Andy Howard
Applications Engineer
28 May, 2002

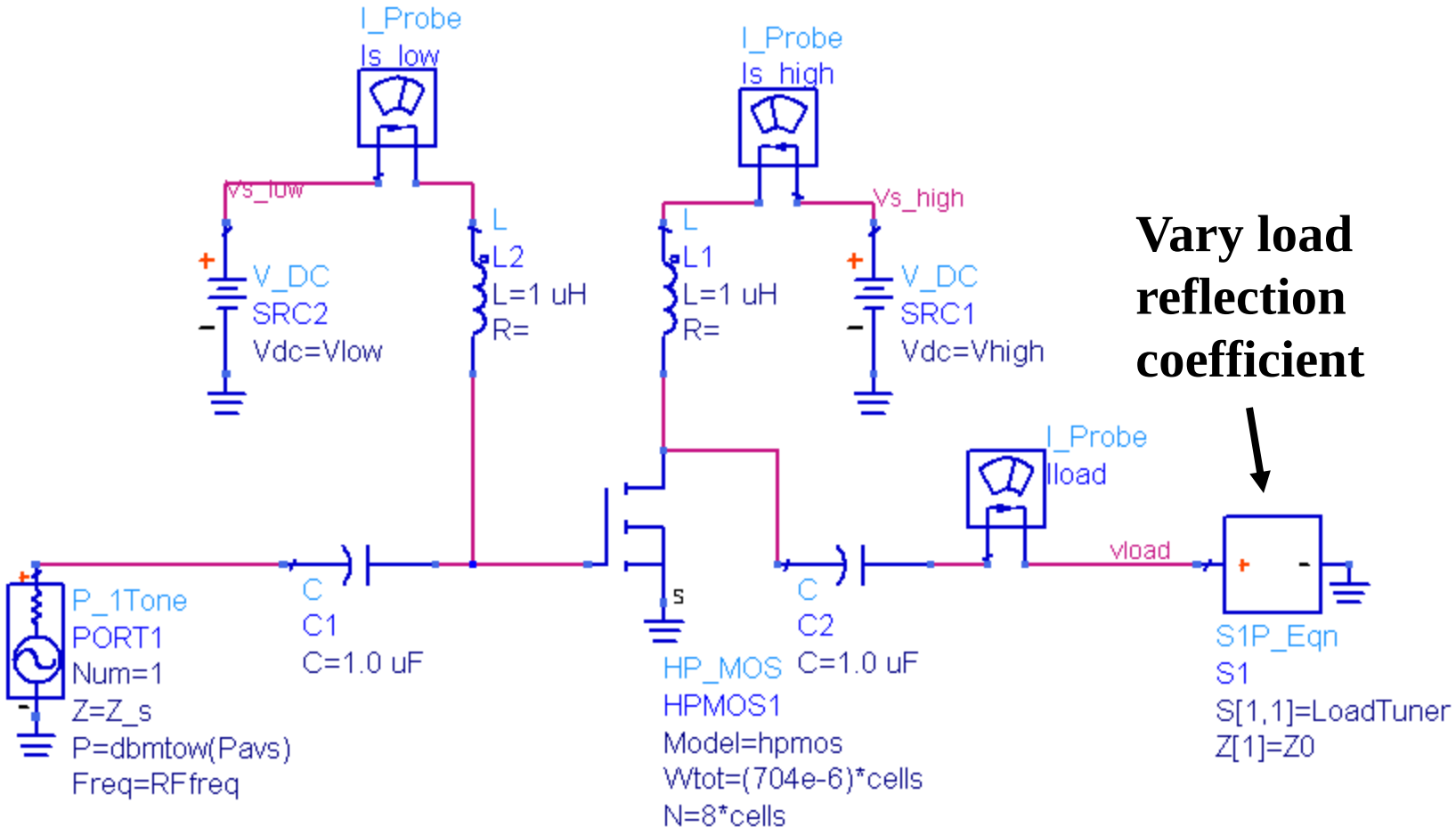
Load Pull Simulation Using ADS

Agilent Technologies

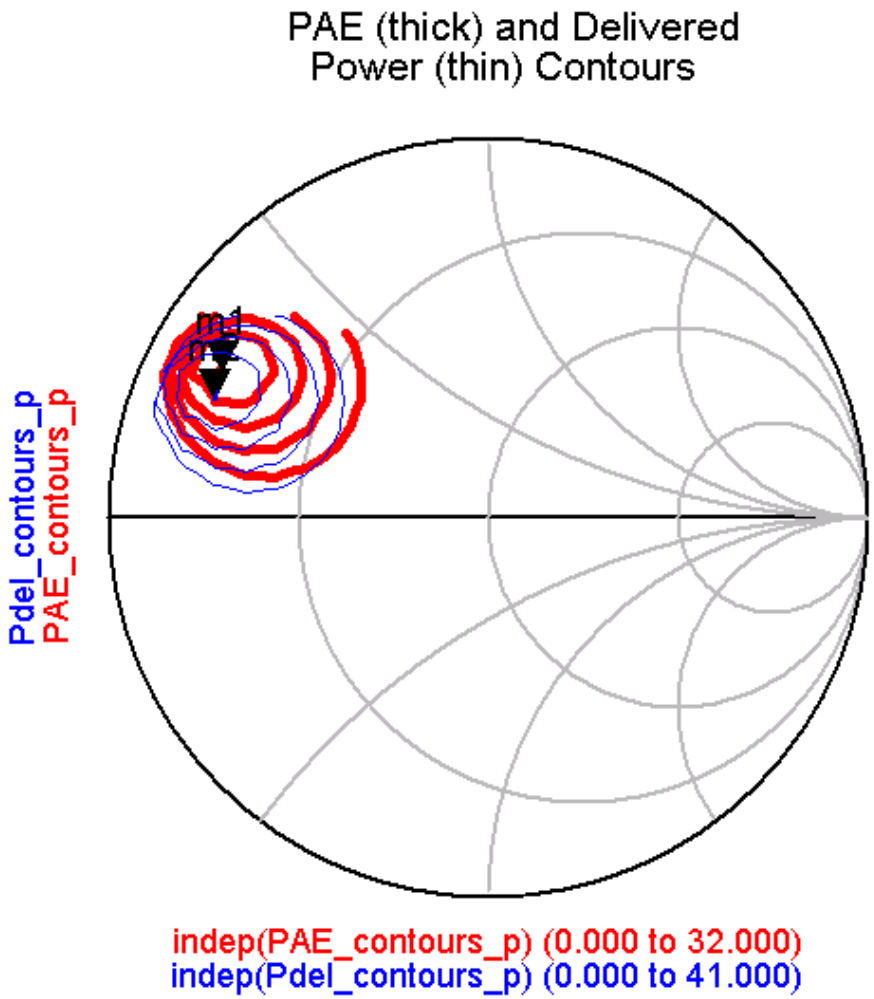
Outline

- Overview
- Specifying and generating desired load reflection coefficients
- Assigning arbitrary reflection coefficients at the harmonic freqs.
- Biasing the device and running a simulation
- Calculating desired responses (delivered power, PAE, etc.)
- Generating contour lines

Load pull simulation varies the load reflection coefficient presented to a device...



...to find the optimal value to maximize power or PAE, etc.



Set Delivered Power contour step size (dB) and PAE contour step size (%), and number of contour lines

