1 )-j )*expp(-1*d1*z0);



(*bb)[0][modbl+col]=
(
 (*bb)[0][modbl+col]-j*ans*exp(j*i*phi0))/sqrt(2);

if (i!=0)  (*bb)[0][modbl+col+mnbl]=( (*bb)[0][modbl+col+mnbl]-j*ans*exp(-1*j*i*phi0))/sqrt(2);

///


ans=(
 -1*d1/q1/q1*(  i/r0*bessel(i,q1*r0)-q1*bessel(i+1,q1*r0 )  )-j*
 ( d1*(-1*j)/q1/q1*i/r0*bessel(i,q1*r0)
 )
)*expp(-1*d1*z0)/sqrt(2);



(*bb)[0][modbl+(nofmodes*bs)+col]=ans*exp(j*i*phi0);

if(i!=0)  (*bb)[0][modbl+4*nofmodes*bs+col+(nofmodes)*bs]=ans*exp(-1*j*i*phi0);
```

```
///


ans=-1/w/mu*(d1/q1/q1*i*d1/r0+i/r0)*besselY(i,q1*r0)*expp(-1*d1*z0);



(*bb)[0][modbl+col+cbl]=ans*exp(j*i*phi0);
if (i!=0) (*bb)[0][modbl+col+mnbl+cbl]=ans*exp(-1*j*i*phi0);




ans=1/w/mu*
(  i/r0*besselY(i,q1*r0)-q1*besselY(i+1,q1*r0))*
  ( d1*( -1*j/q1/q1*d1 )-j )*expp(-1*d1*z0);



(*bb)[0][modbl+col+cbl]=
(
 (*bb)[0][modbl+col+cbl]-1*j*ans*exp(j*i*phi0))/sqrt(2);

if (i!=0)  (*bb)[0][modbl+col+mnbl+cbl]=( (*bb)[0][modbl+col+mnbl+cbl]-1*j*ans*exp(-1*j*i*phi0))/sqrt(2);

///


ans=(
 -1*d1/q1/q1*(  i/r0*besselY(i,q1*r0)-q1*besselY(i+1,q1*r0 )  )-i*j*
 ( d1*(-1*j)/q1/q1*i/r0*besselY(i,q1*r0)
 )
)*expp(-1*d1*z0)/sqrt(2);




(*bb)[0][modbl+dbl+col]=ans*exp(j*i*phi0);

if(i!=0)  (*bb)[0][modbl+4*nofmodes*bs+col+3*(nofmodes)*bs]=ans*exp(-1*j*i*phi0);




}
//  cout<<i);
}

       //  cout<<"B matrix computed \n";
// cout<<(*b));
int aq,ab,ac;
for (aq=0;aq<nofmodes;aq++)
 { for (ab=0;ab<bs;ab++)
```

```
  { for (ac=0;ac<4;ac++)
   (*b)[0][aq=4*bs+ac*bs+ab]=(*bb)[0][ac*nofmodes*bs+aq*bs+ab];
   }
  }


 for (aq=0;aq<nofmodes-1;aq++)
 { for (ab=0;ab<bs;ab++)
  { for (ac=0;ac<4;ac++)

(*b)[0][aq*4*bs+ac*bs+ab+mnbl]=(*bb)[0][ac*nofmodes*bs+(aq+1)*bs+ab+mnbl];


}
  }
/*
FILE *fp;
fp=fopen("b.m","w+b");
int sx,vb;
fprintf(fp,"b=[");
for(sx=0;sx<4*(2*nofmodes-1)*nosam;sx++)
{if(imag((*b)[0][sx])>=0)
fprintf(fp,"%le+%lei \n",real((*b)[0][sx]),imag((*b)[0][sx]));
else
fprintf(fp,"%le%lei \n",real((*b)[0][sx]),imag((*b)[0][sx]));
        }
fprintf(fp,"];");
fclose(fp);
*/
}
 complex aconjb(int kk,int ll)
        {
           complex aaa;

           aaa= conj(E0[kk][0])*E0[ll][0]
              + conj(E0[kk][1])*E0[ll][1]
              + conj(E0[kk][2])*E0[ll][2];
           return(aaa);
        }


complex ssin(complex aa)
       {
           return((exp(j*aa)-exp(-1*j*aa))/2/j);
        }


complex  jj1(complex &aa)
        {
            int i,fac;
            complex p,res,sq;

            sq=aa*aa/4;
```

```
            fac=1;
            p=1;
            res=0;


            for(i=0;i<45;i++)
            {
               res += p;
               p= -1*p*sq/(i+2)/(i+1);
             }
             res*=aa/2;
             return(res);
          }




 void findR()
      {
int kk1,ll1,kk2,ll2;
int bs,bs4,row,col,bb1,cb1,db1;
complex inz,inz1,inz2,q1,q2,cor;
        double d1,d2;
bs=nosam;
bs4=4*bs;
bb1=bs;
cb1=2*bs;
db1=3*bs;

       //  cout<<"computing the noise correlation matrix R\n";



   for(kk1=0;kk1<nosam;kk1++)
     {
  for(kk2=0;kk2<nosam;kk2++)
{



 d1=(*d)[0][kk1];
 q1=sqrt(pow(kmag,2)-pow(d1,2));
 d2=(*d)[0][kk2];
 q2=sqrt(pow(kmag,2)-pow(d2,2));
row=kk1;
col=kk2;

 ////////////////////////////////////////////////////////////////////////////////
//     aa ----- bb
//cout<<"aa");

complex aa,bb,cc,dd,ee;
complex ff,gg,hh,ii,jj;
```

```
aa=integ(q1,q2,1,pp,pp, rmin, rmax); //1
dd=integ(q1,q2,1,pp+1,pp+1, rmin, rmax); //1
bb=integ(q1,q2,-1,pp,pp, rmin, rmax);      // -1
cc=integ(q1,q2,0,(pp),(pp)+1, rmin, rmax);
ee=integ(q1,q2,0,(pp)+1,(pp), rmin, rmax);



if (d1==d2) inz=1.0*1;
else
inz=(expp((d1-d2)*0.5*1)-expp(-1.0*(d1-d2)*0.5*1))/(j*(d1-d2));
if((row<=col) && (col>=(nosam-jum)))
{

