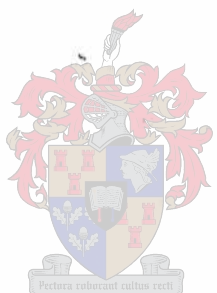
ITERATIVE METHODS IN ELECTROMAGNETIC SCATTERING BASED ON THE MINIMIZATION OF THE ROOT MEAN SQUARE ERROR

by

Pierre Steyn

April 1989



Assignment presented in partial fulfillment of the requirements
for the degree of
MASTERS IN ELECTRONIC ENGINEERING
at the
UNIVERSITY OF STELLENBOSCH.

Supervisor:
D.B. Davidson

DECLARATION

I the undersigned hereby declare that the work contained in this thesis is my own original work and has not previously in its entirety or in part been submitted at any university for a degree.

Pierre Steyn April 1989

SUMMARY

Iterative schemes based on the minimization of the error in scattering problems are presented. In particular, scattering by impenetrable objects is considered. These problems result in operator expressions of convolution type which are solvable using spectral methods. The numeric implementation involves the application of the discrete Fourier transform. A description of the computer implementation of these methods is followed by various numeric results which include a study of the rate of convergence of the schemes and the stability of the solution.

OPSOMMING

Iteratiewe metodes gegrond op die vermindering van die fout in verstrooiingsprobleme word voorgestel. Veral verstrooiing deur ondeurdringbare voorwerpe word ondersoek. Hierdie probleme lei tot operatoruitdrukkings van konvolusie tipe wat deur middel van spektraalmetodes oplosbaar is. Dit behels die toepassing van die diskreet Fourier transform in die numeriese uitvoering. Die rekenaarimplementering van hierdie metodes word beskryf en verskeie numeriese resultate volg wat 'n bestudering van die tempo van konvergensie van die metodes en die stabiliteit van die oplossing insluit.

ACKNOWLEDGEMENTS

Mr D.B. Davidson for his guidance, advice and encouragement.

Prof J.H. Cloete for his guidance and inspiration.

Mr C.F. du Toit for the use of his Pascal routines for calculating Bessel functions and complex functions.

Miss J.M. Case for typing and for encouragement.

NOTATION AND SYMBOLS

D : domain of observation

D': complementary domain to D, ie. all points outside of D

f : the Fourier transform of f (in bold type)

$\mathcal{F}\{f\}$: Fourier transform of a function f

$\mathcal{F}^{-1}\{\mathbf{f}\}$: inverse fourier transform of **f**

p*q : convolution of the functions p and q

Lf : operator L acting on the function f

$\langle f,g \rangle$: the inner product of the functions f and g

$\|f\|$: the norm of f

f^* : the complex conjugate of f

f^T : the transpose of f

$\underline{J}$: underlined variable indicates a vector

$\hat{u}_x$: unit vector in x-direction

$\hat{u}_n$: unit vector normal to a surface

$\underline{J} \times \underline{H}$: the vector product of vectors $\underline{J}$ and $\underline{H}$

Σ : summation symbol

k, k_0 : wavenumber of a medium and of free space respectively

Z_0 : intrinsic impedance of a medium

ÿ : admittivity of a medium

ë : complex permittivity of a medium

$\{p_n\}$: an array or series of values p_n with n=1,2,3...

ABBREVIATIONS

GR : gradient

CGR : conjugate gradient

CST : contrast-source truncation

DFT : discrete Fourier transform

FFT : fast Fourier transform

RMS : root mean square

SDFFT : spectral domain FFT procedure

DCMoM : discrete convolution Method of Moments procedure

CONTENTS

1. Introduction

1.1 Electromagnetic Scattering

Electromagnetic scattering encompasses a variety of physical problems such as the interaction of radio waves with aircraft, the effect of rain and hail on radar signals and the biological effects of microwaves on the human body. Many of these problems can be modelled as bounded structures embedded in a relatively simple medium of infinite extent. Such a configuration is illustrated in figure 1.1 which consists of an object with spatial support Ω being illuminated by a electromagnetic field $\{\underline{E}^i, \underline{H}^i\}$.

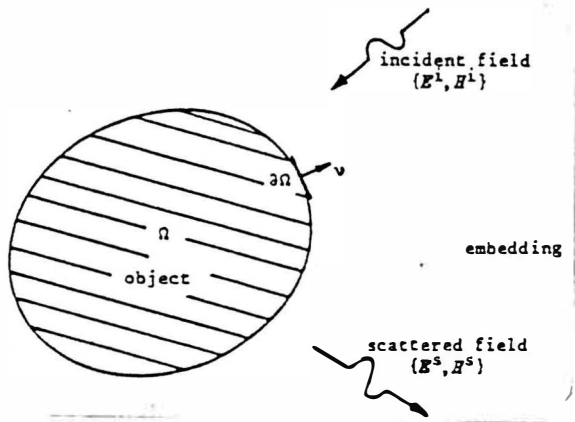


Figure 1.1 : The scattering configuration.(after [1])

The electromagnetic properties of the object differ from that of the medium in which it is embedded. The electromagnetic properties are such that the radiation field from a point source is known.

With the incident field and the properties of the object and the medium known, the problem involves determining the current distribution on the object. From this current the scattered field can be determined. Here we are interested in scattering by impenetrable objects, such as electrically perfect conductors, in a homogeneous medium. The current of interest is thus the surface current.

Exact analytical techniques of solution to these problems are limited to simple geometries. Approximate analytical methods and asymptotic approaches including geometric optics and geometric theory of diffraction can be used to solve many problems to which analytical techniques are not suited. Asymptotic methods are usually limited to the analysis of electrically large scatterers whose geometry can be described in terms of the few canonical shapes for which diffraction coefficients are available.

In the past few decades many computer aided approaches have been developed to solve these problems numerically. Numerical solutions are not fundamentally restricted to scatterers with certain canonical shapes or materials, and in principle they can be carried out to obtain any level of accuracy [8]. They are limited by computer resources such as memory and speed and conventional numerical methods are thus limited to electrically small scatterers.

1.2 Integral Equation Formulation

In this thesis the solution of integral equations arising from integral equation based models in scattering are considered. These have the form of Fredholm integral equations of the first kind

$$g(x) = \int_D K(x,x')f(x')dx' \qquad x \in D \qquad (1.1)$$

In this equation, f is the unknown field quantity in the domain of observation D, g is the known quantity related to the excitation and K is the kernel function of the integral equation.

Further, frequency domain scattering is studied where D is either the spatial domain occupied by the object Ω or the boundary surface $\delta\Omega$ of the object. In the former case

equation (1.1) is a domain integral equation and in the latter a boundary integral equation. The variables x and x' stand for the relevant spatial coordinate system (for example the Cartesian coordinates x, y, z in a three dimensional space). It is assumed that equation (1.1) has a unique solution. Hence, $f(x)=0$, all $x\epsilon D$, if and only if $g(x)=0$, all $x\epsilon D$.

For some scattering problems the kernel function is of the form

$$K(x,x') = K(x-x'). \tag{1.2}$$

Then equation (1.1) has the form of a convolution integral and can be solved using the Fourier transform. The Fourier transform of a function is defined as

$$\mathbf{f}(\alpha) = \mathcal{F}\{f(x)\} = \int_{-\infty}^{\infty} f(x)e^{-j\alpha x}dx \tag{1.3}$$

where x is the spatial domain variable and α is the spectral domain variable. The inverse transform is

$$f(x) = {}^{1}/_{2\pi} \int_{-\infty}^{\infty} \mathbf{f}(\alpha)e^{\,j\alpha x}dx \tag{1.4}$$

The convolution, c(x), of two functions, p(x) and q(x), is defined as

$$c(x) = p*q = \int_{-\infty}^{\infty} p(x-x')q(x)dx' \tag{1.5}$$

The Convolution Theorem states that the Fourier transform of c(x), ie. $\mathbf{c}(x)$, is equal to the product of the transforms of p(x) and q(x).

Thus,

$$\mathbf{c}(\alpha) = \mathbf{p}(\alpha)\mathbf{q}(\alpha)$$

therefore

$$c(x) = f^{-1}\{\mathbf{p}(\alpha)\mathbf{q}(\alpha)\}$$

With K as in equation (1.2), equation (1.1) can be written as

$$g(x) = \int_{-\infty}^{\infty} K(x-x')X_D(x')f(x')dx'$$

$$= f^{-1}\{\mathbf{K}(\alpha)f\{X_D(x)f(x)\}\} \qquad (1.6)$$

where

$$X_D = \begin{cases} 1 & x\epsilon D \\ 0 & x\epsilon D' \end{cases} \qquad (1.7)$$

is the characteristic function. D' is the complementary domain to D.

When $f(x) = 0$, $x\epsilon D'$, then (1.6) simply becomes

$$g(x) = f^{-1}\{\mathbf{K}(\alpha)\mathbf{f}(\alpha)\} \qquad (1.8)$$

1.3 Computational Methods

Once the physical scattering problem has been modelled by a mathematical equation a discretization procedure must be carried out in order to solve numerically. This usually involves replacing the original equation by a finite dimensional matrix equation such as in the Method of Moments. Matrix equations can be solved using direct methods such as Gaussian elimination or iterative methods. However, the larger the electrical size of the problems, the larger the matrix equation becomes. Thus required computer

memory increases. For large systems out-of-core memory may be required which slows down the solution time due to slower accessing speeds. Also large matrices often cannot be accurately solved by direct methods such a Gaussian elimination.

Iterative algorithms have been developed which circumvent the problem of excessive computer time and computer storage for certain classes of problems. This thesis investigates iterative schemes proposed by van den Berg [1,2] which are based on the minimization of the error in field problems.

2. Iterative Minimization of the Root Mean Square Error

The chapter begins with the definition of the functional equation and its related root mean square (RMS) error. An iterative method based upon the minimization of the error is then presented. Variational techniques that enforce a monotonic decrease of the error in each iteration are employed in the method.

The techniques discussed are derived by Peter M. van den Berg [1, 2] and only the results are presented here. The notation used is that of reference [2].

2.1 Functional Equation and Root Mean Square Error

The operator L acting on a function f is defined by

$$Lf = \int_{x' \epsilon D} K(x,x')f(x')dx' \qquad (2.1)$$

In this text the letter L is used for the operator and not K as in [2]. This is to avoid confusion which may arise when K is used to indicate the kernel function. Equation (1.1) can then be written in operator equation form

$$Lf = g \quad , \; x \epsilon D \qquad (2.2)$$

The inner product of two functions f and g defined on D is defined as

$$\langle f,g \rangle = \int_{x \epsilon D} f^*(x)g(x)dx \qquad (2.3)$$

where the asterisk denotes complex conjugate. The norm of a function f is defined as

$$\|f\| = \langle f,f \rangle^{\frac{1}{2}} \qquad (2.4)$$

For an approximate solution f^A of (2.2) the residual is defined as

$$R^A = Lf^A - g \tag{2.5}$$

and the root mean square error as

$$ERR = \langle R^A, R^A \rangle^{\frac{1}{2}}$$

$$= \|R^A\|$$

It is seen that ERR $\geq$ 0, the equality sign holding only when $f^A = f$.

The operator L^T which is adjoint to L is defined by the relation

$$\langle Lg, f \rangle = \langle g, L^{T*} f \rangle \tag{2.7}$$

The adjoint operator expression is

$$g = L^{T*} f \tag{2.8}$$

The notation used here for the adjoint operator follows that of van den Berg. It should be noted that this is somewhat different from other texts which define the adjoint operator to L as L^* (or some other superscript) where

$$\langle Lg, f \rangle = \langle g, L^* f \rangle.$$

The superscript * as used in this latter definition does not indicate the complex conjugate but replaces T*.

2.2 An Iterative Approximation to the Solution of the Functional Equation

This section presents an iterative minimization of the RMS error which leads to the solution of the operator equation (2.2). The method presented is general in that it does not only apply to operator equations of convolution type. At each step of the iterative procedure we write

$$f^{(n)}(x) = f^{(n-1)}(x) + f_{cor}{}^{(n)}(x), \quad n = 1,2,3... \tag{2.9}$$

where $f^{(n)}(x)$ is the approximate solution to $f(x)$ at iteration n and $f_{cor}{}^{(n)}$ is a suitably constructed correction function. The procedure begins with an initial guess $f^{(0)}$ and the associated residual $R^{(0)}$. The residual at iteration n is

$$R^{(n)}(x) = Lf^{(n)} - g \tag{2.10}$$

Substitution of (2.9) into (2.10) leads to

$$R^{(n)} = R^{(n-1)} + Lf_{cor}{}^{(n)} \tag{2.11}$$

The RMS error at iteration n is then

$$ERR^{(n)} = \|R^{(n)}\| \tag{2.12}$$

The correction function $f_{cor}{}^{(n)}$ must now be constructed. Let

$$f_{cor}{}^{(n)}(x) = \alpha_1{}^{(n)}\varphi^{(n)}(x) \tag{2.13}$$

where $\alpha_1{}^{(n)}$ is a variational parameter and φ is a suitably chosen variational function, the choice of which is discussed in the next section.

Using van den Berg results, but generalizing to iteration n with only one variational parameter, we can show that $ERR^{(n)}$ is minimized if [2, equation (4.6)]

$$\alpha_1^{(n)} = -\langle L\varphi^{(n)}, R^{(n-1)}\rangle / \|L\varphi^{(n)}\|^2. \qquad (2.14)$$

To achieve anything at all $\alpha_1^{(n)}$ must not be zero, ie.

$$\langle L\varphi^{(n)}, R^{(n-1)}\rangle \neq 0 \qquad (2.15)$$

which is the improvement condition and places some restriction on the choice of $\varphi^{(n)}$.

Thus, assuming the existence of a variational function, $\varphi^{(n)}$, we have an iterative method using one variational parameter to minimize the RMS error. The algorithm is presented in table 2.1. It is based on van den Berg's table V in [2] but with only one variational parameter and a free choice of variational function. The algorithm is terminated when the error is less than a specified maximum allowable error ERR_{max} or when the number of iterations has reached a specified maximum n_{max}.

Table 2.1 : The iteration scheme with one variational parameter.

initial estimate: $f^{(0)}$

$\downarrow$

residual error: $R^{(0)} = Lf^{(0)} - g$

$\downarrow$

$ERR^{(0)} = \|R^{(0)}\|$

$\downarrow$

$n = n + 1$

$\downarrow$

is $n = n_{max}$? yes $\rightarrow$ write :"error has not fully converged" $\rightarrow$ end

no $\downarrow$

$\downarrow$

determine variational function $\varphi^{(n)}$ and $L\varphi^{(n)}$

$\downarrow$

$\alpha_1^{(n)} = -\langle L\varphi^{(n)}, R^{(n-1)}\rangle / \|L\varphi^{(n)}\|^2$

$\downarrow$

$f_{cor}^{(n)} = \alpha_1^{(n)}\varphi^{(n)}$, $Lf_{cor}^{(n)} = \alpha_1^{(n)} L\varphi^{(n)}$

$\downarrow$

$f^{(n)} = f^{(n-1)} + f_{cor}^{(n)}$, $Lf^{(n)} = Lf^{(n-1)} + Lf_{cor}^{(n)}$

$\downarrow$

$R^{(n)} = Lf^{(n)} - g$

$\downarrow$

$ERR^{(n)} = \|R^{(n)}\|$

$\downarrow$

no $\leftarrow$ is $ERR^{(n)} \leq err_{max}$? yes $\rightarrow$ end

For $n > 1$, $f_{cor}^{(n)}$ can be constructed using more information from the previous iteration.

Let

$$f_{cor}^{(n)} = \alpha_1^{(n)}\varphi^{(n)} + \alpha_2^{(n)} f_{cor}^{(n-1)}, \quad n = 2,3,... \tag{2.16}$$

It can be shown that $ERR^{(n)}$ is minimized if $\alpha_1^{(n)}$ and $\alpha_2^{(n)}$ satisfy the system of two linear algebraic equations [2]:

$$\begin{bmatrix} \langle v_1, v_1\rangle & \langle v_1, v_2\rangle \\ \langle v_2, v_1\rangle & \langle v_2, v_2\rangle \end{bmatrix} \begin{bmatrix} \alpha_1^{(n)} \\ \alpha_2^{(n)} \end{bmatrix} = \begin{bmatrix} -\langle v_1, R^{(n-1)}\rangle \\ 0 \end{bmatrix} \tag{2.17}$$

where

$$v_1 = L\varphi^{(n)} \text{ and } v_2 = Lf_{cor}^{(n-1)}$$

The improvement condition is also equation (2.15). The algorithm based on two variational parameters is presented in table 2.2. It is based on van den Berg's table V in [2] but with only two variational parameters and a free choice of variational function.

Table 2.2 : The iteration scheme with two variational parameters.

initial estimate: $f^{(0)}$

↓

residual error: $R^{(0)} = Lf^{(0)} - g$

↓

$ERR^{(0)} = \|R^{(0)}\|$

↓

[$n = 1$]

↓

determine variational function $\varphi^{(1)}$ and $L\varphi^{(1)}$

↓

$\alpha_1^{(1)} = -\langle L\varphi^{(1)}, R^{(0)}\rangle / \|L\varphi^{(1)}\|^2$

↓

$f_{cor}^{(1)} = \alpha_1^{(1)}\varphi^{(1)}$, $Lf_{cor}^{(1)} = \alpha_1^{(1)}L\varphi^{(1)}$

↓

$f^{(1)} = f^{(0)} + f_{cor}^{(1)}$, $Lf^{(1)} = Lf^{(0)} + Lf_{cor}^{(1)}$

↓

$R^{(1)} = Lf^{(1)} - g$, $ERR^{(1)} = \|R^{(1)}\|$

↓

$n = n + 1$

↓

is $n = n_{max}$? yes → write :"error has not fully converged" → end

no ↓

↓

determine variational function $\varphi^{(n)}$ and $L\varphi^{(n)}$

↓

$v_1 = L\varphi^{(n)}$, $v_2 = Lf_{cor}^{(n-1)}$

↓

solve equation (2.17)

↓

$f_{cor}^{(n)} = \alpha_1^{(n)}\varphi^{(n)} + \alpha_2^{(n)}f_{cor}^{(n-1)}$

↓

$Lf_{cor}^{(n)} = \alpha_1^{(n)}L\varphi^{(n)} + \alpha_2^{(n)}Lf_{cor}^{(n-1)}$

↓

$f^{(n)} = f^{(n-1)} + f_{cor}^{(n)}$, $Lf^{(n)} = Lf^{(n-1)} + Lf_{cor}^{(n)}$

↓

$R^{(n)} = Lf^{(n)} - g$

↓

$ERR^{(n)} = \|R^{(n)}\|$

↓

no ← is $ERR^{(n)} \leq err_{max}$? yes → end

For $n > 2$ van den Berg [2] introduces the possibility of a third variational parameter.

