Formulation notes

# Introduction

I’m hoping to use this as just a quick overview and reference. The details of the formulation can be found in Chapter 10 in Computational Methods for Electromagnetics by Peterson, Raj, Mittra. This is more to a serve as a companion to my code, so that I don’t forget what decisions I made.

## The Plate

This is taken directly from Chapter 10 in Computational Methods for Electromagnetics by Peterson, Raj, Mittra. It is a method of moments approach for scattering from a flat perfectly conducting plate defined on rectangular cells. This will broken into two sections, getting the plate impedance (Z) and then feeding the system (E). Z and V are the matrix array we get when applying method of moments. Ie) E = Z J

The section will hopefully explain the thought process behind thePlate.m and what’s going on there.

### Geometry

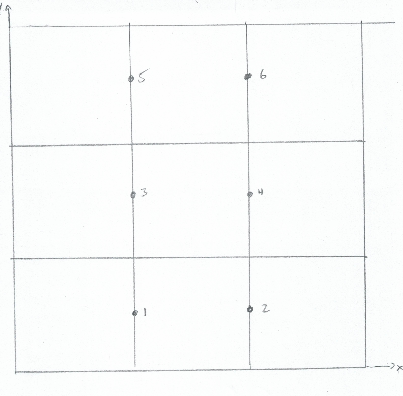
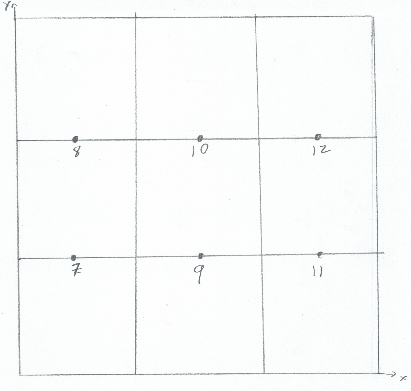
The plate is divided into rectangular cells. This allows for to break our model into its and components. The Electric Field Equation can then be written as

Where is the magnetic potential described by

The rooftop basis function will be used to describe the current. This basis function spans over two cells and is defined by at the center of the shared edge. In the direction of the current polarization, a triangle function used to force the current to die on the outer edges. A pulse function is used in the other direction to allow it to blow.

The important information out of here is that this mean and will be defined on different points. exists the shared edges of the two cells in x, but the center of the cell in y. x-position points are stored as Bxn\_xx and the corresponding y-position points stored as Bxn\_yy. Vice versa for which are stored as Byn\_xx and Byn\_yy.

I define first edge as the lower left hand. The idea was thinking in a cartesian plane where the lower left hand would be closets to (0,0). is defined first increases as we move right and upward. Then to define , is upward and to the left. This all is a bit hard to describe, so an example is provided for 3x3 cell division.

1a) center points defined 1b) center points defined

From here out, it’s pretty much normal method of moments process. The current is expanded as a set of basis function on the defined points. To create N-unknowns for the N-equations, the function is tested using a “razor blade” function (pulse in current direction and delta function in the other).

Going through all of this generates Z. The code stores this value as ZZ. The inverse is also stored as ZZinv. This is speed up “taking the inverse” when solving for the current. Only matrix multiplication will now be needed

### Feeding the plate

The code then feeds the system with theta polarized wave and phi polarized waves (TM and TE). The points are separated in and though. This means take the appropriate dot product ex) and then integrate against the testing function. shares ’s points and shares .

The wave and wave are stored separately as EE\_theta and EE\_phi. The current can now be calculated and are stored as JJ\_theta and JJ\_phi.

### Whats up with kk?

If while reading the code you notice kk=2\*pi scattered around, its just the wavenumber. Recall . The entire plate geometry is defined in lengths of for my sanity. This means that whenever we multiply k by distance we’ll get something proportional to .

