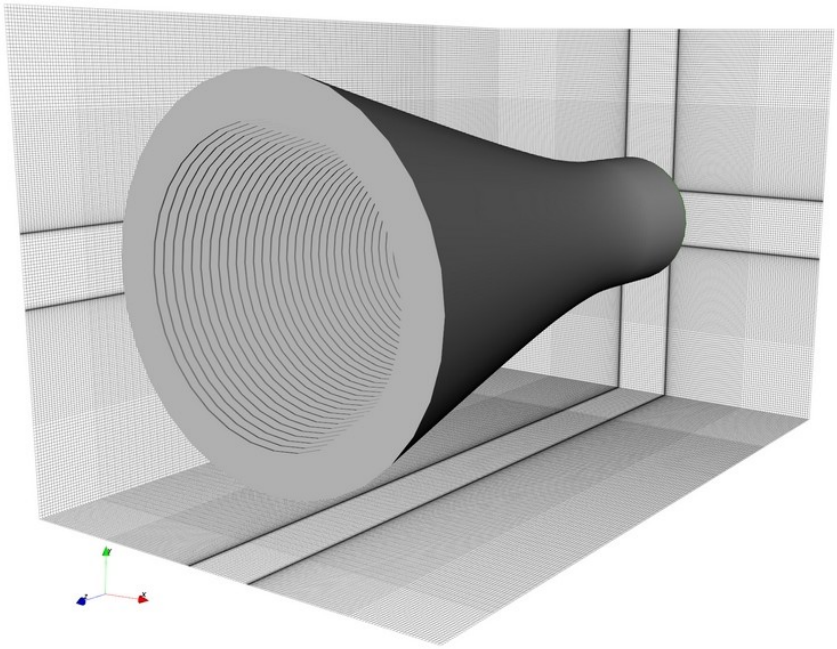
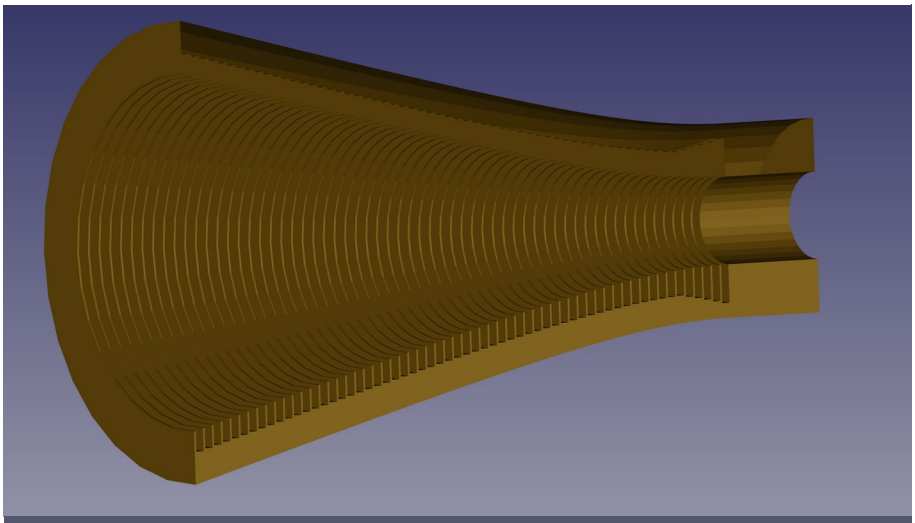
A Comprehensive Corrugated Horn Generator and Simulation with openEMS

Dr Paul Klasmann (DB9HIQ)

paulklasmann@hotmail.com





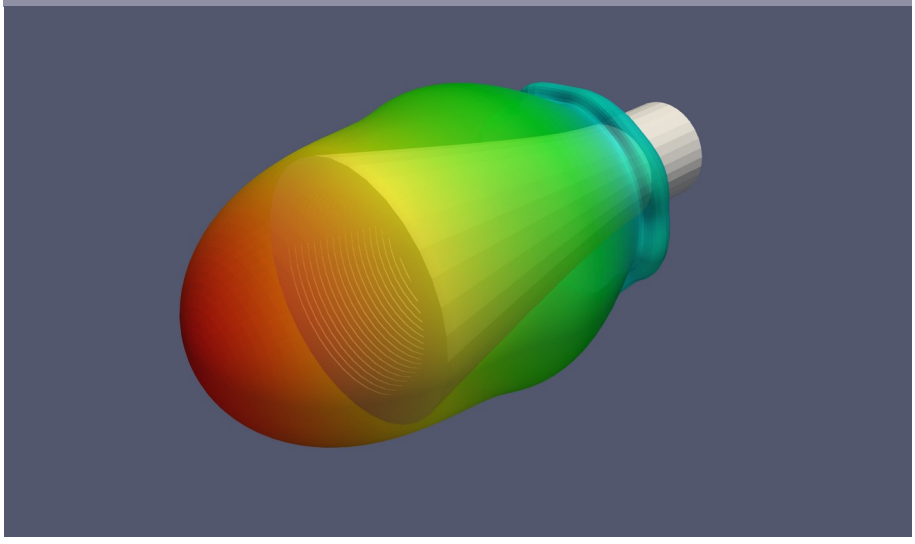


Table of Contents

Aim of Exercise

The following topics will be covered in this tutorial.

- Introduction to the corrugated horn antenna.
- How to define the various horn profile and mode converter options.
- Setting up variables to automate creation of the horn geometry.
- How to define and setup the physical model to simulate with openEMS [1].
- Mesh and simulation setup.
- How to obtain plots for S_{11}.
- How to add field dump boxes to obtain E-field 2D cuts and display them in ParaView [2].
- How to add the near field to far field dump box and plot the 3D farfield radiation patterns.
- How to export the physical model geometry to VTK or STL 3D model formats.
- Final remarks and comparison of the farfield to CST [3] and HFSS [4] EM simulators.

Introduction

Corrugated horn antennas have a number of advantages over smooth walled conical or rectangular pyramidal horn antennas. The key advantages are, near constant beamwidth for a given frequency band, rotationally symmetric radiation patterns for all angles of phi, very low cross polar radiation characteristics, very low sidelobes and very low reflection/VSWR. The bandwidth of the low cross polarisation levels are dependant on the choice of mode converter.

These characteristics make the corrugated horn an ideal candidate for high performance applications such as radiometers, antenna measurements, as feeds in high performance Cassegrain or Gregorian satcom reflector antennas, radio telescopes and radar applications.

Corrugated horns began to appear in the 1960s and much of the theory has been widely published, in particularly a popular book on the subject is “Corrugated Horns for Microwave Antennas”, by P.J.B. Clarricoats and A.D. Olver, IEE Electromagnetic Waves Series 18 [5].

This tutorial is based on the publication by Dr Christophe Granet and Professor Graeme L. James “Design of Corrugated Horns: A Primer” [6]. This article appeared in Dr Tom Milligan’s “Antenna Designer’s Notebook” in IEEE Antennas and Propagation Magazine, Vol.47, No.2, April 2005. It can be found on the internet and there is an errata for some important corrections. Much of the information from the article will be included here for completeness and include the corrections. It is assumed that the reader will have this article to refer to.

In the following sections we will discuss the basic theory of corrugated horns and the design parameters, followed by the various horn profile and mode converter options. A worked example, the same example as described in the article will be given where the same horn geometry and simulation in openEMS will follow. As we will see, openEMS produces results that correlate extremely well with commercial software CST Microwave Studio and Ansys HFSS.

Background

Corrugations in the horn's internal surface are annular slots that are nominally one quarter of a wavelength deep. These are necessary to be able to support a hybrid mode which is a combination of the TE and TM modes and for these modes to have the same propagating velocity. The corrugations provide a high impedance surface which make it possible for the horn to support these hybrid modes. The required propagating mode is the HE_{11} mode which result in a radiation pattern that approaches a Gaussian beam.

In order to generate the HE_{11} mode, a mode converter is required in the throat of the horn that follows the circular waveguide feeding the horn. This consists of 5 to 12 corrugations where the slot depth begins at approximately half a wavelength and gradually decreases to the nominal quarter wavelength of the corrugations that occupy the remaining length of the horn. The article describes three types of mode converter which have been implemented in the openEMS script. Refer to the article for a more thorough background into the theory. There are other types of hybrid mode horn antennas but these will not be considered here and will be the topic for another tutorial.

In addition to the three mode converters, eight different horn profiles have been included and these will have different radiation pattern characteristics. One mode converter and one horn profile must be selected within the script.

There are a number of design parameters that we will need to define.

Design Parameters

The main design parameters are:

f_{min}: lowest operating frequency

f_{max}: highest operating frequency (where $f_{max} \leq 2.4\, f_{min}$)

f_c: the "center frequency" (wavelength λ_c)

f_o: the "output frequency" (wavelength λ_o)

Other physical parameters are:

- Input radius (radius of the circular waveguide feeding the horn)
- Output radius (radius of the horn aperture)
- Depth of slots/corrugations
- Corrugation pitch and pitch to width ratio
- Type of mode converter
- Horn length
- Horn profiles
- Phase-center position

Discussion of Horn Parameters

Frequency of operation

The definition for narrow or broadband operation is determined as follows. For narrowband operation, $f_{max} \leq 1.4 f_{min}$. The center frequency f_c is the geometric mean of f_{max} and f_{min}.

$$f_c = \sqrt{f_{min} f_{max}} \tag{1}$$

The output frequency f_o is chosen to be $f_c \leq f_o \leq 1.05 f_c$.

Broadband operation is defined by $1.4 f_{min} \leq f_{max} \leq 2.4 f_{min}$. Here $f_c \approx 1.2 f_{min}$ and

$1.05 f_c \leq f_o \leq 1.15 f_c$.

Input Radius

The TE_{11} mode is the fundamental mode in circular waveguide and its cutoff wavenumber is:

$$k = \frac{2\pi}{\lambda} = \frac{1.841}{radius\, of\, circular\, waveguide} \tag{2}$$

The input radius of the horn, a_i satisfies the inequality:

$$\frac{2\pi f_{min}}{c} a_i \geq 1.84118 \tag{3}$$

where c is the speed of light. To obtain a return loss of around 15dB at f_{min} we choose:

$$a_i = \frac{3\lambda_c}{2\pi} \tag{4}$$

Where λ_c is the wavelength at the center frequency.

Output Radius

The radius of the horn opening is chosen to provide the necessary beamwidth for the required application. For example, if the horn is illuminating a subreflector for a Cassegrain antenna, we may require an edge taper of -12 to -18 dB at the edge of the subreflector. Refer to Figure.3 in [6]. This design chart is used to select the radius of the horn aperture for edge tapers of -12, -15 and -18 dB for various half-cone angles. Alternatively, choose an aperture to give you the gain required for your application.

Slot depth Calculation

The corrugations in the inside wall of the horn are nominally λ/4 deep, but for an optimal design we use a correction factor κ which is applied to the dimension for each of the corrugations along the length of the horn. The derivation of the correction factor required solving equations that contain Bessel functions to obtain the optimum surface reactance is beyond the scope of this tutorial but the interested reader should refer to [7]. Here, we will summarise the useful and approximate calculation of κ. This correction factor is applied to each corrugation along the length of the horn and the expression is given below in equation (5).

$$\kappa = \exp\left[\frac{1}{2.114\left(k_c a_j\right)^{1.134}}\right] \tag{5}$$

Where k_c is the cutoff wavenumber and a_j is the radius of horn at the j^{th} corrugation. This correction factor will be used in the equations to calculate the corrugation depths.

Corrugation Pitch and Pitch to Width Ratio

The pitch p is chosen to be $\frac{\lambda_c}{10} \le p \le \frac{\lambda_c}{5}$ and for broadband operation it is usual to choose a pitch closer to $\lambda_c/10$.

The pitch-to-width ratio δ is typically in the range 0.7 to 0.9 (except for the ring-loaded mode converter). It has been reported that this parameter influences the level of cross-polarisation [5], [8].

If the pitch is modified, the length of the horn will also change if the number of corrugations is not modified. This should be kept in mind when designing the horn because a change in length will modify the radiation patterns.

Mode Converter

The mode converter is the part of the horn in between the circular waveguide input and the profiled part of the horn. Its purpose is to smoothly convert the TE_{11} mode into the HE_{11} mode that leads to a symmetrical farfield radiation pattern and low cross-polarization.

The mode converter consists of a few (5 to 12) corrugations where the depth transitions from approximately half a wavelength to a quarter of a wavelength over these few corrugations. The transition should be smooth to keep the reflections low and this contributes to cross-polarisation levels.

When the slot/corrugation depth is half a wavelength it looks like a short circuit or has a low surface reactance, as per the smooth wall of the circular waveguide. As subsequent corrugations follow, the depth becomes shorter until it is nominally one quarter of a wavelength and this now looks like the high reactance surface that is required for a corrugated horn to support the HE_{11} mode.

There have been various types of mode converter reported in the literature, here will discuss those presented in [6]. The three types of mode converter are, (1) variable slot depth, (2) ring loaded slots and (3) variable pitch-to-width mode converters.

There were a number of typographical errors in [6] and an errata was issued, however there was still one further error (which caused a lot of head scratching!). The correct equations to calculate the slot geometry for the three mode converters is given below as well as the equations to generate the subsequent slots along the horn profile. All three mode converters have been included in the openEMS script and their corresponding equations are listed here, but refer to Figure 2 in [6] for the parameter definitions on page 77.

An important paper by Thomas *et al*, [9] presents the design of wideband corrugated horn antennas that use ring-loaded-slots to achieve very low cross-polarisation over a continuous bandwidth of up to 2.1:1.

Variable-Depth Slot Mode Converter

This is the most common mode converter and perhaps the simplest. It is used when

$f_{max} \leq 1.8 f_{min}$.

We let d_j be the depth of the j^{th} slot, N_{MC} is the number of slots in the mode converter, σ is a percentage factor for the first slot depth and is chosen to be between 0.4 and 0.5.

When $1 \leq j \leq N_{MC}$, the depth of the j^{th} slot is:

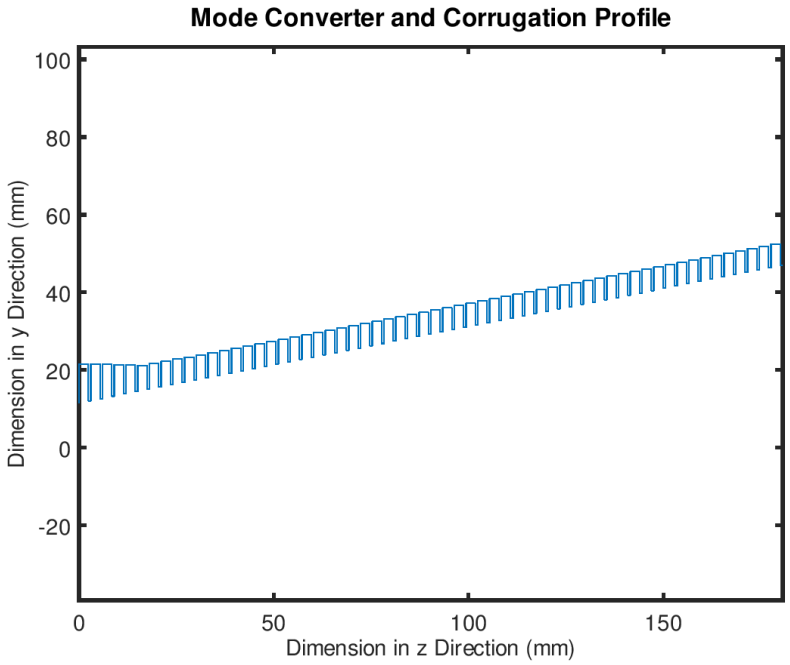
$$d_j = \left\{ \sigma - \left(\frac{j-1}{N_{MC}} \left(\sigma - \frac{1}{4} \exp\left[\frac{1}{2.114 (k_c a_j)^{1.134}} \right] \right) \right) \right\} \lambda_c \tag{6}$$

Let the first slot after the mode converter be $N_{MC}+1$. The total number of slots is N. Then, the depth of the j_{th} slot is given by:

For $N_{MC} + 1 \leq j \leq N$;

$$d_j = \frac{\lambda_c}{4} \exp\left[\frac{1}{2.114 (k_c a_j)^{1.134}} \right] - \left[\frac{j - N_{MC} - 1}{N - N_{MC} - 1} \right] \left\{ \frac{\lambda_c}{4} \exp\left[\frac{1}{2.114 (k_c a_o)^{1.134}} \right] - \frac{\lambda_o}{4} \exp\left[\frac{1}{2.114 (k_o a_o)^{1.134}} \right] \right\} \tag{7}$$

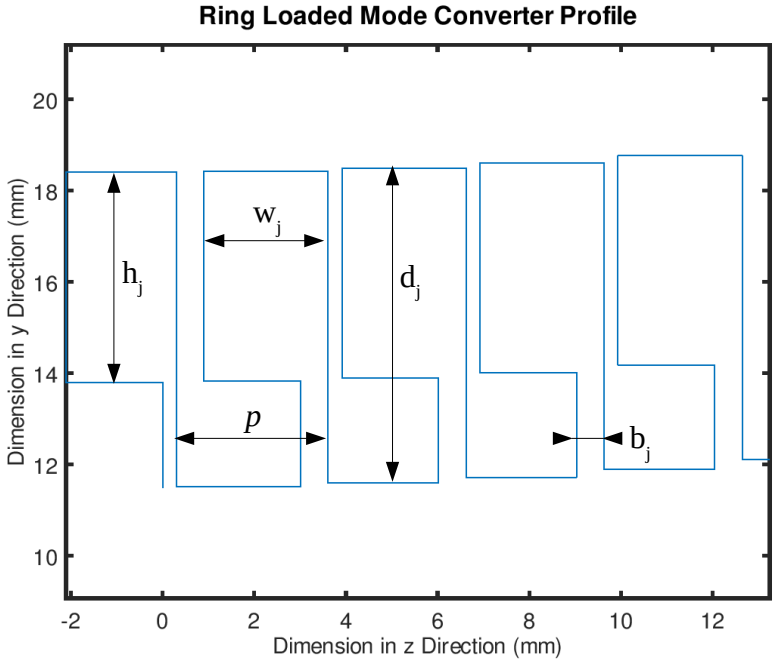
An example of this mode converter and a horn with a linear profile is shown below.



The variable depths of the 5 slot mode converter is followed by the nominally quarter wavelength corrected slot depths for the remaining length of the horn. Later we will see how this profile is completed to form a closed loop that will be used to create the 3D model of the horn.

Ring-Loaded Slot Mode Converter

The ring-loaded-slot mode converter may be used for wide bandwidth horns where $f_{max} \leq 2.4 f_{min}$. Careful consideration should be given to how the mode converter would be manufactured due to the shape of the geometry. The geometry is shown below, refer to [6] to see a more detailed description of the parameters d_j, b_j and h_j. d_j is the depth of the slot, b_j is the width of j^{th} slot at the horn's inside wall, h_j is the depth of the ring and w_j is the width of the j^{th} ring.



For $1 \leq j \leq N_{MC}$ the depth of the j^{th} slot is given by:

$$d_j = \frac{\lambda_c}{4} \exp\left[\frac{1}{2.114(k_c a_j)^{1.134}}\right] \tag{8}$$

For $N_{MC}+1 \leq j \leq N$;

$$d_j = \frac{\lambda_c}{4} \exp\left[\frac{1}{2.114(k_c a_j)^{1.134}}\right] - \left[\frac{j - N_{MC} - 1}{N - N_{MC} - 1}\right]\left\{\frac{\lambda_c}{4} \exp\left[\frac{1}{2.114(k_c a_o)^{1.134}}\right] - \frac{\lambda_o}{4} \exp\left[\frac{1}{2.114(k_o a_o)^{1.134}}\right]\right\} \tag{9}$$

When $1 \leq j \leq N_{MC}$, the width of the b_j^{th} slot is:

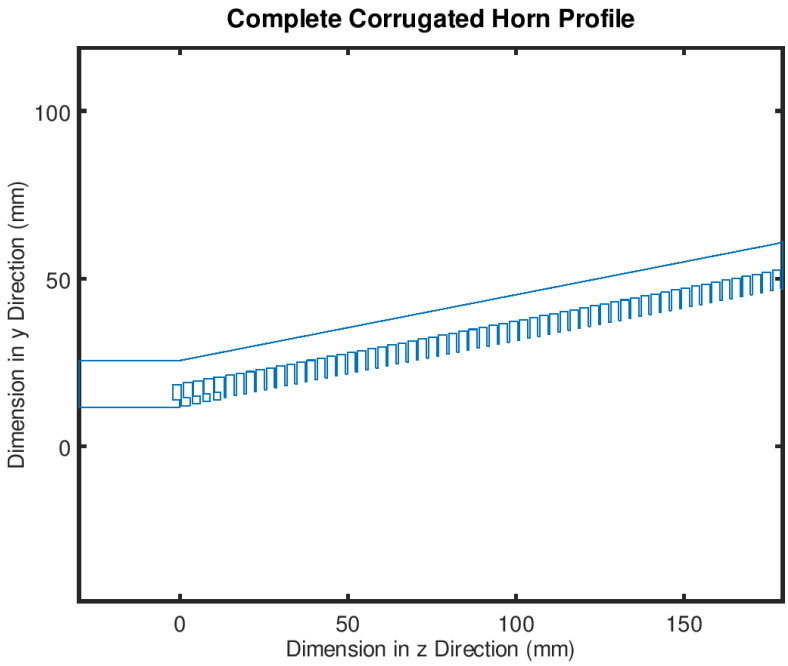
$$b_j = \left[0.1 + (j-1)\frac{\delta - 0.1}{N_{MC}}\right] p \tag{10}$$

The height of the h_j^{th} slot/ring is:

$$h_j = \frac{2}{3} d_j \tag{11}$$

An example of a horn profile with the ring-loaded-slot mode converter is shown here. Care must be taken to ensure that the mode converter is manufacturable. Later, we will see that a secondary delta

value (delta2) is chosen only for this mode converter. If incorrect values for delta2 are chosen, the polyline will self-intersect and this must be avoided.