Let

$$f_{cor}^{(n)} = \alpha_1^{(n)}\varphi^{(n)} + \alpha_2^{(n)}f_{cor}^{(n-1)} + \alpha_3^{(n)}f^{(n-1)} \tag{2.18}$$

It is found that $ERR^{(n)}$ is minimized if $\alpha_1^{(n)}$, $\alpha_2^{(n)}$ and $\alpha_3^{(n)}$ satisfy the system of three linear algebraic equations

$$\begin{bmatrix} \langle v_1,v_1\rangle & \langle v_1,v_2\rangle & \langle v_1,v_3\rangle \\ \langle v_2,v_1\rangle & \langle v_2,v_2\rangle & \langle v_2,v_3\rangle \\ \langle v_3,v_1\rangle & \langle v_3,v_2\rangle & \langle v_3,v_3\rangle \end{bmatrix} \begin{bmatrix} \alpha_1^{(n)} \\ \alpha_2^{(n)} \\ \alpha_3^{(n)} \end{bmatrix} = \begin{bmatrix} -\langle v_1,R^{(n-1)}\rangle \\ 0 \\ 0 \end{bmatrix} \tag{2.19}$$

where $v_1 = L\varphi^{(n)}$, $v_2 = Lf_{cor}^{(n-1)}$ and $v_3 = Lf^{(n-1)}$.

As before, the improvement condition is equation (2.15). The algorithm based on three variational parameters is presented in table 2.3. It is based on van den Berg's table V in [2] but with a free choice of variational function.

Table 2.3 : The iteration scheme with three variational parameters.

initial estimate: $f^{(0)}$ -- $R^{(0)}=Lf^{(0)}-g$, $ERR^{(0)} = \|R^{(0)}\|$

↓

[n = 1]

↓

determine variational function $\varphi^{(1)}$ and $L\varphi^{(1)}$

↓

$\alpha_1^{(1)} = -\langle L\varphi^{(1)}, R^{(0)}\rangle / \|L\varphi^{(1)}\|^2$

↓

$f_{cor}^{(1)} = \alpha_1^{(1)}\varphi^{(1)}$, $Lf_{cor}^{(1)} = \alpha_1^{(1)}L\varphi^{(1)}$

↓

$f^{(1)} = f^{(0)} + f_{cor}^{(1)}$, $Lf^{(1)} = Lf^{(0)} + Lf_{cor}^{(1)}$

↓

$R^{(1)} = Lf^{(1)} - g$, $ERR^{(1)} = \|R^{(1)}\|$

↓

[n = 2]

↓

determine variational function $\varphi^{(2)}$ and $L\varphi^{(2)}$

↓

$v_1 = L\varphi^{(2)}$, $v_2 = Lf_{cor}^{(1)}$

↓

solve equation (2.17) with n = 2

↓

$f_{cor}^{(2)} = \alpha_1^{(2)}\varphi^{(2)} + \alpha_2^{(2)}f_{cor}^{(1)}$

↓

$Lf_{cor}^{(2)} = \alpha_1^{(2)}L\varphi^{(2)} + \alpha_2^{(2)}Lf_{cor}^{(1)}$

↓

$f^{(2)} = f^{(1)} + f_{cor}^{(2)}$, $Lf^{(2)} = Lf^{(1)} + Lf_{cor}^{(2)}$

↓

$R^{(2)} = Lf^{(2)} - g$, $ERR^{(2)} = \|R^{(2)}\|$

↓

→ n = n + 1

↓

is n = n_{max} ? → yes → write :"error not fully converged" → end

no ↓

determine variational function $\varphi^{(n)}$ and $L\varphi^{(n)}$

↓

$v_1 = L\varphi^{(n)}$, $v_2 = Lf_{cor}^{(n-1)}$, $v_3 = Lf^{(n-1)}$

↓

solve equation (2.19)

↓

$f_{cor}^{(n)} = \alpha_1^{(n)}\varphi^{(n)} + \alpha_2^{(n)}f_{cor}^{(n-1)} + \alpha_3^{(n)}f^{(n-1)}$

↓

$Lf_{cor}^{(n)} = \alpha_1^{(n)}L\varphi^{(n)} + \alpha_2^{(n)}Lf_{cor}^{(n-1)} + \alpha_3^{(n)}Lf^{(n-1)}$

↓

$f^{(n)} = f^{(n-1)} + f_{cor}^{(n)}$, $Lf^{(n)} = Lf^{(n-1)} + Lf_{cor}^{(n)}$

↓

$R^{(n)} = Lf^{(n)} - g$

↓

$ERR^{(n)} = \|R^{(n)}\|$

↓

no ← is $ERR^{(n)} \leq err_{max}$? → yes → end

2.3 Iterative Generation of the Variational Functions

In this section we look at the generation of variational functions that can be used in the algorithms presented in tables 2.1 to 2.3. Two techniques, namely the gradient and the contrast source truncation techniques, are investigated. The second technique relies on the operator being of convolution type.

2.3.1 Gradient(GR) Technique

In section 2.2 it was seen that the error at each iteration is minimized only if

$$<L\varphi^{(n)},R^{(n-1)}> \text{ is not zero.}$$

This can be accomplished if we set [2, section 5]

$$\varphi^{(n)}(x) = L^{T*}R^{(n-1)} \tag{2.20}$$

where L^T is the adjoint operator to L.

Then, using equation (2.7)

$$<L\varphi^{(n)},R^{(n-1)}> = <LL^{T*}R^{(n-1)},R^{(n-1)}>$$

$$= <L^{T*}R^{(n-1)},L^{T*}R^{(n-1)}>$$

which will not be zero provided that $R^{(n-1)}$ is not zero. (If $R^{(n-1)}$ is zero then $ERR^{(n-1)}$ would be zero and the solution would have been arrived at.)

With this choice of $\varphi^{(n)}$ it can be shown that

$$<\varphi^{(n)},\varphi^{(n+1)}> = 0$$

exhibiting the orthogonality of the gradients in two successive iterations.

With a initial estimate $f^{(0)}=0$ van den Berg shows that the third variational parameter becomes zero for the gradient

technique. Thus this technique is only used with up to two variational parameters.

This choice of variational function used with the algorithm presented in table 2.1, that is with one variational parameter, is equivalent to the gradient method presented in [1, table I]. With two variational parameters, as in table 2.2, it is equivalent to the gradient method of [1] with the second minimization step which results in the conjugate gradient (CGR) method presented in [1, table I].

2.3.2 Contrast Source Truncation(CST) Technique

This technique can be used when the operator is of the convolution type as defined in section 1.2. The Fourier transform of the operator expression Lf can be written as the product of the Fourier transforms **K** and **f**.

$$\mathcal{F}\{Lf\} = \mathbf{Kf} \tag{2.21}$$

The variational function $\varphi^{(n)}$ is determined as the approximate "contrast source" at D that corresponds as closely as possible to the "field function" $R^{(n-1)}$ at D. It is then defined by

$$R_D^{(n-1)} = L\varphi^{(n)}$$

where

$$R_D^{(n-1)} = X_D R^{(n-1)}$$

Then

$$\mathbf{R}_D^{(n-1)} = \mathbf{K}\boldsymbol{\varphi}^{(n)}$$

thus

$$\boldsymbol{\varphi}^{(n)} = \mathbf{K}^{-1}\mathbf{R}_D^{(n-1)} \tag{2.22}$$

With $\boldsymbol{\varphi}^{(n)}$ determined by equation (2.22), $\varphi^{(n)}$ is obtained by carrying out the inverse Fourier transform. In order to calculate $L\varphi^{(n)}$ the function $\varphi^{(n)}$ must be windowed by setting

$$\varphi^{(n)}(x) = 0, \quad x \epsilon D'$$

The result is the contrast source truncation technique. The determination of $\varphi^{(n)}$ and $L\varphi^{(n)}$ is summarized in table 2.4. [after 2, Table III]

Table 2.4 : The determination of the variational functions $\varphi^{(n)}$ and the $L\varphi^{(n)}$ based on the contrast-source truncation technique.

x domain		spectral domain
$R^{(n-1)}(x)$, $x\epsilon D$ 0, $x\epsilon D'$	$\rightarrow$	$R_D^{(n-1)}$
		$\downarrow$
$\varphi^{(n)}$	$\leftarrow$	$[K]^{-1}R_D^{(n-1)}$

$\varphi^{(n)}$, $x\epsilon D$, is the variational function (conrast source)

$\varphi^{(n)}(x)$, $x\epsilon D$ 0, $x\epsilon D'$	$\rightarrow$	$\varphi^{(n)}$
		$\downarrow$
$L\varphi^{(n)}$	$\leftarrow$	$K\varphi^{(n)}$

2.4 Initial Guess

A degree of freedom in using the iteration scheme is the choice of the initial value $f^{(0)}(x)$. Any choice can be taken and an obvious one is $f^{(0)}(x) = 0$. Another obvious choice for impenetrable objects is the physical optics approximation [4, pp127-128] otherwise known as the Kirchoff approximation. In this approximation the surface current $\underline{J}_s$ in the illuminated portion of the scatterer's surface is approximated by

$$\underline{J}_s \approx 2\hat{u}_n \times \underline{H}^i \qquad (2.23)$$

where $\underline{H}^i$ is the incident magnetic field and $\hat{u}_n$ is the normal to the surface. The Kirchhoff approximation is not an optimal initial estimate. For a non-zero initial estimate $f^{(0)}$ (such as the Kirchhoff approximation) a modified estimate [1]

$$\hat{f}^{(0)}(x) = \Gamma^{(0)} f^{(0)}(x) \tag{2.24}$$

where Γ is a parameter to be derived, may be determined in such a way that $ERR^{(0)}$ is minimized. The residual in this case is

$$\hat{R}^{(0)} = L\hat{f}^{(0)} - g$$

$$= \Gamma^{(0)} Lf^{(0)} - g$$

Thus

$$\langle \hat{R}^{(0)}, \hat{R}^{(0)} \rangle = \int_D (\Gamma^{(0)} Lf^{(0)} - g)^* (\Gamma^{(0)} Lf^{(0)} - g) dx$$

$$= \langle g,g \rangle - \Gamma^{(0)} \langle g, Lf^{(0)} \rangle$$

$$- \Gamma^{(0)*} \langle Lf^{(0)}, g \rangle + |\Gamma^{(0)}|^2 \langle Lf^{(0)}, Lf^{(0)} \rangle$$

$$= \langle g,g \rangle - \Gamma^{(0)} A^{(0)*} - \Gamma^{(0)*} A^{(0)} + |\Gamma^{(0)}|^2 B^{(0)}$$

$$= \langle g,g \rangle - \frac{|A^{(0)}|^2}{B^{(0)}}$$

$$+ \left[|\Gamma^{(0)}|^2 - \frac{\Gamma^{(0)} A^{(0)*}}{B^{(0)}} - \frac{\Gamma^{(0)*} A^{(0)}}{B^{(0)}} + \frac{|A^{(0)}|^2}{|B^{(0)}|^2} \right] B^{(0)}$$

$$= \langle g,g \rangle - \frac{|A^{(0)}|^2}{B^{(0)}} + \left| \Gamma^{(0)} - \frac{A^{(0)}}{B^{(0)}} \right|^2$$

where $A^{(0)} = \langle Lf^{(0)}, g \rangle$ and $B^{(0)} = \|Lf^{(0)}\|^2$

This last equation is minimized if

$$\Gamma^{(0)} = \frac{A^{(0)}}{B^{(0)}}$$

The algorithm to determine the modified initial estimate is shown in table 2.5.

Table 2.5 : Determination of modified initial estimate to minimize $ERR^{(0)}$.

initial estimate $f^{(0)} \neq 0$

↓

evaluate $Lf^{(0)}$

↓

$A^{(0)} = \langle Lf^{(0)}, g \rangle$

↓

$B^{(0)} = \langle Lf^{(0)}, Lf^{(0)} \rangle$

↓

$\Gamma^{(0)} = A^{(0)}/B^{(0)}$

↓

$\hat{f}^{(0)} = \Gamma^{(0)} f^{(0)}$

↓

evaluate $L\hat{f}^{(0)}$

↓

$\hat{R}^{(0)} = L\hat{f}^{(0)} - g$

↓

$ERR^{(0)} = \|\hat{R}^{(0)}\|$

3. Scattering by Perfectly Conducting Strip

In this chapter the numerical implementation of the iterative algorithms presented in tables 2.1 to 2.5 in chapter 2 is discussed. The specific problem to which the algorithm is applied is plane-wave scattering by a perfectly conducting strip.

The emphasis in this implementation is on investigating the convergence of the schemes as well as the effect of factors such as discretization, size of FFT and lossiness of the embedding on the stability of the converged solution.

3.1 Integral Equation Formulation

The problem of electromagnetic scattering by an electrically perfectly conducting strip has been given a fair amount of attention in the available literature. [1,2,3] There is thus data for comparison purposes, especially as far as error convergence is concerned.

The geometry of the problem is illustrated in figure 3.1. The strip is illuminated by a transverse magnetic(TM)-polarized plane wave with electric field vector oriented parallel to the edges of the strip.

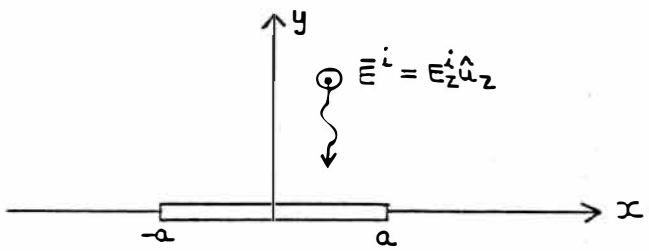


Figure 3.1 : Plane-wave scattering by a strip.

The integral equation formulation (derived in appendix A) for the problem is

$$E_z^i(x) = \int_{-a}^{a} \frac{Z_0 k}{4} H_0^{(1)}(k|x-x'|)J_z(x')dx', \quad -a \le x \le a \qquad (3.1)$$

where k is the wavenumber of the medium, Z_0 is the intrinsic impedance of the medium, and $H_0^{(1)}(x)$ is the zero order Hankel function of the first kind.

For convenience the current can be normalized to $(kZ_0)^{-1}$ then a modified integral equation is obtained:

$$E_z^i(x) = \int_{-a}^{a} {}^1/_4 H_0^{(1)}(k|x-x'|)I_z(x')dx, \quad -a \le x \le a \qquad (3.2)$$

where $I_z(x)$ is the normalized current.

The kernel of this equation is

$$K(x,x') = K(x-x') = {}^1/_4 H_0^{(1)}(k|x-x'|) \qquad (3.3)$$

The known function is

$$g(x) = E_z^i(x)$$

and the unknown is

$$f(x) = I_z(x)$$

The implementation of tables 2.1 to 2.5 to solve equation (3.2) is discussed in the following sections of this chapter.

The physical optics approximation for this case can be derived using the theory in [4, pp 127-128]. The illuminated surface S' is the top surface of the strip which is merely the surface (or domain) D shown in figure 3.1.

On S' the surface current is approximated as

$$\underline{J}_s = 2\ \hat{u}_n \times \underline{H}^i = 2\ \hat{u}_y \times \underline{H}^i$$

With

$$\underline{E}^i = \hat{u}_z, \quad \underline{H}^i = -1/Z_0\ \hat{u}_x$$

thus

$$\underline{J}_s = 2\ \hat{u}_y \times (-1/Z_0\ \hat{u}_x) = 2/Z_0\ \hat{u}_z$$

Normalizing to $1/Z_0 k$ we get

$$I_z(x) = 2k$$

3.2 Discretization

To solve for the integral equation numerically, the surface current can only be calculated at a finite number of points on the strip. The strip is thus divided into N intervals of length h along the x-axis as shown in figure 3.2. The current is evaluated at the center of each interval. These points are marked 1 to N in the figure.

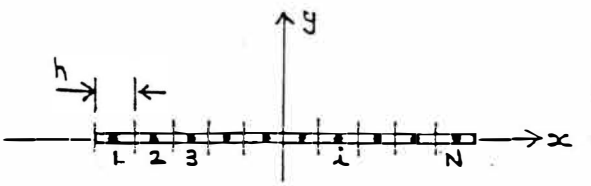


Figure 3.2 : Discretization of the strip.

The interval length is given by

$$h = \frac{2a}{N} \tag{3.4}$$

The normalized surface current is thus approximated as

$$I_z(x) \approx \sum_{n=1}^{N} I_i \delta(-a + {}^{h}/_{2} + (i-1)h) \tag{3.5}$$

where the coefficients I_i, i = 1...N, are to be evaluated.

All functions evaluated in executing the algorithm of table 2.1 numerically are thus represented by vectors of N elements except for the Kernel which also exists for values of x off the strip. Successive samples contained in the vectors are a distance h apart.

3.3 Evaluation of Inner Products

For computational simplicity all integrals of non-convolution type are evaluated by approximating as sums. Thus an integral of the form

$$U = \int_{-a}^{a} y(x)dx \tag{3.6}$$

where y(x) is a function defined on the strip, is approximated as

$$U \approx h \sum_{i=1}^{N} y_i \tag{3.7}$$

where h is the interval length as given in (3.4) and y_i is the value of y(x) at the centre of interval i. The approximation is illustrated in figure 3.3.