### Initializing

The length is in wavelengths, not a ‘real’ length ex) 3. The incident angles are taken in as phiInc and thetaInc. NumCells describes how many cells across one row or column. For example, in the above figure 1, the would NumCells = 3. The division between cells should be at least .

## chaffElt

The plate is now defined. Holes need to be added to the plate to try and optimize the return.

### Initializing

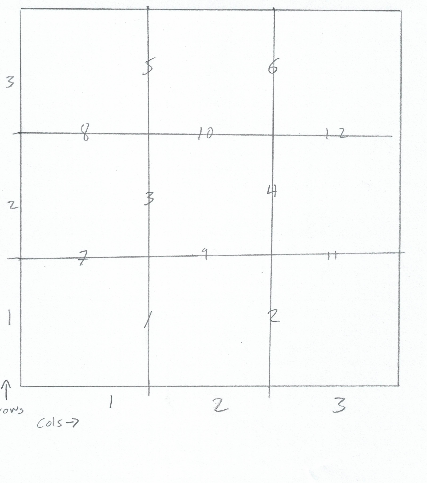
The code takes in the operating frequency, the length of the plate, theta and phi values to optimize over. These values are stored as freq, plateLength, thetaVals, and phiVals. Frequency, thetaVals, and phiVals can be an array of numbers. Angles are in radians.

A plate is then generated using these values. The code forces the division between cells to be and an even number. This for optimization later on. I probably should fix it to work for odd too, but its works pretty well as is.

There is an option to load a previously generated plate. Hopefully want to add a library at some point to try and speed things up. However the code gets the plate, the plate is stored as plateFull and plateNull. When adding holes, we remove those values from the matrix. This means the values are gone and if you want to try a new pattern (as we will for opitimizing) you can’t. This is why two versions of the plate are store. PlateFull, is complete metal plate without any holes while plateNull is where we’ve added holes and removed values.

### nullNew

nullNew takes in a matrix, nullPos, that defines rows and columns. The first column of nullPos is the rows and second column that need to be removed. Here we’re talking about removing cells, but the plate is described in edges. This just means remove the surrounding edges. I think it’s easier to describe in example, so let return to 3x3 cell division.



To remove the middle cell (2,2), edges 3,4,9, and 10. To remove the middle bottom cell (1,2), edges 1,2, and 9 are removed.

### Optimization

There are two ways to optimize. The first just lets the algorithm place whole however it wants (maximizeRCSAvg). The second forces symmetry in each quadrant (maximizeRCSAvgSymm). Because the second, feeds into the first I’m just going to explain that flow. This was written in pieces, so this is a bit of a journey.

maximizeRCSAvgSymm is like an envelope that just includes the setup to the built-in genetic algorithm function ga. This runs a binary for cells to be either “on or off.” ga expects an array, so cells are numbered. The bottom right is 0 and increase as you move rightwards and upwards. The function it attempts to maximize is null2minRCSAvgQuarter.

null2minRCSAvgQuarter only deals with bottom right quarter at first. ga gives it an array that can be 0 or 1. 0 means there is hole while 1 means there is metal. This array is taken and then replicated to get the symmetry we want (quarter2FullArray). This is then fed into null2minRCSAvg.

Null2minRCSAvg takes in an array of zeros and ones. The purpose of this function is really to just change the incident angle. Then, feed the array into null2minRCS with the appropriate scattered angle. We want monostatic rcs maximized, so we use the incident angle. Null2minRCSAvg is the driver function for maximizeRCSAvg too, so this is where symmetric optimization and non-symmetric paths meet.

Finally, the null2minRCS begins by taking the array of zeros and ones we’ve been passing around and pulls the location of the zeros to get the position of holes, nullPos. The function then gets the 4 rcs values in the form of the incident polarization vs the scattered polarization (theta/theta, theta/phi, phi/theta, phi/phi) and averages these values. This averages, avgRCS, is then returned.

Matlab attempts to maximize avgRCS and the solution found is returned.