Eqn Pdel_step=0.5

Eqn PAE_step=4

Eqn NumPAE_lines=5

Eqn NumPdel_lines=5

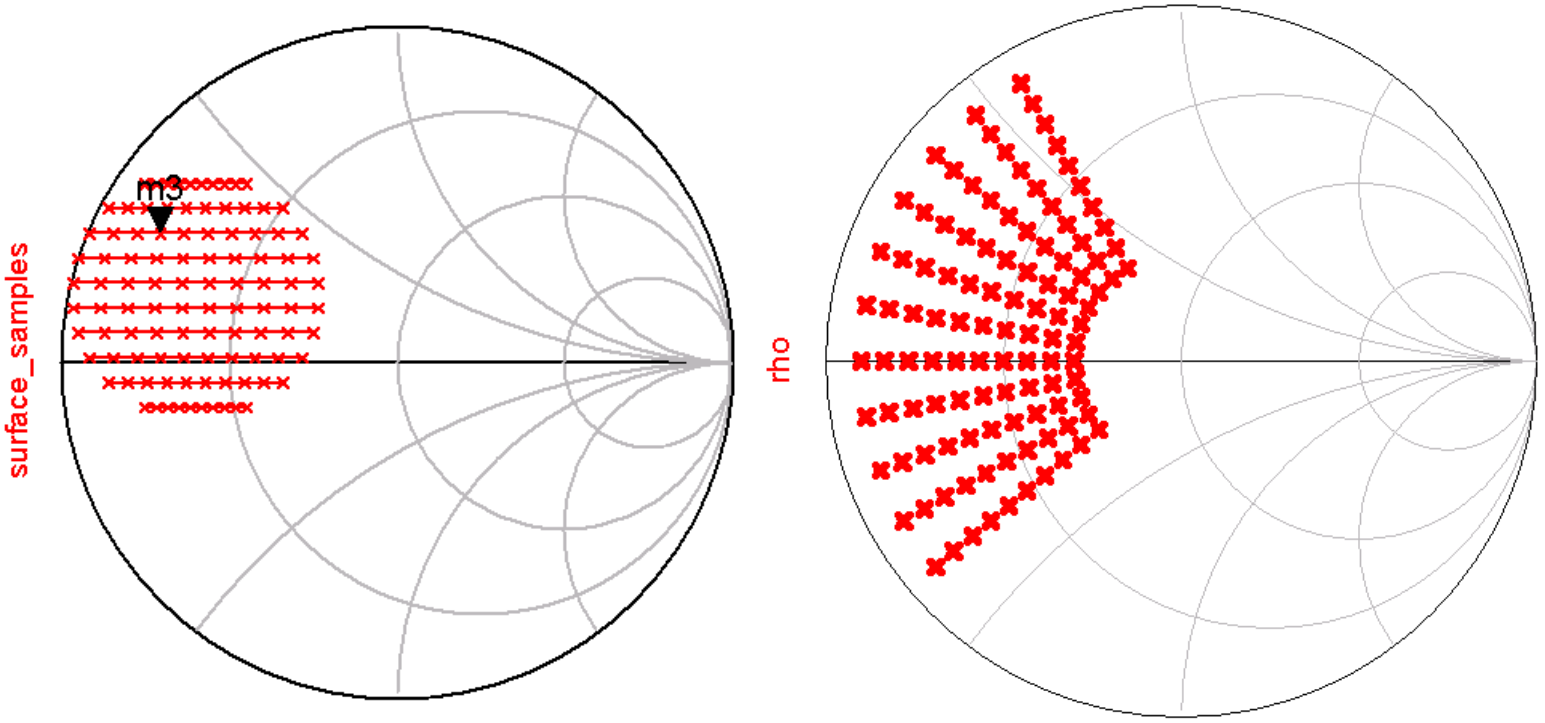
Maximum Power-Added Efficiency, %

40.84

Maximum Power Delivered, dBm

25.67

Specify the load reflection coefficients

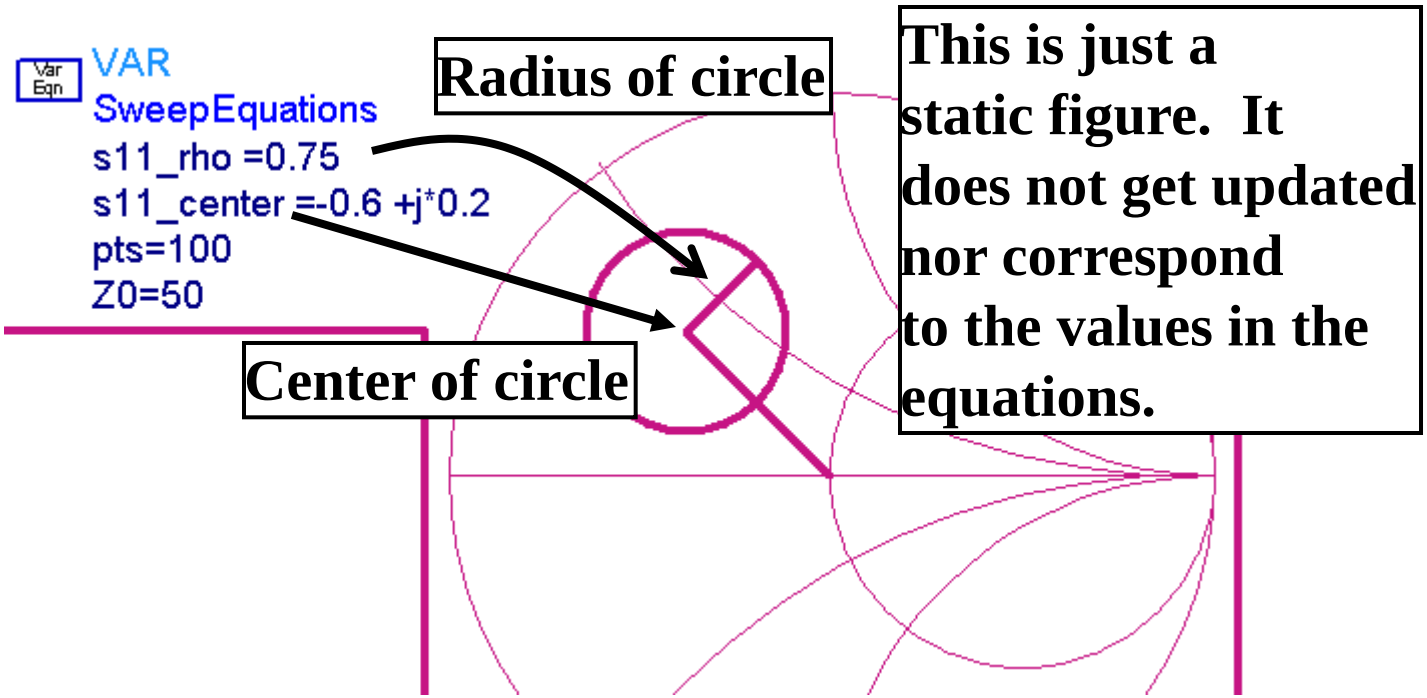


Simulation set up is complicated, but region is sampled more uniformly

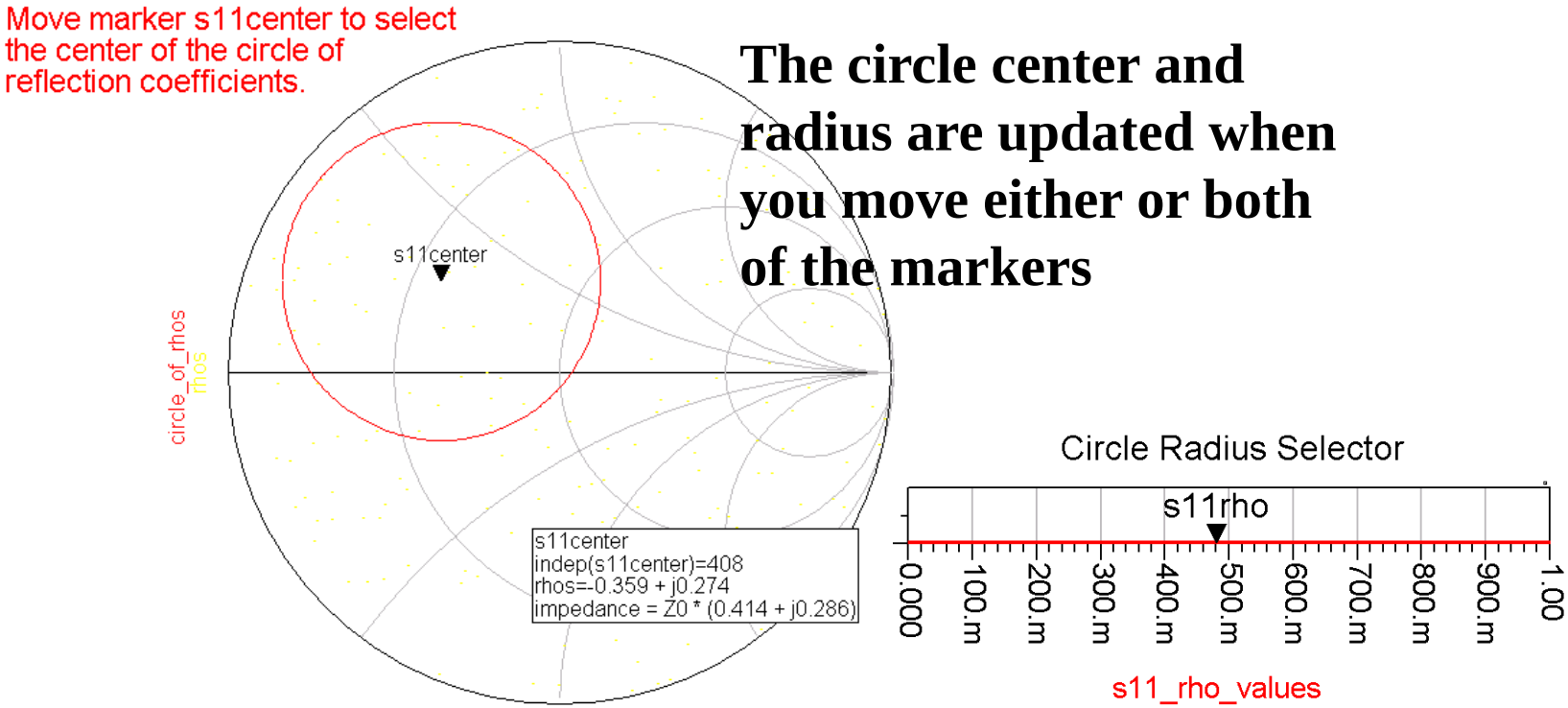
Simulation set up is trivial

Sweeping rho over a circular region

Specify circular region, via s11_rho and s11_center variables



Use ReflectionCoefUtility data display to help set s11_center and s11_rho



The circle center and radius are updated when you move either or both of the markers