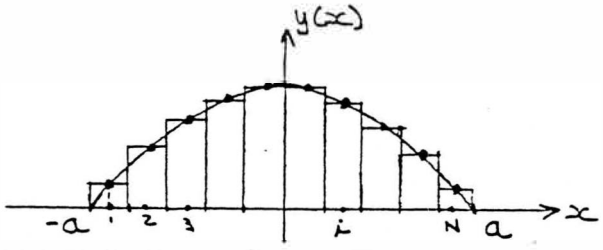


Figure 3.3 : Integral as approximation of area beneath a function.

The numerical approximation is thus the sum of the areas of N pulses of width h and height y_i, i = 1,2...N.

The inner-product is evaluated by applying equation (3.7) to equation (2.3) and obtaining:

$$\langle f.g \rangle \approx h \sum_{n=1}^{N} f_i^* g_i \qquad (3.8)$$

and the norm by applying equation (3.7) to equation (2.7) and obtaining

$$\|f\| = \langle f,f \rangle^{\frac{1}{2}} \approx [h \sum_{n=1}^{N} |f_i|^2]^{\frac{1}{2}} \qquad (3.9)$$

where f_i and g_i are respectively the values of f(x) and g(x) at the centre of interval i.

3.4 Evaluation of Operator Expression

In the generation of the variational functions as required in the algorithms of tables 2.1 to 2.5, evaluation of the operator and the adjoint operator expressions are required. This is seen in section 2.3 where the generation of the variational function is discribed. For the contrast source truncation technique it is required to transform between the

spatial and spectral domains. These steps can be accomplished numerically using the fast Fourier transform (FFT) as the operator is of convolution type.

The integral in the operator expression of equation (2.2) has the form

$$C(x) = \int_{-\infty}^{\infty} K(x-x')p(x')dx' \qquad (3.10)$$

which is a convolution. The function p(x) exists only over the strip surface, ie. $-a \leq x \leq a$.

To evaluate equation (3.10) numerically, the discrete Fourier transforms(DFT) of the functions K(x) and p(x) must be evaluated. The product of these transforms results in the DFT of C(x) which can then be obtained by carrying out the inverse transform. This is all done using the FFT. Generally, the function p(x) already exists as a vector of N elements. We use a M point FFT where M>N, and M is an integer power of 2. To evaluate the FFT of p(x) we form an array $\{p_n\}$ of M-elements by adding zeros to the N-element array as shown in figure 3.4.

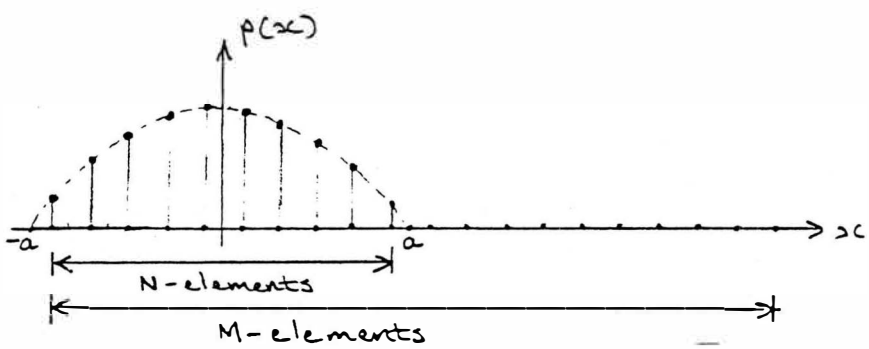


Figure 3.4 : Formation of an M point array by adding zeros to a N point array representing a function defined on the strip surface.

The DFT assumes periodicity in the spatial and spectral domains. The problems is thus changed and now consists of an infinite array of strips with a distance d = (M-N)h between subsequent strips as shown in figure 3.5.

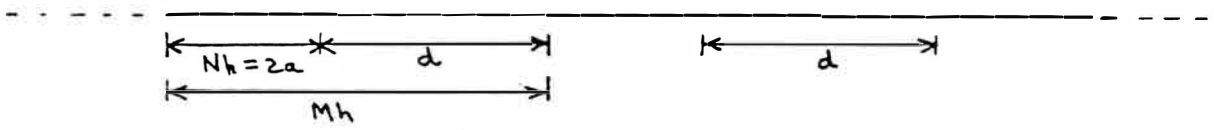


Figure 3.5 : New geometry due to use of the FFT.

The kernel of the integral is defined for all x. Thus aliasing will occur when using the DFT which can be seen as coupling between the strips. However, as it contains the Hankel function its real and imaginary parts decrease monotonically with increasing x. Thus if M is chosen large enough, the distance between the strips will be large and aliasing, or coupling, will be small.

The kernel function K(x) must be sampled. This function is not only defined on the strip surface and when sampled more than N points can be obtained. Looking at equation (3.3) it is seen that K(x) contains a Hankel function of type 1 and order zero. The imaginary part of this function tends to negative infinity when x is zero. The Fourier transform of the Hankel function is simply [9, eqn.(43)][1]

$$\mathcal{F}\{H_0^{(1)}(k|x|)\} = 2(k^2-\alpha^2)^{-\frac{1}{2}} \qquad (3.11)$$

---

1 Equation (43) in [8] is the Fourier transform of $\frac{1}{2}H_0^{(1)}(k|x|)$ thus it is multiplied by a factor 2 to obtain equation (3.11).

By multiplying (3.11) with the factor $^1/_4$ the Fourier transform of the kernel of (3.3) is obtained

$$\mathbf{K}(\alpha) = [4(k^2-\alpha^2)]^{-\frac{1}{2}} \tag{3.12}$$

This equation has a branch point at $\alpha = k$. This situation can be avoided, however, if we assume that the medium surrounding the strip is lossy. A complex wavenumber is thus defined as

$$k = k_0(1 + jl_f) \tag{3.13}$$

where k_0 is the free space wavenumber and l_f is a "loss factor" which must be non zero.

The singularity at $x = 0$ in equation (3.3) can thus be avoided if instead of sampling $K(\alpha)$ in the spatial domain and obtaining the DFT via the FFT, we obtain the DFT by sampling $\mathbf{K}(\alpha)$ in the spectral domain. With the total interval in the spatial domain being Mh the sample interval in the spectral domain is $\alpha_p = {}^{2\pi}/_{Mh}$.

The relationship between an element of the DFT, $\mathbf{K}_n$ and a sample of the continuous transform $\mathbf{K}(\alpha_n)$, derived in reference [6] on p389, is

$$\mathbf{K}(\alpha_n) \approx h\mathbf{K}_n \tag{3.14}$$

where h is as defined in equation (3.4)

Values of the DFT can thus be approximated as

$$\mathbf{K}_n \approx h^{-1}\mathbf{K}(\alpha_n) \tag{3.15}$$

However, because of the relationship in (3.14) the values obtained when carrying out the convolution using the DFT will be h times the actual value. This is proved as follows. The convolution integral is

$$c(x) = \int_{-\infty}^{\infty} K(x-x')p(x')dx'$$

The Fourier transform of p(x) is

$$\mathbf{p}(\alpha) = \int_{-\infty}^{\infty} p(x)e^{-j\alpha x}dx$$

At a specified value of α, α_m we can express $p(\alpha_m)$ as an approximate summation:

$$\mathbf{p}(\alpha_m) \approx \sum_{n=0}^{M-1} p(x_n)\exp(-j\alpha_m x_n)h$$

However, the DFT is defined as

$$\mathbf{p}_m = \sum_{n=0}^{M-1} p(x_n)\exp(-j\alpha_m x_n)$$

Thus

$$p(\alpha_m) \approx hp_m = h\sum_{n=0}^{M-1} p(x_n)\exp(-j\alpha_m x_n)$$

In the same way

$$K(\alpha_m) \approx hK_m = h\sum_{n=0}^{M-1} K(x_n)\exp(-j\alpha_m x_n)$$

Then,

$$K_m p_m = \frac{K(\alpha_m)p(\alpha_m)}{h^2}$$

$$= \frac{1}{h}\,\frac{c(\alpha_m)}{h}$$

$$= h^{-1}c_m$$

Thus,

$$c_m = hK_m p_m$$

The factor h will cancel out if it is left out of equation (3.15) when forming $\{K_m\}$. Thus, we form $\{K_m\}$ as

$$K_m \approx K(\alpha_m) \qquad (3.16)$$

Then $c_m = K_m p_m$ resulting in the correct values of the convolution.

The kernel, $K(x) = {}^1/_4 H_0^{(1)}(k|x|)$ is an even function thus its Fourier transform $K(\alpha)$ is even. The DFT will also be even. However, as the DFT assumes periodicity the DFT values at negative frequencies are repeated at positive frequencies as illustrated in figure 3.6.

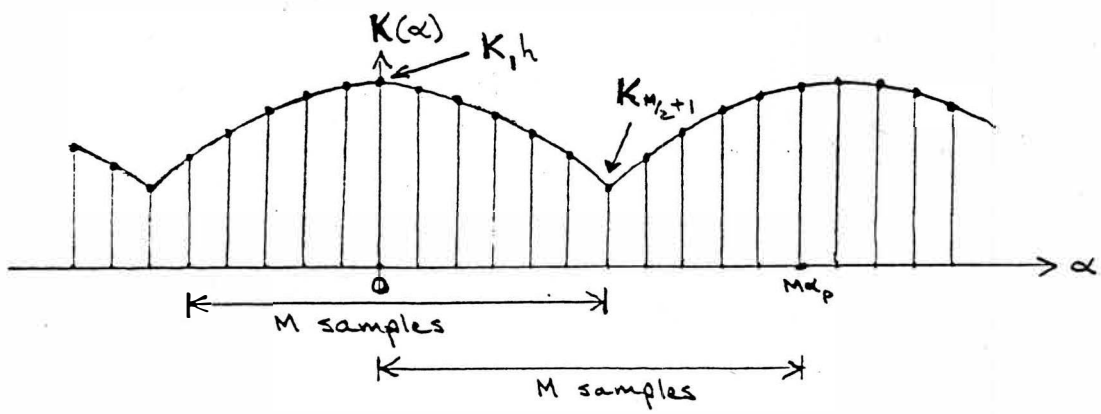


Figure 3.6 : Periodicity due to sampling.

If we make the sample at $\alpha = 0$ our first DFT value $K_1 = K(0)$, then the remaining samples are symmetrical about $K_{\frac{1}{2}M+1}$. Thus, beginning at $\alpha = 0$ we take $\frac{1}{2}M+1$ samples. The remaining $\frac{1}{2}M-1$ samples are obtained according to the rule

$$K_{\frac{1}{2}M+1+i} = K_{\frac{1}{2}M+1-i} \quad , \; i=1\ldots\tfrac{1}{2}M-1 \tag{3.17}$$

If an inverse FFT were now carried out on the DFT of K(x) obtained above the results will be an M element array containing M approximate samples of K(x). However, in order to carry out a linear convolution with a result M-elements in length, the lengths N_1 and N_2 of the arrays being convolved must be such that $N_1+N_2-1=M$ [7, p110-112]. Thus with the array $\{p_n\}$ of length $N_1=N$ the array $\{K_n\}$ representing K(x) must be of length $N_2=M-N+1$ or less. This requires that $\{K_n\}$ must have at least N-1 zeros at its end. Thus, once the sampling has been carried out in the spectral domain, an inverse FFT is carried out in the resulting array. The last N values are set to zero. This array is then transformed back to the spectral domain. The result is modified values of K_n which correspond to the kernel truncated to M-N values in the spatial domain.

p_n and K_n are multiplied to form

$$c_n = K_n p_n \tag{3.18}$$

The inverse FFT carried out on c_n to obtain an M-element array contains the convolution values. However, only the first N-elements of this array are used as the sampled values of c(x).

Looking at figure 3.4 it appears that by placing the samples of p(x) at negative x at the beginning of the array a spatial shift has been carried out on p(x) while this shift has not been executed on K(x). However, this merely means that the array representing c(x) has also been shifted. The

effect of a shift by a distance x_0 is as follows. The convolution is

$$c(x) = \int_{-\infty}^{\infty} K(x-x')p(x')dx'$$

Let $x_1 = x'+x_0$, then

$$c(x) = \int_{-\infty}^{\infty} K(x-x_1+x_0)p(x_1-x_0)dx'$$

Let $x_2 = x+x_0$, then

$$c(x_2-x_0) = \int_{-\infty}^{\infty} K(x_2-x_1)p(x_1-x_0)dx'$$

or, replacing x_2 with x and x_1 with x'

$$c(x-x_0) = \int_{-\infty}^{\infty} K(x-x')p(x'-x_0)dx'.$$

Thus, if $K(x)$ is not shifted then $c(x)$ is shifted by the same amount as $p(x)$.

So far the evaluation of the operator expression of the form of (2.2) has been investigated. However, evaluation of the adjoint operator expression is also required. For the strip the integral has the form

$$d(x) = \int_{-\infty}^{\infty} K^*(x'-x)p(x')dx'$$

$$= \int_{-\infty}^{\infty} \{{}^1/_4 H_0^{(1)}(k|x'-x|)\}^* p(x')dx'$$

$$= \int_{-\infty}^{\infty} \{{}^1/_4 H_0^{(1)}(k|x-x'|)\}^* p(x')dx'$$

Thus, $$d(x) = \int_{-\infty}^{\infty} K^*(x-x')p(x')dx' \qquad (3.19)$$

This last integral is a convolution. The same procedure can be followed as was for the evaluation of equation (3.10) except that the Fourier transform of $K^*(x)$ must be used. This can be easily found from the Fourier transform of $K(x)$ using the property

$$\mathcal{f}\{K^*(x)\} = \mathbf{K}^*(-\alpha) \qquad (3.20)$$

where $$\mathcal{f}\{K(x)\} = \mathbf{K}(\alpha)$$

Thus, for the strip, using equation (3.12) in (3.20)

$$\mathcal{f}\{K^*(x)\} = [{}^1/_2(k^2-(-\alpha)^2)^{-1}]^*$$

$$= [{}^1/_2(k^2-\alpha^2)^{-1}]^*$$

$$= \mathbf{K}^*(\alpha)$$

Thus, the adjoint operator expression is evaluated using the complex conjugate of the array sampled in the spectral domain for evaluation of the operator expression.

It is further required to evaluate the inverse expression,

$$c(x) = \mathcal{f}^{-1}\{\mathbf{K}^{-1}\mathbf{p}\}$$

This is done numerically using the same procedure as for the operator expression except that the inverse of each of the values in the array obtained by sampling in the spectral domain is used.

The above method of constructing the discrete kernel in the spectral domain is similar to the method used in the spectral domain FFT (SDFFT) procedure outlined by Peterson and Mittra in Chapter 5 of [8, pp.119-122]. They construct a discrete, periodic "spectral domain Green's function" of the form [8, eqn.(5.24)]

$$\mathbf{G}_1(\alpha) = \mathbf{S}(\alpha)*[\mathbf{P}(\alpha)\mathbf{W}(\alpha)\mathbf{K}(\alpha)] \qquad (3.20)$$

where

$$\mathbf{S}(\alpha) = \alpha_s \sum_{m=-\infty}^{\infty} \delta(\alpha - m\alpha_s) \qquad (3.21)$$

with $\alpha_s = M\alpha_p = {}^1/_h$ and

$$P(\alpha) = \alpha_p \sum_{q=-\infty}^{\infty} \delta(\alpha - q\alpha_p) \qquad (3.22)$$

$\mathbf{W}(\alpha)$ is a windowing function used to truncate $\mathbf{K}(\alpha)$, in this case, to one period in the spectral domain (length α_s). Multiplication by $P(\alpha)$ results in a discrete function sampled at intervals α_p. Convolution with $\mathbf{S}(\alpha)$ produces a periodic function with period $M\alpha_p$.

Equation (3.20) can be inverse Fourier transformed to obtain the discrete spatial Green's function [8, eqn.(5.25)]

$$\begin{aligned} G_1(x) &= S(x)[P(x)*W(x)*K(x)] \\ &= P(x)*[S(x)\{W(x)*K(x)\}] \end{aligned} \qquad (3.23)$$

where

$$S(x) = \sum_{m=-\infty}^{\infty} \delta(x - mh) \qquad (3.24)$$

and

$$P(x) = \sum_{q=-\infty}^{\infty} \delta(x-qMh) \qquad (3.25)$$

If $G_1(x)$ is now windowed by a function $U(x)$ to ensure that the last N values are zero, i.e.

$$U(x_n) = \begin{array}{ll} 1 & 0 \leq x_n \leq (M-N)h \\ 0 & M-N+1 \leq x_n \leq Mh \end{array}$$

we obtain

$$\begin{aligned} G_1(x) &= U(x)S(x)[P(x)*W(x)*K(x)] \\ &= U(x)P(x)*[S(x)\{W(x)*K(x)\}] \end{aligned} \qquad (3.26)$$

Equation (3.26) is then Fourier transformed to obtain [8, eqn.(5.30)]

$$\begin{aligned} \mathbf{G}_1(\alpha) &= \mathbf{U}(\alpha)*S(\alpha)*[P(\alpha)\mathbf{W}(\alpha)\mathbf{K}(\alpha)] \\ &= S(\alpha)*[\mathbf{U}(\alpha)*P(\alpha)\mathbf{W}(\alpha)\mathbf{K}(\alpha)] \end{aligned} \qquad (3.27)$$

The discrete kernel in the spectral domain generated here is the same as the "spectral Green's function" of the SDFFT procedure with specific choices of $\mathbf{W}(\alpha)$ and $U(x)$.

Peterson and Mittra show that the SDFFT is equivalent to the discrete-convolution Method of Moments (DCMoM) procedure if the basis and testing functions of the latter procedure are suitably chosen [8, chapter 5]. As the MoM procedure involves setting up and solving a matrix equation the SDFFT, and thus the methods used in this thesis, can be given an equivalent matrix interpretation.

