Design and implementation of a wire array antenna analyzer

Hrafnkell Eiriksson

March 5, 2000

Abstract

Design and implementation of a wire array antenna analyzer is discussed. The analyzer is based on the Method of Moments and implemented in C++. Object-oriented design methods are used to make the software as general as possible and to allow for extensions. Implementation of numerical algorithms needed to solve the problem are discussed.

Contents

1 Introduction **2**

2 Theory **3**
2.1 Currents on a dipole . . . 3
2.1.1 The applied field . . . 4
2.2 Halléns integral equation . . . 4
2.3 Halléns integral equation for arrays . . . 5
2.4 The farfield radiation pattern . . . 6

3 Numerical methods **7**
3.1 Method of Moments . . . 7
3.1.1 Testing . . . 7
3.2 Solving Halléns integral equation for a dipole with the Method of Moments . . . 8
3.3 Solving Halléns integral equation for an array . . . 9
3.4 Using a triangle as a basis function . . . 9
3.5 Calculating the farfield pattern . . . 10

4 Results **11**

5 Design of the software **12**
5.1 The concept of object oriented programming . . . 12
5.2 An object oriented array analyzer . . . 12
5.2.1 C++ implementation details . . . 13

A Numerical algorithms **16**
A.1 LU factorization . . . 16
A.2 Simpson's rule of integration . . . 17

1 Introduction

This report documents the design and implementation of a wire array antenna analyzer. The mathematical theory on which it is based is presented and the integral equation that describes the currents on the wires of the array is derived. A numerical method, the method of moments, for solving the integral equation is presented. The moment of methods is based on expanding the current on the wires with a set of basis functions. Results from using different expansion functions are compared.

The array analyzer is implemented in C++ and its object oriented design is presented. Finally an overview of the numerical algorithms used in the program is given.

2 Theory

It is a well known fact that time varying currents radiate. When the currents and boundary conditions of the electromagnetic problem to be solved are known both the electric and magnetic fields can be found. This is usually done with the help of the vector potential [Col85]. The current density and the vector potential are related as follows

$$\mathbf{A}(\mathbf{r}) = \mu \int_V J(\mathbf{r}) \frac{\exp(-jk_0|\mathbf{r}-\mathbf{r}'|)}{4\pi|\mathbf{r}-\mathbf{r}'|} dV' \tag{1}$$

From that the H and E field can be found as

$$H(r) = \frac{1}{\mu}\nabla \times \mathbf{A}(\mathbf{r}) \tag{2}$$

$$\mathbf{E}(\mathbf{r}) = -j\omega\mathbf{A}(\mathbf{r}) + \frac{\nabla\nabla\cdot\mathbf{A}(\mathbf{r})}{j\omega\mu\epsilon} \tag{3}$$

2.1 Currents on a dipole

The simple dipole antenna structure can be viewed as a perfectly conducting cylinder with radius a and length l as can be seen on figure 1. It can be safely assumed that the current on the dipole exists only as

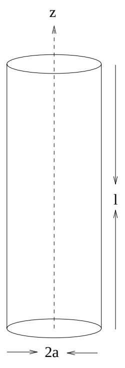


Figure 1: Geometry of a dipole

a surface current J_s as the dipole is perfectly conducting. We can then assume that J_s only has $\hat{z}$ and $\hat{\phi}$ components in the cylindrical coordinate system. If the dipole is made of a thin wire further assumptions can be made about the current. We assume that 1) $a << \lambda$ and 2) $l >> a$. This is referred to as the thin wire approximation.

Thin wire condition 1) allows us to assume that the $\hat{\phi}$ components of J_s are small. If we neglect them then J_s can be written as

$$\mathbf{J_s} = J_s\hat{\mathbf{z}} = J_s(z)\hat{\mathbf{z}} = \hat{\mathbf{z}}\frac{I(z)}{2\pi a} \tag{4}$$

The purpose of the current that is induced on the wire is to cancel the applied field from the generator, E_g, that drives the dipole. The electric field caused by the dipole can be found using eq. (1) and eq. (3). Plugging eq. (4) into eq. (1) gives

$$\mathbf{A} = \mu \int_0^{2\pi}\int_{-l/2}^{l/2} \hat{\mathbf{z}}\frac{I(z')}{2\pi a}\frac{\exp(-jk_0R)}{4\pi R} dz' a d\phi' \tag{5}$$

where $R = |r - r'|$.

Looking at the boundary condition for the E field on the surface of the dipole leads to two conditions. At $\rho = a$ the condition is

$$\begin{aligned} &\hat{\rho} \times (\hat{z}E_g + E) = 0 && (6)\\ \Rightarrow &\hat{\rho} \times \hat{z}E_g + \hat{\rho}E_\rho + \hat{\phi}E_\phi + \hat{z}E_z) = 0 && (7)\\ \Rightarrow &-\hat{\phi}E_g + \hat{z}E_\phi - \hat{\phi}E_z = 0 && (8)\\ \Rightarrow &E_\phi = 0 \qquad E_z = -E_g && (9)\end{aligned}$$

Using the boundary conditions together with eq. (3) gives (at $\rho = a$)

$$E_z = \hat{z} \cdot \mathbf{E} = -j\omega A_z - j\omega \frac{d^2}{dz^2} A_z = -E_g \tag{10}$$

Inserting the expression for the vector potential (eq. 1) into eq. (10) then leads to

$$(k2 + \frac{\partial^2}{\partial z^2}) \int_0^{2\pi} \int_{-l/2}^{l/2} \hat{z} \frac{I(z')}{2\pi a} \frac{\exp(-jk_0 R)}{4\pi R} dz' a d\phi = -j\omega\epsilon E_g \tag{11}$$

If we recall the definition for $R = |r - r'|$ we see that $R = 0$ when the observation point is on the surface of the dipole. This would cause the integral to become singular. To avoid that we replace the dipole with an equivalent line current on the center of the dipole. R now becomes $R = \sqrt{(z - z')^2 + a^2}$. The integral in eq. (11) is now independent of ϕ so integrating over ϕ gives 2π. The resulting equation (eq. 12) is an integral equation that relates the equivalent line current of the dipole to the applied generator field.

$$(k^2 + \frac{\partial^2}{\partial z^2}) \int_{-l/2}^{l/2} \hat{z} I(z') \frac{\exp(-jk_0 R)}{4\pi R} dz' a d\phi = -j\omega\epsilon E_g \tag{12}$$

Equation (12) is known as Pocklingtons integral equation [Col85].