The mode converter may be fabricated as a series of machined slices that are secured into the main body by screws. Don't forget to include at least two or three dowel pins to ensure proper alignment. There must be enough thickness between the rings for the parts to be fabricated.

Variable-Pitch-to-Width Slot Mode Converter

This mode converter is suitable for $f_{max} \leq 2.05 f_{min}$. It is generated with the following equations. For $1 \leq j \leq N_{MC}$, the depth of the j^{th} slot is given by:

$$d_j = \left[\sigma \frac{\lambda_c}{1.15} + \frac{j-1}{N_{MC}-1} \left(\frac{\lambda_c}{4} - \sigma \frac{\lambda_c}{1.15} \right) \right] \exp\left[\frac{1}{2.114 (k_c a_j)^{1.134}} \right] \tag{12}$$

For $N_{MC}+1 \leq j \leq N$;

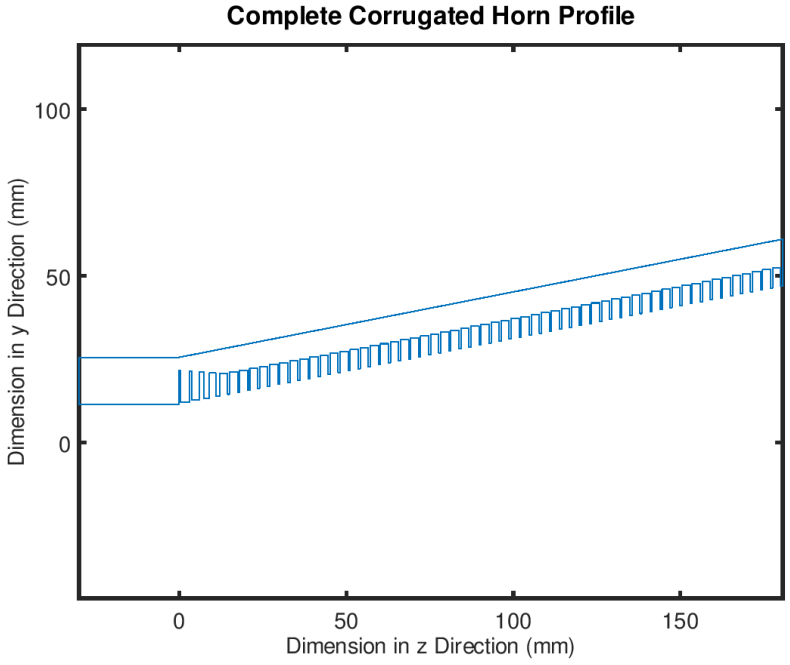
$$d_j = \frac{\lambda_c}{4} \exp\left[\frac{1}{2.114 (k_c a_j)^{1.134}} \right] - \left[\frac{j - N_{MC} - 1}{N - N_{MC} - 1} \right] \left\{ \frac{\lambda_c}{4} \exp\left[\frac{1}{2.114 (k_c a_o)^{1.134}} \right] - \frac{\lambda_o}{4} \exp\left[\frac{1}{2.114 (k_o a_o)^{1.134}} \right] \right\} \tag{13}$$

When $1 \leq j \leq N_{MC}$, the width of the w_j^{th} slot is:

$$w_j = \left[\delta_{min} + \frac{j-1}{N_{MC}-1} (\delta_{max} - \delta_{min}) \right] p \tag{14}$$

Where σ is chosen to be between 0.4 and 0.5.

$0.125 \leq \delta_{min} \leq \delta$ and $\delta_{max} = \delta$ which is the nominal pitch to width ratio. The mode converter can be seen in the example shown below. The width and pitch both change along the length of the mode converter.



Choice of Horn Length

There are no stringent rules to set the length of the horn. It largely depends on the required sidelobe levels, phase center stability, practical limitations for the application and the cost and method of manufacture. The minimum length should be at least 5 wavelengths at the cutoff frequency. Longer horns are usually made from multiple sections that are screwed together. It's important that dowel pins are used to align the sections.

Profile of the Horn

There are numerous profiles for the shape of the corrugated horn [10]. Some of the most popular profiles are the linear, sinusoid, asymmetric sine-squared and the hyperbolic functions. The profile formulations [6] are repeated here because they are all included as options in the openEMS script so that the reader can experiment and chose the best profile for their application. *Granet* [6] suggests that parameter p in some of the formulae is usually chosen to equal 2, but values in the range 0.5 to 5 can be experimented with. The formulae for the horn profiles are listed below.

(1) Linear

$$a(z)=a_i + (a_o-a_i)\frac{z}{L} \tag{15}$$

(2) Sinusoid

Let $A\in[0;1]$

$$a(z)=a_i + (a_o-a_i)\left[(1-A)\frac{z}{L} + A\sin^p\left(\frac{\pi z}{2L}\right)\right] \tag{16}$$

(3) Asymmetric sine-squared

$L = L_1 + L_2$ and $\gamma = L_2/L_1$.

For $0 \le z \le L_1$;

$$a(z) = a_i + \frac{2(a_o-a_i)}{1+\gamma}\sin^2\left(\frac{\pi z}{4L_1}\right) \tag{17}$$

For $L_1 \le z \le L$;

$$a(z)=a_i+\frac{2(a_o-a_i)}{1+\gamma}\left\{\gamma\sin^2\left[\frac{\pi(z+L_2-L_1)}{4L_2}\right]+\frac{1-\gamma}{2}\right\} \tag{18}$$

(4) Tangential

Let $A \in [0;1]$

$$a(z) = a_i + (a_o - a_i)\left[(1-A)\frac{z}{L} + A\tan^p\left(\frac{\pi z}{4L}\right)\right] \tag{19}$$

(5) *xp*

Let $A \in [0;1]$

$$a(z) = a_i + (a_o - a_i)\left[(1-A)\frac{z}{L} + A\left(\frac{z}{L}\right)^p\right] \tag{20}$$

(6) Exponential

$$a(z) = a_i \exp\left[\ln\left(\frac{a_o}{a_i}\right)\frac{z}{L}\right] \tag{21}$$

(7) Hyperbolic

$$a(z) = \sqrt{\left(a_i^2 + \frac{(z^2(a_o^2 - a_i^2))}{L^2}\right)} \tag{22}$$

(8) Polynomial

$$a(z) = a_i + (p+1)(a_o - a_i)\left[1 - \frac{pz}{(p+1)L}\right]\left[\frac{z}{L}\right]^p \tag{23}$$

Phase Center

The phase center of the horn is the position along the axis of the horn that produces minimum variation in phase across the aperture. Another way to describe the phase center, is the point inside the horn where a spherical wave originates from. The phase center may be computed numerically from the field quantities, experimentally or it may be estimated using the formula below (24). The distance inside from the aperture is αL, where L is the length of the horn and α is a value between 0 and 1. An approximation of α is:

$$\alpha = 1 - \exp\left[-4.8\left(\frac{k_c a_o^2}{4\pi L}\right)^2\right] \tag{24}$$

The phase center of a standard horn with unequal E-plane and H-plane beamwidths will be at different locations, and the phase-centre is often referred to as a “fuzzy” location. For a horn with equal beamwidths in the E-plane and the H-plane, the phase center will be better defined at a single point, or at least, a less “fuzzy” location.

If designing a feed horn for a reflector antenna, its phase center must be known. A well designed corrugated horn will have a phase center that is almost constant with frequency.

Finally we will reproduce the example given in [6] and write a Matlab/Octave script to generate the horn profile as a 2D closed loop which will be used by CSXCAD to create a 3D solid horn model to be simulated with openEMS. CSXCAD is the simulation geometry software that comes with openEMS and can be used to view the model, ports and dump boxes and dump planes where the field quantities will be recorded. Each section of the script will be explained and the principles described here will be applicable to designing your own corrugated horn antenna or writing scripts in Matlab or Octave to simulate other types of horn antenna with openEMS.

The following software was used:

Windows 7

openEMS 64bit v0.0.35-42-g6dfc05e

GNU Octave 4.4.1 (and version 5.1.0)

ParaView 5.4.1

Example

We will use the example of a corrugated horn operating at Ku band for a Cassegrain reflector antenna. The standard frequency range is 10.7 to 12.75 GHz for the receive band, and 14.0 to 14.5 GHz for the transmit band. The half-subtended angle from the antenna to the subreflector is 20° and the required edge taper is chosen to be -15 dB.

f_{min} =10.7 GHz

f_{max} = 14.5 GHz

f_{max} = 1.36f_{min}

The geometric mean is chosen as the center frequency and this is calculated with:

$$f_c = \sqrt{f_{min} f_{max}} \quad = 12.46 \text{ GHz}$$

f_o = 1.02f_c = 12.71 GHz

The input radius $a_i = \frac{3\lambda_c}{2\pi}$ = 11.49 mm. From Figure 3 in [6],

$a_o \approx 1.95\lambda_c$ = 46.92 mm.

The pitch $p = \lambda_c/8 \approx 3$ mm and the pitch-to-width ratio $\delta = 0.8$, therefore the slots will be 2.4 mm wide and the teeth will be 0.6 mm wide.

As per the example, we will choose a hyperbolic profile. First the "stepped" profile of the horn is defined by N steps of length p by using the following algorithm.

$Z_{step} = Np$ / (N-1) = L / (N-1)

do j = 1 to N

$z = (j-1)z_{step}$

Where z is the discrete increment of length in the direction of the horns axis. For every value of z, the horn's radius (a_j) will be calculated and this will depend on which profile is chosen. The hyperbolic function is chosen in this example (option 7). The appropriate mode converter equation is selected to calculate slot depth d_j with every associated a_j for the number of corrugation slots chosen for the mode converter.

The geometry becomes defined by a set of N doublets of circular cross sections as explained on p83 [6], also shown here.

n = 0

do j = 1 to N

$length_n = \delta p$

$radius_{n+1} = a_j$

$length_{n+1} = (1-\delta)p$

$n = n + 1$

end do

After the profile is defined we can add a short length of smooth circular waveguide to the throat of the horn and define an outer profile to form a closed loop as will be shown later. The closed loop is also known as a polyline since it is a shape made up from many lines or segments that lie between each node. **See the end of the document for an alternate method to compute the cross sections.**

openEMS Script

The script consists of two main sections, the first being the variable definitions and the generation of the horn profile. The second section defines the openEMS simulation setup using the data generated in the first section. The filename for the complete script is:

Corrugated_horn_with_mode_converter_options_MASTER.m

The following pages will cover each line or group of lines in the script and will explain how to perform the calculations that we have already discussed in the preceding pages.

The lines of code that appear in the text will be in the order that they appear in the simulation script and the full listing will be included in the appendix.

The first three lines will close any figure windows, all user-defined variables (local and global) are cleared from the symbol table and clear the terminal window:

```
close all
clear
clc
```

It is advised to add these commands to any openEMS script so that each time a simulation is started, all previous data in Matlab/Octave is cleared.

The next two lines are flags that control whether or not the simulation is run and whether or not CSXCAD will be opened to inspect the physical model, ports, field dump boxes and mesh:

```
RUN_SIMULATION = 1;
RUN_CSXCAD = 1;
```

Sometimes we want to run the script without the simulation starting when we want to check the geometry or mesh and not waste time running the simulation if it's not setup properly. The same applies to CSXCAD. If this is the case we set one or both flags to 0. Set to 1 to run the simulation or to view the model in CSXCAD. CSXCAD is the included model, port and mesh viewer that comes with openEMS.

With CSXCAD you can select what to view, and is useful to make sure that the mesh coincides with important details of the physical model, the ports, dump boxes and planes that are used to store fields. You may also check that the model and dump boxes are not inside the absorbing boundaries. Some transmission lines may extend into the absorbing boundary, but ports must not be defined inside the PML or MUR absorbing boundaries. Radiating structures should be at least one quarter of a wavelength (at the lowest frequency) away from the inner mesh lines of the absorbing boundary.

Next comes the user editable parameters that define the frequency of operation, physical and dimensional variables. The comments are preceded with the % symbol. To help with readability, all comments are in green. Comments do not form any function other to help the reader and are ignored by the software.

```
% USER EDITABLE PARAMETERS
fmin = 10.7                % Minimum frequency in GHz
fmax = 14.5                % Maximum frequency in GHz
pitch_fraction = 8  % Choose a fraction between 10 to 5 (lambda_c / pitch_fraction)
delta = 0.8                % Pitch to width ratio 0.7 to 0.9
sigma = 0.42               % Percentage factor for first slot depth, 0.4 to 0.5
NMC = 5                    % Number of corrugations in mode converter
wgl = 30;                  % Length of circular feeding waveguide
num_of_corrugations = 60;
% delta2 is only used for case 2, ring loaded slot mode converter
delta2 = 0.2;

% delta_min is only used for case 3,variable pitch to width slot mode converter
delta_min = 0.125;           % Should be greater than 0.125 and less than delta

% Choose profile, 1=LINEAR, 2=SINUSOID, 3=ASYMMETRIC SINE-SQUARED, 4=TANGENTIAL,
% 5=x.rho, 6=EXPONENTIAL, 7=HYPERBOLIC, 8=POLYNOMIAL
horn_profile = 7;          % Must be an integer from 1 to 8

% Choose the type of mode converter, 1=VARIABLE SLOT DEPTH MC,
% 2=RING LOADED SLOTS MC, 3=VARIABLE PITCH TO WIDTH SLOT MC
mode_converter_type = 1;   % Must be an integer from 1 to 3

% END OF USER EDITABLE PARAMETERS
```

The minimum and maximum frequencies are defined, followed by the denominator of the lambda_c/pitch_fraction. Here we choose the denominator to be 8, but any value from 10 to 5 may be chosen. A *delta* value of 0.8 is chosen as the pitch-to-width ratio. *sigma* is set to 0.42 but can be set to 0.4 to 0.5. *NMC* is the number of slots of the mode converter. This is an integer, typically a value from 5 to 12. The smooth circular waveguide preceding the mode converter is defined as *wgl*, here it's set to 30 mm. It should be set to a sensible length, greater than one quarter of a wavelength at the lowest frequency. It's not critical but there should be enough length to add the port box with a length of a few mesh lines.

num_of_corrugations is the total number of corrugations along the length of the horn, the more corrugations, the longer the horn will be. *delta_2* is a secondary delta value used only when the ring-loaded-slot mode converter is selected (case 2).

delta_min is a secondary delta value that is used only when the variable-pitch-to-width mode converter is chosen (case 3). It should be greater than 0.125 and less than *delta*.

The next two definitions are important, they select the type of horn profile and type of mode converter. There are 8 options for the horn profile, as explained in the text. The option number is used in a *case* statement to select the appropriate formula for the profile. There are 3 options for the mode converter and the appropriate formulation is chosen using a *case* statement. The comments state which number corresponds to which profile. This marks the end of the user defined parameters that define the profile of the horn.

The next lines use the variables that we have just entered to perform some calculations.

```
% Calculate center frequency fc based on narrow or wide bandwidth.
fratio = fmax/fmin % ratio of fmax/fmin
if (fratio >= 2.4);            % check fmax/fmin is less than 2.4
  disp('Error, fmax/fmin is greater than 2.4!');
  fc = 0
  elseif (fratio <= 1.4);      % Use Narrowband formula if fmax <= 1.4fmin
    fc = sqrt(fmin*fmax)
     elseif (fmax >= 1.4*fmin && fmax <= 2.4*fmin); % Use wideband formula for
       fc = 1.2*fmin                                  1.4fmin<=fmax<=2.4fmin
endif

if (fratio <= 1.4);
  fo = 1.02*fc                  % For narrowband choose fc <= fo <= 1.05fc
    elseif (fmax >= 1.4*fmin && fmax <= 2.4*fmin);
      fo = 1.10*fc              % For wideband choose 1.05fc <= fo <= 1.15fc
endif

unit = 1e-3;                   % Units in mm
lambda_c = 300/fc              % Center frequency wavelength
lambda_o = 300/fo              % Output frequency
depth_nominal = lambda_c/4     % Nominal slot depth at center frequency
ai = (3 * lambda_c)/(2*pi)     % Radius of input waveguide in mm
ao = 1.95*lambda_c             % Radius of output waveguide in mm
p = lambda_c/pitch_fraction;   % Pitch in mm, lambda_c/10 to lambda_c/5
length = num_of_corrugations*p % Length of horn profile
N = length/p                   % Total number of corrugations
kc = (2*pi)/lambda_c           % Wave number at center frequency
ko = (2*pi)/lambda_o           % Wave number at output frequency
z = 0:p:length;                % z index distance array from 0 to length of horn
r_app = ao*1e-3;               % Aperture radius
A_app = pi*(r_app)^2;          % Aperture area for gain calculation
```

The ratio of the maximum to the minimum frequency is calculated and is used in the *if-ifelse* statement to determine which formula is used to calculate the center frequency. First, there is a check that the frequency ratio is not greater than 2.4. If it is, an error message will be displayed. If it's less or equal to 2.4, the ratio is checked to determine if it is less or equal to 1.4. If it is, the narrowband formula is used (the geometric mean), if it's greater than 1.4, then the wideband formula is used to calculate f_c. This value of f_c is then used with a similar *if* statement to calculate the appropriate value for the output frequency f_o.

Next, a number of parameters are calculated, these are commented for convenience. Note that *z* is the vector that holds the index value for the z distance along the horn's axis and is used to calculate the profile dimensions. The incremental distance is equal to the pitch *p*.

r_app is the radius of the aperture and is used in the equation to calculate the area of the aperture *A_app*. This is used to calculate the gain and efficiency of the horn.

The following section is where the horn profile is calculated and a plot is displayed with an equal aspect ratio (1:1) so that a true representation of the profile can be seen. Each profile type will be shown with an image of the plot so that you can see what the internal profile will look like.

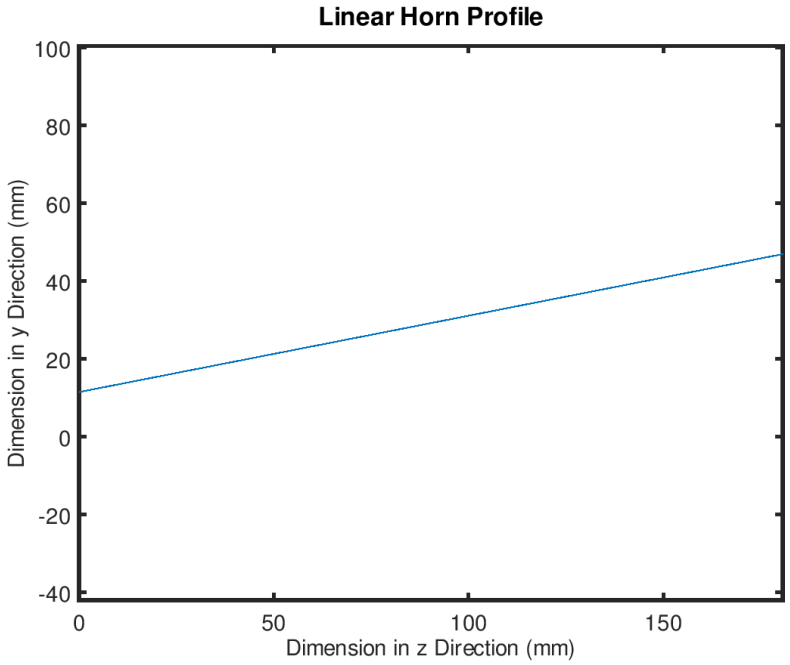
Linear Profile

```
% One of the following horn profiles will be selected depending on value of
% horn_profile chosen above.
switch(horn_profile)
   case 1
   %%% Linear profile %%%
   a = ai+(ao-ai)*z/length;

   plot(z, a);
   set(gca, "linewidth",2, "fontsize", 14 )
   axis equal;
   xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
   ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
   title( 'Linear Horn Profile', 'FontSize', 16 );
```

The *switch* statement checks the value of the variable in the brackets and will execute the corresponding *case* statement. Here you can see how the formula for the linear profile is entered in Matlab/Octave. The radius is calculated for each value of z that was calculated earlier. This creates a single row array called a vector that contains a value for every increment of z and these two vectors are plotted with *plot(z, a);* and the following lines format the plot axis font and labels.