(*R)[row][col]=2*pi*sigma*inz*(aa+(d1)*d2/pow(conj(q1),2)/pow(q2,2)*(2*pow(pp,2)*bb+conj(q1)*q2*dd-pp*q2*cc-pp*conj(q1)*ee));


(*R)[bbl+row][bbl+col]=2*pi*sigma*inz/pow(conj(q1),2)/q2/q2*pow(w*mu,2)*(2*pow(pp,2)*bb+conj(q1)*q2*dd-pp*q2*cc-pp*conj(q1)*ee);

(*R)[col][row]=conj((*R)[row][col]);

(*R)[bbl+col][bbl+row]=conj((*R)[bbl+row][bbl+col]);

////////////////////////////////////////////////////////////////////////////
//  cc----dd
//cout<<"cc");

ff=integYY(q1,q2,1,pp,pp, rmin, rmax); // 1
gg=integYY(q1,q2,-1,pp,pp, rmin, rmax); // -1
hh=integYY(q1,q2,1,pp+1,pp+1, rmin, rmax); //1
ii=integYY(q1,q2,0,pp,pp+1, rmin, rmax);
jj=integYY(q1,q2,0,pp+1,pp, rmin, rmax);

(*R)[cbl+row][cbl+col]=2*pi*sigma*inz*(ff+(d1)*d2/pow(conj(q1),2)/pow(q2,2)*(2*pow(pp,2)*gg+conj(q1)*q2*hh-pp*q2*ii-pp*conj(q1)*jj));

(*R)[dbl+row][dbl+col]=2*pi*sigma*inz*pow(w*mu/conj(q1)/q2,2)*(2*pp*pp*gg+conj(q1)*q2*hh-pp*q2*ii-pp*conj(q1)*jj);

(*R)[cbl+col][cbl+row]=conj((*R)[cbl+row][cbl+col]);

(*R)[dbl+col][dbl+row]=conj((*R)[dbl+row][dbl+col]);


 ////////////////////////////////////////////////////////////////////////////
// ab
//cout<<"end");

(*R)[row][bbl+col]=2*pi*sigma*inz*(d1)*pp*w*mu/pow(conj(q1),2)/pow(q2,2)*( pp*bb-conj(q1)*ee+pp*bb-q2*cc);


(*R)[col][row+bbl]=conj((*R)[row][bbl+col]);
```

```
(*R)[(bbl)+col][row]=conj((*R)[row][bbl+col]);

(*R)[row+bbl][col]=((*R)[row][col+bbl]);
}


////////////////////////////////////////////////////////////////////////////
if((col>=(nosam-jum)) || (row>=(nosam-jum)))
{
// bd
complex bd1,bd2,bd3,bd4;
bd1=integJY(q1,q2,-1,pp,pp, rmin, rmax); // -1
bd2=integJY(q1,q2,1,pp+1,pp+1, rmin, rmax); // 1
bd3=integJY(q1,q2,0,pp,pp+1, rmin, rmax);
bd4=integJY(q1,q2,0,pp+1,pp, rmin, rmax);

(*R)[(bbl)+row][dbl+col]=2*pi*sigma*inz*pow(w*mu/conj(q1)/q2,2)*(  2*pow(pp,2)*bd1+conj(q1)*q2*bd2-pp*q2*bd3-pp*conj(q1)*bd4  );

(*R)[dbl+col][bbl+row]=conj((*R)[(bbl)+row][dbl+col]);


////////////////////////////////////////////////////////////////////////////
//ac
complex ac1;
ac1=integJY(q1,q2,1,pp,pp, rmin, rmax); // 1

(*R)[row][cbl+col]=2*pi*sigma*inz*(ac1+(d1)*d2/pow(conj(q1),2)/q2/q2*( 2*pow(pp,2)*bd1+conj(q1)*q2*bd2-pp*q2*bd3-pp*conj(q1)*bd4 ));

(*R)[cbl+col][row]=conj((*R)[row][cbl+col]);

////////////////////////////////////////////////////////////////////////////

//ad

(*R)[row][dbl+col]=2*pi*sigma*inz*(d1)*pp*w*mu/pow(conj(q1),2)/pow(q2,2)*(  2*pp*bd1-conj(q1)*bd4-q2*bd3  );

(*R)[dbl+col][row]=conj((*R)[row][dbl+col]);

////////////////////////////////////////////////////////////////////////////

//bc

(*R)[(bbl)+row][cbl+col]=2*pi*sigma*inz*d2*pp*w*mu/pow(conj(q1),2)/pow(q2,2)*(pp*bd1-q2*bd3+pp*bd1-conj(q1)*bd4);

(*R)[cbl+col][bbl+row]=conj((*R)[(bbl)+row][cbl+col]);
}
////////////////////////////////////////////////////////////////////////////

//cd
if((row<=col) && (col>=(nosam-jum)))
{

(*R)[cbl+row][dbl+col]=2*pi*sigma*inz*(d1)*pp*w*mu/pow(conj(q1),2)/pow(q2,2)*(pp*gg-conj(q1)*jj+pp*gg-q2*ii);
```

```
 (*R)[cbl+col][dbl+row]=conj((*R)[cbl+row][dbl+col]);


 (*R)[dbl+col][cbl+row]=conj((*R)[cbl+row][dbl+col]);

 (*R)[dbl+row][cbl+col]=((*R)[cbl+row][dbl+col]);
 ////////////////////////////////////////////////////////////////////////////////