2.1.1 The applied field

A dipole can be exited by z-directed uniform field over a gap of width b in the dipole.

$$Eg(z) = \begin{cases} V_g/b & |z| < b/2 \\ 0 & 0 \text{ elsewhere} \end{cases} \tag{13}$$

If we now let the gap size approach 0, $b \to 0$ then we get

$$E_g = V_g \delta(z) \tag{14}$$

This is called the delta-gap feeding model.

2.2 Halléns integral equation

Pocklingtons integral equation can be rewritten into a form that is more suitable for numerical computations [Col85]. This version of the equation is known as Halléns integral equations. This is done by writing eq. (12) as

$$(k_0^2 + \frac{d^2}{dz^2}) Q(z) = -j\omega\epsilon V_g \delta(z) \tag{15}$$

where

$$Q(z) = \int_{-l/2}^{l/2} I(z') \frac{\exp(-jk_0R)}{4\pi R} dz'. \tag{16}$$

The solution to the differential equation eq. (16) is

$$Q(z) = A\cos k_0 z - j\frac{V_g Y_0}{2} sink_0|z| \tag{17}$$

where A and B are unknown constants. This is Halléns integral equation. By solving this equation the current distribution on the dipole can be found. From the current distribution the radiated E and H field can be found with the help of equations (1), (2) and (3).

2.3 Halléns integral equation for arrays

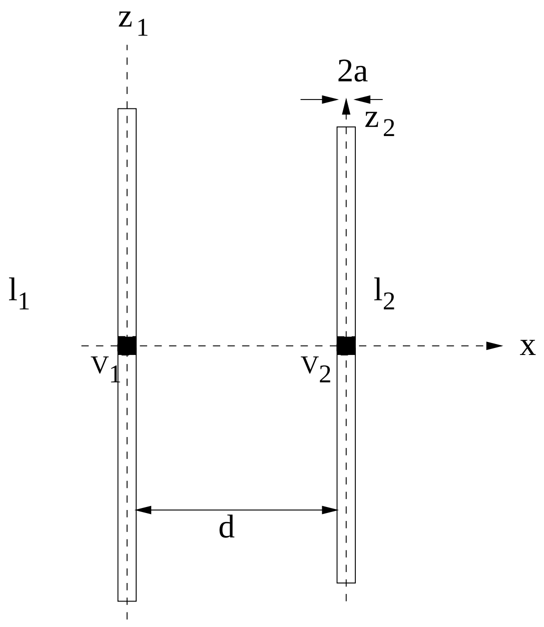


Figure 2: Geometry of a 2 element array

When many dipoles are located close to each other (f.ex. to form an array), the current distribution on one dipole is affected by the radiation from the others [Col85]. In order to find the currents on each element a system of integral equations must be solved. The system has one equation for each element.

Figure 2 shows a simple two element array with two parallel dipoles of radius a and lengths l_1 and l_2. Their separation is the distance d along the x axis. The feed of each dipole is a delta-gap feed. The coordinates z_i' measure the position along the axis of dipole i while z_i is measured on the surface on dipole i.

Let $Q_{ij}(z_i)$ be the vector potential along z_i due to the current $I_j(z_j')$. The scattered electric field along the z direction and on the surface of dipole i is then given by

$$(k_0^2 + \frac{\partial^2}{\partial z_i^2})\,[Q_{i1}(z_i) + Q_{i2}(z_i)] \tag{18}$$

$$= j\omega\epsilon\mu E_i(z_i) = -j\omega\epsilon\mu V_i\delta(z_i) \tag{19}$$

The system of equations can now be cast into a Hallén like equation the same way as was done above. This gives for the two dipole case

$$Q_{11}(z_1) + Q_{12}(z_1) = \frac{-jk_0 Y_0 \mu}{2} V_1 \sin k_0 |z_1| + C_1 \cos k_0 z_1 \tag{20}$$

$$Q_{21}(z_2) + Q_{22}(z_2) = \frac{-jk_0 Y_0 \mu}{2} V_2 \sin k_0 |z_2| + C_2 \cos k_0 z_2 \tag{21}$$

The vector potential functions are given by

$$Q_{ij}(z_i) = \mu \int_{-l_j/2}^{l_j/2} \frac{\exp(-jk_0 R_{ij})}{4\pi R_{ij}} dz_j' \tag{22}$$

where R_{ij} is the distance from the observation point on dipole j to the integration point on dipole i.

The extension of this to more than 2 parallel dipoles is obvious.