Two variables are swept to generate the points within the desired circle

ParamSweep
Sweep1
SweepVar="imag_indexs11"
Start=imag(s11_center)-(max_rho-max_rho/(lines+1))
Stop=imag(s11_center)+(max_rho-max_rho/(lines+1))
Lin=lines

Will explain Start and Stop equations later

HarmonicBalance
HB1
Freq[1]=RFfreq
Order[1]=5
SweepVar="real_indexs11"
Start=real(s11_center)-c_limit
Stop=real(s11_center)+c_limit
Lin=pts_per_line

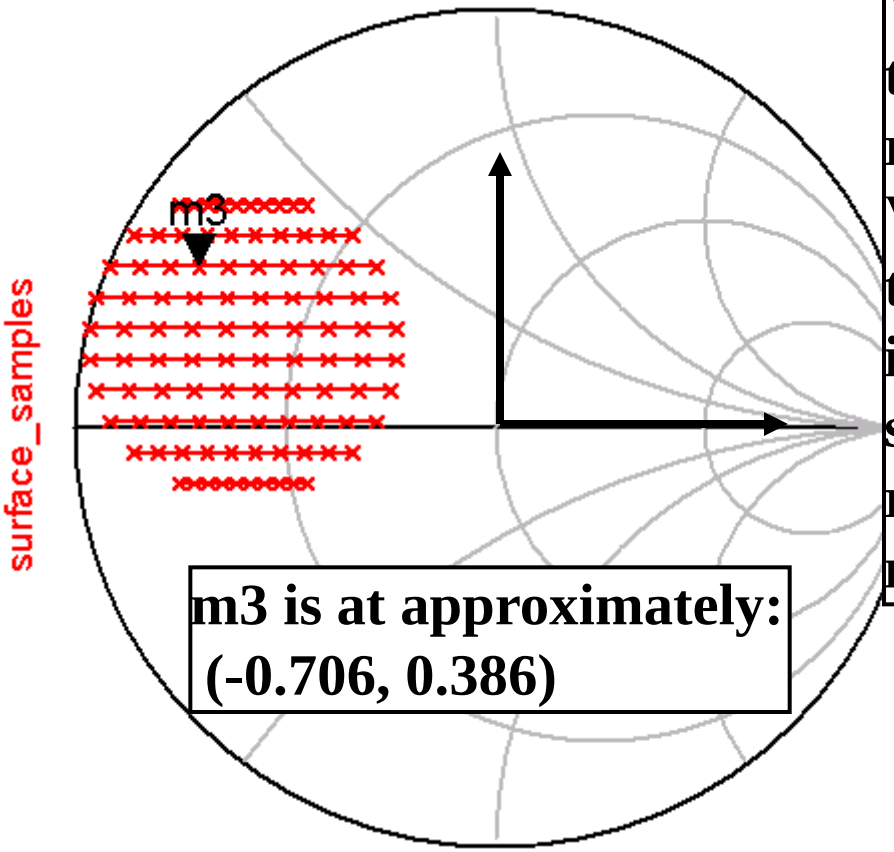
VAR
SweepEquations
real_indexs11=0
imag_indexs11=0
indexs11=real_indexs11+j*imag_indexs11

These initialize the swept variables

indexs11 is the reflection coefficient

More on the swept variables

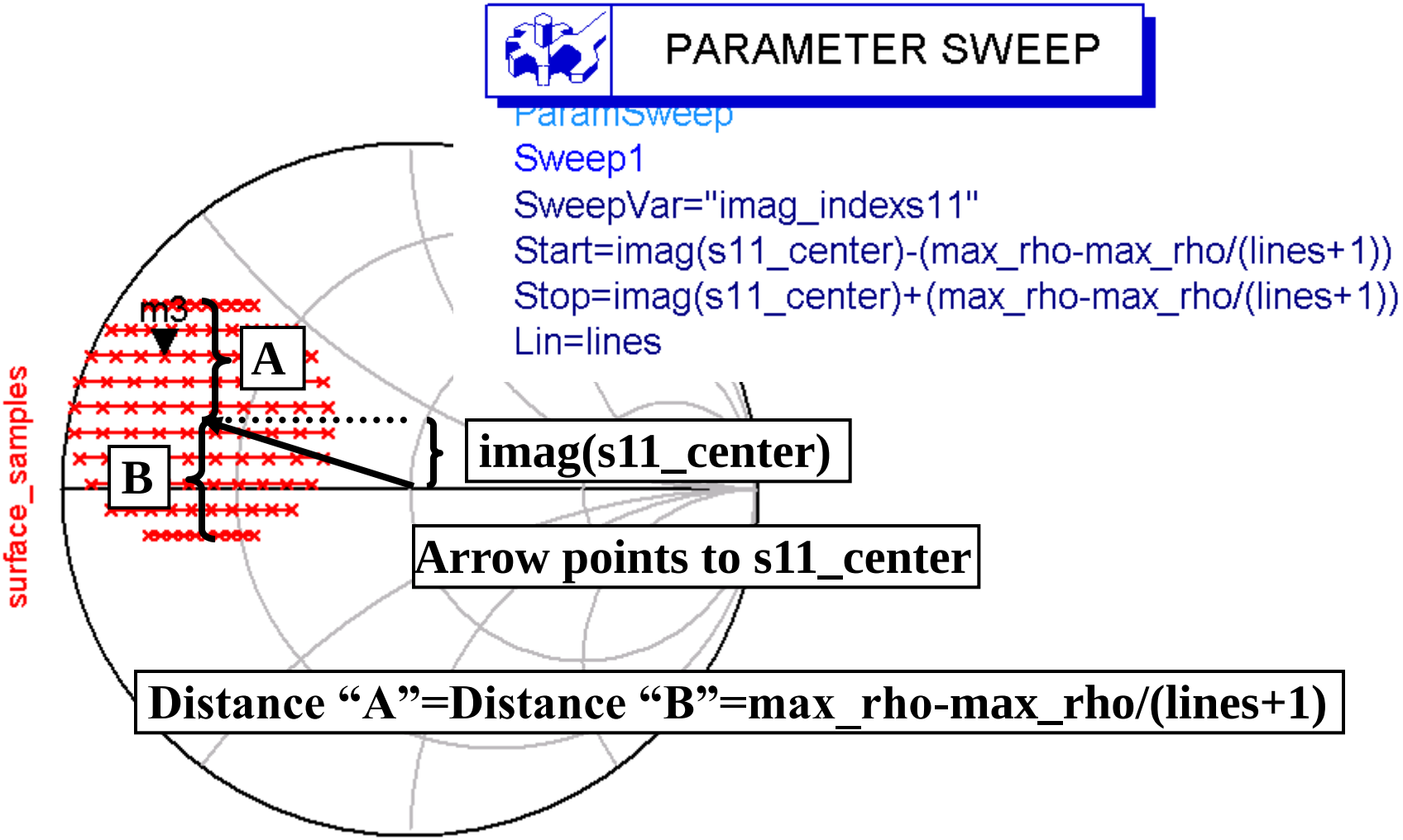
Each point is a data pair: (real_indexs11, imag_indexs11)



m3 is at approximately: (-0.706, 0.386)

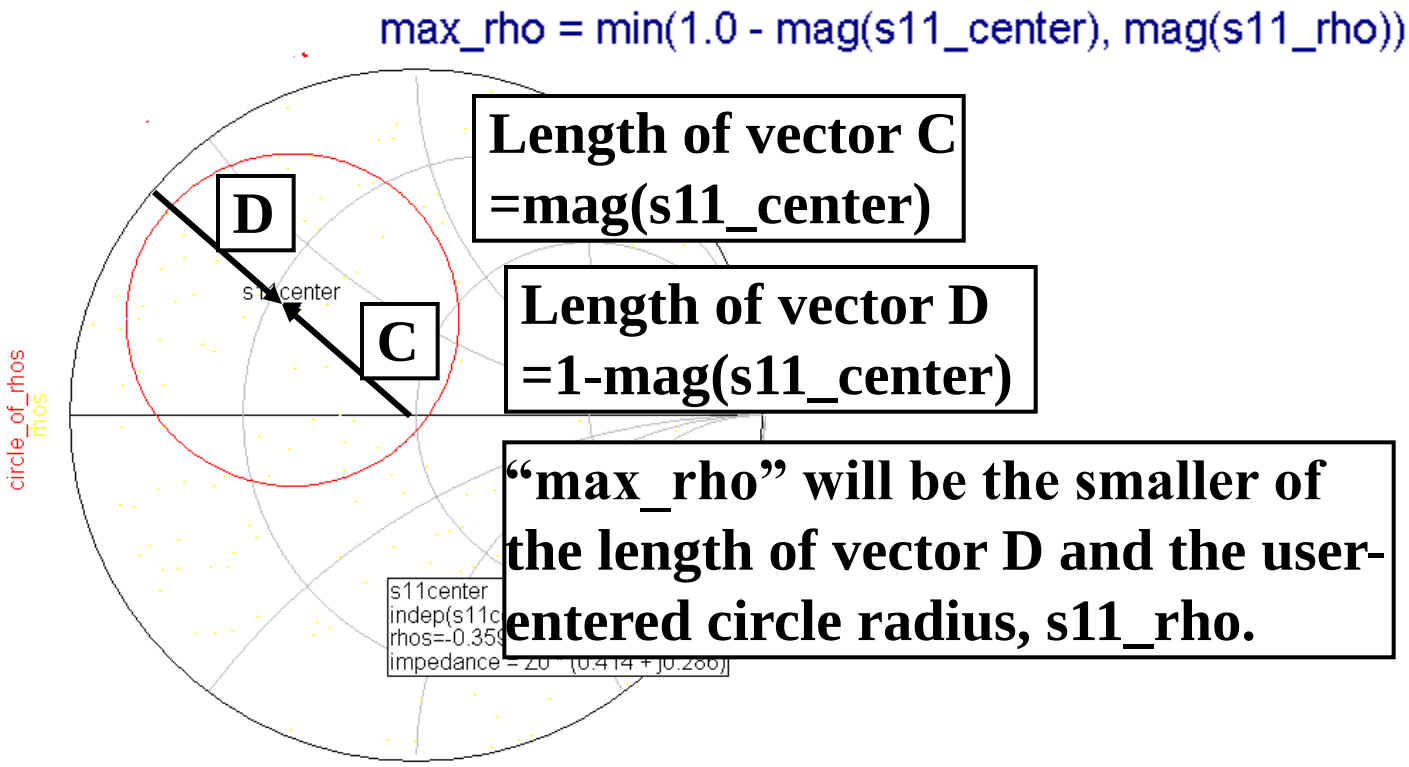
Variable "imag_indexs11" is the y-axis value of simulated reflection coefficient. Variable "real_indexs11" is the x-axis value. The imag_indexs11 variable is stepped uniformly, whereas real_indexs11 is stepped non-uniformly.