Note that the term "vector" in Matlab/Octave refers to a single row or column array, and not a vector in the mathematical sense defining a quantity with magnitude and direction.

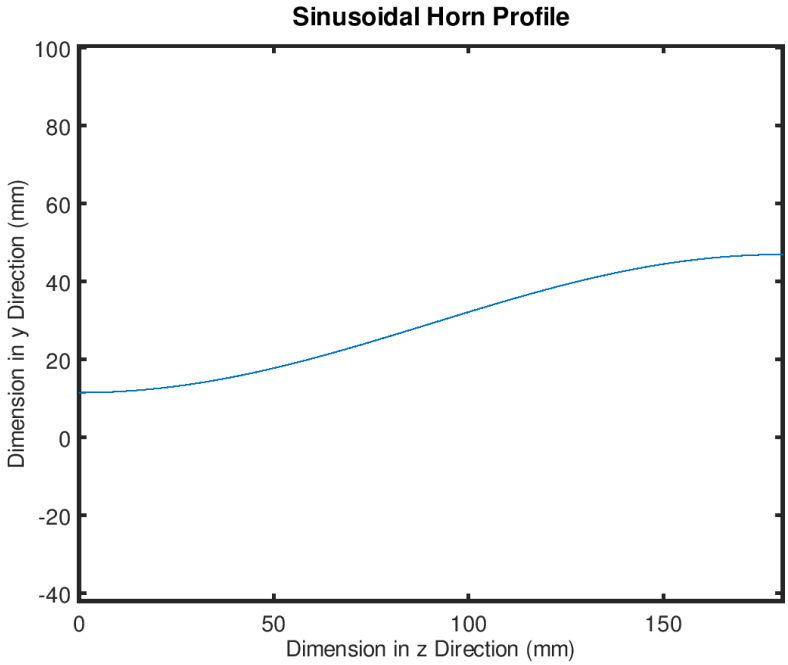
These lines will display the following plot of the linear profile.



Sinusoidal Profile

```
case 2
%%% Sinusoid profile %%%
A = 1;       % Amplitude factor 'A' should be between 0 and 1
rho = 2;     % rho should be between 0.5 and 5, default is 2
a = ai+(ao-ai)*((1-A)*(z/length)+A*power(sin((pi*z)/(2*length)),rho));

plot(z, a);
set(gca, "linewidth",2, "fontsize", 14 )
axis equal;
xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
title( 'Sinusoidal Horn Profile', 'FontSize', 16 );
```

In this case we write the formula for generating a sinusoidal profile and there are two additional parameters that control the shape. These parameters are *A*, which is the sinusoidal amplitude factor and *rho* which is the power value. Try running the script (without running the simulation) with different values to see the effect of changing these values.

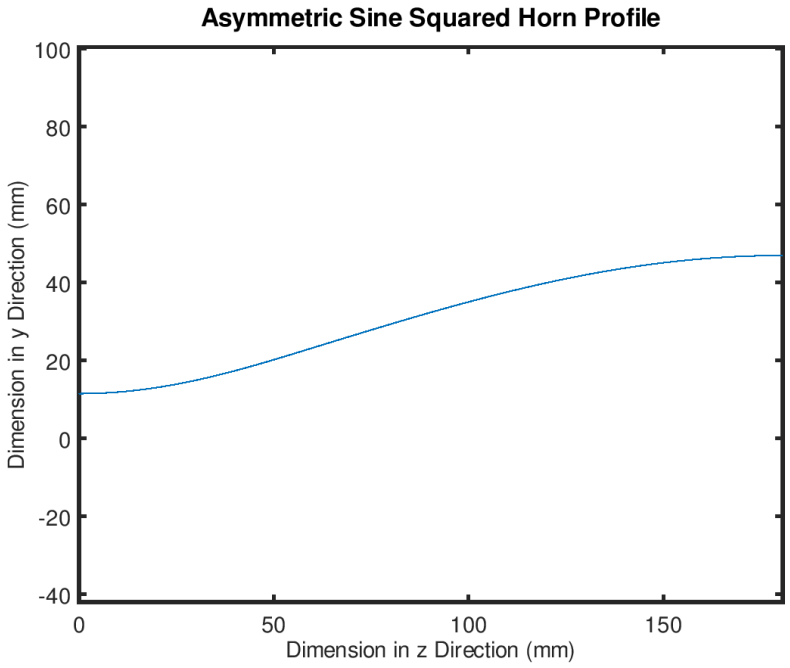


As parameter *A* is reduced, the shape becomes more linear. Increasing *rho* will generate a taper that will have a more linear region near the mode converter before flaring out.

Asymmetric Sine Squared Profile

```
case 3
   %%% Asymmetric Sine-Squared profile %%%
   L1 = length/3;     % Choose a value for L1, must be less than the horns length
   L2 = length-L1;    % L2 is the length between L1 and the end of the horn
   gamma = L2/L1;
   idx = find(z <= L1)      % Find the index of z corresponding to L1
   zelements = size(z,2)    % Total number of points in z axis of the horn

   za = z(1: max(idx))
   aa = ai+((2*(ao-ai))/(1+gamma))*sin((pi*za)/(4*L1)).^2;

   zb = z(max(idx)+1 : zelements)
   ab = ai+((2*(ao-ai))/(1+gamma))*(gamma*sin(((pi*(zb+L2-L1))/(4*L2))).^2+((1-gamma)/
2));

   a = [aa,ab];
   z = [za,zb];

   plot(z, a);
   set(gca, "linewidth",2, "fontsize", 14 )
   axis equal;
   xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
   ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
   title( 'Asymmetric Sine Squared Horn Profile', 'FontSize', 16 );
```

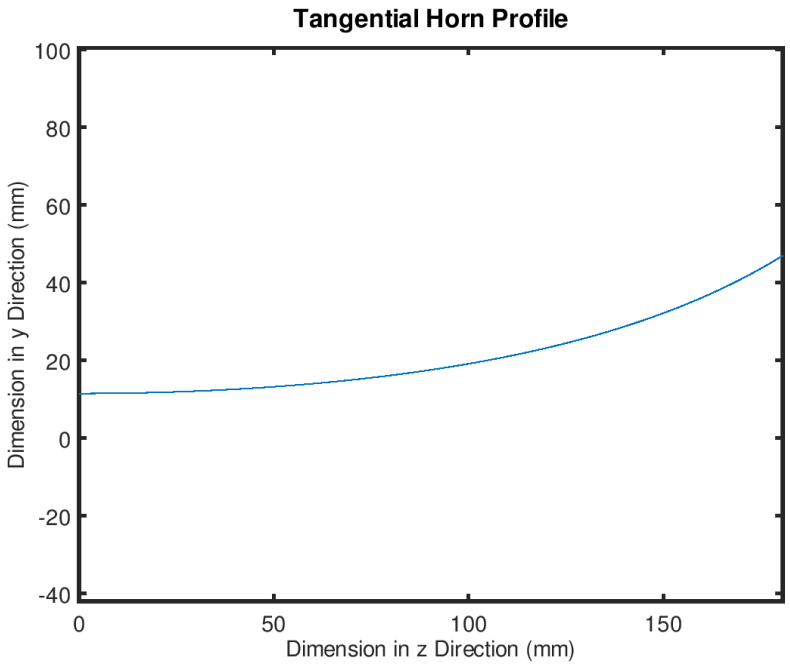
This is similar to the Sinusoidal profile with a different method to control the profile. The difference here is that two different sinusoids may be defined up to a particular length along the horn. The control parameters are *L1* and *L2*. The user only specifies *L1,* and *L2* is adjusted according to the length of the horn. The two sets of coordinates are combined into an *a* and a *z* vector which is then plotted.



Tangential Profile

```
case 4
   %%% Tangential profile %%%
   A = 1;
   rho = 2;
   a = ai+(ao-ai)*((1-A)*(z/length)+A*power(tan((pi*z)/(4*length)),rho));

   plot(z, a);
   set(gca, "linewidth",2, "fontsize", 14 )
   axis equal;
   xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
   ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
   title( 'Tangential Horn Profile', 'FontSize', 16 );
```

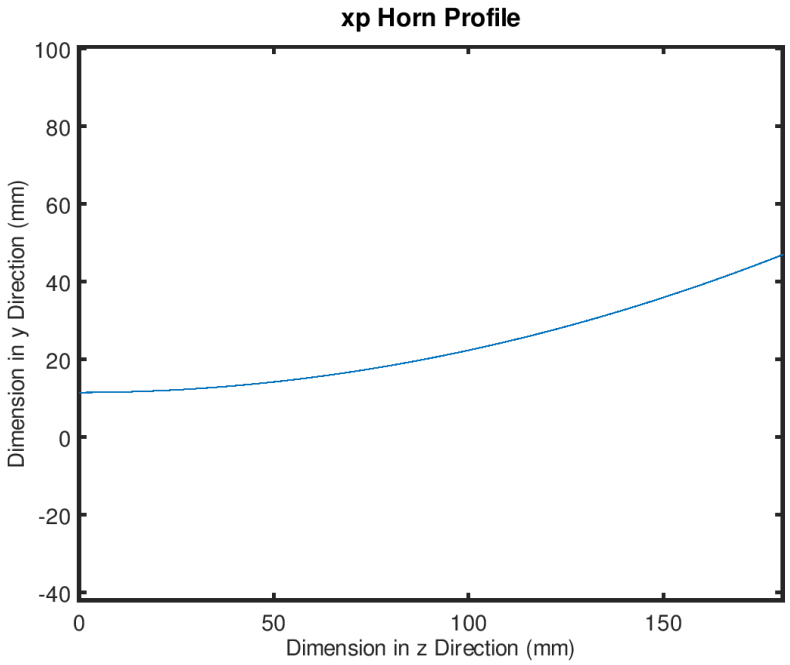
As the name suggests, this profile generates a tangential curve where the amplitude of the tangential component by parameter *A* and *rho* is the power factor as per the sinusoidal profile. As *A* approaches zero, the profile becomes more linear, as *A* is increased, the horn profile resembles that of a trumpet. An example of the tangential profile is shown below.



xp Profile

```
case 5
   %%% x.rho profile %%%
   A = 1;
   rho = 2;
   a = ai+(ao-ai)*((1-A)*(z/length)+A*power(z/length,rho));

   plot(z, a);
   set(gca, "linewidth",2, "fontsize", 14 )
   axis equal;
   xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
   ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
   title( 'xp Horn Profile', 'FontSize', 16 );
```

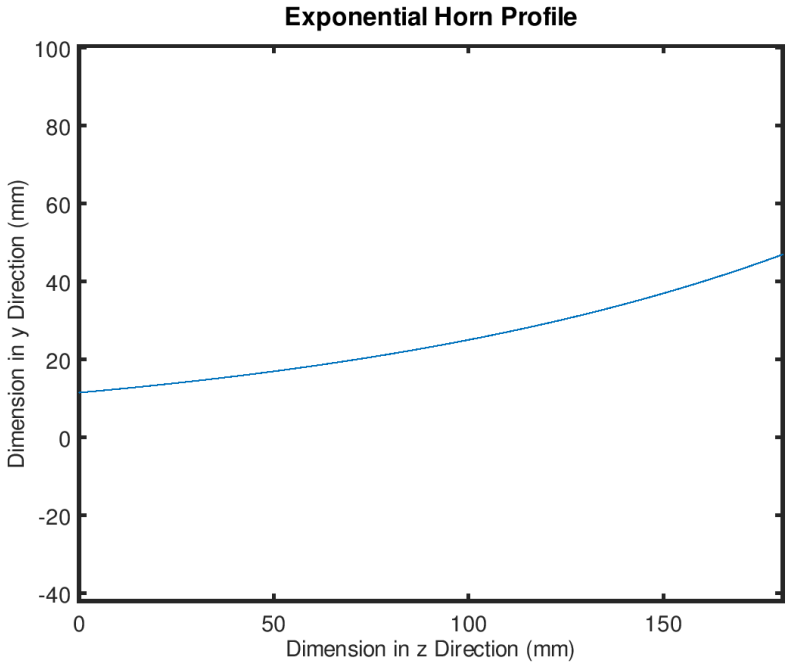
This is similar to the tangential profile, with parameters *A* to control the power amplitude and *rho* controls the rate of increase of the horns diameter along the horn's axis.



Exponential Profile

```
case 6
%%% Exponential profile %%%
a=ai*exp(log(ao/ai)*(z/length));

plot(z, a);
set(gca, "linewidth",2, "fontsize", 14 )
axis equal;
xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
title( 'Exponential Horn Profile', 'FontSize', 16 );
```

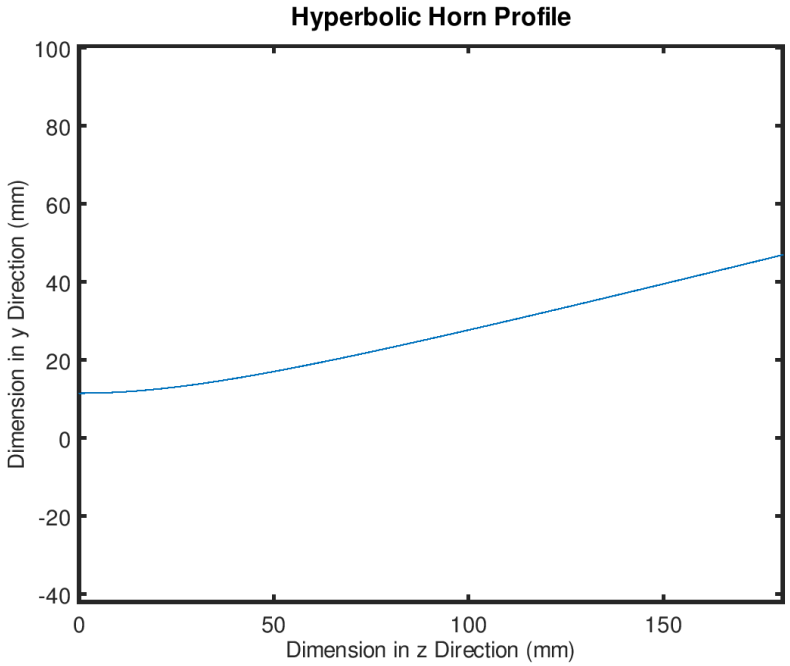
The exponential profile is simple an exponential taper along the length of the horn with the shape determined by the dimensions of the horn.



Hyperbolic Profile

```
case 7
%%% Hyperbolic profile %%%
a = sqrt(ai^2 + (power(z,2) * (ao^2-ai^2) / length^2));

plot(z, a);
set(gca, "linewidth",2, "fontsize", 14 )
axis equal;
xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
title( 'Hyperbolic Horn Profile', 'FontSize', 16 );
```

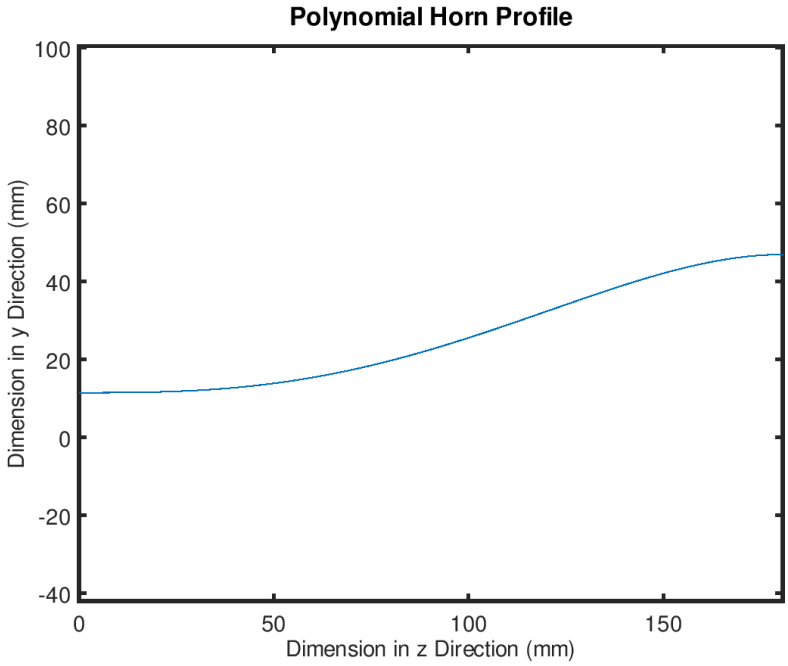
The hyperbolic profile is determined by the horn dimensions and is a popular choice for corrugated horns.



Polynomial Profile

```
case 8
%%% POLYNOMIAL Profile %%%
rho = 2;
a=ai+(rho+1)*(ao-ai)*(1-((rho*z)/((rho+1)*length))).*power(z/length,rho);

plot(z, a);
set(gca, "linewidth",2, "fontsize", 14 )
axis equal;
xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
title( 'Polynomial Horn Profile', 'FontSize', 16 );
```

The final profile is the polynomial curve with the control parameter *rho* which is the polynomial power. By adjusting this parameter the order of the polynomial will generate interesting curves that have been reported to have been used for corrugated horns. Setting *rho* to equal 3, the following profile is generated.



Next we will take a look at the script used to generate the three mode converter profiles.

Another *switch* statement is used and it acts upon the integer 1, 2, or 3 that is stored in the variable name *mode_converter_type*. In this case it is 1, for the variable slot depth mode converter. First a vector *ajmc* is created to store the radius value for slots 1 to the number of corrugations in the mode converter *NMC*. Then an index vector *idx* is created to keep track of the corrugation number, from 1 to *NMC*. *djmc* is the depth of the j^{th} slot.

After the mode converter radii and mode converter points are calculated, the rest of the horn's internal wall radii a_j and slot dimensions d_j are determined. a_j is simply the value of the profile radius that was calculated earlier and we now take each radius value, beginning with the first radius step that comes after the last slot of the mode converter. This is a_j. Vector *idx* is the index vector that is used when calculating the rest of the slot depths along the length of the profile. The depth values for the mode converter and the rest of the profile are combined into one vector *d*.

```
switch(mode_converter_type)

  case 1                              % case 1=VARIABLE SLOT DEPTH MODE CONVERTER
    % Mode Converter depths for element j
    ajmc = a(1:NMC);                      % Index range for mode converter
    idx = 1:NMC;
    djmc =
(sigma-((idx-1)./NMC).*(sigma-(0.25.*exp(1./(2.114.*(kc*ajmc).^1.134))))*lambda_c
    % Depth of remaining corrugations
    aj = a(NMC+1:end);
    idx = NMC+1:N+1;
    dj = ((lambda_c/4).*exp(1./(2.114.*(kc*aj).^1.134)))-((idx-NMC-1)/(N-NMC-
1)).*((lambda_c/4).*exp(1./(2.114.*(kc*aj).^1.134))-(lambda_o/4).*exp(1./
(2.114.*(ko*ao).^1.134)))
    d = [djmc, dj];       % Combining the mode converter and horn depth values
```

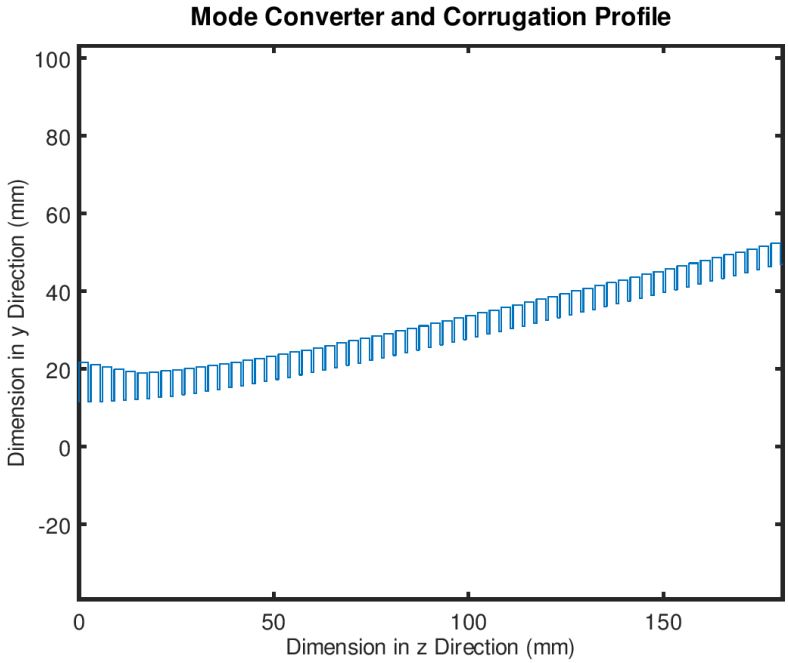
The remaining lines in case 1 are used to generate the length *len* and radius *rad* coordinates. The first two length values are set to equal zero and a *for* loop steps through the algorithm to calculate both the radius and length points for the polygon that will define the corrugated horn profile. *n* is used as a local variable for the *rad* and *len* index value. *z_number* is the total number of points that define the internal profile and the vector *len* is truncated to go from the first element of vector *len*, to the value *z_number*.

