nikiml's Antenna pages - nec.opt tutorial

Nikolay Mladenov

In this tutorial we will optimize the reflectorless GH antenna.

Prerequisites

I will assume that you have installed Python 3.* on Windows and have installed the nec.opt scripts from one of the windows executable packages.

You will also need a good text editor - Windows Notepad and Wordpad are not appropriate. A good free editor is Notepad++.

Complete nec reference can be found here. Nec.opt also allows SYmbols just like in 4nec2 and some other "format extensions" but the compatibility with 4nec2 is preserved.

Nec file

So to start, open your text editor and create a new file named "gh0.nec" and save it in the folder you have selected for the optimizer to run into. Add the following comment section in the beginning of the file (lines starting with CM are comments, CE means "'Comment End").

```
CM
CM GH0 model, optimized with nec.opt by Your Name
CM
CMD--OPT  -s(470,19,13) -t(9,10) -n4 -a10 -r restart.log
CMD--OPT  -F .5*ave_gain_diff+.5*max_gain_diff
CMD--EVAL -u -a10
CM
CE
```

The lines starting with `CMD--OPT` and `CMD--EVAL` are used to pass options to the nec.opt and nec.eval scripts correspondingly and are disguised as regular Nec2 comment cards. Be careful when editing the file in 4nec2 since it may reformat the command option sections.

Options

The options have the following meaning:

-s(470,19,13) - sweep frequencies starting from 470MHz and stepping 13 times by 19MHz each. This means the optimizer will work on the frequencies: 470,489,508,527,546,565,584,603,622,641,660,679 and 698 which is UHF band sampled

with step of 19 MHz.

-t(9,10) - sets the target gain for the antenna to line starting from 9dBi at 470 MHz and going to 10 dBi at 698Mhz (see chart on the right)

-n4 - splits the frequency range in to 4 parts so they can be evaluated in a separate threads/cores

-a10 - instructs the optimizer and evaluator to automatically segment the model (and all generated during the optimization models) with 10 segments per half wavelength at the model design frequency.

-r restart.log - instructs the optimizer try to restart from a previously generated restart.log file.

-F .5*ave_gain_diff+.5*max_gain_diff - that is the target function the optimizer will be minimizing. More on the that later.

-u - is a shortcut for -s(470,6,39) which is a frequency sweep of the UHF band in 39 steps of 6MHz. The optimizer uses the coarser step of 19Mhz just so the optimization goes faster.

Variables

Lets define the nec file variables that will drive our antenna model. They will be based on the drawing to the right and we will give them the following the limits:
w1,w3,w4,h1,h2,h3 - from 0.07m to 0.3m (from 3 inch to a foot), and **feed** - from 0.04m to 0.1m (1.5 to 4 inches) and **gap** - from 0.04m to 0.2m

Because we have the following equation: `feed/2 + w1 = gap/2 + w2` we don't need to define the variable **w2**.

to defined a variable for optimization we use the following format:
`SY var_name=initial_value ' lower_limit, upper_limit`
and variables dependant on other variables have no limits.

Therefore we append the following code to our nec file:

```
SY feed=0.06 ' 0.04, 0.1
SY gap=0.1 ' 0.04, 0.2
SY w1=0.2 ' 0.07, 0.3
SY w3=0.2 ' 0.07, 0.3
SY w4=0.2 ' 0.07, 0.3
SY h1=0.2 ' 0.07, 0.3
SY h2=0.2 ' 0.07, 0.3
SY h3=0.2 ' 0.07, 0.3
```

Geometry

We are now ready to define the geometry of the antenna and we will start with the upper

right portion of the blue wire. Each straight portion of the wire is generated with the following line format:

```
GW tag segs start_x start_y start_z end_x end_y end_z radius
```

Here `tag` is a number that allows are to refer to this wire and `segs` is the number of segments in which the wire is split (this number will be figured out by nec.opt based on the **-a10** option)

Here are the geometry defining lines we append to the nec file, starting with a radius variable:

```
SY radius=0.003175
GW 1 1 0 feed/2 0 0 feed/2+w1 h1 radius ' #00f
GW 2 1 0 feed/2+w1 h1 0 gap/2 h1+h2 radius
GW 3 1 0 gap/2 h1+h2 0 gap/2+w3 h1+h2+h3 radius
GW 4 1 0 gap/2+w3 h1+h2+h3 0 gap/2+w3+w4 h1+h2+h3 radius
```

We can continue like that and build the rest of the antenna, but because the blue portion of the model is symmetric we can use mirror operation to define the remaining three blue sections

```
GX 5 011
```

The GX line mirrors the already defined geometry first across Y and then across Z, incrementing the wire tags by 5 every time.