2.4 The farfield radiation pattern

Using eq. (1) and eq. (3) together allows us to find an explicit expression for the E field radiated from a current distribution. It can be shown that the E field in the *farfield* (that is when the distance from the observation point is larger than $\frac{2D^2}{\lambda}$ where D is the largest dimension of the antenna) is [Col85]

$$\mathbf{E}(\mathbf{r}) \approx jk_0 Z_0 \frac{\exp(-jk_0 r)}{4\pi r} \hat{r} \times \hat{r} \times \int_V \mathbf{J}(\mathbf{r}') \exp(jk_0 \hat{r} \cdot \mathbf{r}') dV' \tag{23}$$

When all the currents are z-directed and parallel along the x-axis eq. (23) simplifies:

$$\mathbf{E}(\mathbf{r}) \approx jk_0 Z_0 \frac{\exp(-jk_0 r)}{4\pi r} \hat{r} \times \hat{r} \times \hat{z} \int_V J(z') \exp(jk_0(z' \cos\theta + x' \sin\theta \cos\phi)) dV' \tag{24}$$

$$= jk_0 Z_0 \frac{\exp(-jk_0 r)}{4\pi r} \int_{wires} J(z') \exp(jk_0(z' \cos\theta + x' \sin\theta \cos\phi)) dz' \tag{25}$$

This expression has two factors

$$E(r,\theta,\phi) = \frac{\exp(-jk_0 r)}{r} F(\theta,\phi) \tag{26}$$

where

$$F(\theta,\phi) = \frac{jk_0 Z_0}{4\pi} \int_{wires} J(r') \exp(jk_0(z' \cos\theta + x' \sin\theta \cos\phi)) dz' \tag{27}$$

is called the *far field radiation pattern.*

3 Numerical methods

Halléns integral equation presented in section 2.2 can not be solved analytically. Numerical methods must be used. The most commonly used method for solving integral equations is the *method of moments*. The method is based on expanding the unknown current (or currents) with a set of basis functions and using that to transform the integral to a sum.

3.1 Method of Moments

An integral equation like eq. (12) can be transformed into the equation

$$\int_a^b G(z,z')I(z')dz' = f(z) \tag{28}$$

where $f(z)$ has the domain $[a,b]$ as its support. $I(z)$ is the unknown and $G(z,z')$ is a kernel function [Col85].

The current in the integral can be written as a sum of weighted basis functions

$$I(z) = \sum_{n=1}^{\infty} I_n \Phi_n(z). \tag{29}$$

When the expansion of the current (eq. (29)) is inserted into eq. (28) the integral is transformed into a sum:

$$\sum_{n=1}^{\infty} I_n \int_a^b G(z,z')\Phi_n(z')dz' = \tag{30}$$

$$\sum_{n=1}^{\infty} I_n G_n(z) = f(z) \tag{31}$$

where

$$G_n(z) = \int_a^b G(z,z')\Phi_n(z')dz' \tag{32}$$

The set of basis functions $\{\Phi_n(z)\}_{n=1}^{\infty}$ is infinite and the equality in eq. (31) holds in some sense (f.ex. least squares, dependent on the basis chosen). In order to be able to work with eq. (31) in a computer we must truncate the sum and choose to use only N basis functions. The equality in eq. (31) now becomes an approximation. An error $r(z)$ can be defined as

$$r(z) = f(z) - \sum_{n=1}^{N} I_n G_n(z) \tag{33}$$

3.1.1 Testing

Equation (31) is still a function of the variable z that has a continuous support domain. In order to be able to work with it in a computer it has to be made discrete (or sampled). We can define a sampling grid with N points z_m over $[a,b]$ and force the error to be $r(z) = 0$ on that grid. We then get

$$\sum_{n=1}^{N} I_n G_n(z_m) = f(z_m) \tag{34}$$

Forcing $r(z)$ to be zero in N discrete points is often referred to as *point testing* or *point matching*.

Equation (34) can be written as linear equation system

$$[G_{nm}]\mathbf{I} = \mathbf{f} \tag{35}$$

where $[G_{nm}]$ is a matrix with the elements $G_n(z_m)$, I is a column vector with the current weights I_n and f is a column vector with the elements $f(z_m)$. This system of linear equations can now be solved with Gaussian elimination or LU factorization.

3.2 Solving Halléns integral equation for a dipole with the Method of Moments

Halléns integral equation has a similar form as eq. (28).

$$\int_{-l/2}^{l/2} I(z')\frac{\exp(-jk_0R)}{4\pi R}dz' = A\cos k_0 z - j\frac{V_g Y_0}{2} sin k_0|z| \tag{36}$$

Here $\frac{\exp(-jk_0R)}{4\pi R}$ is the kernel.

One possible choice for basis functions is the pulse function (see figure 3) The pulse function is defined

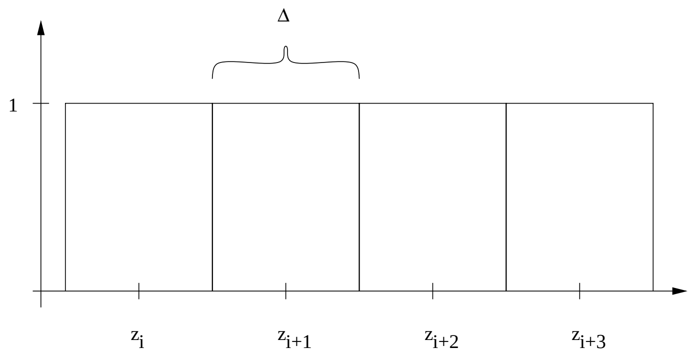


Figure 3: One possibility for choosing a basis is choosing pulse functions. $\Phi_n(z)$ is defined on $[z_n - \Delta/2, z_n + \Delta/2]$.

as

$$\Phi_n(z) = \begin{cases} 1 & |z_n - \Delta/2| < z \\ 0 & \text{elsewhere} \end{cases} \tag{37}$$

where $\Delta = z_n - z_{n-1}$. The elements of matrix in eq. (35) when using pulse functions as basis

$$G_{nm} = \int_{-l/2}^{l/2} \Phi_n(z')\frac{\exp(-jk_0R)}{4\pi R}dz' \tag{38}$$

$$= \int_{z_n-\Delta/2}^{z_n+\Delta/2} \frac{\exp(-jk_0R)}{4\pi R}dz' \tag{39}$$

Halléns integral equation (eq. 36) now becomes

$$\sum_{n=1}^{N} G_{nm}I_n - A\cos k_0 z_m = -\frac{jY_0V_g}{2}\sin k_0|z_m| \tag{40}$$

In eq. (40) there are $N+1$ unknowns (N unknown I_n and the constant A). Therefor we have to choose $N+1$ testing points (the z_m values). We can choose to use the N z_n values that the basis functions are defined around and add one more point. Either endpoint of the dipole is a good choice. That gives $N+1$ unknowns and $N+1$ equations that can then be solved for the current weighting constants I_n.