Finally, the *len* and *rad* vectors defining the profile are plot as a polyline with the axis set to equal to maintain the correct aspect ratio.

```
    % Generate z,y coordinates as len,rad vector
    n = 0;
    len(1) = 0;
    len(2) = 0;
      for i = 1:N;
       rad(i+n) = a(i);
       rad(i+n+1) = a(i)+d(i);
       rad(i+n+2) = a(i)+d(i);
       rad(i+n+3) = a(i+1);
       rad(i+n+4) = a(i+1);
       len(i+n+2) = len(i+n)+delta*p;
       len(i+n+3) = len(i+n+2);
       len(i+n+4) = len(i+n+3)+(1-delta)*p;
       len(i+n+5) = len(i+n+4);
       n = n+3;
      endfor

    z_number = (N*4)+1; % Number of coordinate points for corrugated length of horn
    len = len(1:z_number); % Truncate z axis data points to equal rad vector length

    figure
    plot(len,rad);
    set(gca, "linewidth",2, "fontsize", 14 )
    xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
    ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
    title( 'Mode Converter and Corrugation Profile', 'FontSize', 16 );
    axis equal;   % Scale axis equally for aspect ratio 1:1
```

This brings us to the end of *case 1* for the variable-slot-depth mode converter. Depending on the chosen profile, we now have the internal profile for the horn. Before we get to form a closed polyline, the other two mode converter options will be described. Since the example uses the hyperbolic profile, the following internal profile is generated up to this point.

Hyperbolic horn profile with the variable-slot-depth mode converter.



The following lines for *case 2* construct the ring-loaded-slot mode converter. As per *case 1*, we setup the index vector for the j^{th} radius of the mode converter up from 1 to NMC *ajmc* and a vector for the index for the slot number *idx*. Then calculate the mode converter depth values *djmc*, width *bj* and height of the n^{th} ring *hj*. The depth of the remaining slots/corrugations are similar to the other two mode converters and *d* is the combined vector containing the depths of the mode converter and the rest of the horn's slots.

```
case 2                              % case 2=RING LOADED SLOT MODE CONVERTER
      % Mode Converter depths for element j
      ajmc = a(1:NMC);                  % Index range for mode converter
      idx = 1:NMC;
      djmc = (lambda_c/4).*exp(1./(2.114.*(kc*ajmc).^1.134));
      % Width of bjth slot for mode converter
      bj = (0.1+(idx-1).*((delta2-0.1)./NMC)).*p;
      % Height of hjth slot for mode converter
      hj = (2/3).*djmc;

      % Depth of remaining corrugations
      aj = a(NMC+1:end);
      idx = NMC+1:N+1;
      dj = ((lambda_c/4).*exp(1./(2.114.*(kc*aj).^1.134)))-((idx-NMC-1)/(N-NMC-
1)).*((lambda_c/4).*exp(1./(2.114.*(kc*aj).^1.134))-(lambda_o/4).*exp(1./
(2.114.*(ko*ao).^1.134)));
      d = [djmc, dj];       % Combining the mode converter and horn depth values
```

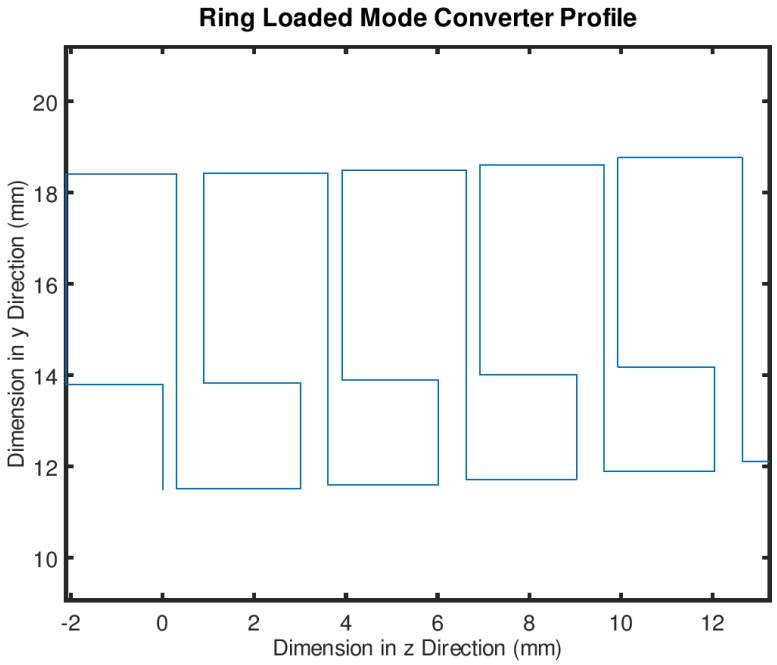
The first 6 coordinates for the first ring-loaded slot are setup before the *for* loop which generates the rest of the slots up to *NMC*, that is why the loop starts with $i = 2$. It was simpler to setup the first ring coordinates this way although it may be possible to calculate all points within the *for* loop.

```
      % Generate z,y coordinates as len,rad vector
      n = 5;
      len = [0, 0, (-delta*p)+bj(1), (-delta*p)+bj(1), bj(1), bj(1)];
      rad = [a(1), a(1)+d(1)-hj(1), a(1)+d(1)-hj(1), a(1)+d(1), a(1)+d(1), a(2)];
        for i = 2:NMC;
          rad(i+n) = a(i);
          rad(i+n+1) = a(i)+d(i)-hj(i);
          rad(i+n+2) = a(i)+d(i)-hj(i);
```

```
    rad(i+n+3) = a(i)+d(i);
    rad(i+n+4) = a(i)+d(i);
    rad(i+n+5) = a(i+1);
    len(i+n) = i*p-p;
    len(i+n+1) = i*p-p;
    len(i+n+2) = len(i+n+1)-(delta*p)+bj(i);
    len(i+n+3) = len(i+n+1)-(delta*p)+bj(i);
    len(i+n+4) = len(i+n+3)+(delta*p)+bj(i);
    len(i+n+5) = len(i+n+3)+(delta*p)+bj(i);
    n = n+5;
  endfor
% Add extra coordinate points before remaining corrugations
len(NMC*(NMC+1)+1) = len(NMC*(NMC+1))+(1-delta)*p;
rad(NMC*(NMC+1)+1) = a(NMC+1);
```

This completes the generation of the ring-loaded slot and the geometry is plot as a polyline in a figure as shown here.

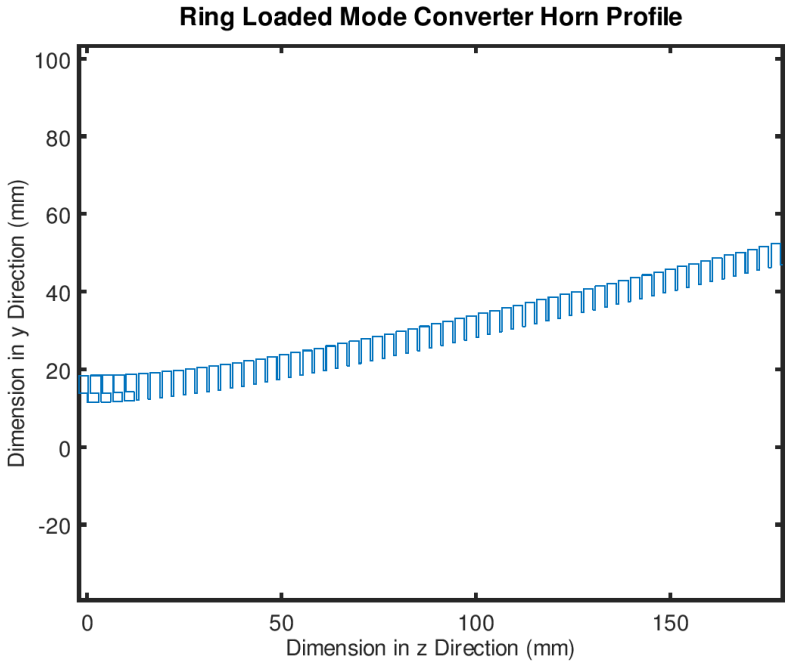
```
figure
plot(len,rad);
set(gca, "linewidth",2, "fontsize", 14 )
xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
title( 'Ring Loaded Mode Converter Profile', 'FontSize', 16 );
axis equal;   % Scale axis equally for aspect ratio 1:1
```



When designing the mode converter, make sure that it can be manufactured!

The remaining slots of the horn are generated here and is plot in another figure. The single value vector, *z_number* stores the number of points generated for the length of the horn up to this point and will be used later.

```
n = n+NMC+1
for i = NMC+1:N;
  rad(n) = a(i);
  rad(n+1) = a(i)+d(i);
  rad(n+2) = a(i)+d(i);
  rad(n+3) = a(i+1);
  rad(n+4) = a(i+1);
  len(n+1) = len(n);
  len(n+2) = len(n+1)+delta*p;
  len(n+3) = len(n+2);
  len(n+4) = len(n+3)+(1-delta)*p;
  n = n+4;
endfor

z_number = (NMC*2)+(N*4)+1; % Number of coordinate points for corrugated length of horn

figure
plot(len,rad);
set(gca, "linewidth",2, "fontsize", 14 )
xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
title( 'Ring Loaded Mode Converter Horn Profile', 'FontSize', 16 );
axis equal;   % Scale axis equally for aspect ratio 1:1
```

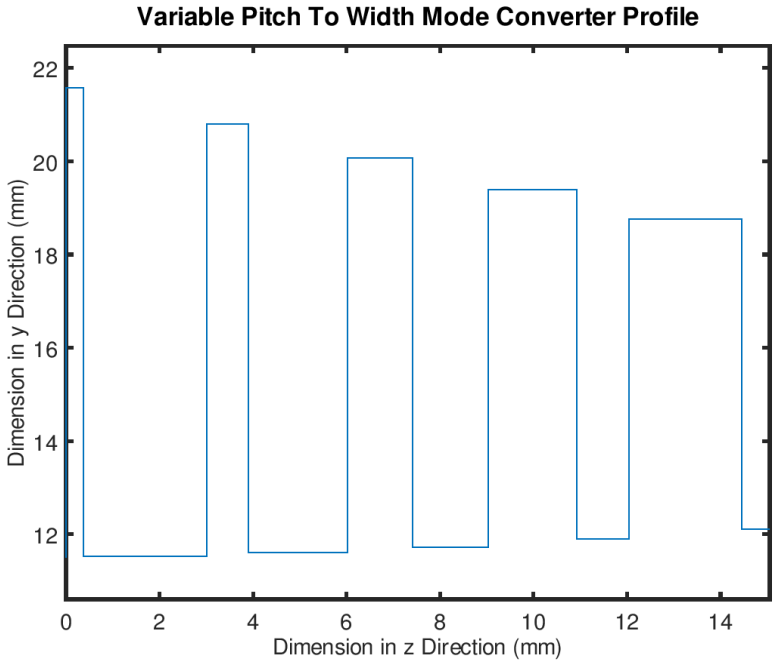


This completes the part of the script to generate the ring-loaded-slot mode converter for *case 2*. The third and final mode converter is the variable-pitch-to-width mode converter which is *case 3*.

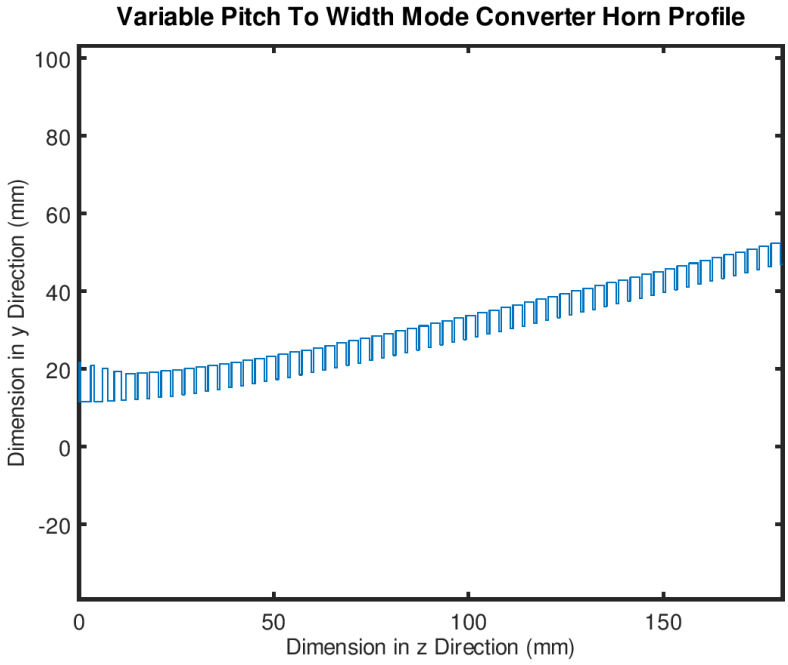
This is very similar to the variable-depth slot mode converter (*case 1*) but the pitch and width of the slots vary up to slot number *NMC*.

```
case 3                  % case 3=VARIABLE PITCH TO WIDTH SLOT MODE CONVERTER
      % Mode Converter depths for element j
      ajmc = a(1:NMC);                      % First indexes for mode converter
      idx = 1:NMC;
      djmc = (sigma*(lambda_c/1.15)+((idx-1)./(NMC-1)).*(lambda_c/4-(sigma*lambda_c/
1.15))).*exp(1./(2.114.*(kc*ajmc).^1.134))
      % Depth of remaining corrugations
      aj = a(NMC+1:end);
      idx = NMC+1:N+1;
      dj = ((lambda_c/4).*exp(1./(2.114.*(kc*aj).^1.134)))-((idx-NMC-1)/(N-NMC-
1)).*((lambda_c/4).*exp(1./(2.114.*(kc*aj).^1.134))-(lambda_o/4).*exp(1./
(2.114.*(ko*ao).^1.134)))
      d = [djmc, dj];       % Combining the mode converter and horn depth values

      % Generate z,y coordinates as len,rad vector
      n = 0;
      len(1) = 0;
      len(2) = 0;
        for i = 1:NMC;
          rad(i+n) = a(i);
          rad(i+n+1) = a(i)+d(i);
          rad(i+n+2) = a(i)+d(i);
          rad(i+n+3) = a(i+1);
          rad(i+n+4) = a(i+1);
          len(i+n+2) = len(i+n)+(delta_min+(((i-1)./(NMC-1))*(delta-delta_min)))*p;
          len(i+n+3) = len(i+n+2);
          len(i+n+4) = len(i+n+3)+(1-(delta_min+(((i-1)./(NMC-1))*(delta-delta_min))))*p;
          len(i+n+5) = len(i+n+4);
          n = n+3;
        endfor

      figure
      plot(len(1:end-1),rad);
      set(gca, "linewidth",2, "fontsize", 14 )
      xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
      ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
      title( 'Variable Pitch To Width Mode Converter Profile', 'FontSize', 16 );
      axis equal;   % Scale axis equally for aspect ratio 1:1

        for i = NMC+1:N;
          rad(i+n) = a(i);
          rad(i+n+1) = a(i)+d(i);
          rad(i+n+2) = a(i)+d(i);
          rad(i+n+3) = a(i+1);
          rad(i+n+4) = a(i+1);
          len(i+n+2) = len(i+n)+delta*p;
          len(i+n+3) = len(i+n+2);
          len(i+n+4) = len(i+n+3)+(1-delta)*p;
          len(i+n+5) = len(i+n+4);
          n = n+3;
        endfor

      z_number = (N*4)+1; % Number of coordinate points for corrugated length of horn
      len = len(1:z_number); % Truncate z axis data points to equal rad vector length

      figure
      plot(len,rad);
      set(gca, "linewidth",2, "fontsize", 14 )
      xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
      ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
      title( 'Variable Pitch To Width Mode Converter Horn Profile', 'FontSize', 16 );
      axis equal;   % Scale axis equally for aspect ratio 1:1

endswitch
```

This segment of the script will generate the following mode converter.



It should be noted that the variable-pitch-to-width mode converter starts with the first slot at the origin of the z axis. The slots progress from narrow to wider until the last slot where the remaining slots along the horn profile will have a constant width as shown in the next figure.



This concludes the section on the mode converters and the following lines add a vertical line at the aperture to define the open face, followed by a polyline that is the same as the horn profile but offset radially to define the external surface. After that, further lines are added to define the circular waveguide preceding the mode converter and the last line closes the polyline to form a closed loop. This is necessary for CSXCAD to form a solid body by a rotation around the x=y=0 coordinate, around the z axis. This type of circularly symmetric structure is also known as a body of revolution (BOR).