4. Numerical Results

The algorithms of figures 2.1 to 2.3 were implemented numerically using Turbo Pascal version 4 on IBM-PC compatibles. The 8087 numeric coprocessor was used with the real type, double. In Turbo Pascal the double type has a precision of 15 to 16 decimal digits and a range of 5×10^{-324} to 1.7×10^{308}.

The results pertain to the problem of scattering by the perfectly conducting strip. The data includes an investigation of the Hankel function obtained through sampling in spectral domains, error convergence of the various techniques for various strip widths, the current obtained by the techniques, the stability of the solution as a function of strip discretization and FFT size and stability of solution as a function of the loss factor.

4.1 Generation of Hankel Function by Spectral Domain Sampling

The Hankel function was generated by sampling the Fourier transform

$$f\{H_0^{(1)}(k|x|)\} = 2(k^2-\alpha^2)^{-\frac{1}{2}}$$

in the spectral domain according to the method of section 3.4. The inverse FFT was carried out to obtain the spatial domain values. These values can then be compared with values obtained from a computer program for calculating Bessel functions. The results for $k_0=1$, $a=1$, $N=16$ and $M=1024$ and $k=k_0(1+0.01j)$ are shown in figure 4.1. The real component is plotted in 4.1(a) and the imaginary in 4.1(b). The difference between the real component calculated by spectral domain sampling and by direct calculation is plotted in 4.1(c). The difference for the imaginary component is plotted in 4.1(d). The direct calculation was done using Pascal procedures for calculating Bessel

functions with complex arguments [10]. In terms of Bessel functions

$$H_0^{(1)}(x) = J_0(x) + jY_0(x)$$

where J_0 is the Bessel function of the first kind of order zero and Y_0 is the Bessel function of the second kind of order zero.

At x=0 there is a relatively large difference, especially for $Im\{H_0^{(1)}(0)\}$ as this should be minus infinity. Otherwise the difference is small in the region of the strip but increases far from the strip.

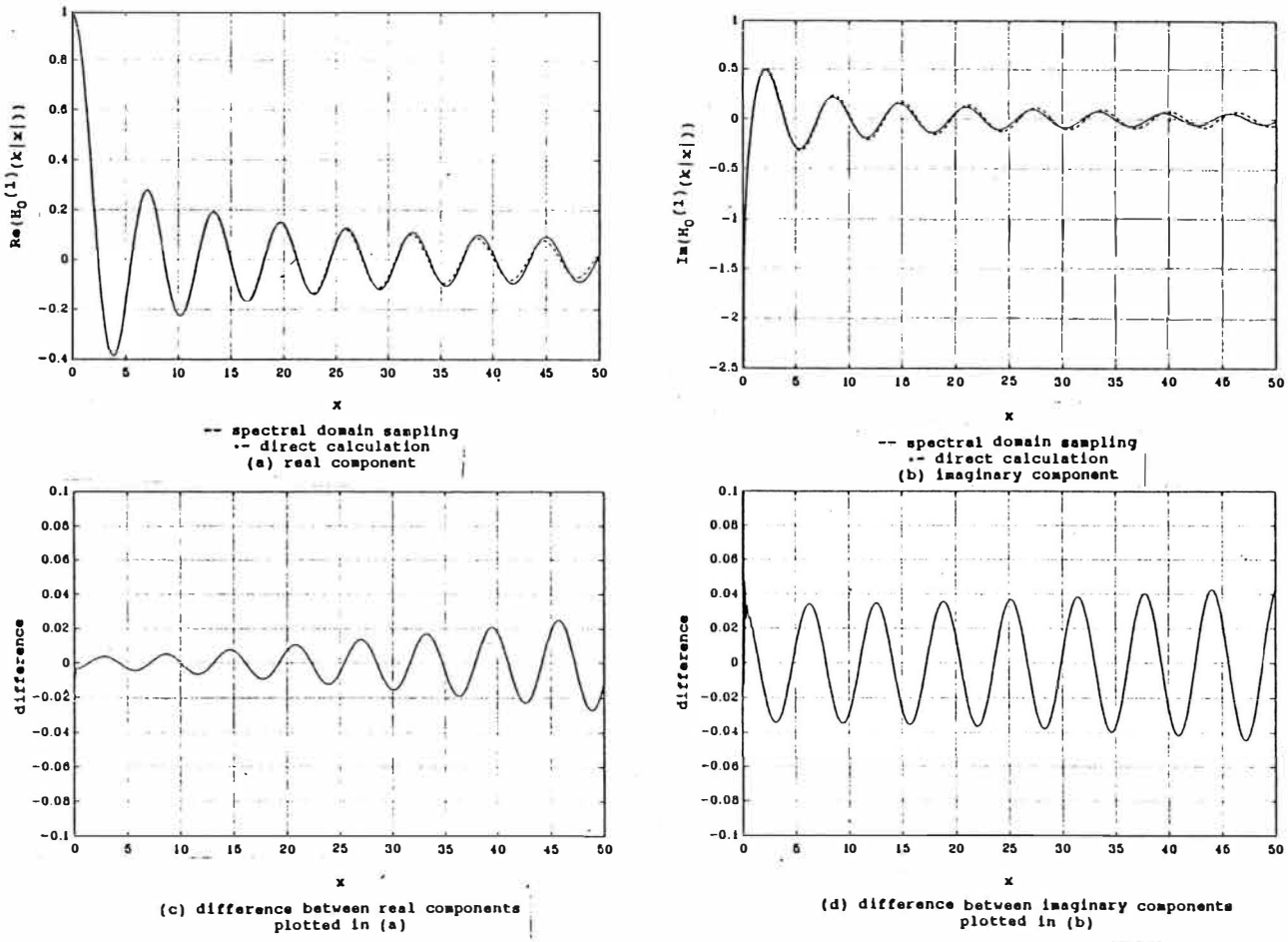


Figure 4.1 : Comparison of Hankel function obtained via spectral domain sampling with direct results (k_0=1, a=1, N=16, M=1024 and $k=k_0[1+j0.01]$).

4.2 Error Convergence

In order to compare results with those in references [1] and [2], the error convergence was investigated for the cases $k_0a = 10$ (figure 4.2), $k_0a = 1$ (figure 4.3) and $k_0a = 0.1$ (figure 4.4). The error presented is the normalized RMS error defined as

$$\overline{ERR} = \frac{ERR^{(n)}}{\|g\|}$$

where $ERR^{(n)}$ is as defined in equation (2.12) and g is the source related function of equation (1.1).

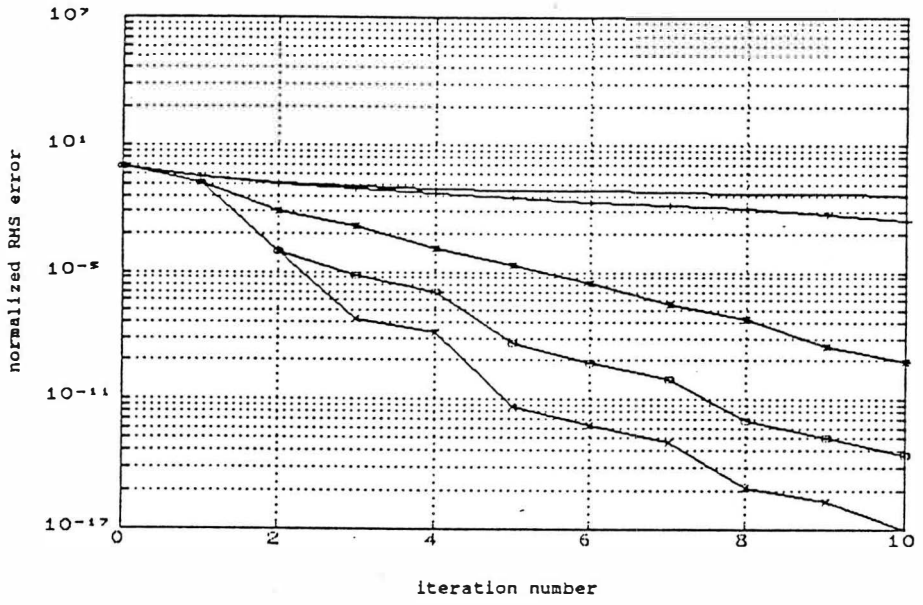
The values of k_0a correspond to strip widths of about 3.18, 0.318 and 0.0318 wavelengths. The number of sample points at the strip amounts to 41, 16 and 6, when $k_0a = 10$, 1 and 0.1 respectively. A 1024-point FFT was used. A complex wavenumber, $k = k_0(1+0.01j)$ was used to avoid the branch point in the Fourier transform of the kernel function. The numerical convergence of the gradient method (GR) and the contrast source truncation technique (CST) are considered. The number following the abbreviations GR and CST indicate the number of variational parameters used.

In figures 4.2(a) ,4.3(a) and 4.4(a) the initial estimate is taken as $f^{(0)} = 0$ while in 4.2(b), 4.3(b) and 4.4(b) the initial estimate is the physical optics (Kirchhoff) approximation with the minimization presented in section 2.4.

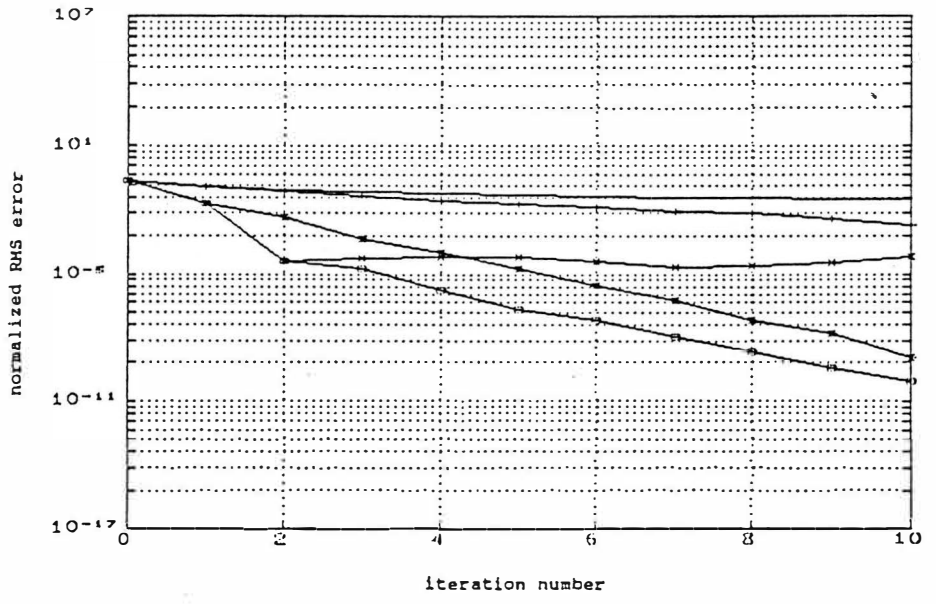
From these figures it can be observed that the choice of a non-zero initial estimate shows hardly any influence on the rate of convergence for the various schemes except for the CST3. It appears to cause numerical instabilities in CST3 which cause the scheme to diverge. An increase in the number of variational parameters for a particular method (GR

or CST) shows a marked improvement in the convergence rate. In all cases the initial convergence of the CST3 is the fastest, however once the error reaches very small values (about 10^{-16}) the scheme tends to diverge. Thus, when this scheme is implemented one must ensure that it is terminated before divergences occur. An error of 10^{-16} is extremely small and the scheme would normally be terminated at a larger error such as 10^{-5}

For smaller strip widths the GR2 scheme shows good convergence, almost equaling that of the CST3. However, the CST3 scheme with zero initial guess is the optimum choice for all k_0a. The current magnitude over the strip after 10 iterations, achieved by the scheme with the smallest error, is shown in figures 4.5, 4.6 and 4.7 respectively. The error in these cases was less than 10^{-17}, 10^{-10} and 10^{-15} respectively. The physical optics approximation is included in these figures for comparison purposes. It is seen that as the strip increases in width the current at the centre approaches the physical optics approximation.

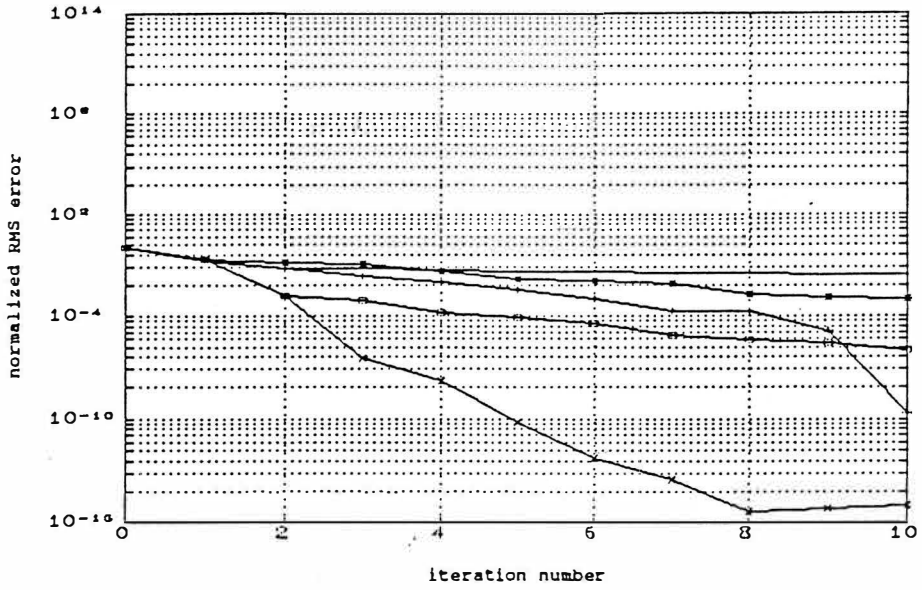


\- GR1 + GR2 * CST1 o CST2 x CST3
(a) Zero initial estimate.

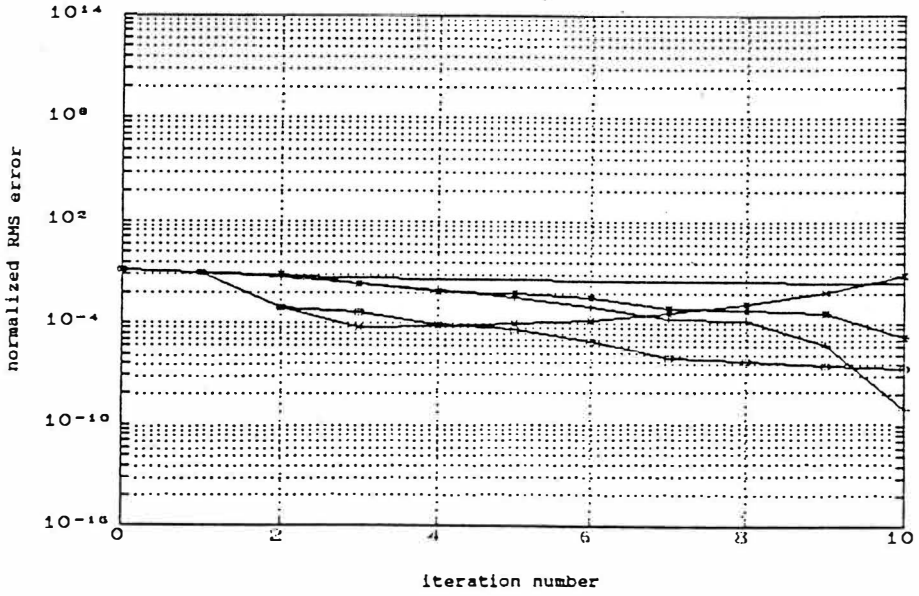


\- GR1 + GR2 * CST1 o CST2 x CST3
(b) Kirchhoff approximation with initial minimization.

Figure 4.2 : The normalized RMS error as a function of the number of iterations (k_0a=10, N=41, M=1024).

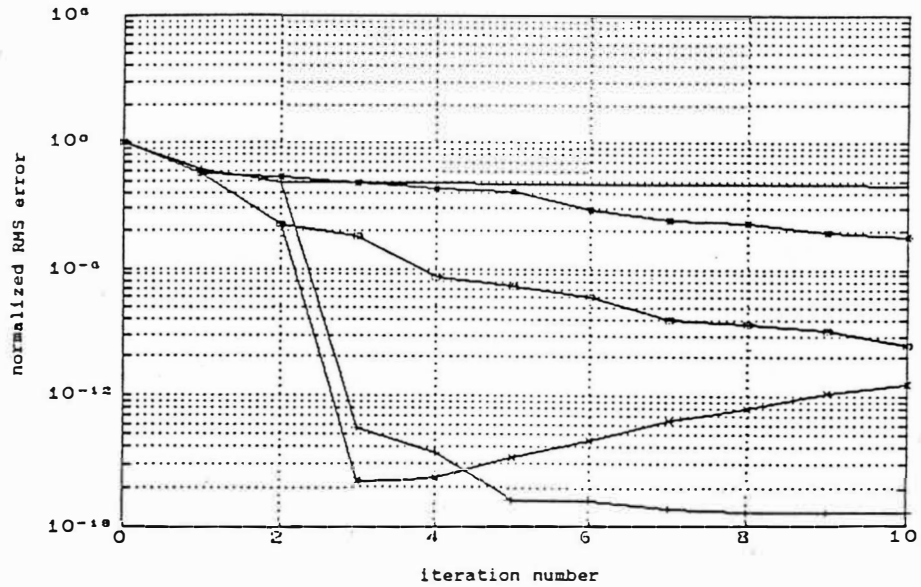


\- GR1 + GR2 * CST1 o CST2 x CST3
(a) Zero initial estimate.