Sweep limits for “imag_indexs11”



What is “max_rho”?

The max_rho equation reduces the radius of the circle of simulated reflection coefficients if any part of the circle would otherwise be outside the Smith chart, which would imply an active load.



Length of vector C =mag(s11_center)

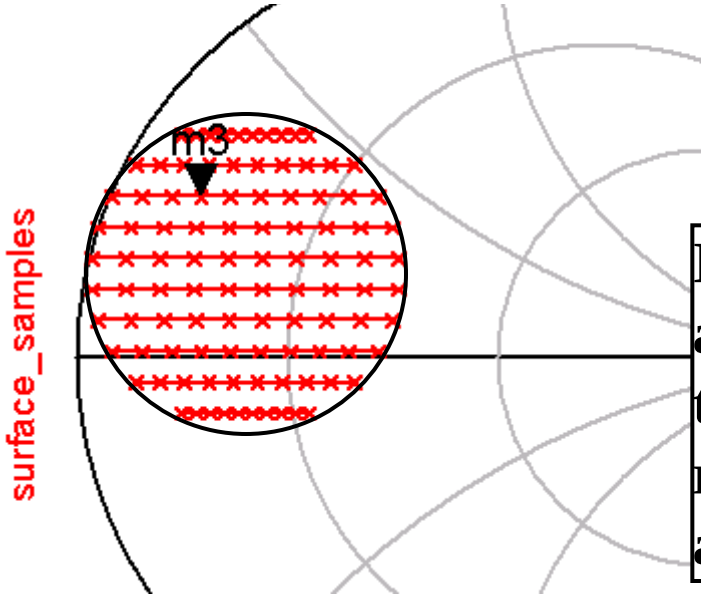
Length of vector D =1-mag(s11_center)

“max_rho” will be the smaller of the length of vector D and the user-entered circle radius, s11_rho.

What about the “max_rho/(lines+1)” term?

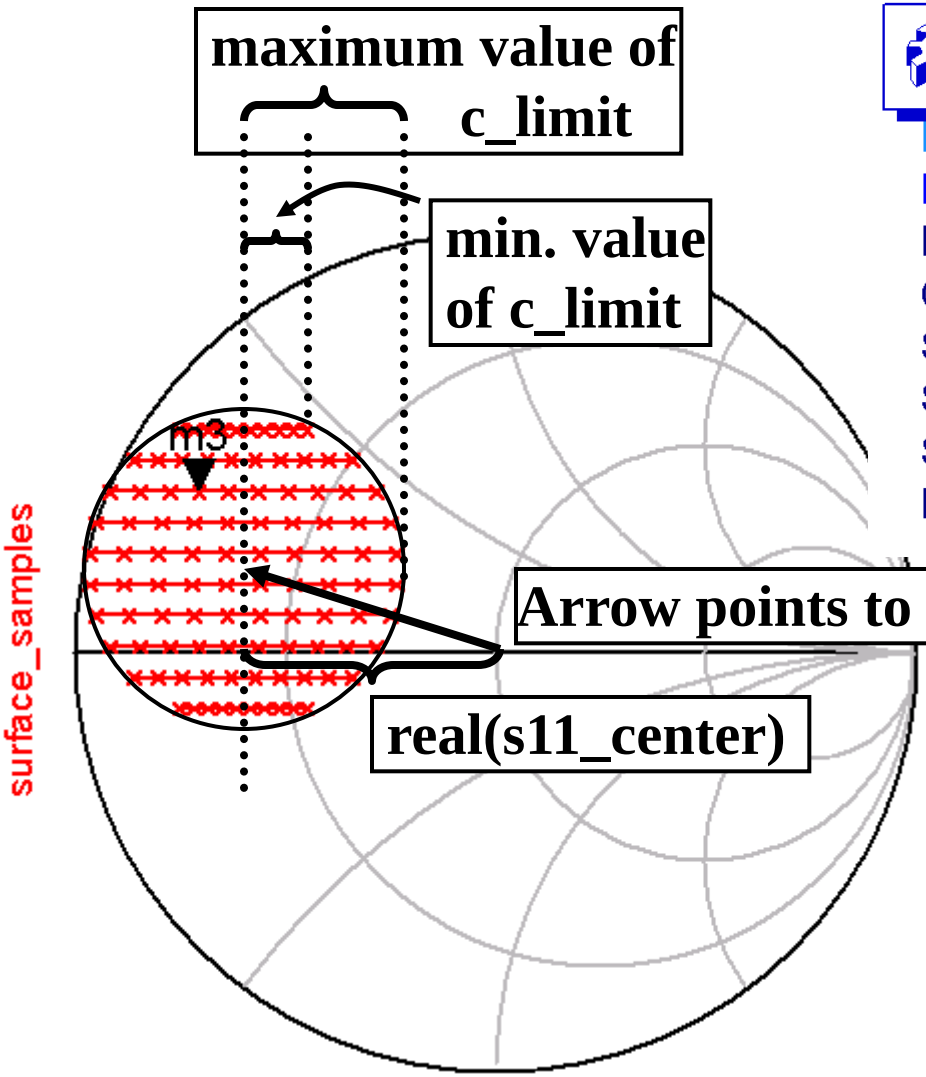
PARAMETER SWEEP

ParamSweep
Sweep1
SweepVar="imag_indexs11"
Start=imag(s11_center)-(max_rho-max_rho/(lines+1))
Stop=imag(s11_center)+(max_rho-max_rho/(lines+1))
Lin=lines



Including this term prevents the first and last line from being tangent to the circle at a single point. This allows real_indexs11 to be swept along the first and last lines defined by imag_indexs11.

Sweep limits for “real_indexs11”