For simplicity, the outer profile of the horn is made to have the same profile as the internal profile. This does not have to be the case, it may take any shape that you define, such as a linear profile. The vector *a_offset* is a copy of vector *a*, which defined the internal profile plus an offset. Here, the offset is half a wavelength at the centre frequency plus 2 mm. This gives enough thickness to the wall at the thinnest points which are in the region of each corrugation. If you wish to check the profile you may uncomment the plot lines after the definition of *a_offset*.

```
% Add the rest of the geometry to create a closed path
% a_offset is the inner horn profile shifted up to give the horn a thickness
a_offset = a.+(lambda_c/2+2);
%figure;                    % Uncomment these three lines for debugging
%plot(z, a_offset);
%axis equal;
```

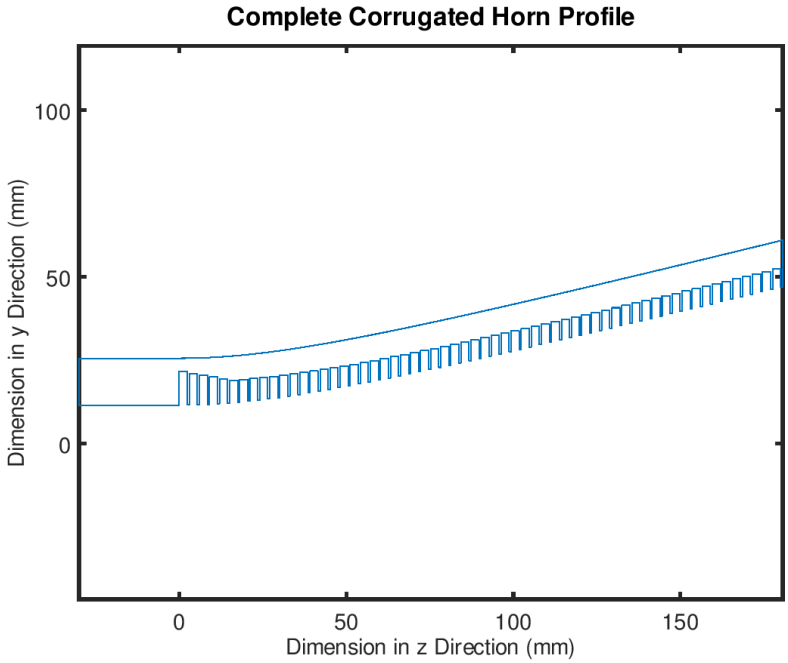
Here we add the vertical line at the front of the horn which will become its front face. Vector *radmsh* is simply a copy of vector *rad* and may be used when defining the mesh lines in the x and y directions of the Cartesian mesh later in the script. Again, you may uncomment the plot lines to check the geometry up to this point if there are any issues.

```
% Add vertical surface at horn aperture
len = [len, len(z_number)];
rad = [rad, a_offset(N)];
radmsh=rad;                % radmsh to fix mesh lines to corrugations
%figure;                   % Uncomment these three lines for debugging
%plot(len, rad);
%axis equal;
```

Now we reverse the order of the vector *a_offset* so that the outer profile is constructed with the largest radius first, directly following the vertical line that we have just defined. This is done with the Matlab/Octave command *fliplr*. These additional vectors that define the various parts of the profile are combined together with the horn's circular waveguide input into the *len* and *rad* vectors. This completes the horn's polyline profile and is plot in a figure so that the complete profile is shown. This will show if there are any problems that should be corrected before proceeding.

```
% Flip outer surface profile so that widest horn dimensions comes next in the outline
coordinates
outer_surface = fliplr(a_offset);
z_flip = fliplr(z);
extent = len(end);  % Fudge to make horn aperture planar for ring loaded slot MC
z_flip(1) = extent; % Fudge to make horn aperture planar for ring loaded slot MC
% Add outer profile and circular waveguide to horn
len = [len, z_flip, -wgl, -wgl, 0];
rad = [rad, outer_surface, ai+(lambda_c/2+2), ai, ai];

figure
plot(len,rad);
set(gca, "linewidth",2, "fontsize", 14 )
xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
title( 'Complete Corrugated Horn Profile', 'FontSize', 16 );
axis equal;   % Scale axis equally for aspect ratio 1:1
```

At this point in the script, the following horn profile is generated.



openEMS Setup

For more information on openEMS commands and usage, refer to the wiki on their website [1]. Now we begin the openEMS simulation setup and start by defining the openEMS standard physical constants such as ε, μ and the speed of light c0.

```
% openEMS setup begins here
% EM related physical constants
physical_constants;
```

Earlier we defined *fmin* and *fmax* in the units of GHz, here we must define *f_start* and *f_stop* for the simulation in Hz, so we multiply the frequency by 1e9 to obtain the frequency in Hz.

```
% frequency range of interest
f_start =  fmin*1e9;
f_stop  =  fmax*1e9;
```

f0 is the frequency at which we want to calculate the fields. Here we define one frequency but a range of frequencies may be defined as a vector.

```
% frequency to calculate fields
f0 = 12.46*1e9;
```

Simulation Parameters

We setup some of the simulation parameters. First define the *FDTD* object by initialising it with *InitFDTD*, with some parameters in the parenthesis. *'NrTS'* is followed by a number, this is the maximum number of time steps for the simulation to run if the *'EndCriteria'* is not met first. This *'EndCriteria'* is the level of power decay in the simulation. In other words, the power should drop to this value before the simulation stops. Sometimes, this does not happen, for example if the structure being simulated is highly resonant, or if energy is trapped and does not propagate, or if there is a problem with the ports setup. For this reason it is recommended to define a maximum number of time steps so that the simulation stops and does not run indefinitely. An *EndCriteria* of

0.5e-3 corresponds to an energy decay of around -33 dB. Note that this energy decay issue is observed to happen with the corrugated horns that have been simulated. This is not unique to openEMS but has also been observed when simulating the same structure with CST Microwave Studio. So it is important to include a maximum number of time steps to stop the simulation when most of the energy had decayed. The number for *NrTS* can be found by experimentation and checking when the energy decay had stabilized. This will be different for different horn geometries.

```
%% setup FDTD parameter & excitation function
FDTD = InitFDTD( 'NrTS', 50000, 'EndCriteria', 0.5e-3 );
```

Now we add the excitation type and parameters with *SetGaussExcite*. There are various types of excitation described in the online documentation, but the Gaussian type is used when the steady state frequency response is required. Use the minimum (f_start) and maximum (f_stop) frequencies to set the frequency range that will be simulated. There are three arguments to the function, the first is the name of the FDTD object, the second is the center frequency of the pulse, and the third is 20 dB cutoff frequency which implies that the total bandwidth is twice the cutoff frequency.

```
FDTD = SetGaussExcite(FDTD,0.5*(f_start+f_stop),0.5*(f_stop-f_start));
```

Boundary conditions

Boundary conditions are defined for each direction in the Cartesian coordinate system for the -x, +x, -y, +y, -z, +z directions of the simulation domain, in that order. A number of boundary conditions are available. For a radiating structure such as an antenna, we need to truncate the simulation volume with absorbing boundaries. There are two types of available absorbing boundaries. These are the simple "MUR" boundary which is the simplest and least computationally intensive, but less effective for an incident wave at larger angles of incidence. The better and more sophisticated absorbing boundary is the PML (Perfectly Matched Layer). This consists of a set thickness which is defined by a number of mesh layers.

The maximum performance can be set by using *PML_20* (PML uses 20 mesh lines) but this requires longer simulation run times and is not usually necessary. *PML_8* seems like a good compromise for low reflections for most situations. We need to know the number of mesh lines used for the PML when defining the field dump boxes, they must not lie inside the PML region. One mesh line away from the PML is fine.

Side note: Each layer of the PML is an absorbing layer, where each layer is progressively more lossy than the last. A function will define the loss profile and would have been optimised for minimum, so we do not need to be concerned with this. For more information on how a PML works, refer to [11].

Other boundary conditions are the PEC (Perfect Electric Conductor) and the PMC (Perfect Magnetic Conductor). These may be used for defining waveguide structures or symmetry planes for reducing simulation time and computer resource requirements for symmetric structures. The PEC may also be used to define infinite ground planes for some types of antennas and microwave components such as monopole antennas, microstrip or stripline. The boundary conditions are assigned to vector BC, which is then added to the simulation setup with the *SetBoundaryCond* statement. This adds the contents of the *BC* vector to the *FDTD* object.

```
BC = {'PML_8' 'PML_8' 'PML_8' 'PML_8' 'PML_8' 'PML_8'}; % boundary conditions
FDTD = SetBoundaryCond(FDTD, BC);
```

Mesh Resolution

Here we set the mesh cell size to equal lambda/20 at the highest frequency, which is one twentieth of a wavelength at *f_stop*. You may choose a smaller or larger number and this will determine how well represented the simulation model is. A finer mesh will be more accurate but will take longer for the simulation to run. Usually it is sufficient to begin with λ/10 or λ/15 to check that the simulation is setup correctly and then to increase it to λ/20 or what is required to ensure that the simulation converges to the required accuracy. Remember that lambda is chosen at the highest frequency, *f_stop*. At this point we initialize the CSX structure, this stores the mesh and model data in an XML file.

```
%% setup CSXCAD geometry & mesh
max_res = c0/(f_stop)/unit/20; % cell size: lambda/20
CSX = InitCSX();               % Initialise CSX structure
```

When simulating a radiating structure, a rule of thumb is to keep a space around the antenna from the absorbing boundary which is at least one quarter of a wavelength at the **lowest** frequency. Here we define *lambda_max* and it will be used to define the extents of the simulation domain and the field dump boxes.

```
% Calculate lambda/4 at lowest frequency to use as distance to nf2ff surfaces
lambda_max = c0/f_start/unit/4;
```

Mesh Definition

The mesh is now defined. This is a very important aspect of setting up the simulation and is often a source of errors in the simulation setup. The object named *mesh* is defined in the x, y and z axis and the syntax is *mesh.x*, *mesh.y* and *mesh.z*.

The mesh is defined to have lines that define the most negative x dimension and the positive x dimension, x=0 at the center and any important features in between.

For mesh.x, we have the most negative dimension as being the outermost dimension of the horn minus 9, which is the PML layers minus 1, minus *lambda_max* which is the length equal to one quarter of a wavelength at the lowest frequency.

Then we have *-radmsh(1:4:end)* which defines mesh lines at the radius points of the internal horn features (excluding the shorted ends of the slots). This ensures that the mesh lines coincide with the horn internals without it being too fine. If the shorted ends of the slots are included, the mesh becomes too fine and the simulation time dramatically increases, so we only include every 4th point in *radmsh*. A mesh line is defined at x=0 and the previous definitions are repeated in the positive x direction.

To complete the *mesh.x* definition, the *SmoothMeshLines* function is used to fill the mesh's x axis with mesh lines that are *max_res* millimetres apart and that a smoothing of factor 1.5 is applied as a way of smoothing the spacing between mesh lines just added and the fixed mesh lines that we previously defined.

```
% Create fixed lines for the simulation box, structure and port
mesh.x = [(-a_offset(end)-(9*max_res)-lambda_max) -radmsh(1:4:end) 0 radmsh(1:4:end)
(a_offset(end)+(9*max_res)+lambda_max)];
mesh.x = SmoothMeshLines( mesh.x, max_res, 1.5); % Create a smooth mesh between specified
fixed mesh lines
```

Since the antenna being simulated is aligned along the z axis, and because it's circular symmetric, we set *mesh.y* to equal *mesh.x* that was defined. This should only apply to body of revolution objects or objects that are symmetric in the x and y axis.

```
mesh.y = mesh.x;                                    % Same as x mesh
```

The final axis to set up is *mesh.z*. First define the most negative point in the -z direction. This is minus the length of the straight waveguide minus a quarter wavelength minus 9 (to allow for the PML). Then we define mesh lines for features as we move forward along the z axis, away from the most negative point that we just defined. *-wgl-1*, corresponds to the cylindrical end-cap that we will later place at the end of the horn's waveguide. *-wgl* is the start of the horn's circular waveguide as well as the location of the port's starting point.

-wgl+10 is the location of the ports stop point, where the voltages and currents are recorded. Location 0 is where the horn's flare begins and the circular waveguide ends. Then we define the z locations for the horn's *len* coordinates for every 2nd point. These will correspond to the radius points in the x and y directions. Finally we add half a wavelength and 9 mesh lines for the PML plus 1. This may seem complicated but hopefully it will make sense when you see the geometry and mesh that will be generated.

The mesh lines for the z axis are filled in and smoothed as per the x and y axis. The *DefineRectGrid* function adds the defined mesh to the CSX object and the units are set, in this case, to millimetres. This completes the mesh definition.

```
% Create fixed lines for the simulation box,port and given number of lines inside the
horn
mesh.z = [-wgl-lambda_max-(9*max_res) -wgl-1 -wgl -wgl+10 0 len(1:2:z_number)
length+2*lambda_max+(9*max_res)];
mesh.z = SmoothMeshLines( mesh.z, max_res, 1.4 );

CSX = DefineRectGrid( CSX, unit, mesh );
```

Horn Geometry

Now that the mesh is setup, we can define the horn. Note that as you setup a simulation for the first time you may need to go back and modify the mesh definition several times as you construct the model, add ports and field dump boxes. The mesh lines must coincide with these objects, if it doesn't, you will encounter errors or have problems running the simulation.

For each item that we add to the simulation setup, we must define its material. This could be a perfect metal, lossy metal, or dielectric etc. Here we add two objects to the CSX object and this is done with the *AddMetal* function. The first object that we define is called 'Corrugated_Horn', we then define the *coords* array from the two vectors that define the radius and length values that we calculated earlier. This is the two dimensional poly line that defines the horn's shape.

The 3D horn is created with the use of the *AddRotPoly* function. The arguments set the CSX object, it is given its material definition by passing the name 'Corrugated_horn' which defines it as a metal, and is given a priority of 10. 'x' sets the plane of the polyline that is defined by the coordinate array

called *coords,* and 'z' sets the axis about which the polyline is rotated around to create the 'body of revolution' object.

```
%% create horn
% horn + waveguide, defined by a rotational polygon
CSX = AddMetal(CSX, 'Corrugated_Horn');
coords = [rad; len];
CSX = AddRotPoly(CSX,'Corrugated_Horn',10,'x',coords,'z');
```

To prevent radiation from coming out of the rear of the horn's circular waveguide we place a short length of conductor (PEC) over its rear face, this is the end-cap. It is defined in the same manner as how the horn was defined, first by using *AddMetal* to define 'Cap' as a perfect electric conductor. Then a cylinder is created with the *AddCylinder* function. Its arguments are the CSX object name, its priority which we assign as 10 and then the start and end coordinates as [x1, y1, z1] and [x2, y2, z2], followed by its outer radius which is the outer line of the horn's profile at the start of the circular waveguide, *a_offset(1)*.

```
% End cap to prevent the radiation coming out of the back of the horn
CSX = AddMetal(CSX, 'Cap');
CSX = AddCylinder(CSX,'Cap',10,[0 0 -wgl],[0 0 (-wgl-1)],a_offset(1));
% End of model geometry
```

Side note: There are a number of primitive shapes that be created in openEMS. A list of the available primitives are:

- Box
- Sphere
- Spherical Shell
- Cylinder
- Cylindrical Shell
- Curve
- Wire
- Polygon
- Extruded Polygon
- Rotational Solid
- Polyhedron

These can be combined to create more complex shapes by using a combination of shapes and using the object priority to give the material a priority when different objects overlap. For example if an object is defined as air is intersecting a metal object, and the air has a higher priority, this is the same as creating a void or cavity in the metal object. This would be useful for constructing a slotted waveguide array or a hollow rectangular waveguide.

Excitation

To apply the excitation to the simulation, we define a waveguide port to the horn's circular waveguide input. Since it is a circular waveguide we use the *AddCircWaveGuidePort* and assign a few arguments.

The start and stop coordinates are defined as [x1, y1, z1] and [x2, y2, z2] respectively. Use the internal waveguide dimensions, *-ai* and *ai* as the negative and positive radius of the waveguide. The start point in the z axis is *-wgl* which is the length of the circular waveguide in the negative z direction. The stop point is 10mm from the end of the waveguide. We could have used a stop point of 5mm instead. The stop point is the reference plane of the port and it should be at least two mesh cells away from the start plane. The excitation originates from the start plane, and the voltage and current monitors are located at the stop plane.

The ports arguments are the CSX structure name "CSX", priority (0), port number (1), start and stop coordinates, circular waveguide radius in meters, mode name ('TE11') which is the fundamental mode, polarisation angle where 0 is horizontal (aligned to the x axis) and pi/2 is vertical (aligned to the y axis), and finally the execution amplitude where 0 is passive and 1 is active. Since we have just one port it must be set to active (1).

```
%% Apply the excitation %%
start=[-ai -ai -wgl];
stop =[ai ai -wgl+10];
[CSX, port] = AddCircWaveGuidePort( CSX, 0, 1, start, stop, ai*unit, 'TE11', 0, 1);
```

Recording Fields (Dump Boxes)

There are numerous types of dumps available, see the documentation for more information. Here we will be recording the dumps only for the vertical and horizontal electric fields. Dump boxes can record a huge amount of data so we usually define the "dump box" as a plane of zero thickness and we use sub-sampling to reduce the amount of data to be stored on disk.

To define a dump box we first use the *AddDump* function with arguments to define the CSX structure name, the name of the dump quantity and the 'SubSampling' argument to be a set of three numbers to record the fields at the x_n^{th}, y_n^{th} and z_n^{th} mesh line. Then the start and stop points for the dump plane are defined and used in the *AddBox* function as shown below. Here we record two dump planes to capture the E-fields on the principle axis, but you can choose any coordinates of the simulation domain that lie inside the PML or MUE absorbing boundary. After the simulation is run we can view the fields at any time step or produce an animation of the fields with ParaView. We could do the same to record the H-fields, electric currents, or SAR in either the time or frequency domain. This is described in the online documentation [1].

```
% Dump box for Electric field at Phi=0 (vertical cut)
CSX = AddDump(CSX,'Et_V_dump', 'SubSampling', '4,4,4');
start=[0 (-a_offset(end)-lambda_max) (-wgl-lambda_max)];
stop =[0 (a_offset(end)+lambda_max) (length+2*lambda_max)];
CSX = AddBox(CSX,'Et_V_dump',0,start,stop);
% Dump box for Electric field at Phi=90 (horizontal cut)
CSX = AddDump(CSX,'Et_H_dump', 'SubSampling', '4,4,4');
start=[(-a_offset(end)-lambda_max) 0 (-wgl-lambda_max)];
stop =[(a_offset(end)+lambda_max) 0 (length+2*lambda_max)];
CSX = AddBox(CSX,'Et_H_dump',0,start,stop);
```

Near Field to Farfield Setup

In order to plot the farfields that are radiated from the antenna, we must define a box that captures the nearfields, this data is used by the *nf2ff* function to perform the transform that calculates the farfields. As we did earlier, define the start and stop coordinates that the *CreateNF2FFBox* function will use to capture the electric and magnetic fields on the surface of this box. The box should not be inside the PML. For a Cartesian mesh we want to record the fields in all 6 directions of the box so the 'Directions' vector should contain 1s for -x, +x, -y, +y, -z, +x directions. 'OptResolution' is used to dump the fields for a given resolution to save disk space, this is optional.

```
% nf2ff calc
start = [mesh.x(9) mesh.y(9) mesh.z(9)];
stop  = [mesh.x(end-8) mesh.y(end-8) mesh.z(end-8)];
[CSX nf2ff] = CreateNF2FFBox(CSX, 'nf2ff', start, stop, 'Directions', [1 1 1 1 1 1],
'OptResolution', max_res*4);
```

Simulation Folder Setup

The next lines setup the folder to store the simulation data. A folder called 'tmp' is created and the .xml file is given the name 'Corrugated_Horn' and this file resides in 'tmp'. When the simulation is run, you will be asked if you want to delete the previously stored data. If so, it will be deleted and a new folder is created called 'tmp'.

```
% Prepare simulation folder
Sim_Path = 'tmp';
Sim_CSX = 'Corrugated_Horn.xml';
[status, message, messageid] = rmdir( Sim_Path, 's'); % Clear previous directory
[status, message, messageid] = mkdir( Sim_Path ); % Create empty simulation folder
```

This line creates the openEMS xml file that contains the FDTD and CSX objects that openEMS will use.

```
% Write openEMS compatible xml-file
WriteOpenEMS([Sim_Path '/' Sim_CSX], FDTD, CSX);
```

Viewing the Structure and Running the Simulation

Here an *if* statement will check the RUN_CSXCAD flag that is defined near the top of the script. If it's set to '1' the *CSXGeomPlot* function will be executed and the model geometry, ports and dump boxes along with the mesh will be shown in CSXCAD which will open when the script is run. If the flag is set to '0', the simulation will run without opening CSXCAD, assuming the RUN_SIMULATION flag is set to '1'.

There are two options that can be used to export the geometry as a VTK or an STL file. VTK may be opened in ParaView to show the structure with the field plots or radiation patterns. STL files can also be opened in ParaView, or in most 3D CAD software, other simulation software or a 3D printer.

```
% Show the structure
if(RUN_CSXCAD == 1)
% CSXGeomPlot([Sim_Path '/' Sim_CSX], ['--export-polydata-vtk=tmp']);
CSXGeomPlot([Sim_Path '/' Sim_CSX], ['--export-STL=tmp']);
end
```

If the RUN_SIMULATION flag is set to '1', the *RunOpenEMS* function will be executed and the option '--numThreads=7' will run the simulation with 7 CPU cores enabled. If you're running on a single multicore CPU it is suggested that you try running the simulation with 3 cores, then increase

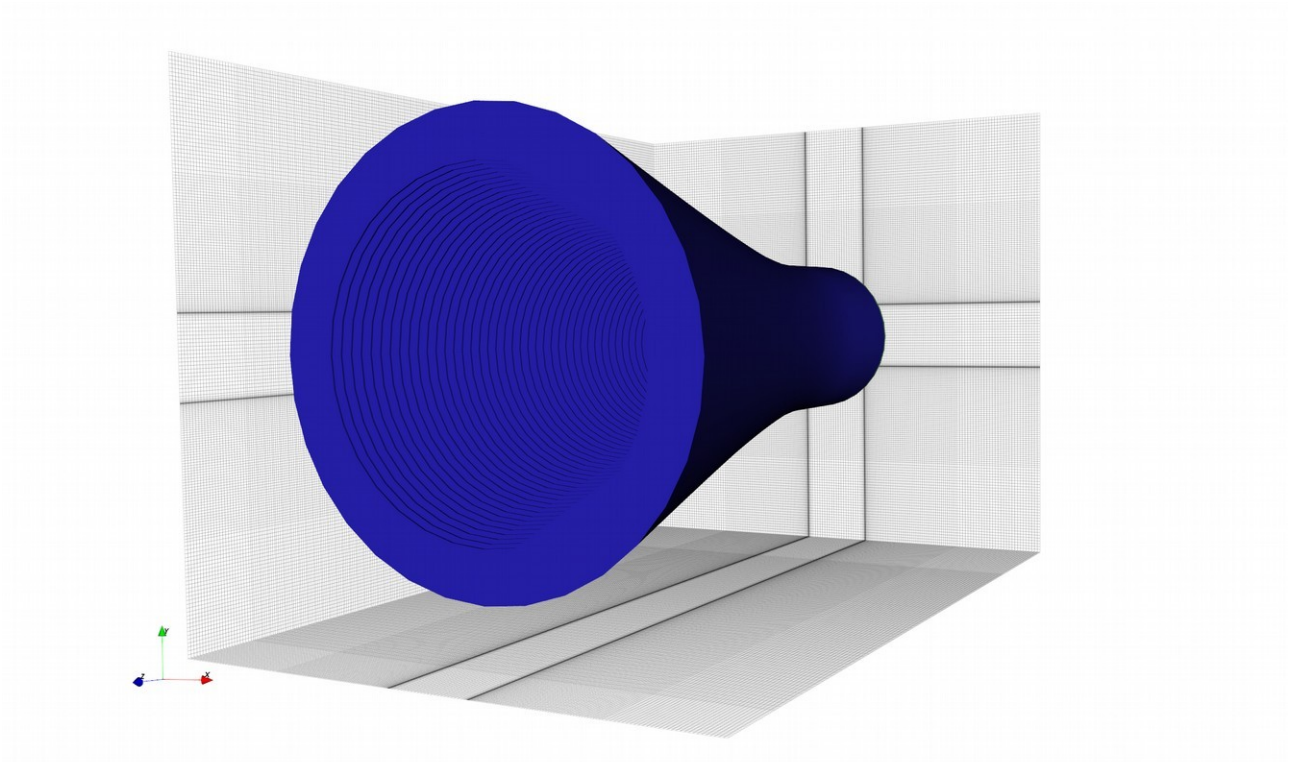
to 4 and then 5 and monitor the number of computations performed per second. Depending on your system, there will be a point where memory bandwidth will become the bottleneck to how fast the simulation can run. If you try to use more cores beyond this, the simulation time may increase, so setting the number of cores to the maximum that you have available is not recommended. Modern Intel Xeon CPUs have a larger memory bandwidth than an Intel i7 or i9 so can take advantage of more cores.