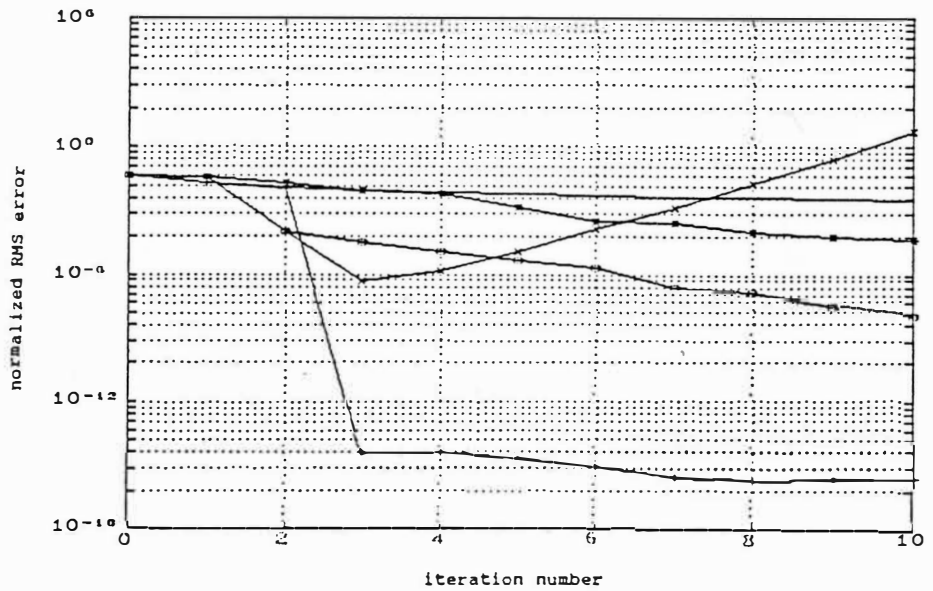
\- GR1 + GR2 * CST1 o CST2 x CST3
(b) Kirchhoff approximation with initial minimization.

Figure 4.3 : The normalized RMS error as a function of the number of iterations (k_0a=1, N=16, M=1024).



\- GR1 + GR2 * CST1 o CST2 x CST3

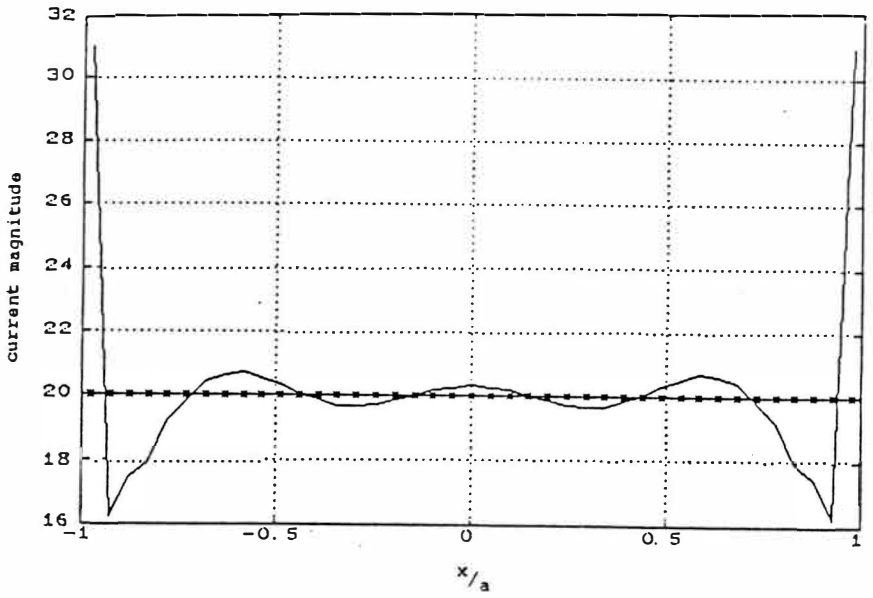
(a) Zero initial estimate.



\- GR1 + GR2 * CST1 o CST2 x CST3

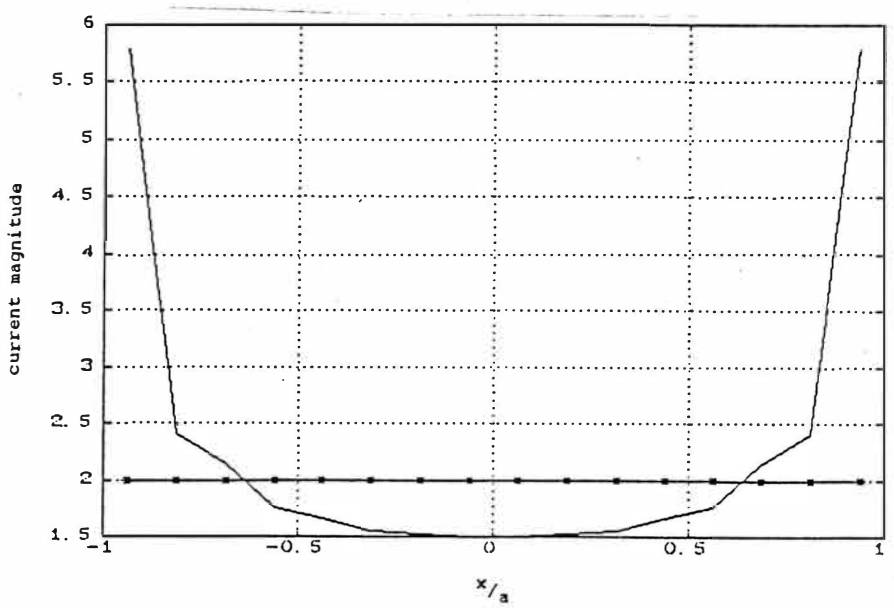
(b) Kirchhoff approximation with initial minimization.

Figure 4.4 : The normalized RMS error as a function of the number of iterations (k_0a=0.1, N=6, M=1024).



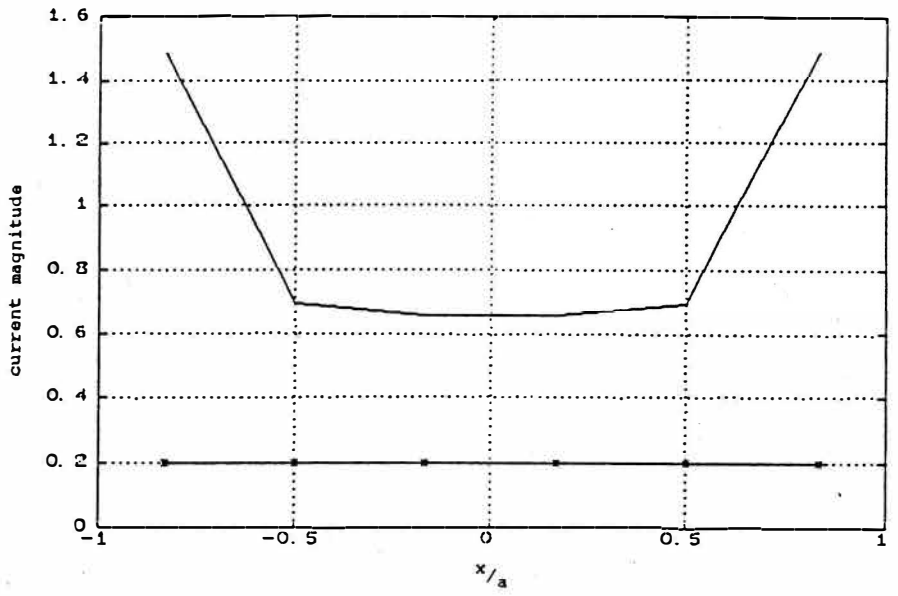
* Kirchhoff approximation

Figure 4.5 : Magnitude of equivalent surface current density distribution normalized to $(kZ_0)^{-1}$ with $k_0a=10$, N=41, M=1024.



* Kirchhoff approximation

Figure 4.6 : Magnitude of equivalent surface current density distribution normalized to $(kZ_0)^{-1}$ with $k_0a=1$, N=16, M=1024.



* Kirchhoff approximation

Figure 4.7 : Magnitude of equivalent surface current density distribution normalized to $(kZ_0)^{-1}$ with $k_0a=0.1$, N=6, M=1024.

4.3 Solution Stability

The stability of the solution as a function of the number of sample points N on the strip and the FFT size, M, was investigated. For various values of M the current at the centre of the strip was determined using CST3 as a function of N. The results for k_0a = 10, 1 and 0.1 are shown in figures 4.8, 4.9 and 4.10 respectively.

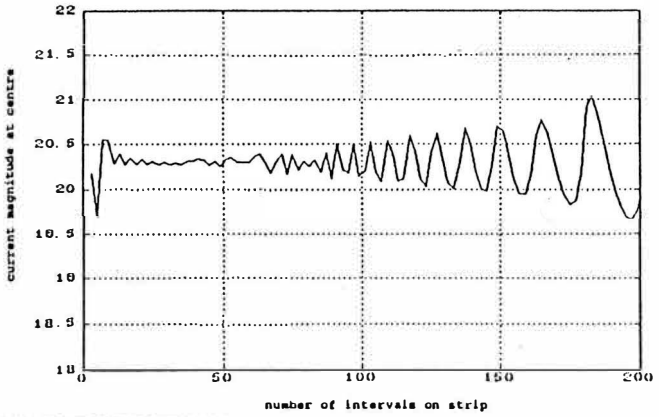
At each value of N the scheme was terminated once the normalized error was less than 10^{-6}. For k_0a = 10 it is seen that even for smaller values of M there is a reasonably large range of N for which the current converges to a constant value. For M = 512 this range is N = 20 to 60. As N increases however, the physical distance between the fictitious copies, discussed in section 3.4, decreases due to decreasing h which leads to an increase in coupling between them. Thus for values of N larger than 60 in the M=512 case the solution becomes unstable.

For smaller strip widths the interval size h is smaller for a given value of N, thus the distance, (M-N)h between fictitious copies is smaller. Thus the maximum N for stable solution decreases.

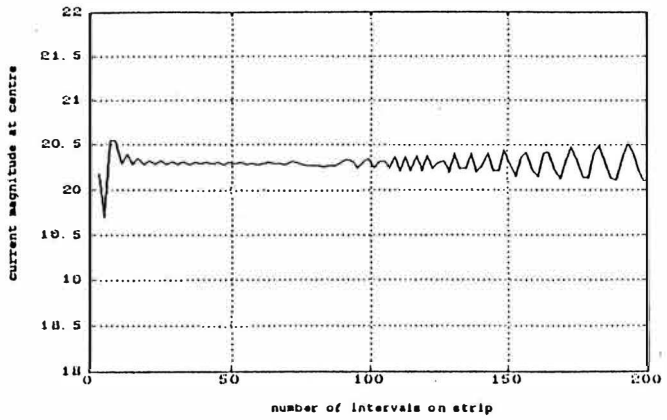
It is seen from figures 4.8 (c) and 4.9(c) that the decrease in this maximum N is roughly a factor 10 for a factor 10 decrease in width.

For k_0a = 0.1, shown in figure 4.10, even for large M(4096) the distance, (M-N)h between copies is small. Thus a stable region is barely perceptible.

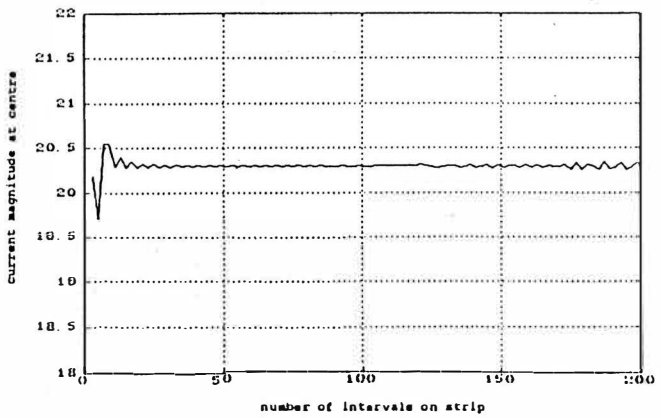
For smaller values of N the solution is unstable in all cases. This is probably due to insufficient number of matching points on the strip.



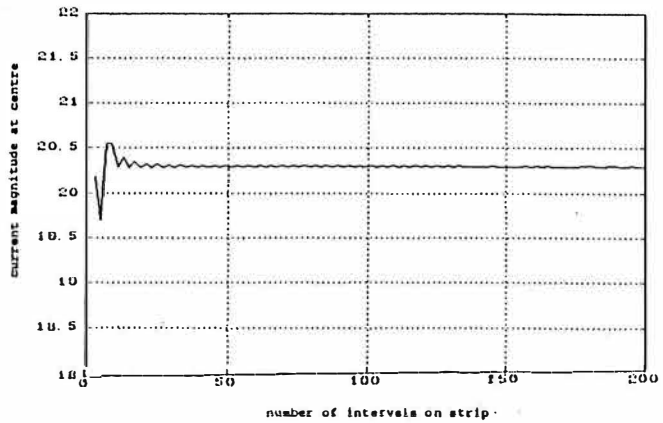
(a) M = 512



(b) M = 1024

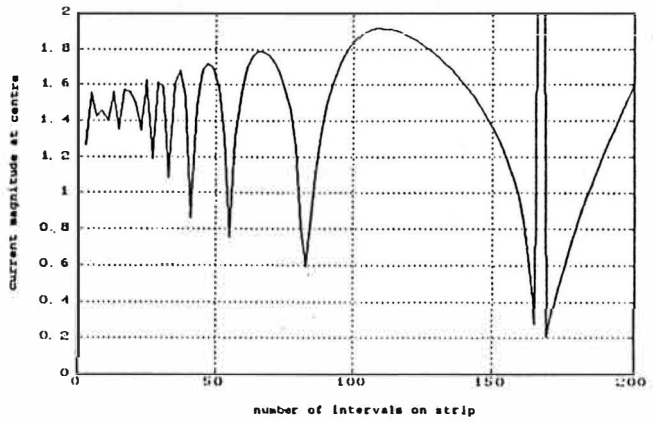


(c) M = 2048

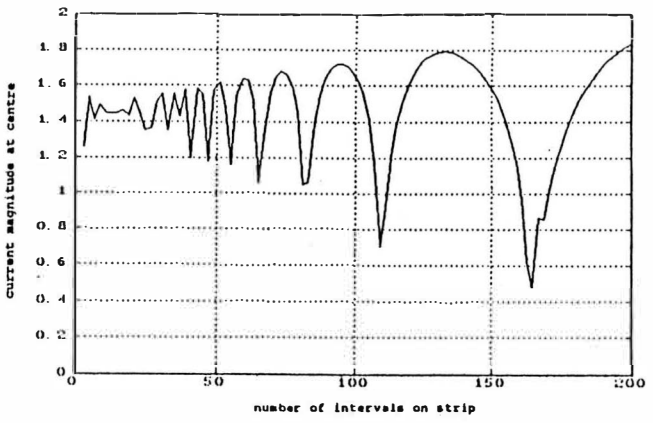


(d) M = 4096

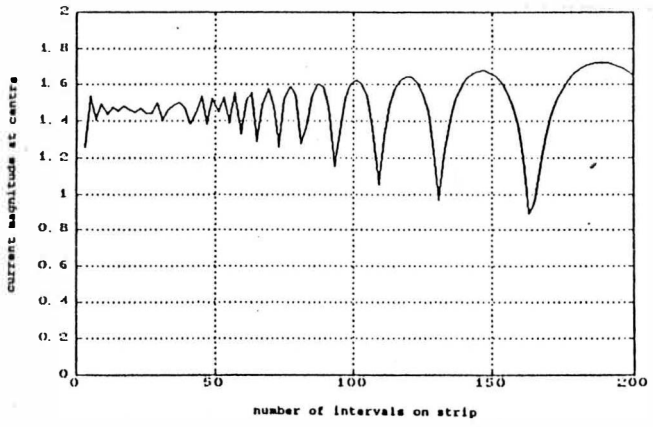
Figure 4.8 : Magnitude of current at centre of strip as a function of the number of intervals taken over the strip ($k_0a=10$, $l_f=0.01$).