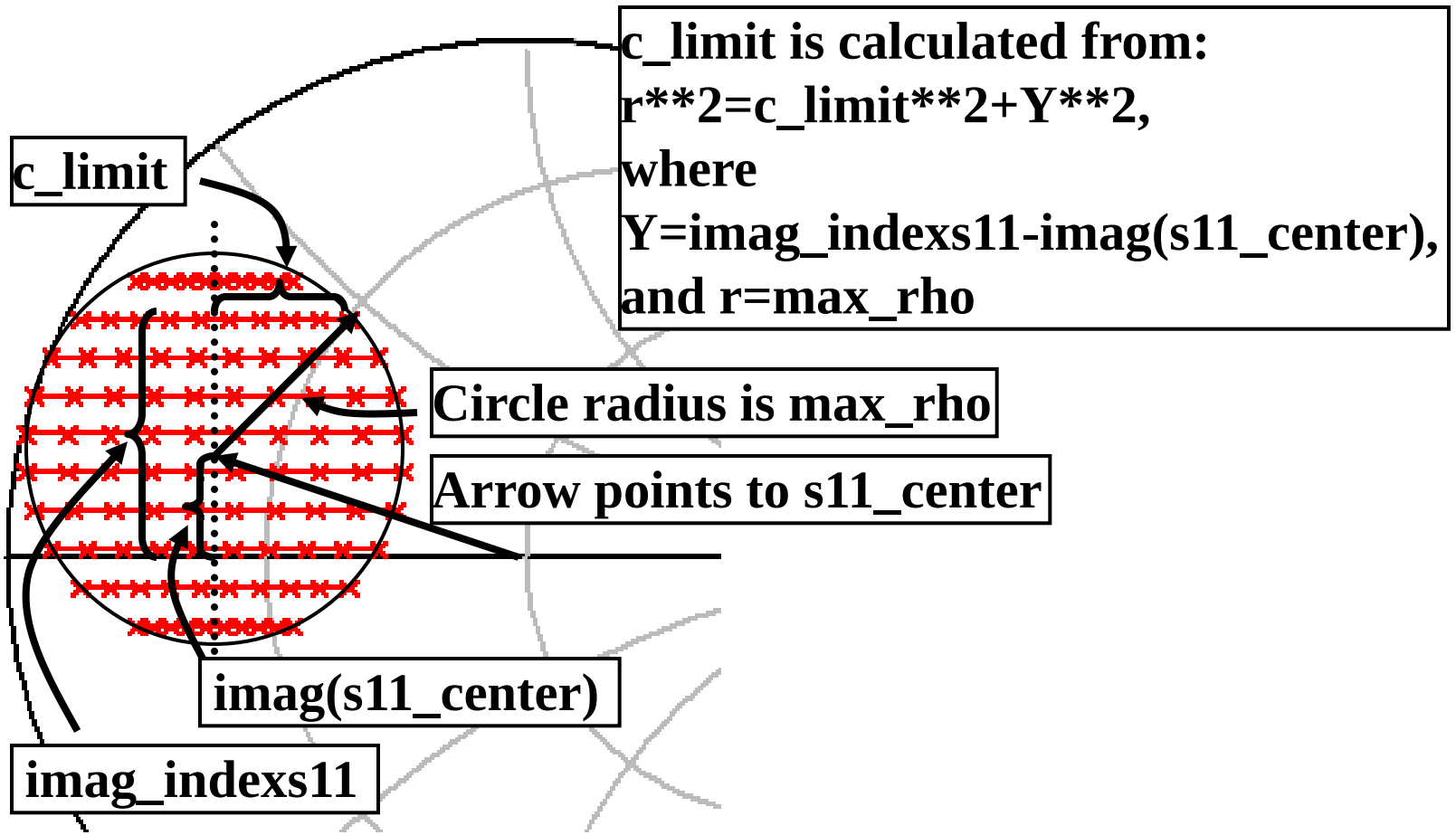
HARMONIC BALANCE

```
HarmonicBalance
HB1
Freq[1]=RFfreq
Order[1]=5
SweepVar="real_indexs11"
Start=real(s11_center)-c_limit
Stop=real(s11_center)+c_limit
Lin=pts_per_line
```

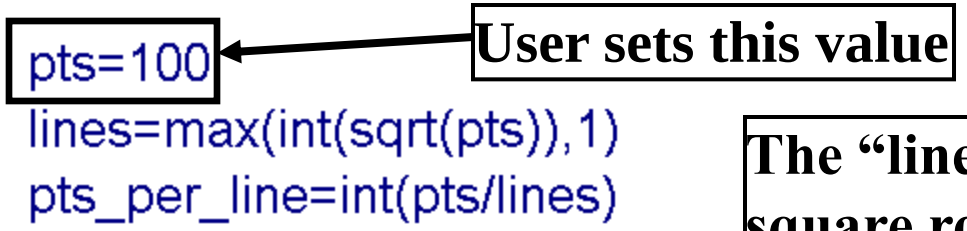
c_limit is the distance from the vertical center line of the circle to the edges, along each of the horizontal lines. It is different for each horizontal line.

How is c_limit calculated?

argument =max_rho^2-(imag(s11_center)-imag_indexs11)^2
c_limit=sqrt(if ((argument) <0) then 0 else argument endif)



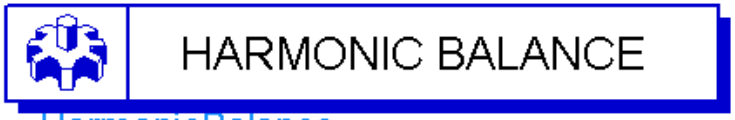
How are the number of lines and the points per line calculated?

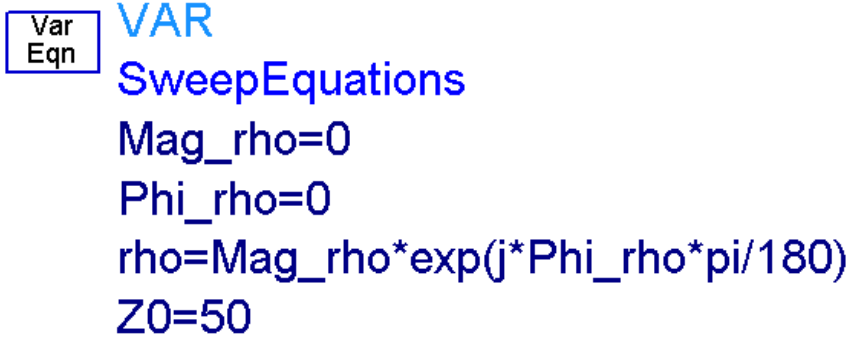


The "lines" equation takes the square root of "pts", then computes the integer part. The max() function ensures that at least one line is used. The number of points per line is computed by keeping the integer part of pts/lines.

Alternate reflection coefficient sweep

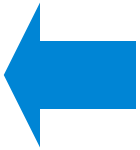
Sweep the phase of the reflection coefficient "Phi_rho" in degrees, and sweep the magnitude "Mag_rho". "rho" below becomes the swept reflection coefficient, instead of indexs11.



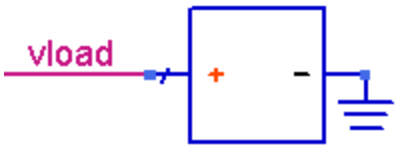


Outline

- Overview
- Specifying and generating desired load reflection coefficients
- Assigning arbitrary reflection coefficients at the harmonic freqs.
- Biasing the device and running a simulation
- Calculating desired responses (delivered power, PAE, etc.)
- Generating contour lines



You could make the load independent of frequency:



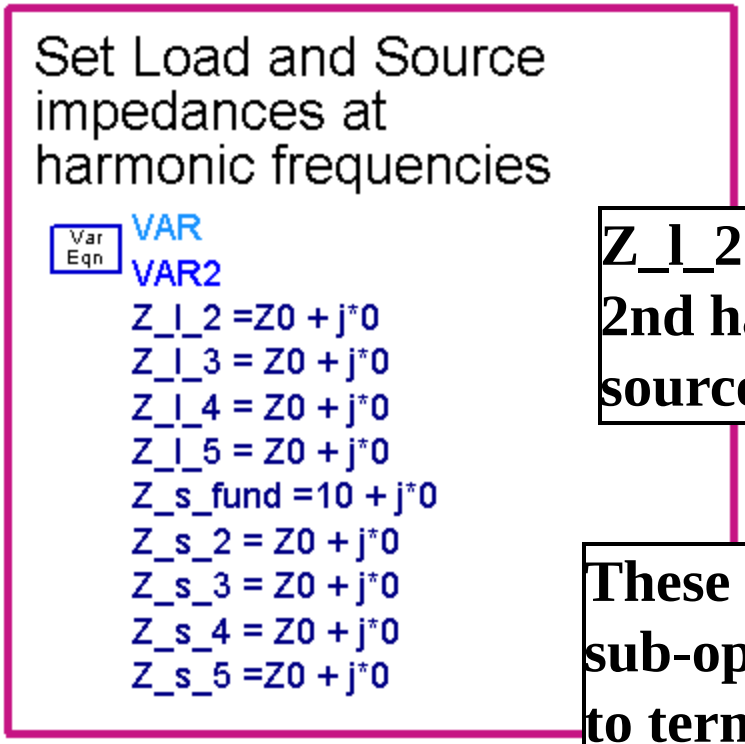
S1P_Eqn
S1
S[1,1]=indexs11
Z[1]=Z0

Var Eqn

VAR
SweepEquations
real_indexs11=0
imag_indexs11=0
indexs11=real_indexs11+j*imag_indexs11

But this would model an unrealistically simple situation and would give sub-optimal results.

Set source and load impedances at harmonic frequencies arbitrarily



Z_l_2 is the load impedance at 2nd harmonic; other load and source impedances defined similarly

These default values are somewhat sub-optimal, as using opens or shorts to terminate the harmonics should give better performance.

Defining a single, frequency-dependent reflection coefficient (1)

The "brute force" way, using a giant if-then-else equation:

These are frequency break points, midway between the fundamental and 2nd harmonic, between the 2nd and 3rd harmonics, etc. RFfreq is the fundamental frequency.

```
VAR
VAR6
f_1 = 1.5*RFfreq
f_2 = 2.5*RFfreq
f_3 = 3.5*RFfreq
f_4 = 4.5*RFfreq
load_tuner = if freq <= f_1 then load_fund elseif freq <= f_2 then load_second_harm elseif freq <= f_3 then load_third_harm elseif freq <
load_fund = (z_fund-50)/(z_fund+50)
load_second_harm = (z_2-50)/(z_2+50)
load_third_harm = (z_3-50)/(z_3+50)
load_fourth_harm = (z_4-50)/(z_4+50)
load_fifth_harm = (z_5-50)/(z_5+50)
```

These equations convert harmonic impedances to reflection coefficients.