 }
 }
 }


 /*
 FILE *fp;
 if (pp==0) fp=fopen("r0.m","w+b");
 if (pp==1) fp=fopen("r1.m","w+b");
 if (pp==2) fp=fopen("r2.m","w+b");
 if (pp==3) fp=fopen("r3.m","w+b");
 if (pp==4) fp=fopen("r4.m","w+b");
 if (pp==5) fp=fopen("r5.m","w+b");
 if (pp==6) fp=fopen("r6.m","w+b");
 if (pp==7) fp=fopen("r7.m","w+b");
 if (pp==8) fp=fopen("r8.m","w+b");
 if (pp==9) fp=fopen("r9.m","w+b");
 if (pp==10) fp=fopen("r10.m","w+b");
 if (pp==11) fp=fopen("r11.m","w+b");
 int sx,vb;
 fprintf(fp,"r%i=[",pp);
 for(sx=0;sx<4*nosam;sx++)
 {
 for(vb=0;vb<4*nosam;vb++)
 {
 if(imag((*R)[sx][vb])>=0)
 fprintf(fp,"%le+%lei \n",real((*R)[sx][vb]),imag((*R)[sx][vb]));
 else
 fprintf(fp,"%le%lei \n",real((*R)[sx][vb]),imag((*R)[sx][vb]));
 }
 }
 fprintf(fp,"];");
 fclose(fp);
 */
       //  cout<<"\n Noise correlation matrix computation complete \n";
 //cout<<<(*R)[0][0];
 complex qaz;
 //cout<<"kill");
 qaz=(*R)[0][0];

        }
```

```
FILE *fk;

void  findg()
      {
           matrix t1,t11,t2,t3,t4;
     /* int xy,mn;
for (xy=0;xy<4*nosam;xy++)
 { for (mn=0;mn<4*(nofmodes-1)*bs;mn++) (*R)[xy][mn+4*(nofmodes)*bs]=(*R)[xy][mn+4*bs];
 }
*/      complex crn;

       // cout<<"Optimizing the gains \n";
   int jkl,mk,nj;
  // matrix *RR;
  // RR=new matrix(4*nosam,4*nosam);
   matrix *bb;
   bb=new matrix(1,4*nosam);



  for (mk=0;mk<4*nosam;mk++) {(*bb)[0][mk]=(*b)[0][pp*4*nosam+mk];}

          t11= solve((*R),!(*bb));


   matrix nba;
   nba=(*bb)*t11;
  // complex abn;
  abn=nba[0][0];
   cout<<"cond"<<pp<<"="<<real(abn)<<","<<imag(abn)<<"\n";
//cout<<t11);
   for (mk=0;mk<4*nosam;mk++)
   {
   (*tp)[pp*4*nosam+mk][0]=t11[mk][0];
   if(pp!=0)
   (*tp)[(pp-1)*4*nosam+mk+nofmodes*4*nosam][0]=t11[mk][0]*exp(2.0*j*pp*phi0);
   }
if(pp==nofmodes-1)
      {
           t3=(*b)*(*tp);
           (*gains)=(*tp)/(t3[0][0]); // ie 1 matrix inversi daha once almamistim
           t2=(*b)*(*gains);
    complex check;
   check=t2[0][0];
                cout<<real(check)<<","<<imag(check)<<"\n";
          crn=1.0/(t3[0][0]);
abn=t3[0][0];
         cout<<" in the optimum case, noise resistance is "<<real(crn)<<"+i"<<imag(crn)<<" ohms \n";
          ssnr= w*M*mu/sqrt(kB*4*T*real(crn));
              cout<<nosam<<"snr="<<ssnr<<"\n";
                }
 }
void saveras(char *fna,int he,int wi,unsigned char *imm)
{
```

```
int ii,jj,kk,ll;
unsigned char *im;

FILE *ff;

if((ff=fopen(fna,"w+b"))==NULL)
{
cout<<" i can not open file for 8 bit save \n";
}
else
{

ii=0x59a66a95;
fwrite(&ii,sizeof(int),1,ff);
ii=wi;
fwrite(&ii,sizeof(int),1,ff);
ii=he;
fwrite(&ii,sizeof(int),1,ff);
ii=8;
fwrite(&ii,sizeof(int),1,ff);
ii=he*wi;
fwrite(&ii,sizeof(int),1,ff);
ii=RT_OLD;
fwrite(&ii,sizeof(int),1,ff);
ii=RMT_NONE;
fwrite(&ii,sizeof(int),1,ff);
ii=0;
fwrite(&ii,sizeof(int),1,ff);
fwrite(immm,sizeof(char),wi*he,ff);
fclose(ff);
}
}

unsigned char *im1;
 double mul;

void  findimage(int he,int wi)
      {
matrix v,vu;
          int ii,jj,kk,ll,mm,kk1,ll1,cc;
          unsigned char *t1,*t2;
          complex *tc1,*tc2,temp,q1;
           double xxo,yyo,zzo,rr0,phiphi0,was,saw,mx,my,test,sss,wiwi,hehe,jo,io,d1;
   double ax[he*wi];
wiwi=wi;
hehe=he;
          was=0.0;
  saw=1e100;
   test=1e100;
          im1=(unsigned char*)calloc(he*wi,1);
          for(ii=0,t1=im1;ii<he;ii++)
         {
           for(jj=0;jj<wi;jj++,t1++)
```