3.3 Solving Halléns integral equation for an array

Halléns integral equation for arrays (eq. (20) and (21) for the two element case) can be solved using the method of moments in a way similar to what was done for the single dipole. Each $Q_{ij}(z)$ becomes a block in the matrix $[G_{nm}]$ and the currents on the wires are represented in a single vector. Equations (20) and (21) become

$$\sum_{n=1}^{N_1} G_{11nm} I_{1n} \quad + \quad \sum_{n=1}^{N_2} G_{12nm} I_{2n} \quad - \quad A_1 \cos k_0 z_{1m} \quad = \quad -\frac{jY_0 V_{1g}}{2} \sin k_0 |z_{1m}| \tag{41}$$

$$\sum_{n=1}^{N_1} G_{21nm} I_{1n} \quad + \quad \sum_{n=1}^{N_2} G_{22nm} I_{2n} \quad - \quad A_2 \cos k_0 z_{1m} \quad = \quad -\frac{jY_0 V_{2g}}{2} \sin k_0 |z_{2m}| \tag{42}$$

where

$$G_{ijnm} = \int_{z_n - \Delta/2}^{z_n + \Delta/2} \frac{\exp(-jk_0 R_{ij})}{4\pi R_{ij}} dz' \tag{43}$$

N_i is the number of basis functions needed (often also referred to as number of subdomains) on wire i of the array and V_{ig} is the generator voltage for the i-th element

Equations (41) and (42) can be written up as a single system of linear equations as described above. The resulting matrix equation will have the structure shown below

$$\begin{bmatrix} [G_{11}] & [\cos k_0 z_{1m}] & [G_{12}] & [0] \\ G_{21} & [0] & [G_{22}] & [\cos k_0 z_{2m}] \end{bmatrix} \begin{bmatrix} \mathbf{I_{1n}} \\ A_1 \\ \mathbf{I_{2n}} \\ A_2 \end{bmatrix} = \frac{-jY_0}{2} \begin{bmatrix} V_{1g} \sin k_0 |\mathbf{z_{1m}}| \\ V_{2g} \sin k_0 |\mathbf{z_{2m}}| \end{bmatrix} \tag{44}$$

Extending this for more elements is obvious.

3.4 Using a triangle as a basis function

Using a pulse function as a basis function for expanding the current corresponds to using a rectangular rule for integration. The next obvious step to try to improve accuracy is to approximate the current as a piecewise linear function. To do that we use the triangular basis function. The $n-$ th basis function is defined over an interval $[z_{n-1}, z_{n+1}]$

$$\Phi_n(z) = \begin{cases} 1/\Delta(z - z_n) + 1 & z_{n-1} < z < z_n \\ -1/\Delta(z - z_n) + 1 & z_n < z < z_{n+1} \\ 0 & \text{otherwise} \end{cases} \tag{45}$$

Note that here the z_n values define the endpoints of the intervals, not their centers (see figure 4)

When using triangles as basis functions we can choose all the z_n points as testing point and then add an extra point for chosen "randomly" to get $N + 1$ testing points.

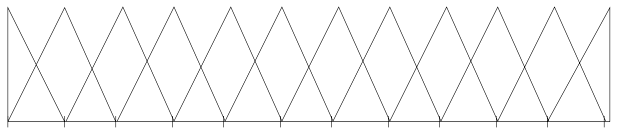


Figure 4: The triangular basis functions. Note that the functions span two intervals and the functions at the ends are different form the other.

3.5 Calculating the farfield pattern

When the currents on an wire array have been found its relatively easy to calculate the farfield radiation pattern from eq. (27). One can chose to do it numerically using an numerical integration method like Simpson's rule of integration (see Appendix A.2). When using simple basis functions as the pulse the integration can be carried out analytically. When using pulse basis functions equation (27) becomes

$$F(\theta,\phi) = \sum_{n=1}^{N_{wires}} [\frac{\exp(jk_0 x_n \sin\theta\cos\phi)}{k_0\cos\phi} 2\sin(jk_0\Delta/2\cos\phi) \sum_{m=1}^{N_{subd}} I_m \exp(jk_0 z_m \cos\phi)] \qquad (46)$$