This is from examples/RF_Board/NADC_PA_prj/NADC_PA_Test

Defining a single, frequency-dependent reflection coefficient (2)

Using an array, defined via the "list( )" function:

The list function defines an array of reflection coefficients, the first value being 0, the second value indexs11, the third value fg(Z_l_2), etc.

Var Eqn

```
VAR
global ImpedanceEquations
;Tuner reflection coefficient=
LoadTuner = LoadArray[iload]
LoadArray = list(0,indexs11, fg(Z_l_2), fg(Z_l_3), fg(Z_l_4), fg(Z_l_5))
iload = int(min(abs(freq)/RFfreq+1.5,length(LoadArray)))
fg(x) = (x-Z0)/(x+Z0)
```

The function fg(x) converts an impedance x into a reflection coefficient.

Defining a single, frequency-dependent reflection coefficient (2 continued)

iload is the index into the array.
when iload=1, LoadTuner = LoadArray[1] =0
when iload=2, LoadTuner = LoadArray[2] =indexs11
when iload=3, LoadTuner = LoadArray[3] =fg(Z_l_2)
etc.

Var Eqn

```
VAR
global ImpedanceEquations
;Tuner reflection coefficient=
LoadTuner = LoadArray[iload]
LoadArray = list(0,indexs11, fg(Z_l_2), fg(Z_l_3), fg(Z_l_4), fg(Z_l_5))
iload = int(min(abs(freq)/RFfreq+1.5,length(LoadArray)))
fg(x) = (x-Z0)/(x+Z0)
```

How is iload calculated?

Var Eqn

```
VAR
global ImpedanceEquations
;Tuner reflection coefficient=
LoadTuner = LoadArray[iload]
LoadArray = list(0,indexs11, fg(Z_I_2), fg(Z_I_3), fg(Z_I_4), fg(Z_I_5))
iload = int(min(abs(freq)/RFfreq+1.5,length(LoadArray)))
```

"freq" is an internal simulator variable. For a 1-tone load-pull simulation it will have values 0, RFfreq, 2*RFfreq, 3*RFfreq, etc., where RFfreq is defined to be the fundamental analysis frequency.
length(LoadArray)=6 in this case.
So when freq=0,
iload = int(min(0/RFfreq +1.5, 6)) = int(min(1.5, 6)) = int(1.5) =1
When freq=RFfreq,
iload =int(min(RFfreq/RFfreq+1.5, 6)) =int(min(2.5, 6)) =int(2.5) =2

More details

Var Eqn

```
VAR
global ImpedanceEquations
;Tuner reflection coefficient=
LoadTuner = LoadArray[iload]
LoadArray = list(0,indexs11, fg(Z_l_2), fg(Z_l_3), fg(Z_l_4), fg(Z_l_5))
iload = int(min(abs(freq)/RFfreq+1.5,length(LoadArray)))
```

When freq=0, analysis at DC is being carried out, and iload=1, so LoadTuner =LoadArray[1]=0. So the reflection coefficient at DC is 0.
When freq=RFfreq, analysis at the fundamental frequency is being carried out, and iload=2,
so LoadTuner=LoadArray[2]=indexs11, which is the load reflection coefficient at the fundamental frequency.
When freq=2*RFfreq, analysis at the 2nd harmonic frequency is being carried out, and iload=3,
so LoadTuner=LoadArray[3]=fg(Z_l_2), which is the load reflection coefficient at the 2nd harmonic frequency. Etc.

Load pull at a harmonic frequency

Var Eqn

```
VAR
global ImpedanceEquations
;Tuner reflection coefficient=
LoadTuner = LoadArray[iload]
LoadArray = list(0,indexs11, fg(Z_l_2), fg(Z_l_3), fg(Z_l_4), fg(Z_l_5))
iload = int(min(abs(freq)/RFfreq+1.5,length(LoadArray)))
```

Define a new variable, Z_l_fund, to fix the load impedance at the fundamental frequency. Then change the LoadArray equation to:

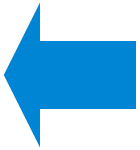
LoadArray = list(0, fg(Z_l_fund), indexs11, fg(Z_l_3),…)

The reflection coefficient at which harmonic frequency is being swept?

What if you wanted to sweep the reflection coefficient at the third harmonic?

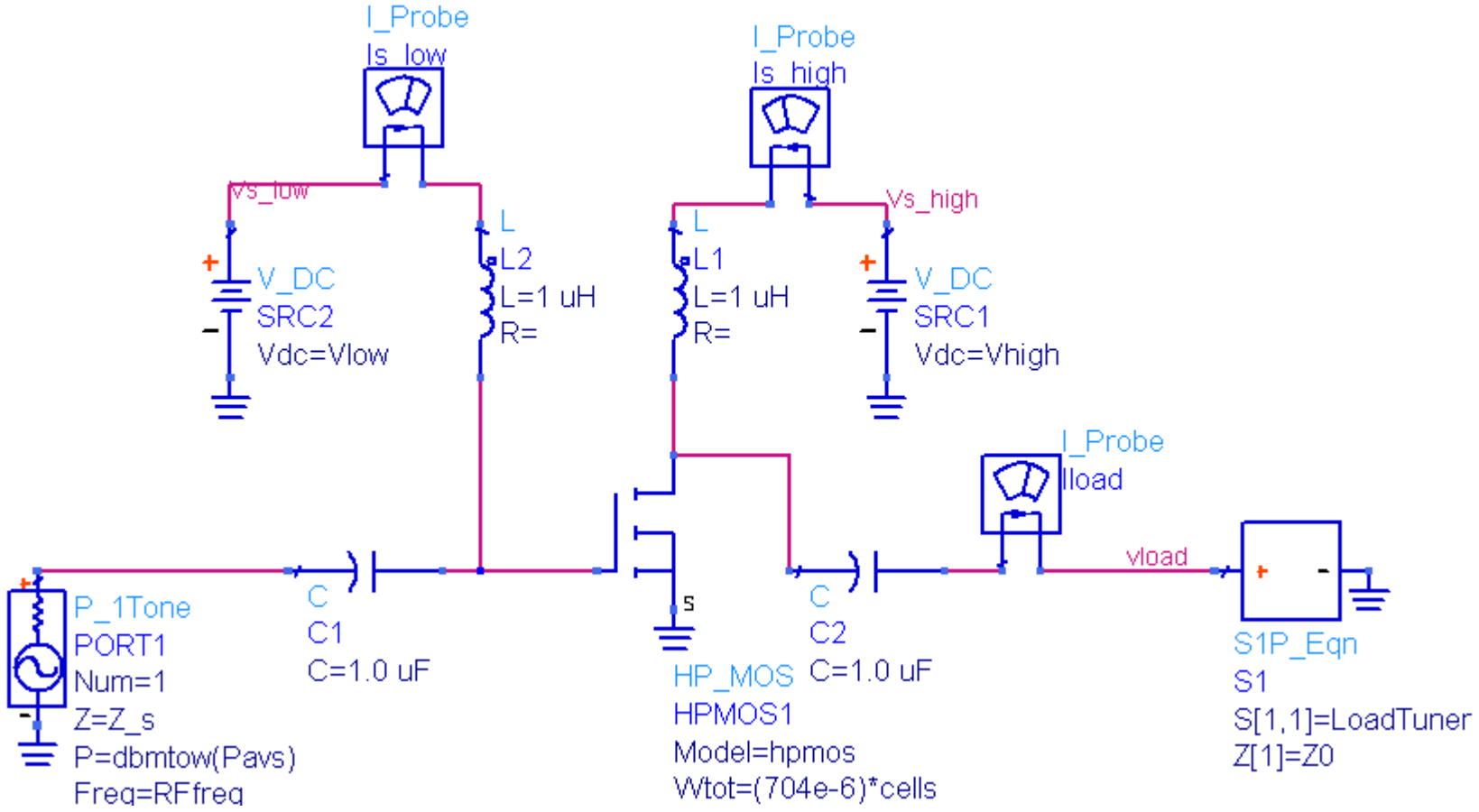
Outline

- Overview
- Specifying and generating desired load reflection coefficients
- Assigning arbitrary reflection coefficients at the harmonic freqs.
- Biasing the device and running a simulation
- Calculating desired responses (delivered power, PAE, etc.)
- Generating contour lines



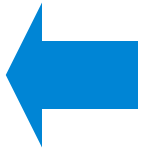
Modify the bias network as desired

But if you delete both current probes and voltage labels "Vs_low" and "Vs_high" then the PAE calculations on the data display won't work.



Outline

- Overview
- Specifying and generating desired load reflection coefficients
- Assigning arbitrary reflection coefficients at the harmonic freqs.
- Biasing the device and running a simulation
- Calculating desired responses (delivered power, PAE, etc.)
- Generating contour lines



Calculate the DC power consumption

```
Eqn Vs_l=exists("real(Vs_low[0])")
Eqn Vs_h=exists("real(Vs_high[0])")
Eqn Is_h=exists("real(Is_high.i[0])")
Eqn Is_l=exists("real(Is_low.i[0])")
Eqn Pdc=Is_h*Vs_h +Is_l*Vs_l +1e-20
```

The [0] index means use the DC component.