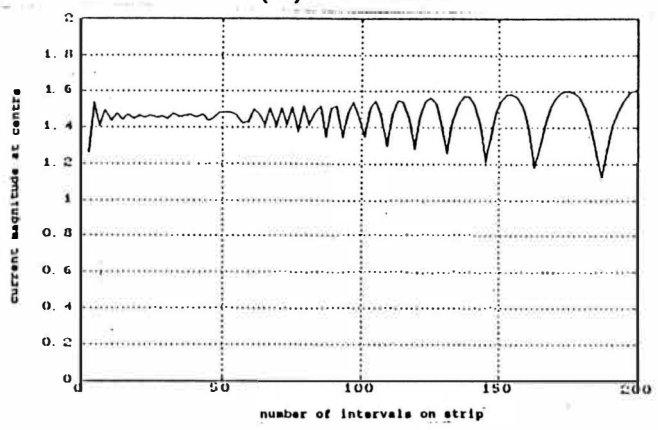
(a) M = 512



(b) M = 1024

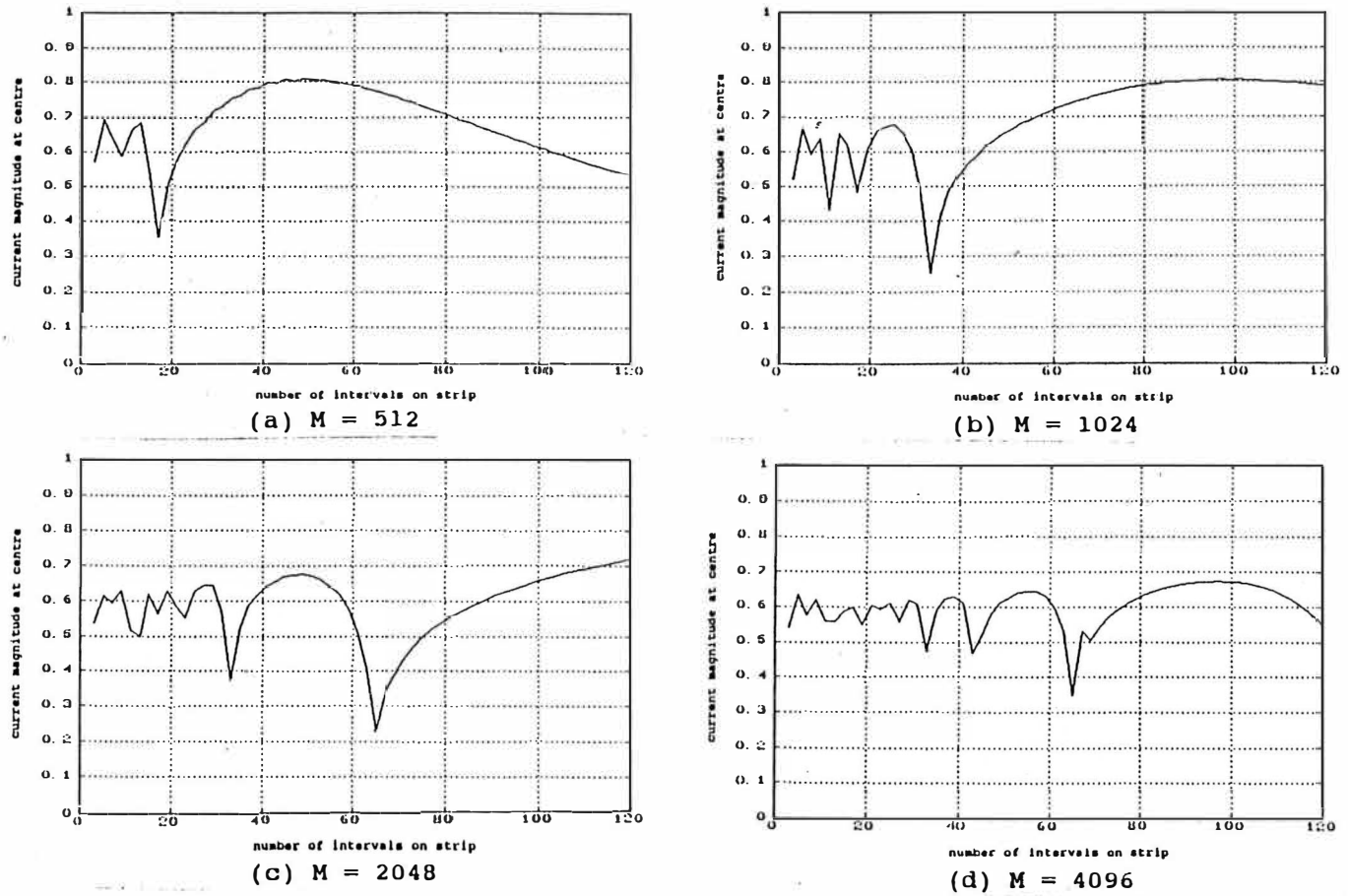


(c) M = 2048



(d) M = 4096

Figure 4.9 : Magnitude of current at centre of strip as a function of the number of intervals taken over the strip ($k_0a=1$, $l_f=0.01$).



(a) M = 512

(b) M = 1024

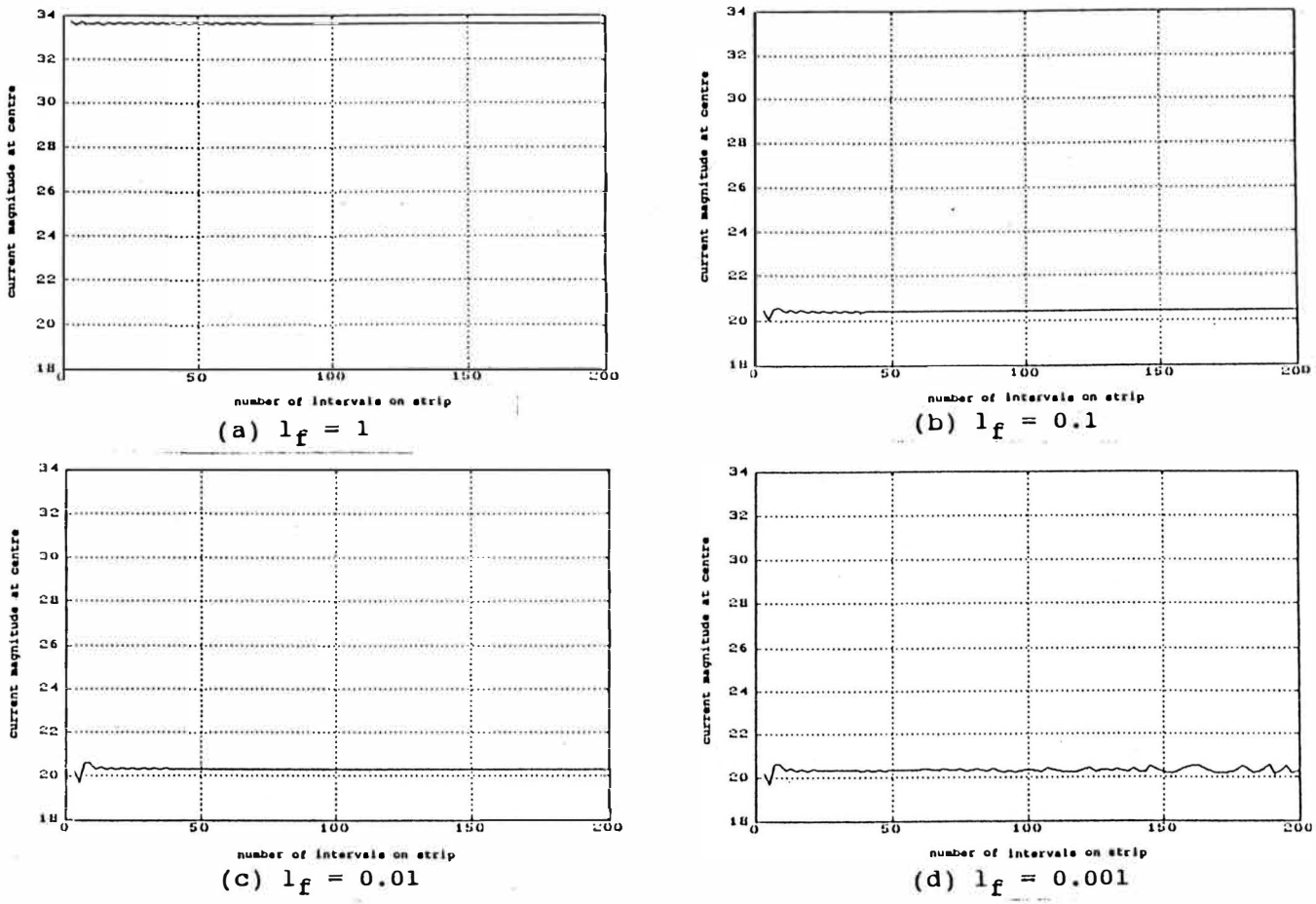
(c) M = 2048

(d) M = 4096

Figure 4.10 : Magnitude of current at centre of strip as a function of the number of intervals taken over the strip ($k_0a=0.1$, $l_f=0.01$).

4.4 The Effect of the Loss Factor

Figure 4.11 shows the effect on solution stability of varying the loss factor l_f of equation (3.13) for **M** = 4096 and k_0a = 10. A higher loss appears to improve the stability. However it is far removed from the physical situation. It is uncertain whether the problem being solved with a high loss factor is a satisfactory approximation to the original one.



(a) $l_f = 1$

(b) $l_f = 0.1$

(c) $l_f = 0.01$

(d) $l_f = 0.001$

Figure 4.11 : Magnitude of current at centre of strip as a function of the number of intervals taken over the strip (k_0a=10, M=4096).

5. Conclusion

Iterative schemes for the solution of scattering by impenetrable objects have been presented. These schemes are based on the minimization of the RMS error. The operator equation arising from the integral equation formulation of the scattering problem involves a Fredholm equation of type one. For some problems the kernel of the integral equation is of one dimensional convolution type. These integrals can be evaluated efficiently using the FFT. The singularity in the kernel function in the spatial domain can be avoided by sampling its Fourier transform in the spectral domain when forming its DFT.

The schemes were applied to the problem of scattering by a perfectly conducting strip. The kernel for this problem is not limited to the domain of observation, thus use of the FFT, which assumes periodicity, results in aliasing. The convergence of the schemes was investigated and of those studied it was found that the CST technique had the fastest convergence when used with three variational parameters. The convergence of the GR technique with two variational parameters approached that of the CST3 for the narrow strip.

The stability of the solution as a function of the number of intervals over the strip and the size of the FFT was studied using CST3. When the distance (M-N)h between fictitious copies of the strip is small coupling between them (aliasing) causes instabilities in the solution.

Thus when these techniques are implemented one must ensure that the choice of N and M produce a solution in a stable region where aliasing is small. For the case of $k_0a=0.1$ the value of M=4096 did not prove large enough to produce a reasonably large stable region.

An increase in the loss improves the stability of the solution. However, it is uncertain whether the original

problem is being satisfactorily approximated by the problem with a high loss factor.

References

[1] P.M. van den Berg, "Iterative computational techniques in scattering based upon the integrated square error criterion," IEEE Trans. Antennas Propagat., vol. AP-32, pp.1063-1071, October 1984.

[2] P.M. van den Berg, "Iterative schemes based on the minimization of the error in field problems," Electromagn., vol. 5, pp.237-262, 1985

[3] W.L. Ko and R. Mittra, "A new approach based on a combination of integral equation and asymptotic techniques for solving electromagnetic scattering problems," IEEE Trans. Antennas Propagat., vol. AP-25, pp.187-197, March 1977.

[4] R.F. Harrington, Time-Harmonic Electromagnetic Fields, McGraw-Hill, New York, 1968.

[5] A. Erdélyi (Ed.), W. Magus, F. Oberhettinger and F.G. Tricomi (Research Associates), Tables of Integral Transforms, vol.1, McGraw-Hill, New York, 1954.

[6] W.H Press, B.P. Flannery, S.A. Teukolsky and W.T. Vetterling, Numerical Recipes, Cambridge University Press, New York, 1987.

[7] A.V. Oppenheim and R.W. Schafer, Digital Signal Processing, Prentice Hall, Englewood Cliffs, New Jersey, 1975.

[8] A.F. Peterson and R. Mittra, "On the implementation of iterative methods for computational electromagnetics," Electromagnetics Lab. Tech. Rep. 85-9, Department of Electrical and Computer Engineering, University of Illinois, Urban, Illinois, December 1985.

[9] P.M. van den Berg and R.E. Kleinman, "The conjugate gradient spectral iterative technique for planar structures," IEEE Trans. Antennas Propagat., vol. AP-36, pp.1418-1423, October 1988.

[10] C.F. du Toit, "The numerical computation of Bessel functions of the first and second kind for integer orders and complex arguments," to appear in IEEE Trans. Antennas Propagat.

Appendix A: Integral Equation Formulation of Electromagnetic Scattering by a Perfectly Conducting Strip

The equivalent surface current $\underline{J}_s(x)$ on the strip as illustrated in figure 3.1 will be entirely z-directed.

Thus,

$$\underline{J}_s(x) = J_{sz}(x)\hat{u}_z \tag{A.1}$$

The electric field produced by this current, ie. the scattered field of the problem is now derived. To begin we look at a infinitely long z-directed current filament shown in figure A-1. From symmetry the fields should be independent of z. We thus study the problem in the xy-plane.

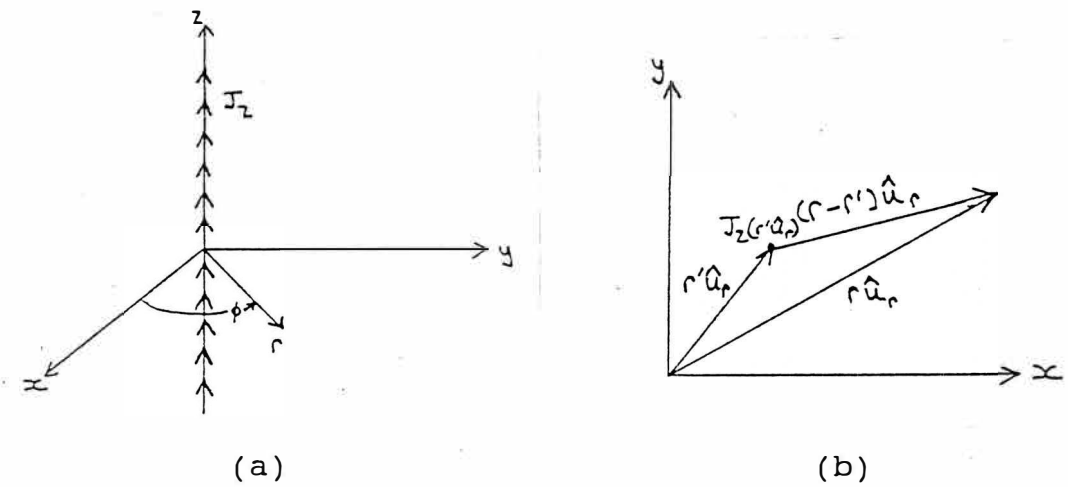


Figure A-1 : An infinite filament of constant a-c current (a) along the z axis and (b) displaced parallel to the z axis [after 4].

The source point is at $r'\hat{u}_r$ and the observation point at $r\hat{u}_r$. The magnetic vector potential produced at $r\hat{u}_r$ by the filament at $r'\hat{u}_r$ is [4, p.225]

$$A_z(r\hat{u}_r) = \frac{J_z(r'\hat{u}_r)}{4j} H_0^{(1)}(k|r\hat{u}_r - r'\hat{u}_r|) \tag{A.2}$$

where $H_0^{(1)}(x)$ is the Hankel function type 1 order 0. A time dependence of e^{-jwt} is assumed thus $H_0^{(1)}$ represents outward travelling waves.

The resultant electric field is

$$E_z = {}^1/_{\ddot{y}} \, (\partial^2/\partial z^2 + k^2) A_z$$

$$= \frac{k^2}{\ddot{y}} \, \frac{J_z(r'\hat{u}_r)}{4j} \, H_0^{(1)}(k|r\hat{u}_r - r'\hat{u}_r|) \qquad (A.3)$$

where $\ddot{y}$ is the admittivity of the medium.

For a dielectric medium, $\ddot{y} = jw\ddot{e}$ where $\ddot{e}$ is the complex permittivity of the medium [4, p.19].

For the strip problem the current filaments only lie in the y=0 plane. The source point is then

$$r'\hat{u}_r = x'\hat{u}_x \qquad -a \leq 'x \leq a.$$

We wish to calculate the scattered component of the tangential electric field on the strip, thus the observation point becomes

$$r\hat{u}_r = x\hat{u}_x \qquad -a \leq x \leq a.$$

Then equation (A3) becomes

$$E_z = \frac{k^2}{4j\ddot{y}} \, J_z(x') H_0^{(1)}(k|x-x'|) \qquad (A.4)$$

The tangential scattered field on the strip surface is then

$$E_z^s(x) = \int_{-a}^{a} \frac{k^2}{4j\ddot{y}} \, J_z(x') H_0^{(1)}(k|x-x'|) dx', \quad -a \leq x \leq a \qquad (A.5)$$

On the strip surface the total tangential electric field must be zero, thus

$$E_z^i(x) = -E_z^s(x)$$

$$= -\int_{-a}^{a} \frac{k^2}{4j\ddot{y}} H_0^{(1)}(k|x-x'|)J_z(x')dx', \quad -a \leq x \leq a$$

But [4, p.48],

$$\ddot{y} = \frac{jk}{Z_0}$$

where Z_0 is the intrinsic impedance of the medium. Thus

$$\frac{k^2}{4j\ddot{y}} = \frac{Z_0 k^2}{-4k} = \frac{-Z_0 k}{4}$$

Therefore

$$E_z^i(x) = \int_{-a}^{a} \frac{Z_0 k}{4} H_0^{(1)}(k|x-x'|)J_z(x')dx', \quad -a \leq x \leq a \qquad (A.6)$$