```
          { xxo=(jj-(wi>>1))*2.0*Ry/wi;
yyo=((he>>1)-ii)*2.0*Ry/he;
rr0=sqrt(xxo*xxo+yyo*yyo);
if(xxo!=0)
{
if(xxo>.0) {phiphi0=atan(yyo/xxo);}
else {if (yyo<0.0)phiphi0=atan(yyo/xxo)-pi;else phiphi0=atan(yyo/xxo)+pi;}
}
else {if(yyo>.0) {phiphi0=pi/2;} else {phiphi0=-pi/2;}}

              if((xxo*xxo + yyo*yyo)<(Ry*Ry))
              {
if (rr0<rmin) {  *t1=0; ax[ii*wi+jj]=0.;}
 else
{ vu=map(rr0,phiphi0,z0);
                  v=vu*(*gains) ;
  temp=v[0][0];
if (abso(temp)>was) {was=abso(temp);mx=xxo;my=yyo;}
if (abso(temp)<saw) {saw=abso(temp);}
*t1=(int)(abso(temp));
ax[ii*wi+jj]=abso(temp);
}
}
else {*t1=0; ax[ii*wi+jj]=0.;} }
}
mul=250./was;
cout<<"mul="<<mul<<"\n";
for(ii=0,t1=im1;ii<he;ii++)
for(jj=0;jj<wi;jj++,t1++)
{
if ((ax[ii*wi+jj]>=0.0) && (ax[ii*wi+jj]<test)) test=ax[ii*wi+jj];
if (ax[ii*wi+jj]>=0.0) *t1=(int)(ax[ii*wi+jj]*mul);
}
            FILE *ffx;
if(r0==.0012) ffx=fopen("aaa0012.bbb","w"); else
if(r0==.0029) ffx=fopen("aaa0029.bbb","w"); else
if(r0==.0048) ffx=fopen("aaa0048.bbb","w"); else
if(r0==.007) ffx=fopen("aaa0070.bbb","w"); else
if(r0==.0096) ffx=fopen("aaa0096.bbb","w"); else
if(r0==.0123) ffx=fopen("aaa0123.bbb","w"); else
if(r0==.0155) ffx=fopen("aaa0155.bbb","w"); else
if(r0==.0191) ffx=fopen("aaa0191.bbb","w"); else
if(r0==.0233) ffx=fopen("aaa0233.bbb","w"); else
if(r0==.028) ffx=fopen("aaa0280.bbb","w"); else
if(r0==.0333) ffx=fopen("aaa0333.bbb","w"); else
if(r0==.0393) ffx=fopen("aaa0393.bbb","w"); else
if(r0==.0461) ffx=fopen("aaa0461.bbb","w"); else
if(r0==.0539) ffx=fopen("aaa0539.bbb","w"); else
if(r0==.0627) ffx=fopen("aaa0627.bbb","w"); else
if(r0==.0727) ffx=fopen("aaa0727.bbb","w"); else
if(r0==.0840) ffx=fopen("aaa0840.bbb","w"); else
if(r0==.0969) ffx=fopen("aaa0969.bbb","w"); else
if(r0==.1115) ffx=fopen("aaa1115.bbb","w"); else
if(r0==.1281) ffx=fopen("aaa1281.bbb","w"); else
```

```
if(r0==.1469) ffx=fopen("aaa1469.bbb","w"); else
if(r0==.1682) ffx=fopen("aaa1682.bbb","w"); else
if(r0==.1924) ffx=fopen("aaa1924.bbb","w"); else
if(r0==.2198) ffx=fopen("aaa2198.bbb","w"); else
if(r0==.25) ffx=fopen("aaa2500.bbb","w");
else
            ffx=fopen("aaa.bbb","w");
for(t1=im1,ii=0;ii<he;ii++)
            {
                 for(jj=0;jj<wi;jj++,t1++)
{
fprintf(ffx,"%ld ", *t1);
}
                 fprintf(ffx,"\n");
             }
             fclose(ffx);
if(r0==0.0012)  saveras("dene0012.ras",he,wi,im1);  else
if(r0==0.0029)  saveras("dene0029.ras",he,wi,im1); else
if(r0==0.0048)  saveras("dene0048.ras",he,wi,im1);  else
if(r0==0.007)  saveras("dene0070.ras",he,wi,im1); else
if(r0==0.0095)  saveras("dene0095.ras",he,wi,im1);  else
if(r0==0.0123)  saveras("dene0123.ras",he,wi,im1); else
if(r0==0.0155)  saveras("dene0155.ras",he,wi,im1);   else
if(r0==0.0191)  saveras("dene0191.ras",he,wi,im1); else
if(r0==0.0233)  saveras("dene0233.ras",he,wi,im1);  else
if(r0==0.0280)  saveras("dene0280.ras",he,wi,im1); else
if(r0==0.0333)  saveras("dene0333.ras",he,wi,im1);  else
if(r0==0.0393)  saveras("dene0393.ras",he,wi,im1); else
if(r0==0.0461)  saveras("dene0461.ras",he,wi,im1);  else
if(r0==0.0539)  saveras("dene0539.ras",he,wi,im1); else
if(r0==0.0627)  saveras("dene0627.ras",he,wi,im1);  else
if(r0==0.0727)  saveras("dene0727.ras",he,wi,im1); else
if(r0==0.0840)  saveras("dene0840.ras",he,wi,im1);  else
if(r0==0.0969)  saveras("dene0969.ras",he,wi,im1); else
if(r0==0.1115)  saveras("dene1115.ras",he,wi,im1);  else
if(r0==0.1281)  saveras("dene1281.ras",he,wi,im1); else
if(r0==0.1469)  saveras("dene1469.ras",he,wi,im1);  else
if(r0==0.1682)  saveras("dene1682.ras",he,wi,im1); else
if(r0==0.1924)  saveras("dene1924.ras",he,wi,im1);  else
if(r0==0.2198)  saveras("dene2198.ras",he,wi,im1); else
if(r0==0.25)  saveras("dene2500.ras",he,wi,im1);
else
saveras("dene.ras",he,wi,im1);

cout<<"was="<<was<<"saw="<<saw<<"test="<<test<<"\n";

        }

void findimagexz(int he,int wi)
{
matrix v,vu;
          int ii,jj,kk,ll,mm,kk1,ll1,cc;
          unsigned char *t1,*t2;
          complex *tc1,*tc2,temp,q1;
```