The only wire left to define now is the source orange wire

```
SY source_radius=radius*0.85
GW 100 3 0 -feed/2 0 0 feed/2 0 source_radius ' #f40
GE
```

The source_wire variable allows us to adjust the radius of the source wire for AGT=1.0, the GE line signifies the end of the geometry definition

To finish the model we append the following lines, note the EX and the FR lines

```
LD      5       0       0       0       24900000 ' means all wires are T6 aluminum
GN      -1  ' no ground plane, free space
EK ' request extended wire kernel
EX 0 100 2 0 1 'the source is in the middle of the 2nd(out of 3) segment of the wire 100
FR      0       0       0       0       800     0 'the design frequency is 800MHz
EN
```

Complete nec file

```
CM
CM GH0 model, optimized with nec.opt by Your Name
CM
CMD--OPT  -s(470,19,13) -t(9,10) -n4 -a10 -r restart.log
CMD--OPT  -F .5*ave_gain_diff+.5*max_gain_diff
CMD--EVAL -u -a10
CM
```

```
CE

SY feed=0.06 ' 0.04, 0.1
SY gap=0.1 ' 0.04, 0.2
SY w1=0.2 ' 0.07, 0.3
SY w3=0.2 ' 0.07, 0.3
SY w4=0.2 ' 0.07, 0.3
SY h1=0.2 ' 0.07, 0.3
SY h2=0.2 ' 0.07, 0.3
SY h3=0.2 ' 0.07, 0.3

SY radius=0.003175
GW 1 1 0 feed/2 0 0 feed/2+w1 h1 radius ' #00f
GW 2 1 0 feed/2+w1 h1 0 gap/2 h1+h2 radius
GW 3 1 0 gap/2 h1+h2 0 gap/2+w3 h1+h2+h3 radius
GW 4 1 0 gap/2+w3 h1+h2+h3 0 gap/2+w3+w4 h1+h2+h3 radius
GX 5 011
SY source_radius=radius*0.85
GW 100 3 0 -feed/2 0 0 feed/2 0 source_radius ' #f40
GE

LD      5       0       0       0       24900000 ' means all wires are T6 aluminum
GN      -1  ' no ground plane, free space
EK ' request extended wire kernel
EX 0 100 2 0 1 'the source is in the middle of the 2nd(out of 3) segment of the wire 100
FR      0       0       0       0       800     0 'the design frequency is 800MHz
EN
```

Evaluating the input file

Right clicking on the saved nec files select evaluate.

This will generate an html output file gh0.nec.html and a console printout:

```
Input file : gh0.nec
Freq sweeps: [(470, 6, 39)]
Autosegmentation: 10 per 0.1875
```