The exists( ) function returns 0 if the variable in quotes is not in the dataset. So, for example, you could delete the wire label Vs_low, re-run the simulation, and the power would still be calculated. If you alter the device biasing and still want to calculate the DC power consumption, you may need to modify the equation for Pdc.

Calculate the power delivered and PAE

Eqn P_In_Watts=0.5*real(V_In[1]*conj(I_In.i[1]))

This power-added efficiency equation that uses the power absorbed at the input is considered to be a better, more accurate figure of merit than the one that uses the power available from the source.

Eqn PAE=100*(Pdel_Watts-P_In_Watts)/Pdc

Eqn Pdel_dBm = 10*log10(Pdel_Watts)+30

0.5*real(V*conj(I)) is a standard equation for calculating power delivered to a complex load. Refer to Desoer and Kuh, Basic Circuit Theory.

Power-added efficiency is calculated as the power delivered to the load minus the power absorbed at the input divided by the DC power consumption.

Pdel_Watts and PAE are functions of the two swept variables

what(Pdel_Watts)
Dependency : [imag_indexs11,real_indexs11] Num. Points : [10, 10] Matrix Size : scalar Type : Real

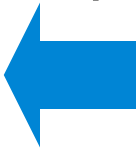
what(PAE)
Dependency : [imag_indexs11,real_indexs11] Num. Points : [10, 10] Matrix Size : scalar Type : Real

The maximum values are computed by finding the maxima across one dimension (real_indexs11) first, then finding the maximum of the remaining array.

Eqn PAEmax=max(max(PAE))

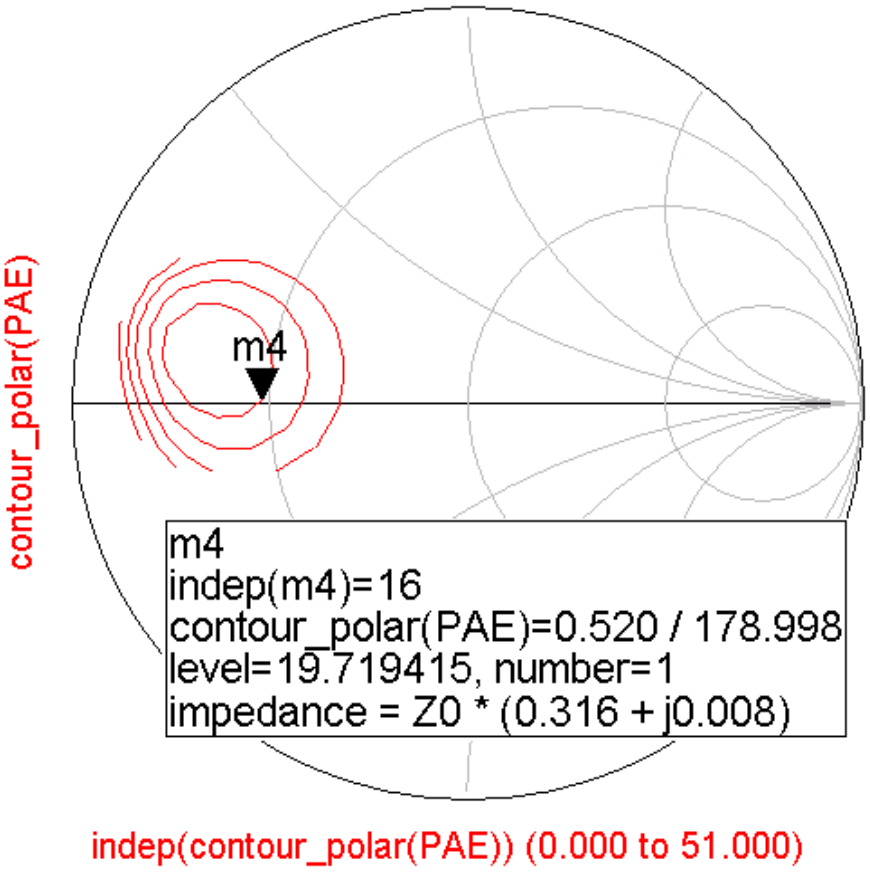
Outline

- Overview
- Specifying and generating desired load reflection coefficients
- Assigning arbitrary reflection coefficients at the harmonic freqs.
- Biasing the device and running a simulation
- Calculating desired responses (delivered power, PAE, etc.)
- Generating contour lines



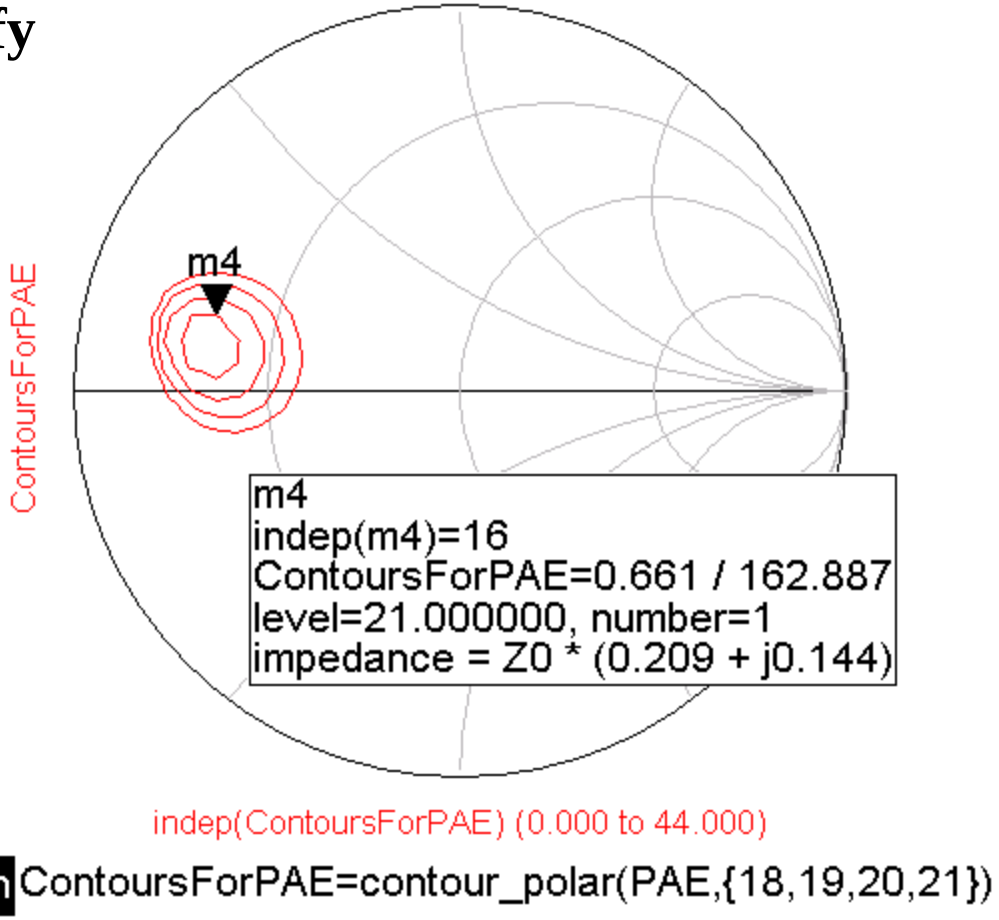
Most simple contour lines

Simplest case is to just use contour_polar( ) function with defaults. Generates six equally-spaced levels between the minimum and maximum of the data.
But you can't easily find the maximum by moving the marker or quickly change the number of lines or their spacing.

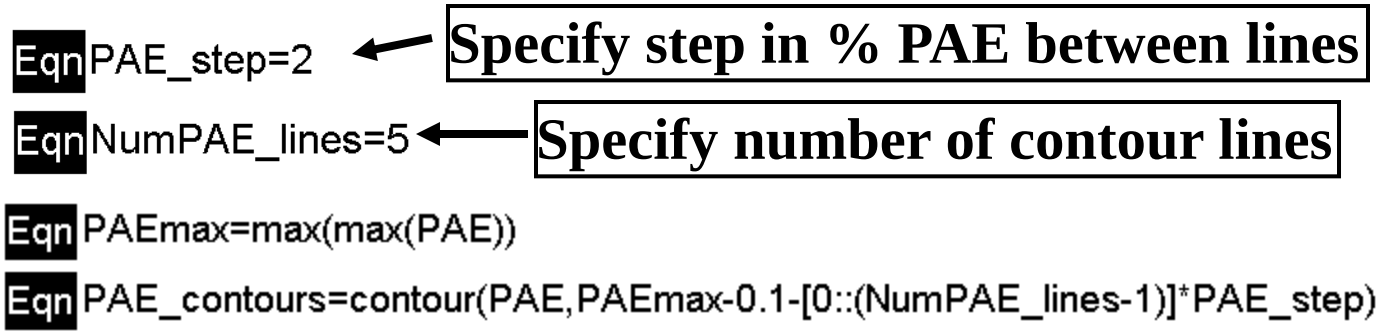


May specify specific contour values

Use curly braces { } to specify specific contour values.
But you can't easily find the maximum by moving the marker, and you may have to modify the values by hand if you change the simulation.