4 Results

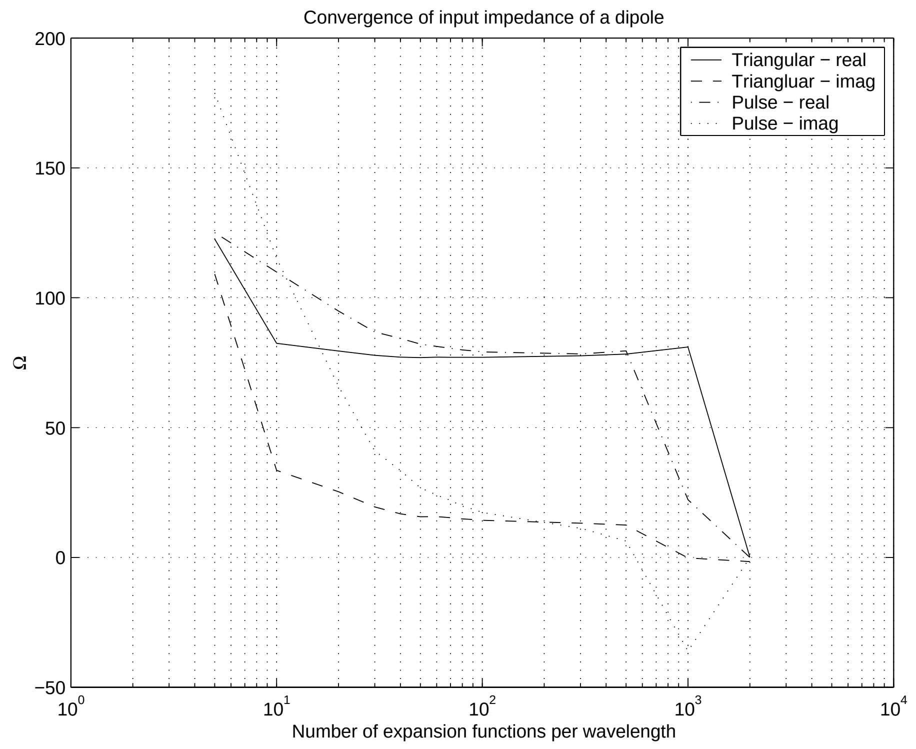


Figure 5: Comparing the input impedance of a $\lambda/2$ dipole when using pulse basis functions and triangular. The impedance seems to converge faster with triangular basis functions as expected.

Its interesting to look at the convergence rate of the input impedance of a half-wave dipole when analyzing it with the program written. As expected, convergence is faster with triangular basis functions than with pulse functions. This is because piecewise linear approximation to the current is better than the rectangular pulses approximation. When using pulses as basis functions the input impedance doesn't even seem to converge totally before the whole system breaks down. Collin shows that breakdown occurs about when the number of subdomains is l/a.

5 Design of the software

5.1 The concept of object oriented programming

Using an object oriented programming language provides the programmer with some very powerful tools to help him organize his program [Str97].

One of the basic concepts in object oriented programming (OOP) is the *class*. A class is an representation of a concept, it is a collection of the data needed to represent its state and the operations (functions, methods) that can be applied to the data. The operations form the *interface* of the class.

Organizing classes into a hierarchy is another of the basic ideas in object oriented programming. At the top of the class hierarchy is the most general representation of the idea or concept being implemented and further down are special versions of the concept. An example could be the basic class ”shape” and further down in the hierarchy there might be a ”circle”. We say that the circle class is inherited from the ”shape” class. The ”shape” and the ”circle” class share the same interface. One possible operation to be applied to a ”shape” would be to draw it.

When writing object oriented code it is important to remember to a) program to an interface, not an implementation and b) premature optimization is the root of all evil.

5.2 An object oriented array analyzer

The result of applying the moment of methods to an antenna array problem as described in section 2 are the currents on each wire. What the currents are depends on the geometry of the problem (thickness, length and position of wires). It seems to be a good idea to represent an antenna array as a class that can be asked what the currents on its wires are. Someone who only wants to use the array analyzer doesn't care how its internals are, he only wants to be able to a) specify the geometry to be analyzed and b) ask what the currents on the wires are. This makes up the interface of the general array analyzer class. Using different basis functions to expand the current can be represented by inherited classes from the basic array analyzer class. This is very convenient when exploring the different results when using different methods for calculating. It allows the programmer to gather the implementation details that are common to all array analyzers into the base class (f.x. to read in geometrical information) and only change the small details that are to be explored in the derived classes.

Every array analyzer that uses the method of moments to analyze the currents on its wires has to do very similar things internally. It has to solve the matrix equation $[G_{nm}]I = V_g$. Before doing that it calculates the elements of the $[G_{nm}]$ matrix and the V_g vector.

The matrix and vector are fundamental concepts used in finding the currents. They too can be represented with a single class. A single matrix class is enough because vector can be considered a matrix where the number of columns or rows is 1. A matrix class has all the basic linear algebra functions such as solving a system of linear equations. They form the interface of the matrix class.

To calculate the elements of the $[G_{nm}]$ matrix, numerical integration must be used. One possible numerical algorithm is the Simpson rule of integration (see appendix A.2). The same numerical integration method can be used to calculate the elements of the matrix independent on the choice of basis functions (and even kernels). The numerical integration method can therefor be implemented as class that represents an ”integration machine” that can integrate any function given to it. The integrator only has one operation in its interface: evaluate the given function.

One possible use of the currents found by the array analyzer is to calculate far field radiation pattern from the antenna array. A far field calculator class could be constructed that has only one operation in its interface: give me the value of the far field pattern in the direction of (θ, ϕ).

5.2.1 C++ implementation details

The `matrix<T>` class is the basic class used for storing data, vectors and matrices in the program. It implements a linear algebra matrix/vector and is a template class. The template type `T` represents the type of data it stores. `T` can f.x. be a `double` or a `complex<double>`. The `matrix<T>` class knows how to LU factor a $N \times N$ matrix (and does it with the algorithm described in appendix A.1), multiply a vector and matrix, sum up the elements of it etc. Arithmetic is implemented using operator overloading.