Freq	RawGain	NetGain	SWR	BeamW	F/R	F/B	Real	Imag	AGT(corr)
470.0	9.674	8.943	2.299	32.4	0.000	0.000	180.45	-159.49	1.03(0.146)
476.0	9.044	8.139	2.532	31.5	0.000	0.000	145.59	-129.05	1.03(0.146)
482.0	8.274	7.208	2.750	31.0	0.000	0.000	122.68	-97.71	1.03(0.146)
488.0	7.384	6.183	2.934	30.9	0.000	0.000	108.18	-67.56	1.03(0.146)
494.0	6.394	5.095	3.069	31.8	0.000	0.000	99.62	-39.18	1.03(0.146)
500.0	5.354	3.997	3.151	36.4	0.000	0.000	95.41	-12.56	1.03(0.146)
506.0	4.304	2.927	3.179	130.7	-1.010	0.000	94.56	12.50	1.03(0.146)
512.0	3.344	1.980	3.160	135.4	-2.390	0.000	96.49	36.27	1.03(0.146)
518.0	2.554	1.231	3.102	138.7	-3.510	0.000	100.89	58.97	1.03(0.146)
524.0	2.022	0.761	3.015	140.3	-4.290	0.000	107.68	80.76	1.03(0.128)
530.0	1.742	0.559	2.907	140.3	-4.740	0.000	116.88	101.70	1.03(0.128)
536.0	1.712	0.619	2.785	139.1	-4.960	0.000	128.67	121.75	1.03(0.128)
542.0	1.882	0.885	2.655	136.8	-4.940	0.000	143.32	140.75	1.03(0.128)
548.0	2.162	1.265	2.520	134.0	-4.770	0.000	161.15	158.28	1.03(0.128)

```
 554.0    2.502    1.707   2.384   130.9  -4.510   0.000   182.49   173.70  1.03( 0.128)
 560.0    2.852    2.158   2.248   127.5  -4.220   0.000   207.55   186.02  1.03( 0.128)
 566.0    3.192    2.596   2.115   124.0  -3.980   0.000   236.24   193.83  1.03( 0.128)
 572.0    3.502    3.002   1.984   120.7  -3.750   0.000   267.85   195.42  1.03( 0.128)
 578.0    3.772    3.362   1.857   117.4  -3.540   0.000   300.74   188.96  1.03( 0.128)
 584.0    4.011    3.685   1.735   114.2  -3.340   0.000   331.98   173.19  1.03( 0.119)
 590.0    4.181    3.932   1.618   111.5  -3.170   0.000   357.71   148.29  1.03( 0.119)
 596.0    4.291    4.110   1.506   108.8  -3.040   0.000   374.04   116.63  1.03( 0.119)
 602.0    4.341    4.218   1.400   106.3  -2.930   0.000   378.58    82.66  1.03( 0.119)
 608.0    4.321    4.246   1.301   104.1  -2.900   0.000   371.64    51.50  1.03( 0.119)
 614.0    4.231    4.192   1.210   102.2  -2.900   0.000   356.05    27.22  1.03( 0.119)
 620.0    4.061    4.046   1.126   100.6  -2.930   0.000   335.73    11.70  1.03( 0.119)
 626.0    3.811    3.808   1.051    98.9  -2.990   0.000   314.39     4.80  1.03( 0.119)
 632.0    3.481    3.480   1.025    97.5  -3.080   0.000   294.69     5.20  1.03( 0.119)
 638.0    3.071    3.063   1.088    96.2  -3.200   0.000   278.17    11.12  1.03( 0.119)
 644.0    2.598    2.576   1.153    95.1  -3.340   0.000   265.54    20.94  1.03( 0.112)
 650.0    2.038    1.997   1.216    94.1  -3.520   0.000   257.03    33.32  1.03( 0.112)
 656.0    1.428    1.364   1.274    93.1  -3.700   0.000   252.61    47.21  1.03( 0.112)
 662.0    0.798    0.711   1.327    92.2  -3.860   0.000   252.17    61.82  1.03( 0.112)
 668.0    0.168    0.059   1.374    91.1  -3.990   0.000   255.65    76.51  1.03( 0.112)
 674.0   -0.392   -0.522   1.415    89.8  -4.030   0.000   263.02    90.70  1.03( 0.112)
 680.0   -0.832   -0.980   1.448    87.9  -3.930   0.000   274.36   103.77  1.03( 0.112)
 686.0   -1.062   -1.225   1.476    85.6  -3.610   0.000   289.79   114.97  1.03( 0.112)
 692.0   -1.062   -1.237   1.497    83.1  -3.060   0.000   309.40   123.31  1.03( 0.112)
 698.0   -0.812   -0.996   1.512    80.5  -2.250   0.000   333.20   127.44  1.03( 0.112)
```

We see that the AGT values (1.03) are close to 1 and we don't need to make correction of the source wire radius now, and we can proceed with...

Optimizing the input file

This is done by right clicking on the nec files selecting **optimize**.

The optimizer generates a best***.nec file every time it discovers one better the the previous best. The *** is the score for the target function for this file. A score of 0 roughly corresponds to reaching the gain target (option **-t(9,10)**)

The progress of the optimization process is output in a console window:

```
    1. Min score 2.08885, Mean score 7.12211, Improved 24 members, IterTime(38 sec)
.............................................
    2. Min score 0.712851, Mean score 6.10965, Improved 21 members, IterTime(33 sec)
.............................................
    3. Min score 0.712851, Mean score 5.24163, Improved 20 members, IterTime(31 sec)
.............................................
    4. Min score 0.712851, Mean score 4.45466, Improved 15 members, IterTime(28 sec)
.............................................
    5. Min score 0.652565, Mean score 3.8238, Improved 14 members, IterTime(27 sec)
```

One can wait for the optimizer to finish, but it is more practical to just interrupt it by pressing Ctrl+C once the Min score and the Means score are close enough. If the score is measuring a quantity equivalent to dB gain, close enough can be around .01, but this is a

matter of personal choice.