```
            double xxo,yyo,zzo,rr0,phiphi0,was,saw,mx,my,test,sss,wiwi,hehe,jo,io,d1;
   double ax[he*wi];
wiwi=wi;
hehe=he;
          was=0.0;
  saw=1e100;
   test=1e100;
          im1=(unsigned char*)calloc(he*wi,1);
//cout<<he<<"\n";
          for(ii=0,t1=im1;ii<he;ii++)
         { //cout<<ii<<"-+-+";
//io=ii;
           for(jj=0;jj<wi;jj++,t1++)
           {
//jo=jj;
 zzo=(jj-(wi>>1))*1/wi;
rr0=((he>>1)-ii)*2.0*Ry/he;
if (abs(rr0)<rmin) {  *t1=0; ax[ii*wi+jj]=0.;}
 else
{
if(rr0>=0.) vu=map(rr0,phi0,zzo);
else vu=map(-rr0,phi0+pi,zzo);
                  v=vu*(*gains) ;temp=v[0][0];
if (abso(temp)>was) {was=abso(temp);mx=xxo;my=yyo;}
if (abso(temp)<saw) {saw=abso(temp);}
*t1=(int)(abso(temp));
ax[ii*wi+jj]=abso(temp);
}}}
mul=250./was;
cout<<"mul="<<mul<<"\n";

 for(ii=0,t1=im1;ii<he;ii++)
for(jj=0;jj<wi;jj++,t1++)
{
if ((ax[ii*wi+jj]>=0.0) && (ax[ii*wi+jj]<test)) test=ax[ii*wi+jj];
if (ax[ii*wi+jj]>=0.0) *t1=(int)(ax[ii*wi+jj]*mul);
}

            FILE *ffx;
if(r0==.0012) ffx=fopen("aaaxz0012.bbb","w"); else
if(r0==.0029) ffx=fopen("aaaxz0029.bbb","w"); else
if(r0==.0048) ffx=fopen("aaaxz0048.bbb","w"); else
if(r0==.007) ffx=fopen("aaaxz0070.bbb","w"); else
if(r0==.0095) ffx=fopen("aaaxz0095.bbb","w"); else
if(r0==.0123) ffx=fopen("aaaxz0123.bbb","w"); else
if(r0==.0155) ffx=fopen("aaaxz0155.bbb","w"); else
if(r0==.0191) ffx=fopen("aaaxz0191.bbb","w"); else
if(r0==.0233) ffx=fopen("aaaxz0233.bbb","w"); else
if(r0==.028) ffx=fopen("aaaxz0280.bbb","w"); else
if(r0==.0333) ffx=fopen("aaaxz0333.bbb","w"); else
if(r0==.0393) ffx=fopen("aaaxz0393.bbb","w"); else
if(r0==.0461) ffx=fopen("aaaxz0461.bbb","w"); else
if(r0==.0539) ffx=fopen("aaaxz0539.bbb","w"); else
if(r0==.0627) ffx=fopen("aaaxz0627.bbb","w"); else
```

```
 if(r0==.0727) ffx=fopen("aaaxz0727.bbb","w"); else
 if(r0==.0840) ffx=fopen("aaaxz0840.bbb","w"); else
 if(r0==.0969) ffx=fopen("aaaxz0969.bbb","w"); else
 if(r0==.1115) ffx=fopen("aaaxz1115.bbb","w"); else
 if(r0==.1281) ffx=fopen("aaaxz1281.bbb","w"); else
 if(r0==.1469) ffx=fopen("aaaxz1469.bbb","w"); else
 if(r0==.1682) ffx=fopen("aaaxz1682.bbb","w"); else
 if(r0==.1924) ffx=fopen("aaaxz1924.bbb","w"); else
 if(r0==.2198) ffx=fopen("aaaxz2198.bbb","w"); else
 if(r0==.25) ffx=fopen("aaaxz2500.bbb","w");

 else
            ffx=fopen("aaaxz.bbb","w");
 for(t1=im1,ii=0;ii<he;ii++)
            {
                 for(jj=0;jj<wi;jj++,t1++)
 {fprintf(ffx,"%ld ", *t1);}
 fprintf(ffx,"\n");
             }
             fclose(ffx);
 if(r0==0.0012)  saveras("denexz0012.ras",he,wi,im1);  else
 if(r0==0.0029)  saveras("denexz0029.ras",he,wi,im1); else
 if(r0==0.0048)  saveras("denexz0048.ras",he,wi,im1);  else
 if(r0==0.007)  saveras("denexz0070.ras",he,wi,im1); else
 if(r0==0.0095)  saveras("denexz0095.ras",he,wi,im1); else
 if(r0==0.0123)  saveras("denexz0123.ras",he,wi,im1); else
 if(r0==0.0155)  saveras("denexz0155.ras",he,wi,im1);  else
 if(r0==0.0191)  saveras("denexz0191.ras",he,wi,im1); else
 if(r0==0.0233)  saveras("denexz0233.ras",he,wi,im1);  else
 if(r0==0.0280)  saveras("denexz0280.ras",he,wi,im1); else
 if(r0==0.0333)  saveras("denexz0333.ras",he,wi,im1);  else
 if(r0==0.0393)  saveras("denexz0393.ras",he,wi,im1); else
 if(r0==0.0461)  saveras("denexz0461.ras",he,wi,im1);  else
 if(r0==0.0539)  saveras("denexz0539.ras",he,wi,im1); else
 if(r0==0.0627)  saveras("denexz0627.ras",he,wi,im1);  else
 if(r0==0.0727)  saveras("denexz0727.ras",he,wi,im1); else
 if(r0==0.0840)  saveras("denexz0840.ras",he,wi,im1);  else
 if(r0==0.0969)  saveras("denexz0969.ras",he,wi,im1); else
 if(r0==0.1115)  saveras("denexz1115.ras",he,wi,im1);  else
 if(r0==0.1281)  saveras("denexz1281.ras",he,wi,im1); else
 if(r0==0.1469)  saveras("denexz1469.ras",he,wi,im1);  else
 if(r0==0.1682)  saveras("denexz1682.ras",he,wi,im1); else
 if(r0==0.1924)  saveras("denexz1924.ras",he,wi,im1);  else
 if(r0==0.2198)  saveras("denexz2198.ras",he,wi,im1); else
 if(r0==0.25)  saveras("denexz2500.ras",he,wi,im1);
 else
 saveras("denexz.ras",he,wi,im1);
 cout<<"was="<<was<<"saw="<<saw<<"test="<<test<<"\n";}
 void trans(matrix a, matrix b)
 {
 int c1,e1,c2,e2,ro1,co1,ro2,co2;
 for(c1=0;c1<4;c1++)
 {
 for(c2=0;c2<4;c2++)
```