Appendix B: Program Listing

The program listed here was used to determine the rate of convergence for the various schemes. Data generated by the program was used to create figures 4.2 to 4.7.

```
program iterative_techniques(input,output);

{$m 32768,0,655360}
{$n+}

{solution of scattering by perfectly conducting strip}
{by iterative minimization of rms error               }
{methods implemented:GR1,GR2 = gradient               }
{    CST1,CST2,CST3 = contrast source truncation      }

  uses
    dos;

  type
    real = double;

  const
    {TNNearlyZero = 1e-38;}   {single}
    TNNearlyZero = 1e-308;    {double}
    {TNNearlyZero = 1e-4932;} {extended}

  const
    re = 0;
    im = 1;
    mag = 0;
    arg = 1;
    maxi = 4096;
    mini = 205;
    veclen = 4;

  type
    complex = array [re..im] of real;
    polar = array [mag..arg] of real;
    scarray  = array [1..mini] of complex;
    lrarray = array [1..maxi] of real;
    srarray  = array [1..mini] of real;
    vector = array [1..veclen] of complex;
    matrix = array [1..veclen] of vector;
    cmplxptr = ^complex;
    plrptr = ^polar;
    scarrayptr = ^scarray;
    lrarrayptr = ^lrarray;
    filename = string[8];
```

```
const
  zero:complex = (0,0);
  one:complex  = (1,0);
  two:complex  = (2,0);
  mone:complex = (-1,0);

function magnitude(z:complex;
             square:boolean):real;

  {calculates magnitude of complex number}
  {if square is true the result is the magnitude squared}

  var
    r:real;

  begin  {function magnitude}
    r:=sqr(z[re])+sqr(z[im]);
    if not square then r:=sqrt(r);
    magnitude:=r
  end;   {function magnitude}

procedure cneg(z:complex;
          var mz:complex);

  begin
    mz[re]:=-z[re];
    mz[im]:=-z[im]
  end;

procedure cmult(z1,z2:complex;
               var zp:complex);

  {calculates the product of two complex numbers}

  begin  {procedure cmult}
    zp[re]:=z1[re]*z2[re]-z1[im]*z2[im];
    zp[im]:=z1[re]*z2[im]+z1[im]*z2[re]
  end;   {procedure cmult}

procedure cadd(z1,z2:complex;
             var zs:complex);

  {calculates sum of two complex numbers}

  begin  {procedure cadd}
    zs[re]:=z1[re]+z2[re];
    zs[im]:=z1[im]+z2[im]
  end;   {procedure cadd}

procedure csub(z1,z2:complex;
             var zd:complex);

  {calculates difference between two complex numbers}
```

```
  begin  {procedure csub}
    zd[re]:=z1[re]-z2[re];
    zd[im]:=z1[im]-z2[im]
  end;   {procedure csub}

procedure cdiv(zn,zd:complex;
               var zq:complex);

  {calculates the ratio of two complex numbers}

  var
    f:real;

  begin  {procedure cdiv}
    f:=sqr(zd[re])+sqr(zd[im]);
    zq[re]:=(zn[re]*zd[re]+zn[im]*zd[im])/f;
    zq[im]:=(zn[im]*zd[re]-zn[re]*zd[im])/f
  end;   {procedure cdiv}

procedure conj(z:complex;
           var zc:complex);

  {calculates conjugate of a complex number}

  begin  {procedure conj}
    zc[re]:=z[re];
    zc[im]:=-z[im]
  end;   {procedure conj}

function sumint(lolim:real;  {lower integration limit}
                uplim:real;  {upper integration limit}
                number:integer;  {number of values}
                y:srarray):real; {function values}

  {numerical integration using area summation }
  {NB : number of values = number of intervals}
  {ie, samples lie at centre of corresponding interval}

  var
    ans:real;
    i:integer;

  begin
    ans:=0;
    for i:=1 to number do
      ans:=ans+y[i];
    ans:=ans*(uplim-lolim)/number;
    sumint:=ans
  end;

procedure comint(lolim:real;  {lower integration limit}
                 uplim:real;  {upper integration limit}
                 number:integer;  {number of values}
                 z:scarray;  {complex function values}
```

```
            var za:complex);  {complex result}

  {numerical integration of a complex function }
  {calculates the real and imaginary parts seperatly}
  {using sumint}

  var
    x,y:srarray;
    i:integer;

  begin
    for i:=1 to number do
      begin
        x[i]:=z[i,re];
        y[i]:=z[i,im]
      end;
    za[re]:=sumint(lolim,uplim,number,x);
    za[im]:=sumint(lolim,uplim,number,y)
  end;

procedure FFT(NumPoints:integer;
                datar:lrarrayptr;
                datai:lrarrayptr);

  var
    ii,jj,n,mmax,i,j,k        : integer;
    wtemp,wr,wpr,wpi,wi,theta : real;
    tempr,tempi               : real;

  begin  {procedure FFT}
    n := NumPoints div 2;
    j := 1;
    for ii := 2 to NumPoints-1 do
      begin
        k := NumPoints div 2;
        while j > k do
          begin
            j := j-k;
            k := k div 2;
          end;
        j := j+k;
        if j > ii then
          begin
            tempr := datar^[j];
            tempi := datai^[j];
            datar^[j] := datar^[ii];
            datai^[j] := datai^[ii];
            datar^[ii] := tempr;
            datai^[ii] := tempi
          end;
      end;
    mmax := 1;
    while (n >= mmax) do
      begin
```

```
        theta := -pi/mmax;
        wpr := -2*sqr(sin(0.5*theta));
        wpi := sin(theta);
        wr := 1;
        wi := 0;
        for ii := 1 to mmax do
          begin
            for jj := 0 to ((n-ii) div mmax) do
              begin
                i := ii+jj*mmax*2;
                j := i+mmax;
                tempr := wr*datar^[j] - wi*datai^[j];
                tempi := wr*datai^[j] + wi*datar^[j];
                datar^[j] := datar^[i] - tempr;
                datai^[j] := datai^[i] - tempi;
                datar^[i] := datar^[i] + tempr;
                datai^[i] := datai^[i] + tempi;
              end;
            wtemp := wr;
            wr := wr*wpr - wi*wpi + wr;
            wi := wi*wpr + wtemp*wpi + wi;
          end;
        mmax := 2*mmax;
      end;
  end;   {procedure FFT}

procedure IFFT(NumPoints:integer;
                    datar:lrarrayptr;
                    datai:lrarrayptr);

  var
    ii,jj,n,mmax,i,j,k        : integer;
    wtemp,wr,wpr,wpi,wi,theta : real;
    tempr,tempi               : real;
    mult                      : real;

  begin  {procedure IFFT}
    n := NumPoints div 2;
    j := 1;
    for ii := 2 to NumPoints-1 do
      begin
        k := NumPoints div 2;
        while j > k do
          begin
            j := j-k;
            k := k div 2;
          end;
        j := j+k;
        if j > ii then
          begin
            tempr := datar^[j];
            tempi := datai^[j];
            datar^[j] := datar^[ii];
            datai^[j] := datai^[ii];
```

```
            datar^[ii] := tempr;
            datai^[ii] := tempi
          end;
      end;
    mmax := 1;
    while (n >= mmax) do
      begin
        theta := pi/mmax;
        wpr := -2*sqr(sin(0.5*theta));
        wpi := sin(theta);
        wr := 1;
        wi := 0;
        for ii := 1 to mmax do
          begin
            for jj := 0 to ((n-ii) div mmax) do
              begin
                i := ii+jj*mmax*2;
                j := i+mmax;
                tempr := wr*datar^[j] - wi*datai^[j];
                tempi := wr*datai^[j] + wi*datar^[j];
                datar^[j] := datar^[i] - tempr;
                datai^[j] := datai^[i] - tempi;
                datar^[i] := datar^[i] + tempr;
                datai^[i] := datai^[i] + tempi;
              end;
            wtemp := wr;
            wr := wr*wpr - wi*wpi + wr;
            wi := wi*wpr + wtemp*wpi + wi;
          end;
        mmax := 2*mmax;
      end;
    mult:=1/NumPoints;
    for ii:=1 to NumPoints do
      begin
        datar^[ii]:=mult*datar^[ii];
        datai^[ii]:=mult*datai^[ii];
      end;
  end;   {procedure IFFT}

procedure Gaussian_Elimination(
          Dimen:integer;  {Dimension of the square matrix}
          Coefficients:matrix;  {Square matrix}
          Constants:vector;  {Constants of each equation}
      var Solution:vector;  {Unique solution to the set of
                             equations}
      var Error:integer);  {Flags if something goes wrong}

  {origin:Turbo Pascal Numerical Methods Toolbox}
  {(C) Copyright 1986 Borland International}
  {(version date:4 Sep 87)}
  {modified for complex coefficients}
  {Purpose : Calculate the solution of a linear set of}
  {          equations using Gaussian elimination and }
  {          backwards substitution.}
```

```
{-  User-defined Types:
{-              complex = array [re..im] of real;}
{-              vector  = array [1..veclen] of complex;}
{-              matrix  = array [1..veclen] of vector; }
{-              Errors:  0: No errors;}
{-                       1: Dimen < 1}
{-                       2: no solution exists}

var  {dummy vars}
  pig:complex;

procedure Initial(Dimen        : integer;
              var Coefficients : matrix;
              var Constants    : vector;
              var Solution     : vector;
              var Error        : integer);

  {- This procedure test for errorsin the value
      of Dimen.  This procedure also finds the
      solution for the trivial case Dimen = 1. -}

  begin
    Error := 0;
    if Dimen < 1 then
      Error := 1
    else
      if Dimen = 1 then
        if magnitude(Coefficients[1,1],false) <
                                        TNNearlyZero then
          Error := 2
        else
          cdiv(Constants[1],
               Coefficients[1,1],Solution[1])
  end; { procedure Initial }

procedure EROswitch(var Row1 : vector;
                    var Row2 : vector);

  {- elementary row operation - switching two rows -}

  var
    DummyRow : vector;

  begin
    DummyRow := Row1;
    Row1 := Row2;
    Row2 := DummyRow;
  end; { procedure EROswitch }

procedure EROmultAdd(Multiplier   : complex;
                     Dimen        : integer;
                 var ReferenceRow : vector;
                 var ChangingRow  : vector);
```

```
    {- row operation - adding a multiple of one row
                       to another -}

    var
      Term : integer;

    begin
      for Term := 1 to Dimen do
        begin
          cmult(Multiplier,ReferenceRow[Term],pig);
          cadd(ChangingRow[Term],pig,ChangingRow[Term])
        end
    end; { procedure EROmultAdd }

  procedure UpperTriangular(Dimen        : integer;
                        var Coefficients : matrix;
                        var Constants    : vector;
                        var Error        : integer);

    {- This procedure makes the coefficient matrix
       upper triangular.  The operations which
       perform this are also performed on the
       Constants vector.  If one of the main
       diagonal elements of the upper triangular
       matrix is zero, then the Coefficients
       matrix is singular and no solution exists
       (Error = 2 is returned).  -}

    var
      Multiplier        : complex;
      Row, ReferenceRow : integer;
      v:complex;

    procedure Pivot(Dimen        : integer;
                    ReferenceRow : integer;
                var Coefficients : matrix;
                var Constants    : vector;
                var Error        : integer);

      {- This procedure searches the ReferenceRow
         column of the Coefficients matrix for
         the first non-zero element below the
         diagonal. If it finds one, then the
         procedure switches rows so that the
         non-zero element is on the diagonal. -}
      {- It also switches the corresponding elements
         in the Constants vector. If it doesn't
         find one, the matrix is singular and no
         solution exists (Error = 2 is returned). -}

      var
        NewRow : integer;
        Dummy : complex;
```

```
  begin
    Error := 2;          { No solution exists }
    NewRow := ReferenceRow;
    while (Error > 0) and (NewRow < Dimen) do
      { Try to find a row with a non-zero }
      { diagonal element    }
      begin
        NewRow := Succ(NewRow);
        if magnitude(Coefficients[NewRow,
                    ReferenceRow],false) >
                                  TNNearlyZero then
          begin
            EROswitch(Coefficients[NewRow],
                      Coefficients[ReferenceRow]);
            { Switch these two rows }
            Dummy := Constants[NewRow];
            Constants[NewRow] :=
                                Constants[ReferenceRow];
            Constants[ReferenceRow] := Dummy;
            Error := 0;    { Solution may exist }
          end;
      end;
  end; { procedure Pivot }

begin { procedure UpperTriangular }
  ReferenceRow := 0;
  while (Error = 0) and (ReferenceRow < Dimen - 1) do
    begin
      ReferenceRow := Succ(ReferenceRow);
      { Check to see if the main diagonal element is
        zero }
      if magnitude(Coefficients[ReferenceRow,
            ReferenceRow],false) < TNNearlyZero then
        Pivot(Dimen, ReferenceRow, Coefficients,
            Constants, Error);
      if Error = 0 then
        for Row := ReferenceRow + 1 to Dimen do
          { Make the ReferenceRow element of this row
            zero }
          if magnitude(Coefficients[Row,
            ReferenceRow],false) > TNNearlyZero then
            begin
              cneg(Coefficients[Row,ReferenceRow],
                   pig);
              cdiv(pig,
              Coefficients[ReferenceRow,ReferenceRow],
              Multiplier);
              EROmultAdd(Multiplier, Dimen,
                         Coefficients[ReferenceRow],
                         Coefficients[Row]);
              cmult(Multiplier,
                    Constants[ReferenceRow],pig);
              cadd(Constants[Row],pig,Constants[Row])
```

```
                end;
          end; { while }
        if magnitude(Coefficients[Dimen, Dimen],false) <
                                          TNNearlyZero then
          Error := 2;    { No solution }
      end; { procedure UpperTriangular }

    procedure BackwardsSub(Dimen        : integer;
                       var Coefficients : matrix;
                       var Constants    : vector;
                       var Solution     : vector);

      {- This procedure applies backwards substitution
           to the upper triangular Coefficients matrix
           and Constants vector. The resulting vector
           is the solution to the set of equations and
           is returned in the vector Solution.          -}

      var
        Term, Row : integer;
        Sum : complex;

      begin
        Term := Dimen;
        while Term >= 1 do
          begin
            Sum:=zero;
            for Row := Term + 1 to Dimen do
              begin
                cmult(Coefficients[Term,Row],
                          Solution[Row],pig);
                cadd(Sum,pig,Sum)
              end;
            csub(Constants[Term],Sum,pig);
            cdiv(pig,Coefficients[Term, Term],
                          Solution[Term]);
            Term := Pred(Term)
          end;
      end; { procedure BackwardsSub }

    begin { procedure Gaussian_Elimination }
      Initial(Dimen, Coefficients, Constants,
                          Solution, Error);
      if Dimen > 1 then
        begin
          UpperTriangular(Dimen, Coefficients,
                                Constants, Error);
          if Error = 0 then
          BackwardsSub(Dimen, Coefficients,
                                Constants, Solution)
        end
    end; { procedure Gaussian_Elimination }

  const
```

```
    vlen=410;     {must be 2*mini}

  type
    matvec=array [1..vlen] of double;
    matmat=record
        typ:longint;
        nrows:longint;
        ncols:longint;
        imagf:longint;
        namlen:longint;
        name:array [1..9] of char;
        values:matvec
      end;
    matname=string[8];

  procedure setvec(n:integer;
                mname:matname;
                  vec:matvec;
              var mat:matmat);

    {This procedure creates a .mat column vector of real
values.}

    var
      i:integer;

    begin  {procedure setvec}
      with mat do
        begin
          for i:=1 to vlen do values[i]:=0;
          typ:=0;
          nrows:=n;
          ncols:=1;
          imagf:=0;
          namlen:=9;
          name[1]:=mname[1];
          name[2]:=mname[2];
          name[3]:=mname[3];
          name[4]:=mname[4];
          name[5]:=mname[5];
          name[6]:=mname[6];
          name[7]:=mname[7];
          name[8]:=mname[8];
          name[9]:=chr(0);
          for i:=1 to n do values[i]:=vec[i];
        end
    end;   {procedure setvec}

  procedure setmat(n:integer;
                mname:matname;
                    x:matvec;
                    y:matvec;
              var mat:matmat);
```

```
  {This procedure creates a .mat matrix consisting
    of two columns of real values.  Each column
    is input as a vector}

  var
    i:integer;

  begin  {procedure setmat}
    with mat do
      begin
        for i:=1 to vlen do values[i]:=0;
        typ:=0;
        nrows:=n;
        ncols:=2;
        imagf:=0;
        namlen:=9;
        name[1]:=mname[1];
        name[2]:=mname[2];
        name[3]:=mname[3];
        name[4]:=mname[4];
        name[5]:=mname[5];
        name[6]:=mname[6];
        name[7]:=mname[7];
        name[8]:=mname[8];
        name[9]:=chr(0);
        for i:=1 to nrows do values[i]:=x[i];
        for i:=1 to nrows do values[i+nrows]:=y[i]
      end
  end;   {procedure setmat}


procedure savemat(mname:matname;
                  mat:matmat);

  var
    file1:file of matmat;

  begin  {procedure savemat}
    assign(file1,mname+'.mat');
    rewrite(file1);
    write(file1,mat);
    close(file1)
  end;   {procedure savemat}

const
  a = 1;
  epsilon0 = 8.854e-12;

var  {global variables}
  intrin0:real;        {intrinsic impedance of free space}

  M:integer;           {no. of points used in FFT}
  N:integer;           {no. of intervals over strip}
```

```
  k0:real;              {free space wavenumber}
  kc:complex;           {complex wavenumber}
  h:real;               {interval length}

  fkernelr:lrarrayptr; {real part of Fourier
                          transform of Kernel}