Restarting the optimization

Since the option `-r restart.log` is given to the optimizer, if a file named `restart.log` is present along your input file it will be used to restart the optimization from the state stored in this file.

If the optimizer is interrupted with Control+C it will automatically generated a restart file named `restart.`*`TodaysDate`*`.log`. All that is needed in this case is to rename this file to `restart.log` and rerun the Optimize command on the input file.

I cases where the optimizer was interrupted by other means (computer crash, power outage, etc.) and not restart log was generated automatically, it can be generated by right-clicking on the opt_log file and selecting **"Generate restart file"**. This will create the missing `restart.`*`TodaysDate`*`.log` and the optimizer can be restarted as before.

"Generate restart file" command also gives statistical data about the optimization state:

	Min	Max	Average	StdDev
Score	0.94941	7.85009	3.69400	1.85587
feed	0.04068	0.09991	0.07061	0.01592
gap	0.04582	0.17493	0.10592	0.02986
h1	0.08513	0.26351	0.15962	0.04264
h2	0.07337	0.26788	0.14781	0.05271
h3	0.07113	0.20922	0.12974	0.03571
w1	0.08008	0.26706	0.16092	0.04526
w3	0.07130	0.28963	0.16795	0.06354
w4	0.07363	0.27426	0.13915	0.05280
R0mg	1.85322	9.04331	5.24146	1.96822
R0ag	-0.18608	6.65686	2.14653	1.87788
R0ms	1.81477	5.86019	3.05330	0.73712
R0as	1.59782	3.08566	2.16571	0.32094

Those stats show how close is the optimizer to completion. If all Min, Max pairs are converging then the optimization is in "Local search state", the score would only change gradually and not by a lot, and the process is almost completed. Otherwise the optimizer is still doing "Global search" and big jumps of the best Score are still possible.

Score and target function

A score of an antenna is the value of the target function when evaluated with the results of the model evaluation. Since the optimizer always minimizes, it considers an antenna to be the best when it has the lowest possible score.

When designing the target function we should not forget the fact that it is always minimized. Therefore if we want to maximize the average net gain of an antenna we need to minimize the target function `-ave_net_gain`.

Another important fact is that the best antennas gain curve is naturally tilted upwards. This is because the amount of currents induced in the antenna is proportional to the amount of metal its made of - measured in wavelengths. So if an antenna works as efficiently at two frequencies **F** and **2F** it should have 2 times (~3dB) more gain at the higher frequency. For UHF the ratio of max frequency over min frequency is 698/470 which converted in dB is approximately 1.71dB. For VHF-hi it is 216/174 ~ 0.9dB.

Lets look at the optimization results for few different target functions

When optimizing for average net gain (option `-F -ave_net_gain`) we will get the result on the right. It has a average gain of 10.4dB but the difference between the min gain (7.4dBi) and the maximum gain (11.9) is 4.5dB. And for a general antenna we would probably like to have a bit higher minimum gain even if we have to sacrifice a little average gain.

If we try to maximize the minimum gain (option `-F -min_net_gain`) we will get the result on the left. The average gain is now 9.8dB, which is quite a bit less, and what we got for it is more gain from 470 to about 508. This may not seem like a good trade-of, but that is not the worst part. A bigger problem is that the optimizer, with this target function, will consider the flat gain curve at 8.5 dBi (in red) to be even better, and if it is possible will produce a model with that gain curve. Meaning that it will happily trade another 1.2dB average gain for just a tiny improvement at the band ends.

The solution to the shortcomings of the above two target functions is to use them in combination. A small average gain token in the target function will allow the optimizer to distinguish models with identical minimum gain and allow it to select the one with best average gain. For example we can use one of the following hybrid target functions:

- `-F -.5*min_net_gain-.5*ave_net_gain` (result on the right)
- `-F .5*max_gain_diff+.5*ave_gain_diff` (result below)

The second option is different because it uses the token **gain_diff**. This token is designed for direct minimization (there is no need to negate it), but what is more important it measures the gain not from 0 but from a predefined target gain curve.

In this case the target curve is defined with the option `-t(9,10)` which is a straight line going from 9dBi at 470 to 10dBi at 698. This is in agreement with the natural tendency of the gain curve to tilt upwards.

In conclusion - it is obvious that there is some trade-of to be made between maximizing average gain or the minimum gain, and the amount of the trade of can be controlled by adjusting the weights of the **max_gain_diff** and **ave_gain_diff** tokens in the target function. What are the best weights is subjective and one can only find them through experimentation.

nikolay dot mladenov at gmail dot com