More complex but more flexible contour equations



First contour line will be for PAEmax-0.1. This -0.1 term is included so the first line will be a small circle that you can see, rather than being a single pixel that you can't.
If NumPAE_lines=5, then [0::(NumPAE_lines-1)] generates an array from 0 to 4, in steps of 1.
So the second PAE line will be for PAEmax-0.1 -1*PAE_step.
The third PAE line will be for PAEmax-0.1 -2*PAE_step.
Etc.
With this approach, you can quickly change the number of lines and the spacing between them.

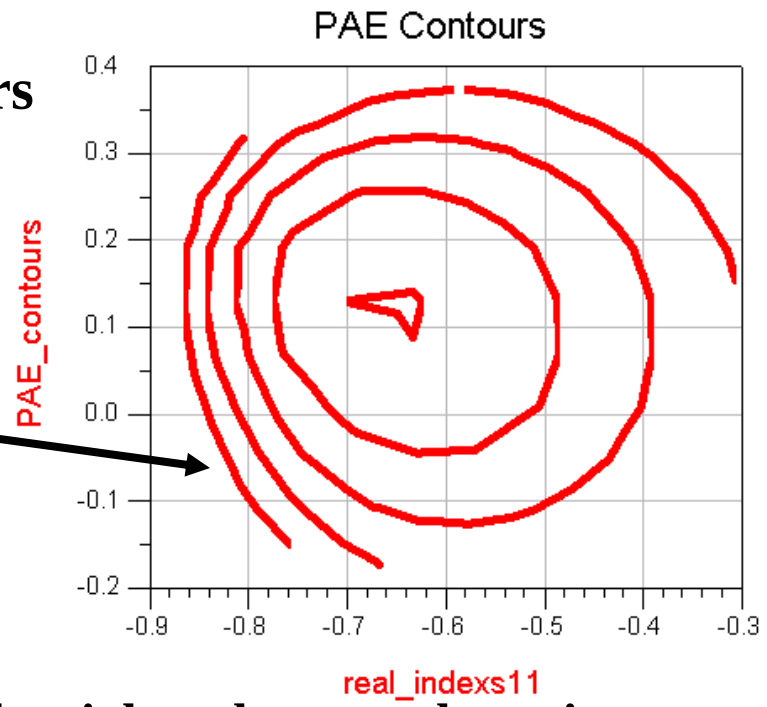
Generating contour lines for rectangular plots

The contour( ) function generates contours for a rectangular plot, which may give you more resolution than a Smith chart.

Eqn PAE_contours=contour(PAE,PAEmax-0.1-[0::(NumPAE_lines-1)]*PAE_step)

real_indexs11	PAE_contours
level=13.531, number=1	
-0.805	0.317
-0.808	0.314
-0.811	0.312
-0.836	0.266
-0.848	0.252
-0.856	0.213
-0.864	0.191
-0.864	0.158
-0.866	0.130
-0.863	0.102
-0.858	0.070
-0.854	0.045
-0.843	0.009
-0.837	-0.015
-0.821	-0.052
-0.807	-0.083
-0.789	-0.112
-0.758	-0.150
level=15.531, number=1	
-0.580	0.373

These pairs of points define the left-most arc



Although the right column and y-axis are labeled "PAE_contours", they are really "imag_indexs11"

Generating contour lines for Smith chart

The contour( ) function generates y-axis coordinates (PAE_contours - really imag_indexs11) paired with x-axis coordinates (real_indexs11). But you have to convert these to complex numbers for plotting on the Smith chart.

Eqn PAE_contours=contour(PAE,PAEmax-0.1-[0::(NumPAE_lines-1)]*PAE_step)

Eqn PAE_contours_p=[indep(PAE_contours)+j*PAE_contours]

real_indexs11	PAE_contours
level=13.531, number=1	
-0.805	0.317
-0.808	0.314
-0.811	0.312
-0.836	0.266
-0.848	0.252

PAE_contours_p
level=13.531, number=1
-0.805 + j0.317
-0.808 + j0.314
-0.811 + j0.312
-0.836 + j0.266
-0.848 + j0.252

The PAE_contours_p equation takes the X and Y coordinate pairs of the contour lines and converts them to complex numbers, X +j*Y, which can be plotted on a Smith chart.

This equation generates contour lines for the Smith chart directly:

Eqn PAE_conts_polar=contour_polar(PAE,PAEmax-0.1-[0::(NumPAE_lines-1)]*PAE_step)

Generating contours after sweeping magnitude and phase of rho

Eqn PAE_contours=contour(PAE,PAEmax-0.1-[0::(NumPAE_lines-1)]*PAE_step)

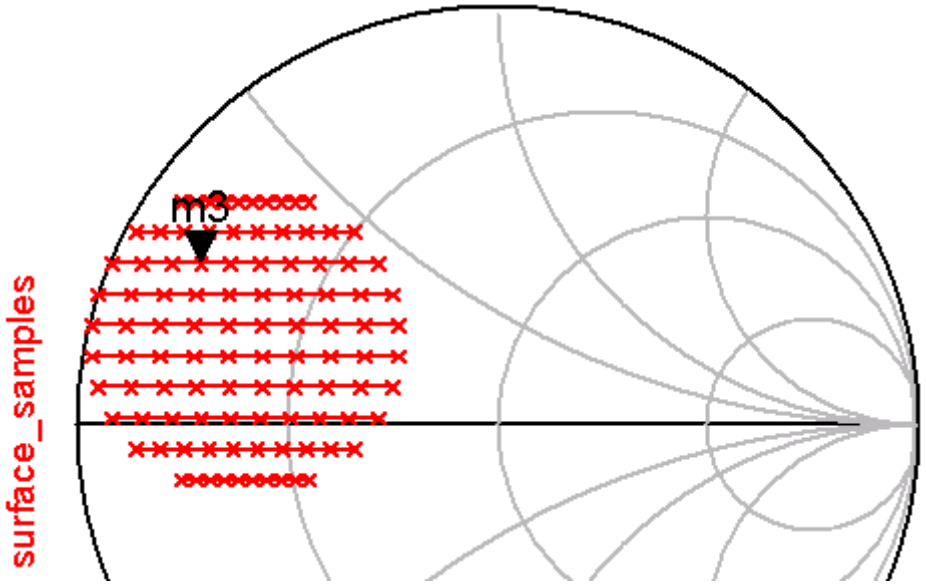
Eqn PAE_conts_forSmithCh=indep(PAE_contours)*exp(j*PAE_contours*pi/180)

Mag_rho	PAE_contours
level=15.538, number=1	
0.433	200.213
0.429	200.000
0.367	194.990
0.335	190.000
0.314	182.038
0.307	180.000
0.302	170.338

Mag_rho	PAE_conts_forSmithCh
level=15.538, number=1	
0.433	-0.407 - j0.150
0.429	-0.403 - j0.147
0.367	-0.354 - j0.095
0.335	-0.330 - j0.058
0.314	-0.313 - j0.011
0.307	-0.307 + j3.757E-17
0.302	-0.298 + j0.051

PAE_contours can be plotted on a rectangular plot. The X-axis will be the magnitude of the reflection coefficient, and the Y-axis will be the phase of the reflection coefficient, in degrees. These must be converted to complex numbers via the PAE_conts_forSmithCh equation, for plotting on a Smith chart.

Plotting the actually-simulated reflection coefficients



This data shows the simulation results when the reflection coefficient is the value selected by the marker's location.

Impedance at marker m3

5.775 + j12.609

PAE, %	Power Delivered (dBm)
40.84	25.64

These "surface_samples" are computed from the swept variables, real_indexs11 and imag_indexs11:

Eqn surface_samples=real_indexs11+j*expand(imag_indexs11)

The "expand( )" function just adds an extra independent variable to the imag_indexs11 variable so it can be added to real_indexs11.

PAE and power delivered are functions of the swept variables

what(PAE)
Dependency : [imag_indexs11,real_indexs11] Num. Points : [10, 10] Matrix Size : scalar Type : Real

what(Pdel_dBm)
Dependency : [imag_indexs11,real_indexs11] Num. Points : [10, 10] Matrix Size : scalar Type : Real

Find the indices of imag_indexs11 and real_indexs11 that correspond to marker m3's location

Eqn imag_index=find_index(imag_indexs11,imag(m3))

Eqn real_index=find_index(real_indexs11[imag_index,::],real(m3))

PAE[imag_index,real_index]
40.84

Pdel_dBm[imag_index,real_index]
25.64

Summary

- ADS offers much flexibility for simulating load pull
- Set-up and post-processing equations are complex, but you don't have to know them to run simulations. The set-up and post-processing can be simplified.
- Other, more advanced capabilities are available, including source-pull, two-tone intermodulation distortion while doing a load pull, and optimization within a load pull sweep.