```
{
for(e1=0;e1<nosam-jum;e1++)
{
for(e2=0;e2<nosam-jum;e2++)
{ro1=c1*b.n/4+e1;co1=c2*b.m/4+e2;ro2=c1*a.n/4+e1;co2=c2*a.m/4+e2;b[ro1][co1]=a[ro2][co2];
}}}}
}



main()
{
int ssaacc;
setparams();
cout<<"ss1= \n";
scanf("%lf",&ss1);
cout<<"ss2= \n";
scanf("%lf",&ss2);
double satt;
//cout<<"acc\n";
//scanf("%li",&acc);
int catt,latt;
cout<<"enter inner radius\n";
scanf("%lf",&rmin);
cout<<"enter outer radius \n";
scanf("%lf",&rmax);
cout<<"enter number of Bessel modes m \n";
scanf("%li",&nofmodes);

cout<<"enter dinit \n";
scanf("%lf",&red);
scanf("%lf",&imd);
dinit=red+j*imd;
cout<<real(dinit)<<imag(dinit)<<"\n";

cout<<"enter point of interest radial distance \n";
scanf("%lf",&satt);
r0=satt;
cout<<"enter point of interest angle (*pi) \n";
scanf("%lf",&satt);
phi0=satt*pi;
double tol;
//cout<<"enter tol \n";
//scanf("%lf",&tol);
cout<<"enter jum \n";
scanf("%li",&jum);

//cout<<"enter sac \n";
//scanf("%li",&ssaacc);

cout<<"enter L \n";
scanf("%lf",&l);

Ry=rmax;
```

```
//jum=3;
/////

int indco=0;
nosam=0;
d=new matrix(1,1);
(*d)[0][0]=0;
int qa=0,saco=0,linesaco=0,lineqa=0;
double olsnr=.0;
int pnco=0;
int linepnco=0;
while(lineqa==0)
{//1
linesaco=lineqa=0;
while(lineqa==0)
{//2
pnco=0;
indco=nosam;
while(pnco<1)
{//3
saco=qa=0;
while(saco<1)
{//4
if(sqrt(pow(jum*qa*ss1,2)+pow(lineqa*ss2,2))<50) acc=31;
else if(sqrt(pow(jum*qa*ss1,2)+pow(lineqa*ss2,2))<100) acc=39;
else if(sqrt(pow(jum*qa*ss1,2)+pow(lineqa*ss2,2))<150) acc=49;
else if(sqrt(pow(jum*qa*ss1,2)+pow(lineqa*ss2,2))<200) acc=65;
else if(sqrt(pow(jum*qa*ss1,2)+pow(lineqa*ss2,2))<250) acc=69;
else acc=77;
cout<<"linepnco="<<linepnco<<"linesaco="<<linesaco<<"pnco="<<pnco<<"saco="<<saco<<"lineqa="<<lineqa<<"\n";
int cu,ct,bt,gr;
for(gr=0;gr<1;gr++)
{//10
nosam+=jum;
dnew=new matrix(1,nosam);
int jh,lh;
for(jh=0;jh<nosam-jum;jh++) (*dnew)[0][jh]=(*d)[0][jh];
for(jh=0;jh<jum;jh++) (*dnew)[0][nosam+jh-jum]=pow(-1,pnco)*(jh+jum*qa+pnco)*ssi+dinit;

cout<<"new d's:";
for(jh=0;jh<jum;jh++) cout<<"("<<real((*dnew)[0][nosam-jum+jh]);
cout<<"\n";

cout<<"new q's:";
complex qq;
for(jh=0;jh<jum;jh++)
{
qq=sqrt(pow(kmag,2)-pow((*dnew)[0][nosam-jum+jh],2));
cout<<"("<<real(qq)<<","<<imag(qq)<<")";
}
cout<<"\n";
```

```
delete d;
d=new matrix(1,nosam);
int ff;
for(ff=0;ff<nosam;ff++) (*d)[0][ff]=(*dnew)[0][ff];
int ht;
delete dnew;
 delete b;
delete gains;
delete tp;
b=new matrix(1,4*(2*nofmodes-1)*nosam);
gains=new matrix(4*(2*nofmodes-1)*nosam,1);
tp=new matrix(4*(2*nofmodes-1)*nosam,1);
findh();
int de,be;

R=new matrix(4*(nosam),4*(nosam));
//cout<<"8\n";
for (pp=0;pp<nofmodes;pp++)
 {//5
//cout<<"9\n";
if(nosam!=jum)
{//6
//cout<<"7\n";
if(pp==0) {trans((*R0),(*R));}
if(pp==1) {trans((*R1),(*R));}
if(pp==2) {trans((*R2),(*R));}
if(pp==3) {trans((*R3),(*R));}
if(pp==4) {trans((*R4),(*R));}
if(pp==5) {trans((*R5),(*R));}
if(pp==6) {trans((*R6),(*R));}
if(pp==7) {trans((*R7),(*R));}
if(pp==8) {trans((*R8),(*R));}
if(pp==9) {trans((*R9),(*R));}
if(pp==10) {trans((*R10),(*R));}
if(pp==11) {trans((*R11),(*R));}
if(pp==12) {trans((*R12),(*R));}
if(pp==13) {trans((*R13),(*R));}
if(pp==14) {trans((*R14),(*R));}
if(pp==15) {trans((*R15),(*R));}
if(pp==16) {trans((*R16),(*R));}
if(pp==17) {trans((*R17),(*R));}
if(pp==18) {trans((*R18),(*R));}
if(pp==19) {trans((*R19),(*R));}
if(pp==20) {trans((*R20),(*R));}
if(pp==21) {trans((*R21),(*R));}
if(pp==22) {trans((*R22),(*R));}
//cout<<"i\n";
}//6
//cout<<"o\n";
findR();
//cout<<"4\n";
findg();
//cout<<"6\n";
if(pp==0) {if(nosam!=jum)delete R0;R0=new matrix(4*nosam,4*nosam);(*R0)=(*R);}
```