The `integrator<Arg, Res>` class implements Simpson's rule of integration (as described in appendix A.2). It is also a template class where the type `Arg` represents the type of data the function to be integrated accepts as arguments and the type `Res` represents the type of data the function to be integrated returns as results This allows the integrator class to allocate `matrix<T>` vectors for the data of correct type. `integrator<Arg, Res>` takes as an argument to its constructor how often to divide the interval that should be integrated over and an function object representing the function to integrate. The basic function object to integrate is represented by the base class `bas_func<Arg, Res>` where `Arg` and `Res` have the same meaning and in the `integrator<Arg, Res>` class. Classes with the base type `bas_func<Arg, Res>` represent the kernel multiplied by the basis function. Two classes are derived from the `bas_func<Arg, Res>` class, `const_func<Arg, Res>` and `trap_func<Arg, Res>`. `const_func<Arg, Res>` is used when expanding the current with pulse basis functions and `trap_func<Arg, Res>` when using triangular basis functions.

The workhorse of the array analyzer is the `bas_array` class. When an instance of it is constructed its constructor reads in the geometry to analyze and how many basis functions (subdomains) to use for every wavelength. It then proceeds to fill out the G_{nm} matrix (just called `G` in the implementation) and the V_g vector (called `V` in the implementation). Different versions of the `array` class are used when using pulse basis functions (`const_array`) or triangular basis functions (`trap_array`).

The farfield calculations are done by classes derived from `bas_farfield`. Only one derived class has been implemented, the `const_farfield` class. It implements calculations of the farfield when using pulse expansion functions using eq. (46)

An UML class diagram showing the relationship between classes is showin in figure 5.2.1

The `const_array` and `const_farfield` classes can be used together with OpenGL (`http://www.opengl.org`
to create a 3 dimensional farfield pattern viewer (see the `fargl.cc` source file). The output of that viewer is shown in figure 6.

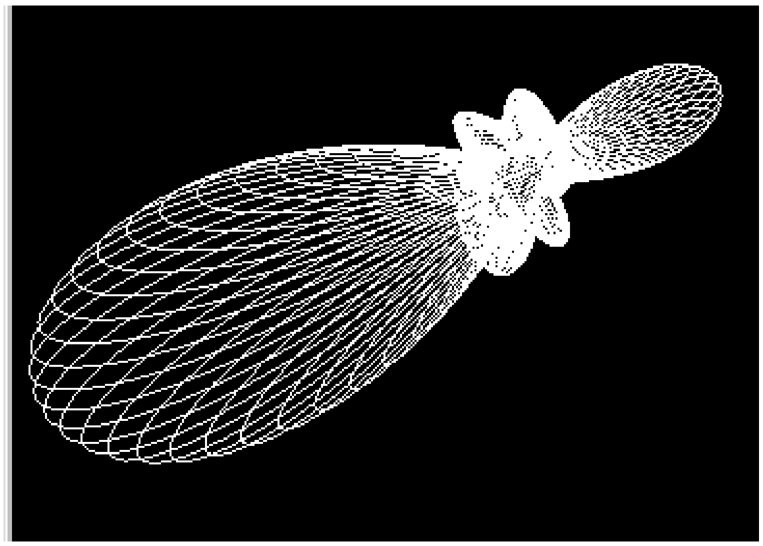
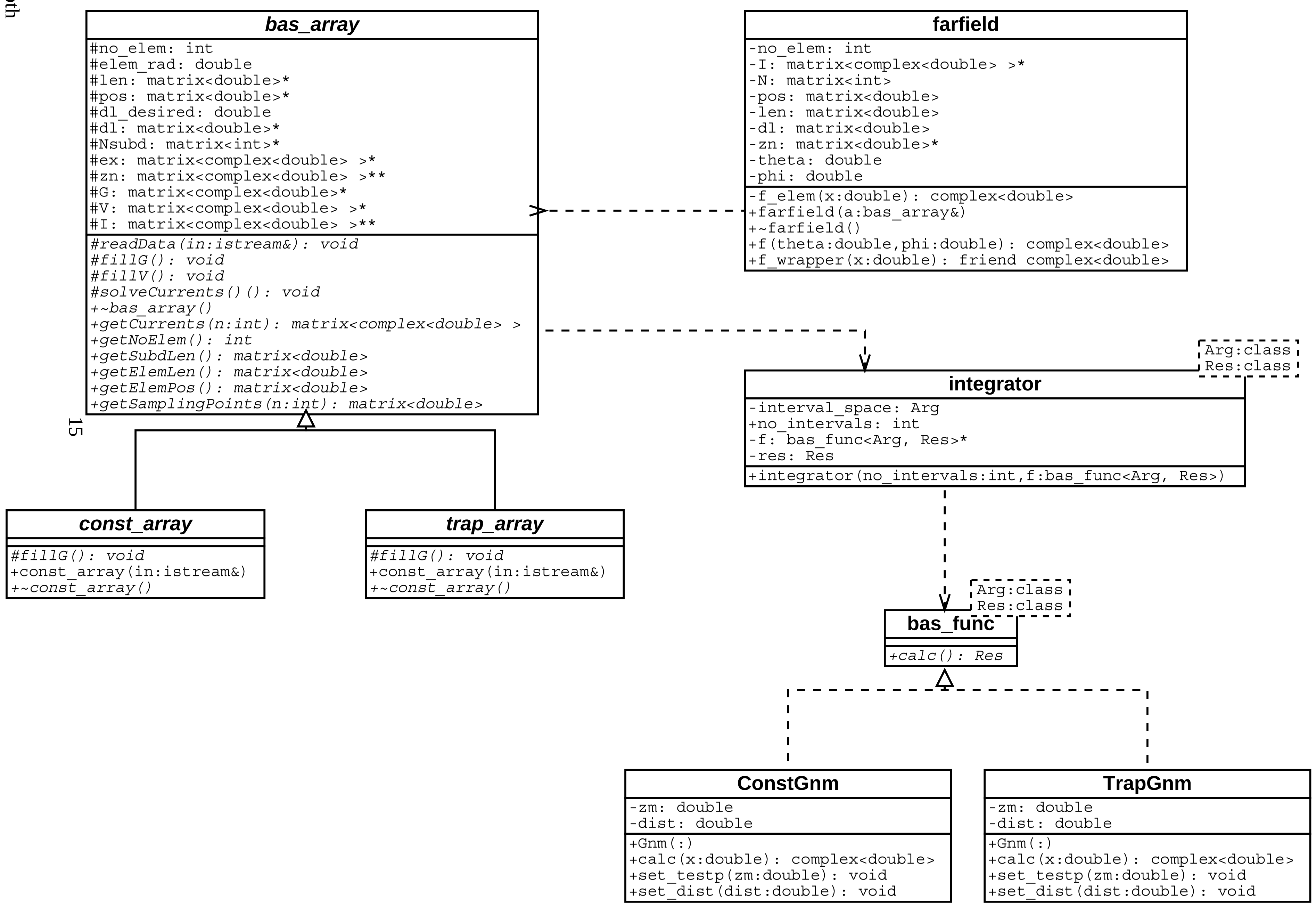