If the RUN_SIMULATION flag is set to '0' the simulation is not run. It is useful to set it to '0' when you are experimenting with generating the horn's profile and you don't want to let the simulation run until you are ready to try it.

If you comment out the *RunOpenEMS* line and uncomment the two lines above it, it will generate a mesh representation that show the mesh lines in all PEC regions for debugging purposes in ParaView.

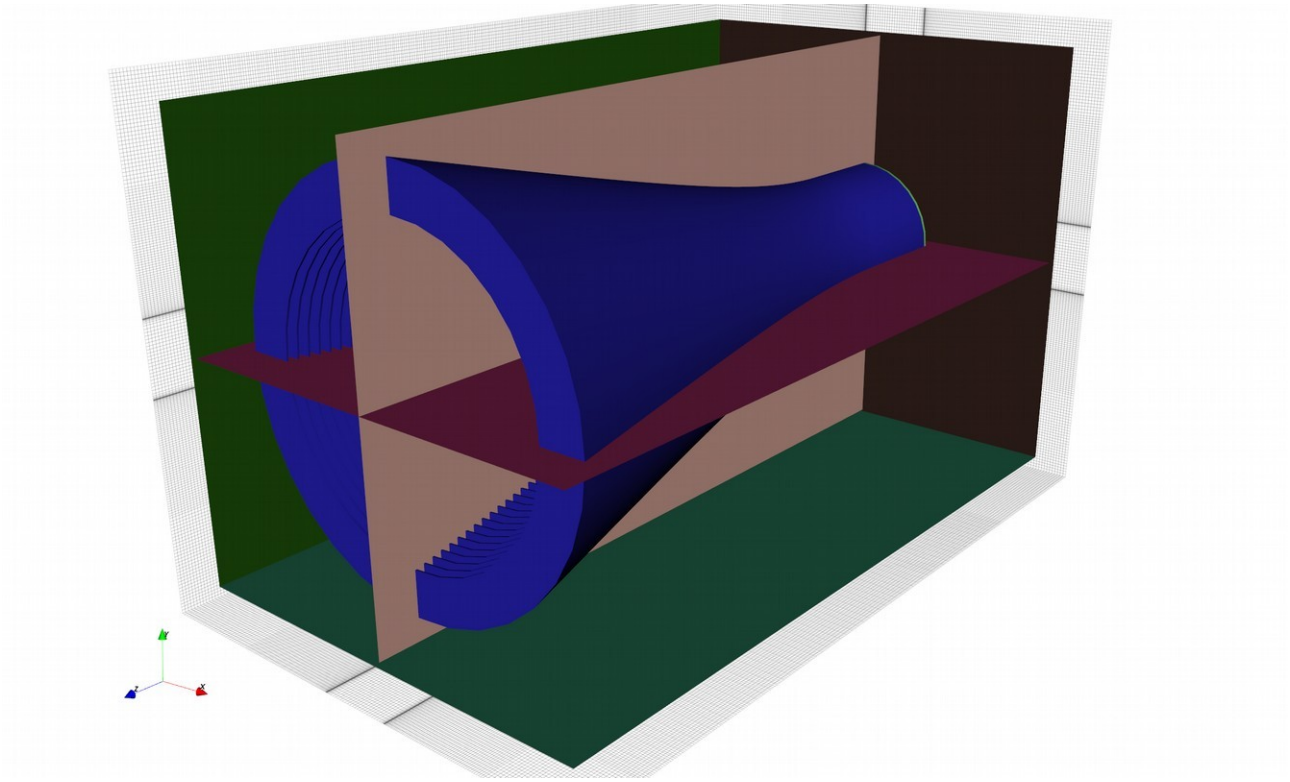
```
% Run openEMS
if(RUN_SIMULATION == 1)
%openEMS_opts = '--debug-PEC --no-simulation';    % Uncomment to visualise mesh in
ParaView
%RunOpenEMS(Sim_Path, Sim_CSX, openEMS_opts);
RunOpenEMS(Sim_Path, Sim_CSX, '--numThreads=7');
end
```

Here are some views from CSXCAD if the simulation is run to this point in the script. The model of the horn is shown with the mesh in the background. The position of the mesh can be changed for each axis to verify that there are mesh lines where you expect them to be.



The default colour is blue, this can be changed in CSXCAD by right clicking on the item in the "Properties and Structures" menu and clicking "View" to change the colour from the palate. See the front cover for the same horn in light grey. Note that the cross-section view was done in FreeCAD after scaling the horn by 1000, and changing the colour/material.

In this screenshot some of the *nf2ff* box planes and both E-field dump planes are shown. The list of dump planes and ports are shown to the left of the CSXCAD window and can be activated or deactivated in the view by selecting or deselecting the lightbulb icons. Note that the *nf2ff* box planes are some distance in from the boundary. They are 9 mesh cells in from each boundary so that they are not inside the PML which has been defined to be 8 cells deep.



Post-processing

When the simulation has successfully completed, the following part of the script is executed to post process the data to give us the useful information that describe the antenna's characteristics.

Here the frequency range over which the S-parameters will be calculated is defined by creating a vector using the *linspace* function to go from *f_start* to *f_stop* and there are 201 points within this range and are stored in *freq*.

S-Parameters

The function *calcPort* is called and takes the arguments for the port number, the simulation path where the data structures are saved, and the frequency vector defined above as *freq*. Refer to the documentation to see other *calcPort* arguments that can set the reference impedance, shifting of the reference plane, switching the direction of propagation and the signal type. These are not used here.

```
% Postprocessing & do the plots
freq = linspace(f_start,f_stop,201);
port = calcPort(port, Sim_Path, freq);
```

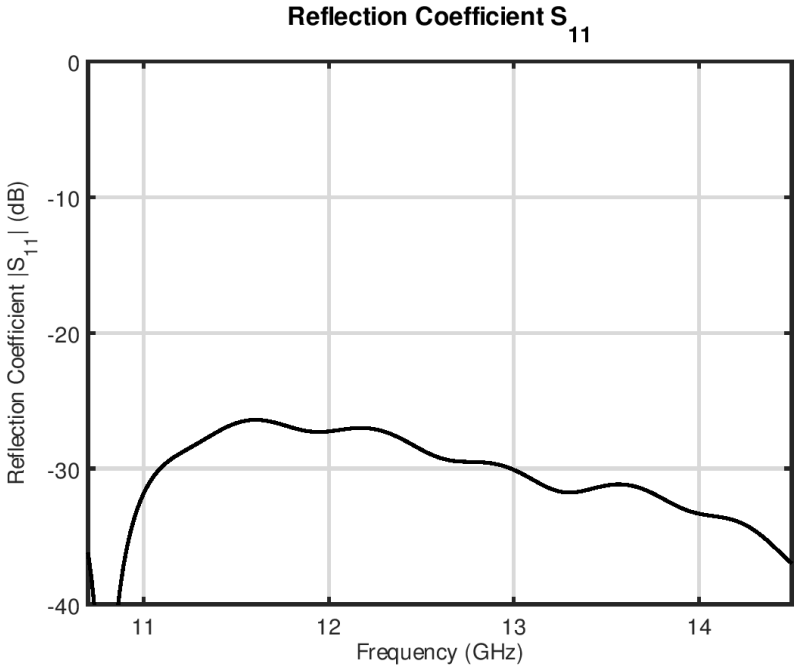
The following two lines calculate the input impedance and S-parameter S_{11} for each frequency point. The port here is '1' since we have only one port, *uf* is the voltage at each frequency point and *if* is the current for the same frequency point. *tot* is the total quantity, *ref* is the reflected quantity, and *inc* is the incident quantity. Z_{in} is the impedance and is simply the total voltage divided by the

total current (Ohms Law). S_{11} is the reflection at the port and is the reflected voltage divided by the incident voltage at each frequency point.

```
Zin = port.uf.tot ./ port.if.tot;
s11 = port.uf.ref ./ port.uf.inc;
```

This plots the reflection coefficient S_{11} vs frequency.

```
% Plot reflection coefficient S11
figure
plot(freq/1e9, 20*log10(abs(s11)), 'k-', 'Linewidth', 2);
xlim([fmin fmax]);
ylim([-40 0]);
set(gca, "linewidth",2, "fontsize", 14)
grid on
title('Reflection Coefficient S_{11}', 'FontSize', 16);
xlabel('Frequency (GHz)','FontSize', 14);
ylabel('Reflection Coefficient |S_{11}| (dB)','FontSize', 14);
drawnow
```



Radiated Farfields

Here the nearfield to farfield plots are defined, we set the range for theta, select which phi cuts to plot and this is passed to the *CalcNF2FF* function which is assigned to *nf2ff* data structure.

```
% NFFF plots

% Calculate the far field at phi=0, 45 and at phi=90 degrees
thetaRange = (0:0.2:359) - 180;
disp('calculating far field at phi=[0 45 90] deg...');
nf2ff = CalcNF2FF(nf2ff, Sim_Path, f0, thetaRange*pi/180, [0 45 90]*pi/180);
```

Dlog is the maximum directivity that is obtained from the nf2ff data structure with the use of the *Dmax* quantity. *G_a* is the gain calculated from the wavelength and the area of the horn's aperture. *e_a* is the antennas efficiency which is *Dmax* divided by the theoretical gain. These quantities are displayed in the command window when the simulation has run using the *disp* function.

```
Dlog=10*log10(nf2ff.Dmax);      % Calculate maximum Directivity in dB
G_a = 4*pi*A_app/(c0/f0)^2;     % Calculate theoretical gain for given aperture
e_a = nf2ff.Dmax/G_a;           % Calculate Efficiency
```

```
% Display some antenna parameters from above calculations
disp(['radiated power: Prad = ' num2str(nf2ff.Prad) ' Watt']);
disp(['directivity: Dmax = ' num2str(Dlog) ' dBi']);
disp(['aperture efficiency: e_a = ' num2str(e_a*100) '%']);
```

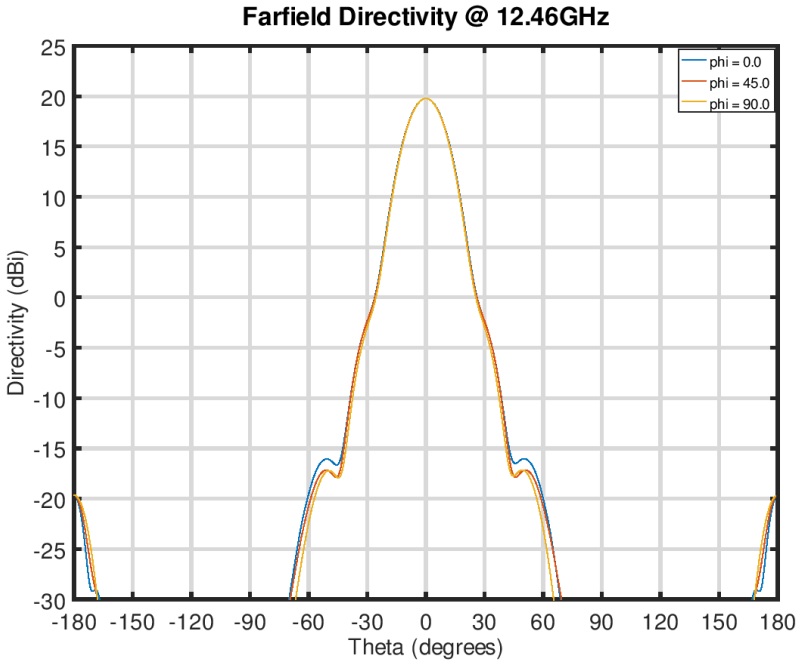
The output copied from the command window is:

```
radiated power: Prad = 4.2116e-25 Watt
directivity: Dmax = 19.7805 dBi
aperture efficiency: e_a = 63.203%
```

The next section of the script sets up a figure window and plots the directivity for the three phi cuts that are defined. Following that is a plot that shows the co and cross polarisation farfield plots for phi = 45°. The worst case cross polarisation levels occur at phi = 45° for a horn with a symmetric aperture and this is used to determine the overall cross polarisation performance of a horn antenna and is directly related to the radiated cross polarisation performance for a reflector antenna when the corrugated horn illuminates it.

```
% Directivity
figure
plotFFdB(nf2ff,'xaxis','theta','param',[1 2 3]);
ylim([-30 25]);
xlim([-180 180]);
grid on
set(gca,"linewidth",2, "fontsize", 14, "XTick", -180:30:180, "YTick", -30:5:40)
title('Farfield Directivity @ 12.46GHz','FontSize', 16);
xlabel('Theta (degrees)','FontSize', 14);
ylabel('Directivity (dBi)','FontSize', 14);
drawnow
```

The directivity plot is shown in this figure.



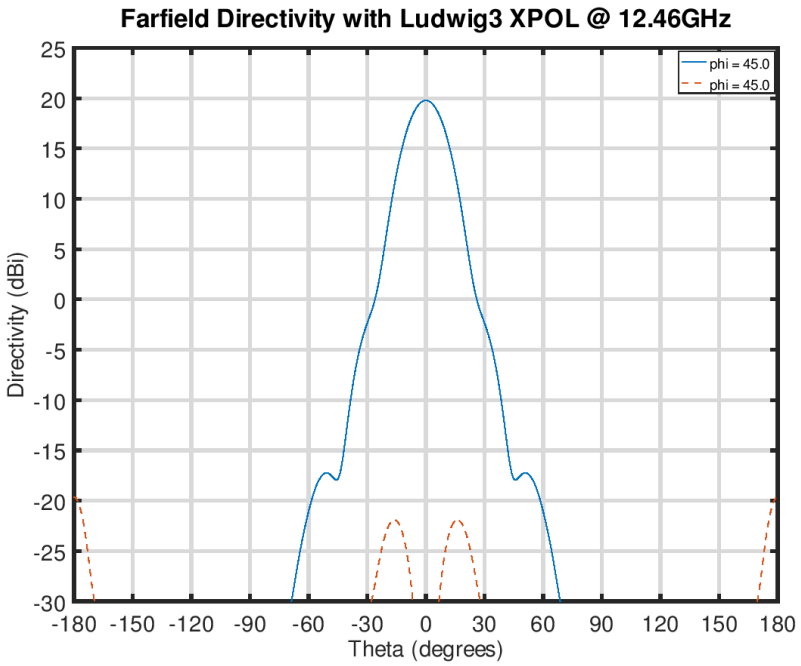
Note the circularly symmetric properties of the radiation patterns, this is one of the qualities of a well designed corrugated horn antenna. Another feature are the very low sidelobes.

At the time of writing this tutorial, the *plotFFcocx* function is not supplied with openEMS but will be included with this tutorial. It can also be found on the openEMS forum and is based on the *plotFFdB* function, modified by Dr Bruce Veidt to include the Ludwig3 transform to calculate the cross polarisation. The Ludwig3 definition of cross polarisation is a standard IEEE method to

calculate cross polarisation radiation patterns. The transform can be found in [12]. Here is the transform for completeness and it may be used directly within the script if you do not wish to use the *plotFFcocx* function.

```
Convert from E_theta and E_phi to co- and cross-pol as per
Ludwig's third definition.
Ref.: Milligan, Modern Antenna Design, 2005, p.22
[E_co] = [cos(phi)  -sin(phi)] [E_phi  ]
[E_cx] = [sin(phi)   cos(phi)] [E_theta]

% Plot Ludwig3 cross polar
plotFFcocx(nf2ff,'xaxis','theta','param',[2]);
ylim([-30 25]);
xlim([-180 180]);
grid on
set(gca,"linewidth",2, "fontsize", 14, "XTick", -180:30:180, "YTick", -30:5:40)
title('Farfield Directivity with Ludwig3 XPOL @ 12.46GHz','FontSize', 16);
xlabel('Theta (degrees)','FontSize', 14);
ylabel('Directivity (dBi)','FontSize', 14);
drawnow
```

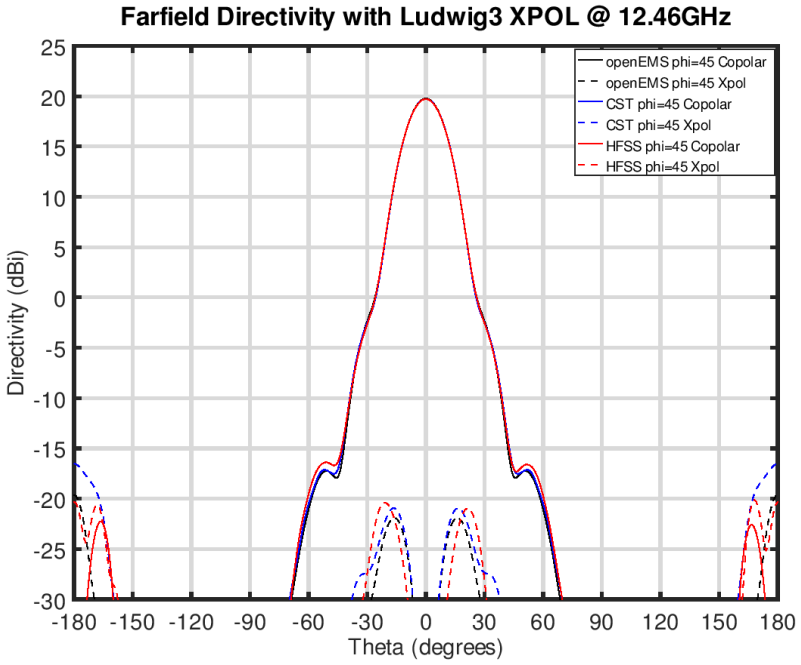
The figure of the cross polarisation plot is shown below.



This plot shows that the peak cross-polarisation levels are approximately 42 dB below the peak co-polar level of 19.78 dBi.

Co and Cross Polarisation Farfield Comparison with CST and HFSS

As a matter of interest, the same antenna was exported as an STL file to CST Microwave Studio and Ansys HFSS to simulate the same antenna and the farfields were plot for the same frequency of 12.46 GHz to obtain equivalent data to compare to openEMS. A modified version of the *plotFFcocx* function was used to show the data on the same plot for a phi angle of 45°.