```
if(pp==1) {if(nosam!=jum)delete R1;R1=new matrix(4*nosam,4*nosam);(*R1)=(*R);}
if(pp==2) {if(nosam!=jum)delete R2;R2=new matrix(4*nosam,4*nosam);(*R2)=(*R);}
if(pp==3) {if(nosam!=jum)delete R3;R3=new matrix(4*nosam,4*nosam);(*R3)=(*R);}
if(pp==4) {if(nosam!=jum)delete R4;R4=new matrix(4*nosam,4*nosam);(*R4)=(*R);}
if(pp==5) {if(nosam!=jum)delete R5;R5=new matrix(4*nosam,4*nosam);(*R5)=(*R);}
if(pp==6) {if(nosam!=jum)delete R6;R6=new matrix(4*nosam,4*nosam);(*R6)=(*R);}
if(pp==7) {if(nosam!=jum)delete R7;R7=new matrix(4*nosam,4*nosam);(*R7)=(*R);}
if(pp==8) {if(nosam!=jum)delete R8;R8=new matrix(4*nosam,4*nosam);(*R8)=(*R);}
if(pp==9) {if(nosam!=jum)delete R9;R9=new matrix(4*nosam,4*nosam);(*R9)=(*R);}
if(pp==10) {if(nosam!=jum)delete R10;R10=new matrix(4*nosam,4*nosam);(*R10)=(*R);}
if(pp==11) {if(nosam!=jum)delete R11;R11=new matrix(4*nosam,4*nosam);(*R11)=(*R);}
if(pp==12) {if(nosam!=jum)delete R12;R12=new matrix(4*nosam,4*nosam);(*R12)=(*R);}
if(pp==13) {if(nosam!=jum)delete R13;R13=new matrix(4*nosam,4*nosam);(*R13)=(*R);}
if(pp==14) {if(nosam!=jum)delete R14;R14=new matrix(4*nosam,4*nosam);(*R14)=(*R);}
if(pp==15) {if(nosam!=jum)delete R15;R15=new matrix(4*nosam,4*nosam);(*R15)=(*R);}
if(pp==16) {if(nosam!=jum)delete R16;R16=new matrix(4*nosam,4*nosam);(*R16)=(*R);}
if(pp==17) {if(nosam!=jum)delete R17;R17=new matrix(4*nosam,4*nosam);(*R17)=(*R);}
if(pp==18) {if(nosam!=jum)delete R18;R18=new matrix(4*nosam,4*nosam);(*R18)=(*R);}
if(pp==19) {if(nosam!=jum)delete R19;R19=new matrix(4*nosam,4*nosam);(*R19)=(*R);}
if(pp==20) {if(nosam!=jum)delete R20;R20=new matrix(4*nosam,4*nosam);(*R20)=(*R);}
if(pp==21) {if(nosam!=jum)delete R21;R21=new matrix(4*nosam,4*nosam);(*R21)=(*R);}
if(pp==22) {if(nosam!=jum)delete R22;R22=new matrix(4*nosam,4*nosam);(*R22)=(*R);}
}//5
delete R;

if((ssnr-olsnr<tol) || (abs(imag(abn))>real(abn)*.25) || (real(abn)<=0))
{//7
saco+=1;
nosam=nosam-jum;
int jh;
dnew=new matrix(1,nosam);
for(jh=0;jh<nosam;jh++) (*dnew)[0][jh]=(*d)[0][jh];
delete d;
d=new matrix(1,nosam);
for(jh=0;jh<nosam;jh++) (*d)[0][jh]=(*dnew)[0][jh];
}//7

else {
olsnr=ssnr;
saco+=1;
delete gfin;
gfin=new matrix(4*(2*nofmodes-1)*nosam,1);
(*gfin)=(*gains);
}

}//10

  //delete b;
//delete gains;
//delete tp;
qa+=1;
} //4
pnco+=1;
qa=0;
```

```
}//3
cout<<"::"<<indco<<nosam<<"\n";
if(indco==nosam) linesaco+=1;
else linesaco=0;
lineqa+=1;
}//2
linepnco+=1;
lineqa+=1;

}//1

cout<<"ultimate snr="<<olsnr<<"\n";
delete gains;
gains=new matrix(4*(2*nofmodes-1)*nosam,1);
(*gains)=(*gfin);