Figure 6: The output of the 3D farfield pattern viewer.

pth


bas_array
#no_elem: int
#elem_rad: double
#len: matrix<double>*
#pos: matrix<double>*
#dl_desired: double
#dl: matrix<double>*
#Nsubd: matrix<int>*
#ex: matrix<complex<double> >*
#zn: matrix<complex<double> >**
#G: matrix<complex<double>*
#V: matrix<complex<double> >*
#I: matrix<complex<double> >**
#readData(in:istream&): void
#fillG(): void
#fillV(): void
#solveCurrents()(): void
+~bas_array()
+getCurrents(n:int): matrix<complex<double> >
+getNoElem(): int
+getSubdLen(): matrix<double>
+getElemLen(): matrix<double>
+getElemPos(): matrix<double>
+getSamplingPoints(n:int): matrix<double>
farfield
-no_elem: int
-I: matrix<complex<double> >*
-N: matrix<int>
-pos: matrix<double>
-len: matrix<double>
-dl: matrix<double>
-zn: matrix<double>*
-theta: double
-phi: double
-f_elem(x:double): complex<double>
+farfield(a:bas_array&)
+~farfield()
+f(theta:double,phi:double): complex<double>
+f_wrapper(x:double): friend complex<double>
integrator
Arg:class
Res:class
-interval_space: Arg
+no_intervals: int
-f: bas_func<Arg, Res>*
-res: Res
+integrator(no_intervals:int,f:bas_func<Arg, Res>)
const_array
#fillG(): void
+const_array(in:istream&)
+~const_array()
trap_array
#fillG(): void
+const_array(in:istream&)
+~const_array()
bas_func
Arg:class
Res:class
+calc(): Res
ConstGnm
-zm: double
-dist: double
+Gnm(:)
+calc(x:double): complex<double>
+set_testp(zm:double): void
+set_dist(dist:double): void
TrapGnm
-zm: double
-dist: double
+Gnm(:)
+calc(x:double): complex<double>
+set_testp(zm:double): void
+set_dist(dist:double): void

A Numerical algorithms

A.1 LU factorization

The standard method of solving systems of linear equations is Gaussian elimination [Hag88]. An example of such a system is

$$\begin{bmatrix} 2 & 4 & 2 \\ 4 & 7 & 7 \\ -2 & -7 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 13 \\ 7 \end{bmatrix} \tag{47}$$

It is usually done in two phases. The first phase is *forward elimination* where the the coefficient matrix is converted to an upper triangular form. In the example above the unknown x_1 has to be removed from rows 2 and 3. This is done by subtracting it 2 times from the second equation and -1 times from the third equation. This gives

$$\begin{bmatrix} 2 & 4 & 2 \\ 0 & 7 & 7 \\ 0 & -7 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \\ 11 \end{bmatrix} \tag{48}$$

The unknown x_2 is then eliminated from the third equation by subtracting 3 times the second equation from the third equation and thereby giving the desired upper triangular structure

$$\begin{bmatrix} 2 & 4 & 2 \\ 0 & 7 & 7 \\ 0 & 0 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \\ -4 \end{bmatrix} \tag{49}$$

We can encode the steps taken to convert the original coefficient matrix (which we shall now call A) to the upper triangular matrix (which we call U) in a third matrix L. We set element (i, j) in to the factor we needed to multiply row i with to subtract it from row j to eliminate the i-th unknown from it. In the example above the matrix L would be

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 3 & 1 \end{bmatrix} \tag{50}$$

Here we have set the diagonal of the matrix to 1 for reasons which are about to be clear. This type of matrix is called unit lower triangular.

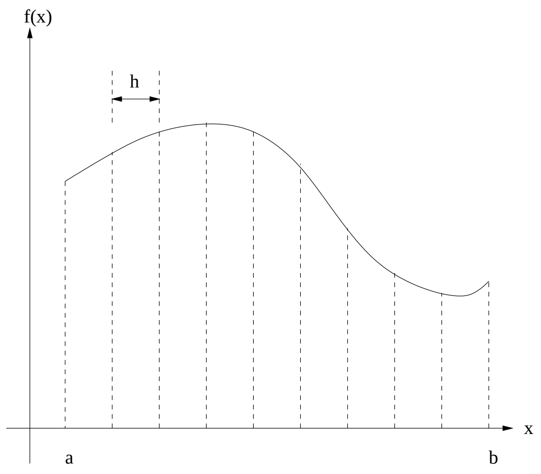
The reason for setting the diagonal to 1 is that it can be shown that Gaussian elimination is equivalent to factoring the original matrix A into the product between a unit lower triangular matrix L and an upper triangular matrix U. We can write

$$\mathbf{A} = \mathbf{LU} \tag{51}$$

The second phase of the Gaussian elimination is *back substitution*. In that the values for the unknowns (the $x_i, i = 1..3$ in the example) are calculated. We start by solving for the highest numbered unknown, then substituting the solution for it into the equation above and solve for the next highest numbered and so on. In the example above this would give $x_3 = -4/-2 = 2$, then $x_2 = (5 - 3 \cdot 2)/-1 = 1$ and finally $x_1 = (4 - 4 \cdot 1 - 2 \cdot 2)/2 = -2$