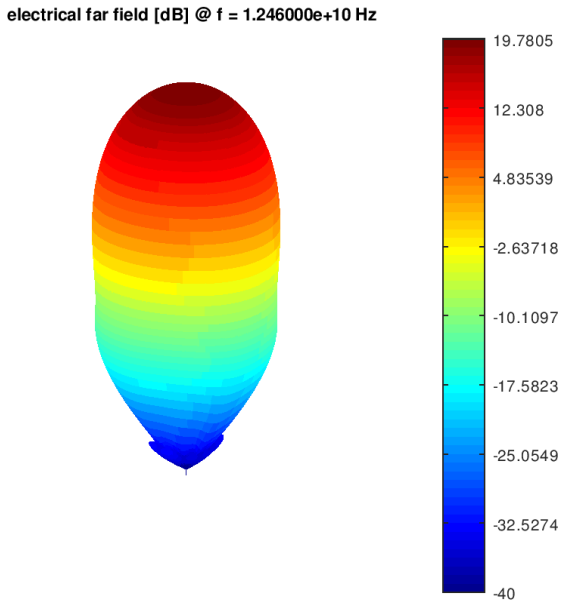
The results show extremely good correlation and gives the user a great deal of confidence in the use of openEMS to simulate high performance microwave antennas! The author can confirm that the data was plotted "as is" and has not been modified in any way.

3D Radiation Patterns

The 3D radiation patterns can be displayed in Matlab/Octave in a figure window or exported as VTK data to be displayed in ParaView. First define the phi and theta ranges. You may want more data points over angles of interest as shown below, or you can define one data range with small increments but this will create larger data files. The phi angle is the radial slice, usually around the z axis. Theta is the angle that can be regarded as the angle of elevation with a range from 0° being along the axis of the antenna (zenith), looking forward, to 180° (directly behind the antenna).

```
%% Calculate 3D pattern
phiRange = sort(unique([-180:5:-100 -100:2.5:-50 -50:1:50 50:2.5:100 100:5:180]));
thetaRange = sort(unique([0:1:50 50:2:100 100:5:180]));

disp('calculating 3D far field...');
nf2ff = CalcNF2FF(nf2ff, Sim_Path, f0, thetaRange*pi/180, phiRange*pi/180,
'Verbose',2,'Outfile','nf2ff_3D.h5');

figure
plotFF3D(nf2ff, 'logscale', -40);        % plot 3D far field in dB
```

This is the 3D plot produced by Octave.



The final lines save farfield data to VTK files to be viewed with ParaView.

```
% Save far field in VTK to plot in ParaView
E_far_normalized = nf2ff.E_norm{1}/max(nf2ff.E_norm{1}(:));
DumpFF2VTK([Sim_Path '/Farfield.vtk'],E_far_normalized,thetaRange,phiRange,'scale',
0.008, 'logscale', -30, 'maxgain', Dlog);
```

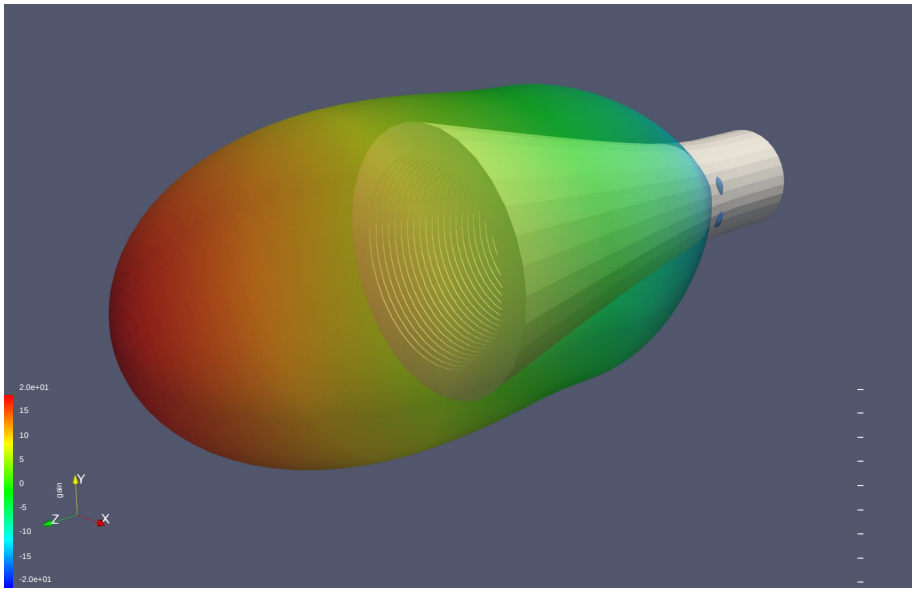
Some screenshots from ParaView showing the 3D farfield with the antenna are shown on the following page. ParaView is a very powerful program and gives the user many options to format and display data. The quality of the plots and images are on par with the best commercial software available.

Start ParaView and select “Open” then browse to the ‘tmp’ folder where simulation data is stored. Select the files you wish to open. In this case the ‘Corrugated_Horn.stl’ and ‘Farfield.vtk’ have been selected. You will not see anything until you click on the ‘eye’ icon next to the file names in the ‘Pipeline Browser’. This will make the items visible in the main window. Click on the horn filename to select it and you may then change the colour and other attributes. Click on the Farfield file and go to the ‘Color Map Editor’ and open the ‘Choose Preset’ colour ranges and select ‘Blue to Red Rainbow’ to make the farfield plot look like the screenshots below.

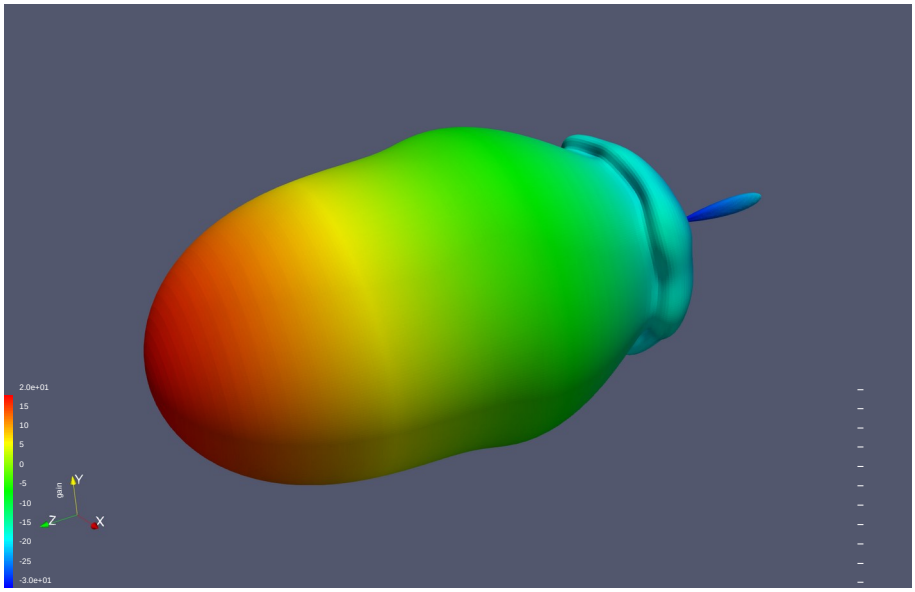
Each item displayed can be made transparent, or left opaque. Experiment with different settings to generate the view that you want. The legend can be shown or switched off.

The size of the 3D plot can be modified with the scale value in the DumpFF2VTK function arguments. Here it’s set to 0.008.

Transparent far field view overlaid with the antenna.



Opaque farfield view.

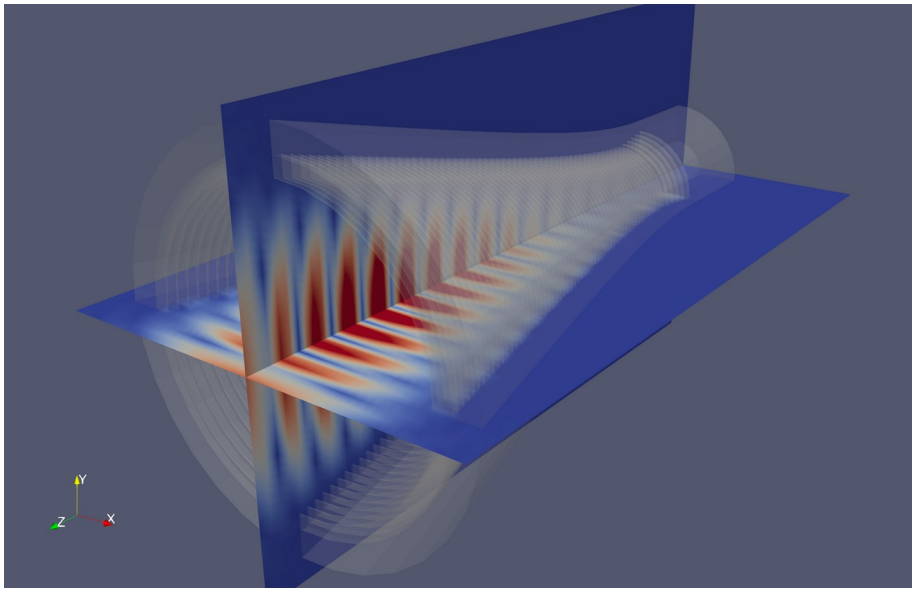


Electric Field Visualisation

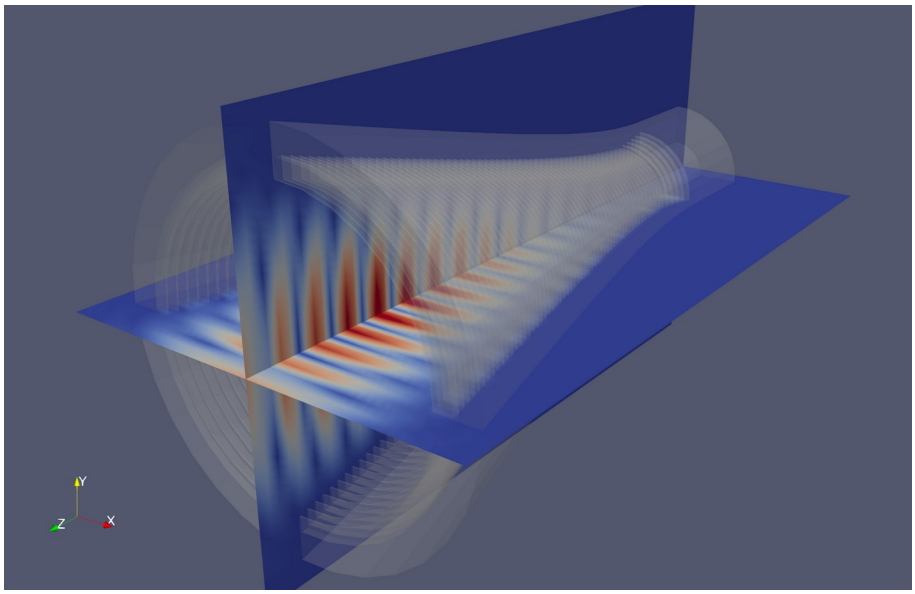
The electric fields (or magnetic fields) can be viewed in ParaView by loading the Et_H_dump_..vtr and Et_E_dump_..vtr files from the open file dialogue. This will open a number of files for each of the above showing the E-Fields in the horizontal and vertical planes that were defined in the script. Data is present for time-step intervals that was stored and this enables the user to generate an animation of the fields to get a better understand of what's happening in the simulated structure.

The colour scaling can be modified for a particular time-step or frame, or can be normalised to the maximum values in the dataset. Here we see the colour scaling set to 0.6 and 1 and the opacity of the horn has been reduced to show it as a transparent object.

E-field visualisation scaled to 0.6



E-field visualisation scaled to 1



Conclusion

The information presented in [6] has been described and implemented with all available configurations with Matlab/Octave. The script used to generate the high performance corrugated horns is used directly by the free and open-source FDTD software openEMS within the same script for simplicity. The important results from openEMS are presented with equivalent simulation results from leading commercial FDTD and FEM software which should give users of openEMS a great deal of confidence in the results when simulating other waveguide antennas other than corrugated horn antennas.

By following the script presented here, many of the important functions and the workflow of openEMS has been discussed so that users of the software may adapt it to their own needs and other antenna types. The use of ParaView is an alternative to displaying results in Matlab/Octave and we have demonstrated it's use to display antenna farfields and electric field plots in high quality two and three dimensional plots.

Bibliography

[1] T. Liebig, 'openEMS - Open Electromagnetic Field Solver'. [Online]. Available: https://www.openems.de/start.

[2] 'ParaView'. [Online]. Available: https://www.paraview.org.

[3] 'CST Microwave Studio', *Dassault Systemes*. [Online]. Available: www.3ds.com/products-services/simulia/products/cst-studio-suite/.

[4] 'HFSS'. [Online]. Available: www.ansys.com.

[5] P. J. B. Clarricoats and A. D. Olver, *Corrugated Horns for Microwave Antennas*. .

[6] C. Granet and G. L. James, 'Design of Corrugated Horns: A Primer', *IEEE Antennas and Propagation Magazine*, vol. 47, no. 2, pp. 76–84, Apr-2005.

[7] G. L. James, 'Analysis and Design of TE11-to-HE11 Corrugated Cylindrical Waveguide Mode Converters', *IEEE Trans. Microw. Theory Tech.*, vol. MTT-29, pp. 1059–1066, 1981.

[8] A. D. Olver, P. J. B. Clarricoats, A. A. Kishk, and L. Shafai, *Microwave Horns and Feeds*. 1994.

[9] B. Thomas, G. L. James, and K. J. Greene, 'Design of Wide-Band Corrugated Conical Horns for Cassegrain Antennas', *IEEE Trans. Antennas Propag.*, vol. AP-34, no. 6, pp. 750–757, Jun. 1986.

[10] C. Granet, 'Profile Options for Feed Horn Design', in *Proceedings of the Asia Pacific Microwave Conference*, 2000, vol. 1, pp. 1448–1452.

[11] J.-P. Berenger, 'A Perfectly Matched Layer for the Absorption of Electromagnetic Waves', *J. Comput. Phys.*, no. 114, pp. 185–200, 1994.

[12] T. A. Milligan, *Modern Antenna Design*, 2nd ed. IEEE Press, Wiley-Interscience, 2005.

Thanks to Ioana Cojocaru, Dr Thomas Milligan, Dr Christophe Granet and Dr Bruce Veidt for checking this document for errors and suggestions for improvements.