if(r0==.0012) fk= fopen("gains0012.dat", "w+t"); else
if(r0==.0029) fk= fopen("gains0029.dat", "w+t"); else
if(r0==.0048) fk= fopen("gains0048.dat", "w+t"); else
if(r0==.007) fk= fopen("gains0070.dat", "w+t"); else
if(r0==.0096) fk= fopen("gains0095.dat", "w+t"); else
if(r0==.0123) fk= fopen("gains0123.dat", "w+t"); else
if(r0==.0155) fk= fopen("gains0155.dat", "w+t"); else
if(r0==.0191) fk= fopen("gains0191.dat", "w+t"); else
if(r0==.0233) fk= fopen("gains0233.dat", "w+t"); else
if(r0==.0280) fk= fopen("gains0280.dat", "w+t"); else
if(r0==.0333) fk= fopen("gains0333.dat", "w+t"); else
if(r0==.0393) fk= fopen("gains0393.dat", "w+t"); else
if(r0==.0461) fk= fopen("gains0461.dat", "w+t"); else
if(r0==.0539) fk= fopen("gains0539.dat", "w+t"); else
if(r0==.0627) fk= fopen("gains0627.dat", "w+t"); else
if(r0==.0727) fk= fopen("gains0727.dat", "w+t"); else
if(r0==.084) fk= fopen("gains0840.dat", "w+t"); else
if(r0==.0969) fk= fopen("gains0969.dat", "w+t"); else
if(r0==.1115) fk= fopen("gains1115.dat", "w+t"); else
if(r0==.1281) fk= fopen("gains1281.dat", "w+t"); else
if(r0==.1469) fk= fopen("gains1469.dat", "w+t"); else
if(r0==.1682) fk= fopen("gains1682.dat", "w+t"); else
if(r0==.1924) fk= fopen("gains1924.dat", "w+t"); else
if(r0==.2198) fk= fopen("gains2198.dat", "w+t"); else
if(r0==.25) fk= fopen("gains2500.dat", "w+t"); else
fk= fopen("gains.dat", "w+t");
        fprintf(fk,"%ld \n",nofmodes);
         for(int i=0;i<4*(2*nofmodes-1)*nosam;i++)
         fprintf(fk,"%lg, %lg     \n",
                        real((*gains)[i][0]), imag((*gains)[i][0]));
fclose(fk);



findimage(64,64);
findimagexz(64,64);
//end
}
```

Bibliography

[1] S. W. Young, "Magnetic Resonance Imaging : Basic Principles," Ravan Press New York, 1988.

[2] O. Ocali and E. Atalar, "Ultimate Intrinsic Signal-to-Noise ratio in MRI," *Magnetic Resonance in Medicine*, vol. 39, pp. 462-473 ,1997

[3] H. Vesselle and R. E. Collin, "The signal-to-noise ratio of nuclear magnetic resonance surface coils and application to a lossy dielectric cylinder model -part i:Theory," *IEEE Trans. on Biomedical Engineering*, vol. 42, pp. 497–506, 1995.

[4] W. A. Edelstein, G. H.Glover, C. J. Hardy and R. W. Redington, "The intrinsic signal-to-noise ratio in NMR imaging," *Magnetic Resonance in Medicine*, vol. 3, pp. 604–618, 1986.

[5] E. Yamashita, editor, "Analysis Methods for Electromagnetic Wave Problems,' vol. 1, Artech House, Boston. London, 1990.

[6] R. Rudduck and C. L. Chen, "New plane wave spectrum formulations for the near fields of circular and strip apertures," *IEEE Trans. on Antennas and Propagation*, vol. AP-24, pp. 238–39, 1976.

[7] P. C. Clemmow, "The Plane Wave Representation of Electromagnetic Fields," PERGAMON PRESS, 1966.

[8] C. A. Balanis, "Advanced Engineering Electromagnetics," John Wiley and Sons, 1989.

[9] B. Frank, "Introduction to Bessel functions," Dover Pub., 1958.

[10] Ocali O and Atalar E. Intravascular Magnetic Resonance Imaging Using a Loopless Catheter Antenna. Magnetic Resonance in Medicine,vol. 37, pp. 112-118, 1997.

[11] D. Bertsekas, "Constrained optimization and Lagrange multiplier methods," Academic Press, 1982.

[12] Schnall MD, Lenkinski RE, Pollack HM, Imai Y, and Kressel HY. Prostate: MR imaging with an endorectal surface coil. Radiology 1989; 172(2): p. 570-574.

[13] 1. Shunk KA, Lima JAC, Heldman AW, and Atalar E. Transesophageal Magnetic Resonance Imaging. Magnetic Resonance in Medicine 1999; 41(4): p. 722-726.

[14] Atalar E, Bottomley PA, Ocali O, Correia LCL, Kelemen MD, Lima JAC, and Zerhouni EA. High Resolution Intravascular MRI and MRS by Using a Catheter Receiver Coil. Magnetic Resonance in Medicine 1996; 36(4): p. 596-605.

[15] Correia LCL, Atalar E, Kelemen MD, Ocali O, Hutchins GM, Flag JL, Gerstenblith G, Zerhouni EA, and Lima JAC. Intravascular magnetic resonance imaging of aortic atherosclerotic plaque composition. Arteriosclerosis Thrombosis and Vascular Biology 1997; 17(12): p. 3626-3632.

[16] Hurst GC, Hua J, Duerk JL, and Cohen AM. Intravascular (catheter) NMR receiver probe: preliminary design analysis and application to canine iliofemoral imaging. Magn Reson Med 1992; 24(2): p. 343-357.

[17] Leung DA, Debatin JF, Wildermuth S, McKinnon GC, Holtz D, Dumoulin CL, Darrow RD, Hofmann E, and von Schulthess GK. Intravascular MR tracking catheter: preliminary experimental evaluation. AJR Am J Roentgenol 1995; 164(5): p. 1265-1270.

[18] Martin AJ, Plewes DB, and Henkelman RM. MR imaging of blood vessels with an intravascular coil. J Magn Reson Imaging 1992; 2(4): p. 421-429.

[19] Martin AJ and Henkelman RM. Intravascular MR imaging in a porcine animal model. Magn Reson Med 1994; 32(2): p. 224-229.

[20] Zimmermann GG, Erhart P, Schneider J, vonSchulthess GK, and Debatin JF. Intravascular MR Imaging of Atherosclerotic Plaque : Ex Vivo Analysis of Human Femoral Arteries with Histologic Correlation. Radiology 1997; 204(3): p. 769-774.

[21] Quick HH, Zimmermann-Paul GG, Hofmann E, vonShulthess GK, and Debatin JF. In Vitro Assessment of a Perfused Intravascular MR-Imaging Balloon. in ISMRM Sixth Scientific Meeting and Exhibition. 1998. Syndey, Australia.

[22] Yang X, Atalar E, and Zerhouni EA. Intravascular MR Imaging and Intravascular MR-Guided Interventions. International Journal of Cardiovascular Interventions 1999; 2: p. 85-96.