We can now formally proceed to solve a system like

$$\mathbf{Ax} = \mathbf{b}$$



ht

Figure 7: Most numerical integration methods are based on calculating the function $f(x)$ to be integrated in a discrete set of points with a spacing of h and then approximating the behavior of the function between those points.

. Using LU factorization this system can be written as

$$\mathbf{Ax} = (\mathbf{LU})\mathbf{x} = \mathbf{L}(\mathbf{Ux}) = \mathbf{Ly}$$

where $\mathbf{y} = \mathbf{Ux}$. Hence the original system is equivalent to the two new systems

$$\mathbf{Ly} = \mathbf{b} \tag{52}$$

and

$$\mathbf{Ux} = \mathbf{y} \tag{53}$$

The solution to the first equation is $\mathbf{y}$. Knowing $\mathbf{y}$ we can proceed to solve $\mathbf{x}$ by solving $\mathbf{Ux} = \mathbf{y}$. The system $\mathbf{Ly} = \mathbf{b}$ is a lower triangular system that can be solved by *forward substitution*. Forward substitution is like backward substitution but we start at the top instead of the bottom. The system $\mathbf{Ux} = \mathbf{b}$ can then be solved by backward substitution.

In summary, the system $\mathbf{Ax} = \mathbf{b}$ is solved in three steps:

1. $\mathbf{LU}$ factor $\mathbf{A}$
2. Solve $\mathbf{Ly} = \mathbf{b}$ by forward substitution
3. Solve $\mathbf{Ux} = \mathbf{y}$ by backward substitution

The same LU factorization can be used to solve the system of equations for many different $\mathbf{b}$ vectors.

A.2 Simpson's rule of integration

Most numerical integration methods are based on calculating the value of the function to integrated in a discrete set of points with a certain interval and approximating the behavior of the function with a simple function between two or more of the precalculated points (see figure 7). Approximating the function between two points with a constants gives the rectangular rule, picewise linear approximation gives the trapezoidal rule, and picewise quadratic approximation will give Simpson's rule [Kre99]. Simpson's rule is important because it is sufficiently accurate for most problems, but still sufficiently simple.

To derive Simpsons rule we divide the interval of integration $a \leq x \leq b$ into an even number, say $n = 2m$, of equal subintervals of length $h = (b-a)/2m$ with endpoints $x_0(=a), x_1, \ldots, x_{2m-1}, x_{2m}(=b)$. We now take the first two subintervals and approximate $f(x)$ in the interval $x_0 \leq x \leq x_2 = x_0 + 2h$ by the Lagrange polynomial $p_2(x)$ through $(x_0, f(x_0))$,$(x_1, f(x_1))$,$(x_2, f(x_2)$. The polynomial $p_2(x)$ is given as

$$p_2(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} f(x_0) + \tag{54}$$

$$\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} f(x_1) + \tag{55}$$

$$\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} f(x_2) \tag{56}$$

The denominators are $2h^2$, $-h^2$ and $2h^2$, respectively. Setting $s = (x - x_1)/h$ we have $x - x_0 = (s+1)/h$, $x - x_1 = sh$, $x - x_2 = (s-1)/h$ and we obtain

$$p_2(x) = \frac{1}{2} s(s-1) f(x_0) - (s+1)(s-1) f(x_1) + \tag{57}$$

$$\frac{1}{2}(s+1) s f(x_2) \tag{58}$$

We now integrate with respect to x from x_0 to x_2. This corresponds to integrating with respect to s from -1 to 1. Since $dx = hds$ the result is

$$\int_{x_0}^{x_2} f(x)dx \approx \int_{x_0}^{x_2} p_2(x)dx = \frac{h}{3}\left(f(x_0) + 4f(x_1) + f(x_2)\right) \tag{59}$$

A similar formula holds for the other intervals. By summing all these m formulas we obtain *Simpson's rule*:

$$\int_a^b f(x)dx \approx \frac{h}{3}(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \tag{60}$$

$$\cdots + 2f(x_{2m-2}) + 4f(x_{2m-1} + f(x_{2m}))) \tag{61}$$

An algorithm for Simpson's rule could then be:
In: a, b, m, $f_0, \ldots, f_{2m}$ Out: Approximate value of integral.
Compute:

1. $s_0 = f_0 + f_{2m}$
2. $s_1 = f_1 + f_3 + \cdots + f_{2m-1}$
3. $s_2 = f_2 + f_4 + \cdots + f_{2m-2}$
4. $h = (b-a)/2m$
5. return $\frac{h}{3}(s_0 + 4s_1 + 2s_2)$

References

[Col85] Robert E. Collin. *Antennas and radiowave propagation*. McGraw-Hill, 1985.

[Hag88] William W. Hager. *Applied Numerical Linear Algebra*. Prentice-Hall, 1988.

[Kre99] Erwin Kreyszig. *Advanced Engineering Mathematics, 8th edition*. John Wiley & Sons, 1999.

[Str97] Bjarne Stroustrup. *The C++ Programming Language, third edition*. Addison Wesley, 1997.