  fkerneli:lrarrayptr; {imag part of Fourier
                          transform of Kernel}

  {fkernelc:lcarrayptr;} {Fourier transform of
                           conj. of Kernel}

procedure kernel_init;

  {*** sample Kernel in spectral domain ***}
  {*** zero padding in spatial domain ***}

  var
    i,j,k,l,mm:integer;
    alpha,dalpha,b,ba,xtot:real;
    kc2:complex;
    numre,numim:real;
    den:complex;
    dvar:complex;
    r2:real;
    aa,bb:real;

    F,G,sr,si:extended;
    zr:complex;

  function sqt_1(x:extended):extended;

    {this function written by C.F. du Toit}
    {  √(1+x)-1  for x<0.0578 lose 1.2 digits }

    var
      sc:real;

    begin
      sc:=(0.0109100341796875*x-
             0.013092041015625)*x+0.01611328125;
      sc:=(sc*x-0.0205078125)*x+0.02734375;
      sc:=(sc*x-0.0390625)*x+0.0625;
      sqt_1:=((sc*x-0.125)*x+0.5)*x
    end;

  begin  {procedure kernel_init}
    xtot:=M*h;
    dalpha:=2*pi/xtot;
    alpha:=0;
    mm:=M div 2;
    numre:=0.5;       {kc[re]/2;}
    numim:=0;         {kc[im]/2;}
    aa:=kc[re];  bb:=kc[im];
```

```
kc2[re]:=sqr(aa)+sqr(bb);
kc2[im]:=2*aa*bb;  {kc*kc}
for i:=1 to mm+1 do
  begin
    den[re]:=kc2[re]-sqr(alpha);
    den[im]:=kc2[im];

    {csqrt(den,den);}
    {calculate sqrt of den}

    sr:=Sqr(den[re]);
    si:=Sqr(den[im]);
    if sr*0.0578<=si then
      begin
        f:=(Sqrt(sr+si)+den[re])*0.5;
        if den[im]<0 then
          zr[im]:=-sqrt(F-den[re])
        else
          zr[im]:=sqrt(F-den[0]);
        zr[re]:=sqrt(F)
      end
    else
      begin
        F:=Abs(den[0])*0.5;
        G:=sqrt(sqt_1(si/sr)*F);
        zr[re]:=sqrt(sqrt(sr+si)*0.5+F);
        if den[re]<0 Then
          begin
            if den[im]<0 then zr[im]:=
                          -zr[re] else zr[im]:=zr[re];
            zr[re]:=G
          end
        else if den[im]<0 then zr[im]:=
                          -G Else zr[im]:=G;
      end;
    den:=zr;

    r2:=sqr(den[re])+sqr(den[im]);
    aa:=den[re];  bb:=den[im];
    fkernelr^[i]:=(numre*aa+numim*bb)/r2;
    fkerneli^[i]:=(numim*aa-numre*bb)/r2;
    alpha:=alpha+dalpha
  end;
j:=mm;
for i:=mm+2 to M do
  begin
    fkernelr^[i]:=fkernelr^[j];
    fkerneli^[i]:=fkerneli^[j];
    j:=j-1
  end;
ifft(M,fkernelr,fkerneli);
k:=mm+1;
l:=N div 2;
j:=mm;
```

```
    fkernelr^[k]:=0;
    fkerneli^[k]:=0;
    for i:=k+1 to k+l do
      begin
        fkernelr^[i]:=0;
        fkernelr^[j]:=0;
        fkerneli^[i]:=0;
        fkerneli^[j]:=0;
        j:=j-1;
      end;
    fft(M,fkernelr,fkerneli);

  end;   {procedure kernel_init}

function log10(x:real):real;

  {calculates the logarithm of x to the base 10}

  begin  {function log10}
    log10:=ln(x)/ln(10)
  end;   {function log10}

procedure inprod(limit:real; numsam:integer; f,g:scarray;
                 var zip:complex);

  {calculates the inner product of the functions f and g}

  var
    ii:integer;
    intfun:scarray;
    hog:complex;
    zr,zi,wr,wi:real;

  begin  {procedure inprod}
    for ii:=1 to numsam do
      begin
        zr:=f[ii,re]; zi:=-f[ii,im];    {conjugate}
        wr:=g[ii,re]; wi:=g[ii,im];
        intfun[ii,re]:=zr*wr-zi*wi;
        intfun[ii,im]:=zr*wi+zi*wr
      end;
    comint(-limit,limit,numsam,intfun,zip);
  end;   {procedure inprod}


function norm(limit:real;
              numsam:integer;
              f:scarray;
              square:boolean):real;

  {calculates norm of the function f}
  {if square is true the result is the norm squared}

  var
```

```
    ii:integer;
    magf2:srarray;
    r:real;

  begin  {function norm}
    for ii:=1 to numsam do magf2[ii]:=
                              sqr(f[ii,re])+sqr(f[ii,im]);
    r:=sumint(-limit,limit,numsam,magf2);
    if not square then r:=sqrt(r);
    norm:=r
  end;   {function norm}

procedure s_to_l(llength:integer;
                 slength:integer;
                   short:scarray;
                   longr:lrarrayptr;
                   longi:lrarrayptr);

  {*** zero padding of short array to obtain
       long array ***}

  var
    i:integer;

  begin  {procedure s_to_l}
    for i:=1 to slength do
      begin
        longr^[i]:=short[i,re];
        longi^[i]:=short[i,im]
      end;
    for i:=slength+1 to llength do
      begin
        longr^[i]:=0;
        longi^[i]:=0
      end
  end;   {procedure s_to_l}

procedure l_to_s(slength:integer;
                 longr:lrarrayptr;
                 longi:lrarrayptr;
                 var short:scarray);

  {*** obtain short array from long array ***}

  var
    i:integer;

  begin  {procedure l_to_s}
    for i:=1 to slength do
      begin
        short[i,re]:=longr^[i];
        short[i,im]:=longi^[i]
      end
  end;   {procedure l_to_s}
```

```
procedure conv(lenfft:integer;
               numsam:integer;
                    f:scarray;
                var g:scarray);

  var
    fftargr,fftargi:lrarrayptr;
    i:integer;
    aa,bb,cc,dd:real;

  begin  {procedure conv}
    new(fftargr);
    new(fftargi);
    s_to_l(lenfft,numsam,f,fftargr,fftargi);
    fft(lenfft,fftargr,fftargi);
    for i:=1 to lenfft do
      begin
        aa:=fftargr^[i];  bb:=fftargi^[i];
        cc:=fkernelr^[i]; dd:=fkerneli^[i];
        fftargr^[i]:=aa*cc-bb*dd;
        fftargi^[i]:=aa*dd+bb*cc
      end;
    ifft(lenfft,fftargr,fftargi);
    l_to_s(numsam,fftargr,fftargi,g);
    dispose(fftargr);
    dispose(fftargi);
    fftargr:=nil;
    fftargi:=nil
  end;   {procedure conv}

procedure convc(lenfft:integer;
                numsam:integer;
                     f:scarray;
                 var g:scarray);

  var
    i:integer;
    fftargr,fftargi:lrarrayptr;
    aa,bb,cc,dd:real;

  begin  {procedure convc}
    new(fftargr);
    new(fftargi);
    s_to_l(lenfft,numsam,f,fftargr,fftargi);
    fft(lenfft,fftargr,fftargi);
    for i:=1 to lenfft do
      begin
        aa:=fftargr^[i];  bb:=fftargi^[i];
        cc:=fkernelr^[i]; dd:=-fkerneli^[i];  {CONJUGATE}
        fftargr^[i]:=aa*cc-bb*dd;
        fftargi^[i]:=aa*dd+bb*cc
      end;
    ifft(lenfft,fftargr,fftargi);
```

```
    l_to_s(numsam,fftargr,fftargi,g);
    dispose(fftargr);
    dispose(fftargi);
    fftargr:=nil;
    fftargi:=nil
  end;   {procedure convc}

var
  mu0:real;              {permeability of free space}
  dum:real;              {2/intrin0}

  fin,fout:text;         {output and input files}
  res:string[5];         {output file name}
  tech:integer;          {technique:1=GR,2=CST}
  novar:integer;         {no. of variational constants}
  ka:real;               {wavenumber * half strip width}
  losfac:real;           {kc[re]/kc[im]}
  noit:integer;          {no. of iterations}
  ig:integer;            {initial guess, if ig=0 then
                           zero init. guess else
                           Kirchhoff approx. with
                           initial minimization step}

  methstr:string[3];     {method abreviation}

  lamda:real;            {free space wavelength}

  rms:matvec;            {store normerr at each iteration
                           to create .mat file}
  rmsmat:matmat;         {matrix created from rms}
  rmsname:matname;       {name for .mat file containing
                           error}

  xval:matvec;           {x values at sample points}
  cur:matvec;            {current magnitude at sample
                           points}
  curmat:matmat;         {matrix created from cur}
  curname:matname;       {name for .mat file containing
                           current}

  g:scarray;             {incident field}
  normg:real;            {norm of g}
  fn:scarray;            {current after iteration n}
  rhon:scarray;          {residual after iteration n}
  phin:scarray;          {variational function in iteration
                           n}
  Lphin:scarray;         {convolution of phin and kernel}
  itno:integer;          {iteration number}
  errn:real;             {rms error after iteration n}
  normerr:real;          {normalized rms error}
  Lfn:scarray;           {convolution of fn and Kernel}
  fcorn:scarray;         {correction factor of current
                           during iteration n}
  Lfcorn:scarray;        {convolution of fcorn and Kernel}
```

```
  v:array [1..3] of scarray;  {iteration variables}
  aa:matrix;                  {iteration matrix}
  b:vector;                   {iteration variable}
  c1:vector;                  {cofactors of a}
  d:complex;                  {determinant of a}
  bod:complex;                {b[1]/d}
  alpha:vector;               {variational constants}
  gerr:integer;

  x1:real;                    {value of x at centre of
                                interval 1}
  xi:real;                    {value of x at centre of
                                interval i}
  vr:real;                    {magnitude of current at
                                interval centre}

  dog,cat,rat:complex; {dummy vars}

procedure write_heading_fout;

  begin
    WRITELN(fout,'method:',methstr,novar:2,'  M = ',M:4);
    WRITELN(fout,'ka = ',ka,'  N = ',N:4,
             '  loss factor = ',losfac)
  end;

procedure write_error_fout;

  begin
    WRITELN(fout,itno:3,'   ',normerr,
             '   ',log10(normerr))
  end;

procedure write_current_fout;

  begin
    WRITELN(fout,xi,'  ',vr)
  end;


procedure cst_technique;

  {generates variational parameters for CST ttechnique}

  var
    argfftr,argffti:lrarrayptr;  {argument of FFT}
    ii:integer;                  {loop counter}
    num,den:complex;
    aa,bb,cc,dd,r2:real;

  begin  {procedure cst_technique}
    new(argfftr);
    new(argffti);
    s_to_l(M,N,rhon,argfftr,argffti);
```

```
    fft(M,argfftr,argffti);
    for ii:=1 to M do
      begin
        aa:=argfftr^[ii];  bb:=argffti^[ii];
        cc:=fkernelr^[ii]; dd:=fkerneli^[ii];
        r2:=sqr(cc)+sqr(dd);
        argfftr^[ii]:=(aa*cc+bb*dd)/r2;
        argffti^[ii]:=(bb*cc-aa*dd)/r2
      end;
    ifft(M,argfftr,argffti);
    l_to_s(N,argfftr,argffti,phin);
    dispose(argfftr);
    dispose(argffti);
    argfftr:=nil;
    argffti:=nil;
    conv(M,N,phin,Lphin)
  end;   {procedure cst_technique}

procedure gr_technique;

  {generates variational parameters for GR technique}

  begin  {procedure gr_technique}
    convc(M,N,rhon,phin);
    conv(M,N,phin,Lphin);
  end;   {procedure gr_technique}

procedure newfunc;

  {calculates values at the end of an iteration}

  var
    ii:integer;  {loop counter}

  begin  {procedure newfunc}
    for ii:=1 to N do
      begin
        cadd(fn[ii],fcorn[ii],fn[ii]);
        cadd(Lfn[ii],Lfcorn[ii],Lfn[ii]);
        csub(Lfn[ii],g[ii],rhon[ii])
      end;
    errn:=norm(a,N,rhon,false);
    normerr:=errn/normg
  end;   {procedure newfunc}

procedure novar1;

  var
    ii:integer;  {loop counter}

  begin  {procedure novar1}
    v[1]:=Lphin;
    inprod(a,N,v[1],v[1],aa[1,1]);
    inprod(a,N,v[1],rhon,dog);
```

```
    cneg(dog,b[1]);
    cdiv(b[1],aa[1,1],alpha[1]);
    for ii:=1 to N do
      begin
        cmult(alpha[1],phin[ii],fcorn[ii]);
        cmult(alpha[1],Lphin[ii],Lfcorn[ii])
      end;
    newfunc
  end;   {procedure novar1}

procedure novar2;

  var
    ii,jj:integer;  {loop counters}

  begin  {procedure novar2}
    v[1]:=Lphin;
    v[2]:=Lfcorn;
    for ii:=1 to 2 do
      for jj:=1 to 2 do inprod(a,N,v[ii],v[jj],aa[ii,jj]);
    inprod(a,N,v[1],rhon,dog);
    cneg(dog,b[1]);  b[2]:=zero;

    Gaussian_Elimination(2,aa,b,alpha,gerr);
    if gerr <> 0 then writeln(fout,'GAUSS ERROR !');

    for ii:=1 to N do
      begin
        cmult(alpha[1],phin[ii],dog);
        cmult(alpha[2],fcorn[ii],cat);
        cadd(dog,cat,fcorn[ii]);
        cmult(alpha[1],Lphin[ii],dog);
        cmult(alpha[2],Lfcorn[ii],cat);
        cadd(dog,cat,Lfcorn[ii])
      end;
    newfunc
  end;   {procedure novar2}

var  {STRICTLY NOT GLOBALS}
  i,j:integer;      {loop counters}

  onecarray,twocarray,zcarray:scarray;

  A0,B0,eta0:complex;

  hrs,mins,sex,sex100:word;
  stime,ftime,ttime:single;


begin  {PROGRAM}

  WRITELN('RUNNING....');

  for i:=1 to mini do onecarray[i]:=one;
```

```
for i:=1 to mini do zcarray[i]:=zero;

mu0:=pi*4e-7;
intrin0:=sqrt(mu0/epsilon0);
dum:=2/intrin0;

assign(fin,'input.dat');
reset(fin);

repeat

  gettime(hrs,mins,sex,sex100);
  stime:=3600.0*hrs+60.0*mins+sex+sex100/100.0;

  READLN(fin,res,tech,novar,M,ka,N,losfac,noit,ig);

  WRITELN('***',res);

  rmsname:=res+'rms';

  if tech=1 then methstr:='gr' else methstr:='cst';

  assign(fout,rmsname+'.prn');
  rewrite(fout);
  write_heading_fout;

  k0:=ka/a;
  lamda:=2*pi/k0;
  kc[re]:=k0;
  kc[im]:=losfac*k0;

  h:=2*a/N;

  new(fkernelr);
  new(fkerneli);

  {new(fkernelc);}

  kernel_init;

  {incident field}
  g:=onecarray;
  normg:=norm(a,N,g,false);

  if ig = 0 then
    begin

      {zero initial guess}
      fn:=zcarray;
      Lfn:=zcarray;          {conv(M,N,fn,Lfn);}
      for i:=1 to N do csub(Lfn[i],g[i],rhon[i]);
    end
  else
    begin
```

```
    {initial guess : physical optics approximation}
    for i:=1 to N do
      begin
        fn[i,re]:=2*kc[re];
        fn[i,im]:=2*kc[im]
      end;
    conv(M,N,fn,Lfn);
    inprod(a,N,Lfn,g,A0);
    inprod(a,N,Lfn,Lfn,B0);
    cdiv(A0,B0,eta0);
    for i:=1 to N do cmult(eta0,fn[i],fn[i]);
    conv(M,N,fn,Lfn);
    for i:=1 to N do csub(Lfn[i],g[i],rhon[i]);
  end;

itno:=0;
errn:=norm(a,N,rhon,false);
normerr:=errn/normg;

write_error_fout;
rms[itno+1]:=normerr;

if tech = 1 then
  begin  {tech=1}
    if novar > 2 then novar:=2;
    case novar of
        1:begin  {novar=1}
            itno:=1;
            while itno<=noit do
              begin
                gr_technique;
                novar1;
                write_error_fout;
                rms[itno+1]:=normerr;
                itno:=succ(itno)
              end
          end;   {novar=1}
        2:begin  {novar=2}
            itno:=1;
            gr_technique;
            novar1;
            write_error_fout;
            rms[itno+1]:=normerr;
            itno:=2;
            while itno<=noit do
              begin
                gr_technique;
                novar2;
                write_error_fout;
                rms[itno+1]:=normerr;
                itno:=succ(itno)
              end
          end    {novar=2}
```

```
      end   {case novar of}
  end    {tech=1}
else
  begin  {tech=2}
    if novar > 3 then novar:=3;
    case novar of
        1:begin  {novar=1}
            itno:=1;
            while itno<=noit do
              begin
                cst_technique;
                novar1;
                write_error_fout;
                rms[itno+1]:=normerr;
                itno:=succ(itno)
              end
          end;   {novar=1}
        2:begin  {novar=2}
            itno:=1;
            cst_technique;
            novar1;
            write_error_fout;
            rms[itno+1]:=normerr;
            itno:=2;
            while itno<=noit do
              begin
                cst_technique;
                novar2;
                write_error_fout;
                rms[itno+1]:=normerr;
                itno:=succ(itno)
              end
          end;   {novar=2}
        3:begin  {novar=3}
            itno:=1;
            cst_technique;
            novar1;
            write_error_fout;
            rms[itno+1]:=normerr;
            itno:=2;
            cst_technique;
            novar2;
            write_error_fout;
            rms[itno+1]:=normerr;
            itno:=3;
            while itno<=noit do
              begin
                cst_technique;
                v[1]:=Lphin;
                v[2]:=Lfcorn;
                v[3]:=Lfn;
                for i:=1 to 3 do
                  for j:=1 to 3 do
                        inprod(a,N,v[i],v[j],aa[i,j]);
```

```
                    inprod(a,N,v[1],rhon,dog);
                    cneg(dog,b[1]);
                    b[2]:=zero;  b[3]:=zero;

                    Gaussian_Elimination(3,aa,b,
                                         alpha,gerr);
                    if gerr <> 0 then
                      writeln(fout,'GAUSS ERROR !');

                    for i:=1 to N do
                      begin
                        cmult(alpha[1],phin[i],dog);
                        cmult(alpha[2],fcorn[i],cat);
                        cmult(alpha[3],fn[i],rat);
                        cadd(cat,rat,cat);
                        cadd(dog,cat,fcorn[i]);
                        cmult(alpha[1],Lphin[i],dog);
                        cmult(alpha[2],Lfcorn[i],cat);
                        cmult(alpha[3],Lfn[i],rat);
                        cadd(cat,rat,cat);
                        cadd(dog,cat,Lfcorn[i])
                      end;
                    newfunc;
                    write_error_fout;
                    rms[itno+1]:=normerr;
                    itno:=succ(itno)
                  end
              end    {novar=3}
          end   {case novar of}
      end;   {tech=2}

gettime(hrs,mins,sex,sex100);
ftime:=3600.0*hrs+60.0*mins+sex+sex100/100.0;
if ftime < stime then
  ttime:=24*3600-stime+ftime
else
  ttime:=ftime-stime;
hrs:=trunc(ttime/3600);
ttime:=ttime-3600*hrs;
mins:=trunc(ttime/60);
ttime:=ttime-60*mins;
sex:=trunc(ttime);
ttime:=ttime-sex;
sex100:=round(ttime*100);
writeln(fout,'*** Execution Time = ',
        hrs:2,':',mins:2,':',sex:2,',',sex100,' ***');

close(fout);

setvec(succ(noit),rmsname,rms,rmsmat);
savemat(rmsname,rmsmat);

x1:=-a+h/2;
curname:=res+'cur';
```

```
    assign(fout,curname+'.prn');
    rewrite(fout);
    write_heading_fout;

    for i:=1 to N do
      begin
        xi:=x1+(i-1)*h;
        vr:=magnitude(fn[i],false);
        write_current_fout;
        xval[i]:=xi;
        cur[i]:=vr
      end;

    close(fout);

    setmat(N,curname,xval,cur,curmat);
    savemat(curname,curmat);

    dispose(fkernelr);
    dispose(fkerneli);
    fkernelr:=nil;
    fkerneli:=nil;

    {dispose(fkernelc);
    fkernelc:=nil}

  until eof(fin);
  close(fin);

end.
```