18th September 2019

Appendix A

Complete Script Listing

File: Corrugated_horn_with_mode_converter_options_MASTER.m

```
% Script to generate corrugated horn profiles for simulation with openEMS.
%
% Formulations and example from "Design of Corrugated Horns: A Primer", IEEE
% Antennas and Propagation Magazine,Vol.47,No 2,April 2006, by Christophe Granet
% Author: Paul Klasmann
% Date: March 2019
%
% 1) Enter flags for RUN_SIMULATION and RUN_CSXCAD, 1 to run and 0 not to run.
%    It's advised not to run the simulation while setting up the horns geometry.
% 2) Enter values in the user editable parameters.
% 3) If using the ring loaded slot mode converter, enter a value for delta2,
%    otherwise ignore this variable.
% 4) If using the variable pitch to slot width mode converter, enter a value for
%    delta_min, otherwise ignore this variable.
% 5) Select an integer number from 1 to 8 for the horn_profile.
% 6) Select an integer number from 1 to 3 for the mode converter type.
% 7) Length of horn profile is chosen in the length = xx*p line. Enter a value
%    that is multiplied by the pitch "p".  This will give the length of the
%    profiled length of the horn.  Line 81.
%
% Note 1: If you wish to only generate the complete horn profile you may comment
%       out all lines from 408 if you do not have openEMS installed.
% Note 2: Set circular waveguide length wgl to something sensible for given diameter.

close all
clear
clc

RUN_SIMULATION = 0;   % Set to 1 to run openEMS, 0 not to run
RUN_CSXCAD = 1;       % Set to 1 to open CSXCAD to show 3D geometry, 0 not to open CSXCAD

% USER EDITABLE PARAMETERS
fmin = 10.7                % Minimum frequency in GHz
fmax = 14.5                % Maximum frequency in GHz
pitch_fraction = 8  % Choose a fraction between 10 to 5 (lambda_c / pitch_fraction)
delta = 0.8               % Pitch to width ratio 0.7 to 0.9
sigma = 0.42               % Percentage factor for first slot depth, 0.4 to 0.5
NMC = 5                    % Number of corrugations in mode converter
wgl = 30;                  % Length of circular feeding waveguide
num_of_corrugations = 60;
% delta2 is only used for case 2, ring loaded slot mode converter
delta2 = 0.1;

% delta_min is only used for case 3,variable pitch to width slot mode converter
delta_min = 0.125;           % Should be greater than 0.125 and less than delta

% Choose profile, 1=LINEAR, 2=SINUSOID, 3=ASYMMETRIC SINE-SQUARED, 4=TANGENTIAL,
% 5=x.rho, 6=EXPONENTIAL, 7=HYPERBOLIC, 8=POLYNOMIAL
horn_profile = 7;          % Must be an integer from 1 to 8

% Choose the type of mode converter, 1=VARIABLE SLOT DEPTH MC,
% 2=RING LOADED SLOTS MC, 3=VARIABLE PITCH TO WIDTH SLOT MC
mode_converter_type = 1;   % Must be an integer from 1 to 3

% END OF USER EDITABLE PARAMETERS

% Calculate center frequency fc based on narrow or wide bandwidth.
fratio = fmax/fmin             % ratio of fmax/fmin
if (fratio >= 2.4);            % check fmax/fmin is less than 2.4
  disp('Error, fmax/fmin is greater than 2.4!');
  fc = 0
  elseif (fratio <= 1.4);      % Use Narrowband formula if fmax <= 1.4fmin
    fc = sqrt(fmin*fmax)
     elseif (fmax >= 1.4*fmin && fmax <= 2.4*fmin); % Use wideband formula for
1.4fmin<=fmax<=2.4fmin
       fc = 1.2*fmin
endif

if (fratio <= 1.4);
  fo = 1.02*fc    % For narrow band choose fc <= fo <= 1.05fc
    elseif (fmax >= 1.4*fmin && fmax <= 2.4*fmin);
      fo = 1.10*fc    % For wideband choose 1.05fc <= fo <= 1.15fc
```

```
endif

unit = 1e-3;                    % Units in mm
lambda_c = 300/fc               % Center frequency wavelength
lambda_o = 300/fo               % Output frequency
depth_nominal = lambda_c/4      % Nominal slot depth at center frequency
ai = (3 * lambda_c)/(2*pi)      % Radius of input waveguide in mm
ao = 1.95*lambda_c              % Radius of output waveguide in mm
p = lambda_c/pitch_fraction;    % Pitch in mm, lambda_c/10 to lambda_c/5
length = num_of_corrugations*p  % Length of horn profile
N = length/p                    % Total number of corrugations
kc = (2*pi)/lambda_c            % Wave number at center frequency
ko = (2*pi)/lambda_o            % Wave number at output frequency
z = 0:p:length;                 % z index distance array from 0 to length of horn
r_app = ao*1e-3;                % Aperture radius
A_app = pi*(r_app)^2;           % Aperture area for gain calculation

% One of the following horn profiles will be selected depending on value of
% horn_profile chosen above.
switch(horn_profile)
   case 1
   %%% Linear profile %%%
   a = ai+(ao-ai)*z/length;

   plot(z, a);
   set(gca, "linewidth",2, "fontsize", 14 )
   axis equal;
   xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
   ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
   title( 'Linear Horn Profile', 'FontSize', 16 );

   case 2
   %%% Sinusoid profile %%%
   A = 1;       % Amplitude factor 'A' should be between 0 and 1
   rho = 2;     % rho should be between 0.5 and 5, default is 2
   a = ai+(ao-ai)*((1-A)*(z/length)+A*power(sin((pi*z)/(2*length)),rho));

   plot(z, a);
   set(gca, "linewidth",2, "fontsize", 14 )
   axis equal;
   xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
   ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
   title( 'Sinusoidal Horn Profile', 'FontSize', 16 );

   case 3
   %%% Asymmetric Sine-Squared profile %%%
   L1 = length/3;    % Choose a value for L1, must be less than the horns length
   L2 = length-L1;   % L2 is the length between L1 and the end of the horn
   gamma = L2/L1;
   idx = find(z <= L1)     % Find the index of z corresponding to L1
   zelements = size(z,2)   % Total number of points in z axis of the horn

   za = z(1: max(idx))
   aa = ai+((2*(ao-ai))/(1+gamma))*sin((pi*za)/(4*L1)).^2;

   zb = z(max(idx)+1 : zelements)
   ab = ai+((2*(ao-ai))/(1+gamma))*(gamma*sin(((pi*(zb+L2-L1))/(4*L2))).^2+((1-gamma)/2));

   a = [aa,ab];
   z = [za,zb];

   plot(z, a);
   set(gca, "linewidth",2, "fontsize", 14 )
   axis equal;
   xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
   ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
   title( 'Asymmetric Sine Squared Horn Profile', 'FontSize', 16 );

   case 4
   %%% Tangential profile %%%
   A = 1;
   rho = 2;
   a = ai+(ao-ai)*((1-A)*(z/length)+A*power(tan((pi*z)/(4*length)),rho));

   plot(z, a);
   set(gca, "linewidth",2, "fontsize", 14 )
   axis equal;
   xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
   ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
   title( 'Tangential Horn Profile', 'FontSize', 16 );
```

```
    case 5
    %%% x.rho profile %%%
    A = 1;
    rho = 2;
    a = ai+(ao-ai)*((1-A)*(z/length)+A*power(z/length,rho));

    plot(z, a);
    set(gca, "linewidth",2, "fontsize", 14 )
    axis equal;
    xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
    ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
    title( 'xp Horn Profile', 'FontSize', 16 );

    case 6
    %%% Exponential profile %%%
    a=ai*exp(log(ao/ai)*(z/length));

    plot(z, a);
    set(gca, "linewidth",2, "fontsize", 14 )
    axis equal;
    xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
    ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
    title( 'Exponential Horn Profile', 'FontSize', 16 );

    case 7
    %%% Hyperbolic profile %%%
    a = sqrt(ai^2 + (power(z,2) * (ao^2-ai^2) / length^2));

    plot(z, a);
    set(gca, "linewidth",2, "fontsize", 14 )
    axis equal;
    xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
    ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
    title( 'Hyperbolic Horn Profile', 'FontSize', 16 );

    case 8
    %%% POLYNOMIAL Profile %%%
    rho = 3;
    a=ai+(rho+1)*(ao-ai)*(1-((rho*z)/((rho+1)*length))).*power(z/length,rho);

    plot(z, a);
    set(gca, "linewidth",2, "fontsize", 14 )
    axis equal;
    xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
    ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
    title( 'Polynomial Horn Profile', 'FontSize', 16 );

endswitch

switch(mode_converter_type)

  case 1                            % case 1=VARIABLE SLOT DEPTH MODE CONVERTER
    % Mode Converter depths for element j
    ajmc = a(1:NMC);                    % Index range for mode converter
    idx = 1:NMC;
    djmc = (sigma-((idx-1)./NMC).*(sigma-(0.25.*exp(1./(2.114.*(kc*ajmc).^1.134))))))*lambda_c;
    % Depth of remaining corrugations
    aj = a(NMC+1:end);
    idx = NMC+1:N+1;
    dj = ((lambda_c/4).*exp(1./(2.114.*(kc*aj).^1.134)))-((idx-NMC-1)/(N-NMC-1)).*((lambda_c/
4).*exp(1./(2.114.*(kc*aj).^1.134))-(lambda_o/4).*exp(1./(2.114.*(ko*ao).^1.134)));
    d = [djmc, dj];       % Combining the mode converter and horn depth values

    % Generate z,y coordinates as len and rad vector
    n = 0;
    len(1) = 0;
    len(2) = 0;
      for i = 1:N;
       rad(i+n) = a(i);
       rad(i+n+1) = a(i)+d(i);
       rad(i+n+2) = a(i)+d(i);
       rad(i+n+3) = a(i+1);
       rad(i+n+4) = a(i+1);
       len(i+n+2) = len(i+n)+delta*p;
       len(i+n+3) = len(i+n+2);
       len(i+n+4) = len(i+n+3)+(1-delta)*p;
       len(i+n+5) = len(i+n+4);
       n = n+3;
      endfor
```

```
    z_number = (N*4)+1; % Number of coordinate points for corrugated length of horn
    len = len(1:z_number); % Truncate z axis data points to equal rad vector length

    figure
    plot(len,rad);
    set(gca, "linewidth",2, "fontsize", 14 )
    xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
    ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
    title( 'Mode Converter and Corrugation Profile', 'FontSize', 16 );
    axis equal;   % Scale axis equally for aspect ratio 1:1

    case 2                              % case 2=RING LOADED SLOT MODE CONVERTER
      % Mode Converter depths for element j
      ajmc = a(1:NMC);                  % Index range for mode converter
      idx = 1:NMC;
      djmc = (lambda_c/4).*exp(1./(2.114.*(kc*ajmc).^1.134));
      % Width of bjth slot for mode converter
      bj = (0.1+(idx-1).*((delta2-0.1)./NMC)).*p;
      % Height of hjth slot for mode converter
      hj = (2/3).*djmc;

      % Depth of remaining corrugations
      aj = a(NMC+1:end);
      idx = NMC+1:N+1;
      dj = ((lambda_c/4).*exp(1./(2.114.*(kc*aj).^1.134)))-((idx-NMC-1)/(N-NMC-1)).*((lambda_c/
4).*exp(1./(2.114.*(kc*aj).^1.134))-(lambda_o/4).*exp(1./(2.114.*(ko*ao).^1.134)));
      d = [djmc, dj];       % Combining the mode converter and horn depth values

      % Generate z,y coordinates as len,rad vector
      n = 5;
      len = [0, 0, (-delta*p)+bj(1), (-delta*p)+bj(1), bj(1), bj(1)];
      rad = [a(1), a(1)+d(1)-hj(1), a(1)+d(1)-hj(1), a(1)+d(1), a(1)+d(1), a(2)];
        for i = 2:NMC;
          rad(i+n) = a(i);
          rad(i+n+1) = a(i)+d(i)-hj(i);
          rad(i+n+2) = a(i)+d(i)-hj(i);
          rad(i+n+3) = a(i)+d(i);
          rad(i+n+4) = a(i)+d(i);
          rad(i+n+5) = a(i+1);
          len(i+n) = i*p-p;
          len(i+n+1) = i*p-p;
          len(i+n+2) = len(i+n+1)-(delta*p)+bj(i);
          len(i+n+3) = len(i+n+1)-(delta*p)+bj(i);
          len(i+n+4) = len(i+n+3)+(delta*p)+bj(i);
          len(i+n+5) = len(i+n+3)+(delta*p)+bj(i);
          n = n+5;
        endfor
      % Add extra coordinate points before remaining corrugations
      len(NMC*(NMC+1)+1) = len(NMC*(NMC+1))+(1-delta)*p;
      rad(NMC*(NMC+1)+1) = a(NMC+1);

      figure
      plot(len,rad);
      set(gca, "linewidth",2, "fontsize", 14 )
      xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
      ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
      title( 'Ring Loaded Mode Converter Profile', 'FontSize', 16 );
      axis equal;   % Scale axis equally for aspect ratio 1:1

      n = n+NMC+1
      for i = NMC+1:N;
        rad(n) = a(i);
        rad(n+1) = a(i)+d(i);
        rad(n+2) = a(i)+d(i);
        rad(n+3) = a(i+1);
        rad(n+4) = a(i+1);
        len(n+1) = len(n);
        len(n+2) = len(n+1)+delta*p;
        len(n+3) = len(n+2);
        len(n+4) = len(n+3)+(1-delta)*p;
        n = n+4;
      endfor

      z_number = (NMC*2)+(N*4)+1; % Number of coordinate points for corrugated length of horn

      figure
      plot(len,rad);
      set(gca, "linewidth",2, "fontsize", 14 )
      xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
```

```
      ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
      title( 'Ring Loaded Mode Converter Horn Profile', 'FontSize', 16 );
      axis equal;   % Scale axis equally for aspect ratio 1:1

    case 3                % case 3=VARIABLE PITCH TO WIDTH SLOT MODE CONVERTER
      % Mode Converter depths for element j
      ajmc = a(1:NMC);                    % First indexes for mode converter
      idx = 1:NMC;
      djmc = (sigma*(lambda_c/1.15)+((idx-1)./(NMC-1)).*(lambda_c/4-(sigma*lambda_c/1.15))).*exp(1./
(2.114.*(kc*ajmc).^1.134))
      % Depth of remaining corrugations
      aj = a(NMC+1:end);
      idx = NMC+1:N+1;
      dj = ((lambda_c/4).*exp(1./(2.114.*(kc*aj).^1.134)))-((idx-NMC-1)/(N-NMC-1)).*((lambda_c/
4).*exp(1./(2.114.*(kc*aj).^1.134))-(lambda_o/4).*exp(1./(2.114.*(ko*ao).^1.134)))
      d = [djmc, dj];       % Combining the mode converter and horn depth values

      % Generate z,y coordinates as len,rad vector
      n = 0;
      len(1) = 0;
      len(2) = 0;
        for i = 1:NMC;
          rad(i+n) = a(i);
          rad(i+n+1) = a(i)+d(i);
          rad(i+n+2) = a(i)+d(i);
          rad(i+n+3) = a(i+1);
          rad(i+n+4) = a(i+1);
          len(i+n+2) = len(i+n)+(delta_min+(((i-1)./(NMC-1))*(delta-delta_min)))*p;
          len(i+n+3) = len(i+n+2);
          len(i+n+4) = len(i+n+3)+(1-(delta_min+(((i-1)./(NMC-1))*(delta-delta_min))))*p;
          len(i+n+5) = len(i+n+4);
          n = n+3;
        endfor

      figure
      plot(len(1:end-1),rad);
      set(gca, "linewidth",2, "fontsize", 14 )
      xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
      ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
      title( 'Variable Pitch To Width Mode Converter Profile', 'FontSize', 16 );
      axis equal;   % Scale axis equally for aspect ratio 1:1

        for i = NMC+1:N;
          rad(i+n) = a(i);
          rad(i+n+1) = a(i)+d(i);
          rad(i+n+2) = a(i)+d(i);
          rad(i+n+3) = a(i+1);
          rad(i+n+4) = a(i+1);
          len(i+n+2) = len(i+n)+delta*p;
          len(i+n+3) = len(i+n+2);
          len(i+n+4) = len(i+n+3)+(1-delta)*p;
          len(i+n+5) = len(i+n+4);
          n = n+3;
        endfor

      z_number = (N*4)+1; % Number of coordinate points for corrugated length of horn
      len = len(1:z_number); % Truncate z axis data points to equal rad vector length

      figure
      plot(len,rad);
      set(gca, "linewidth",2, "fontsize", 14 )
      xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
      ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
      title( 'Variable Pitch To Width Mode Converter Horn Profile', 'FontSize', 16 );
      axis equal;   % Scale axis equally for aspect ratio 1:1

endswitch

% Add the rest of the geometry to create a closed path
% a_offset is the inner horn profile shifted up to give the horn a thickness
a_offset = a.+(lambda_c/2+2);
%figure;                  % Uncomment these three lines for debugging
%plot(z, a_offset);
%axis equal;

% Add vertical surface at horn aperture
len = [len, len(z_number)];
rad = [rad, a_offset(N)];
radmsh=rad;                % radmesh to fix mesh lines to corrugations
%figure;                   % Uncomment these three lines for debugging
```

```
%plot(len, rad);
%axis equal;

% Flip outer surface profile so that widest horn dimensions comes next in the outline coordinates
outer_surface = fliplr(a_offset);
z_flip = fliplr(z);
extent = len(end);  % Fudge to make horn aperture planar for ring loaded slot MC
z_flip(1) = extent; % Fudge to make horn aperture planar for ring loaded slot MC
% Add outer profile and circular waveguide to horn
len = [len, z_flip, -wgl, -wgl, 0];
rad = [rad, outer_surface, ai+(lambda_c/2+2), ai, ai];

figure
plot(len,rad);
set(gca, "linewidth",2, "fontsize", 14 )
xlabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
ylabel( 'Dimension in y Direction (mm)', 'FontSize', 14 );
title( 'Complete Corrugated Horn Profile', 'FontSize', 16 );
axis equal;   % Scale axis equally for aspect ratio 1:1

% openEMS setup begins here
% EM related physical constants
physical_constants;

% frequency range of interest
f_start =  fmin*1e9;
f_stop  =  fmax*1e9;

% frequency to calculate fields
f0 = 12.46*1e9;

%% setup FDTD parameter & excitation function
FDTD = InitFDTD( 'NrTS', 50000, 'EndCriteria', 0.5e-3 );
FDTD = SetGaussExcite(FDTD,0.5*(f_start+f_stop),0.5*(f_stop-f_start));
BC = {'PML_8' 'PML_8' 'PML_8' 'PML_8' 'PML_8' 'PML_8'}; % boundary conditions
FDTD = SetBoundaryCond(FDTD, BC);

%% setup CSXCAD geometry & mesh
max_res = c0/(f_stop)/unit/20; % cell size: lambda/20
CSX = InitCSX();               % Initialise CSX structure

% Calculate lambda/4 at lowest frequency to use as distance to nf2ff surfaces
lambda_max = c0/f_start/unit/4;

% Create fixed lines for the simulation box, structure and port
mesh.x = [(-a_offset(end)-(9*max_res)-lambda_max) -radmsh(1:4:end) 0 radmsh(1:4:end) (a_offset(end)
+(9*max_res)+lambda_max)];
mesh.x = SmoothMeshLines( mesh.x, max_res, 1.5); % Create a smooth mesh between specified fixed mesh
lines
mesh.y = mesh.x;                                 % Same as x mesh
% Create fixed lines for the simulation box,port and given number of lines inside the horn
mesh.z = [-wgl-lambda_max-(9*max_res) -wgl-1 -wgl -wgl+10 0 len(1:2:z_number)
length+2*lambda_max+(9*max_res)];
mesh.z = SmoothMeshLines( mesh.z, max_res, 1.4 );

CSX = DefineRectGrid( CSX, unit, mesh );

%% create horn
% horn + waveguide, defined by a rotational polygon
CSX = AddMetal(CSX, 'Corrugated_Horn');
coords = [rad; len];
CSX = AddRotPoly(CSX,'Corrugated_Horn',10,'x',coords,'z');

% End cap to prevent the radiation coming out of the back of the horn
CSX = AddMetal(CSX, 'Cap');
CSX = AddCylinder(CSX,'Cap',10,[0 0 -wgl],[0 0 (-wgl-1)],a_offset(1));
% End of model geometry

%% Apply the excitation %%
start=[-ai -ai -wgl];
stop =[ai ai -wgl+10];
[CSX, port] = AddCircWaveGuidePort( CSX, 0, 1, start, stop, ai*unit, 'TE11', 0, 1);

% Dump box for Electric field at Phi=0 (vertical cut)
CSX = AddDump(CSX,'Et_V_dump', 'SubSampling', '4,4,4');
start=[0 (-a_offset(end)-lambda_max) (-wgl-lambda_max)];
stop =[0 (a_offset(end)+lambda_max) (length+2*lambda_max)];
CSX = AddBox(CSX,'Et_V_dump',0,start,stop);

% Dump box for Electric field at Phi=90 (horizontal cut)
```

```
CSX = AddDump(CSX,'Et_H_dump', 'SubSampling', '4,4,4');
start=[(-a_offset(end)-lambda_max) 0 (-wgl-lambda_max)];
stop =[(a_offset(end)+lambda_max) 0 (length+2*lambda_max)];
CSX = AddBox(CSX,'Et_H_dump',0,start,stop);

% nf2ff calc
start = [mesh.x(9) mesh.y(9) mesh.z(9)];
stop  = [mesh.x(end-8) mesh.y(end-8) mesh.z(end-8)];
[CSX nf2ff] = CreateNF2FFBox(CSX, 'nf2ff', start, stop, 'Directions', [1 1 1 1 1 1],
'OptResolution', max_res*4);

% Prepare simulation folder
Sim_Path = 'tmp';
Sim_CSX = 'Corrugated_Horn.xml';
[status, message, messageid] = rmdir( Sim_Path, 's'); % Clear previous directory
[status, message, messageid] = mkdir( Sim_Path ); % Create empty simulation folder

% Write openEMS compatible xml-file
WriteOpenEMS([Sim_Path '/' Sim_CSX], FDTD, CSX);

% Show the structure
if(RUN_CSXCAD == 1)
% CSXGeomPlot([Sim_Path '/' Sim_CSX], ['--export-polydata-vtk=tmp']);
CSXGeomPlot([Sim_Path '/' Sim_CSX], ['--export-STL=tmp']);
end

% Run openEMS
if(RUN_SIMULATION == 1)
%openEMS_opts = '--debug-PEC --no-simulation';   % Uncomment to visualise mesh in Paraview
%RunOpenEMS(Sim_Path, Sim_CSX, openEMS_opts);
RunOpenEMS(Sim_Path, Sim_CSX, '--numThreads=7');
end

% Postprocessing & do the plots
freq = linspace(f_start,f_stop,201);
port = calcPort(port, Sim_Path, freq);

Zin = port.uf.tot ./ port.if.tot;
s11 = port.uf.ref ./ port.uf.inc;

% Plot reflection coefficient S11
figure
plot(freq/1e9, 20*log10(abs(s11)), 'k-', 'Linewidth', 2);
xlim([fmin fmax]);
ylim([-40 0]);
set(gca, "linewidth",2, "fontsize", 14)
grid on
title('Reflection Coefficient S_{11}', 'FontSize', 16);
xlabel('Frequency (GHz)','FontSize', 14);
ylabel('Reflection Coefficient |S_{11}| (dB)','FontSize', 14);
drawnow

% NFFF plots

% Calculate the far field at phi=0, 45 and at phi=90 degrees
thetaRange = (0:0.2:359) - 180;
disp('calculating far field at phi=[0 45 90] deg...');
nf2ff = CalcNF2FF(nf2ff, Sim_Path, f0, thetaRange*pi/180, [0 45 90]*pi/180);

Dlog=10*log10(nf2ff.Dmax);      % Calculate maximum Directivity in dB
G_a = 4*pi*A_app/(c0/f0)^2;     % Calculate theoretical gain for given aperture
e_a = nf2ff.Dmax/G_a;           % Calculate Efficiency

% Display some antenna parameters from above calculations
disp(['radiated power: Prad = ' num2str(nf2ff.Prad) ' Watt']);
disp(['directivity: Dmax = ' num2str(Dlog) ' dBi']);
disp(['aperture efficiency: e_a = ' num2str(e_a*100) '%']);

% Directivity
figure
plotFFdB(nf2ff,'xaxis','theta','param',[1 2 3]);
ylim([-30 25]);
xlim([-180 180]);
grid on
set(gca,"linewidth",2, "fontsize", 14, "XTick", -180:30:180, "YTick", -30:5:40)
title('Farfield Directivity @ 12.46GHz','FontSize', 16);
xlabel('Theta (degrees)','FontSize', 14);
ylabel('Directivity (dBi)','FontSize', 14);
drawnow
```

```
% Plot Ludwig3 cross polar
plotFFcocx(nf2ff,'xaxis','theta','param',[2]);
ylim([-30 25]);
xlim([-180 180]);
grid on
set(gca,"linewidth",2, "fontsize", 14, "XTick", -180:30:180, "YTick", -30:5:40)
title('Farfield Directivity with Ludwig3 XPOL @ 12.46GHz','FontSize', 16);
xlabel('Theta (degrees)','FontSize', 14);
ylabel('Directivity (dBi)','FontSize', 14);
drawnow

% Polar plot
figure
leg=[];   %legend
polarFF(nf2ff,'xaxis','theta','param',[1 2 3],'logscale',[-30 35], 'xtics', 12);
title('Farfield Directivity @ 12.46GHz','FontSize', 16);
xlabel('Theta (degrees)','FontSize', 14);
ylabel('Directivity (dBi)','FontSize', 14);
drawnow

%% Calculate 3D pattern
%phiRange = sort(unique([-180:5:-100 -100:2.5:-50 -50:1:50 50:2.5:100 100:5:180]));
%thetaRange = sort(unique([0:1:50 50:2:100 100:5:180]));
phiRange = sort(unique([-180:1:-100 -100:1:-50 -50:1:50 50:1:100 100:1:180]));
thetaRange = sort(unique([0:1:50 50:1:100 100:1:180]));

disp('calculating 3D far field...');
nf2ff = CalcNF2FF(nf2ff, Sim_Path, f0, thetaRange*pi/180, phiRange*pi/180,
'Verbose',2,'Outfile','nf2ff_3D.h5');

figure
colormap jet;
plotFF3D(nf2ff, 'logscale', -40);        % plot 3D far field in dB

% Save far field in VTK to plot in ParaView
E_far_normalized = nf2ff.E_norm{1}/max(nf2ff.E_norm{1}(:));
DumpFF2VTK([Sim_Path '/Farfield.vtk'],E_far_normalized,thetaRange,phiRange,'scale', 0.008,
'logscale', -30, 'maxgain', Dlog);

%%% END OF SCRIPT %%%
```

Alternative Method to Compute the Horn Cross-sections

Dr Granet suggests using the following algorithm to calculate the N-doublets for calculating the cross-sections.

n = 0

do j = 1 to N

  n = n +1

  radius(n) = a(j)+d(j)

  length(n) = δ p

  n = n + 1

  radius(n) = a(j)

  length(n) = (1-δ) p

end do

END OF DOCUMENT