Documentation: Python scripts for near field to far field measures and for angular scan

Author:
Galaad BARRAUD

May 2025

Contents

Overview **2**

Requirements **2**

File Structure **2**

Function: matrix(dossier, ligne_cible, colonne_cible) **2**

Function: matrix_single_freq(chemin_fichier, A, B, pas, col) **3**

Function: file_to_array(dossier, ligne_cible, colonne_cible, rot=False) **3**

Function: boustrophedon(A, B, pas_x, pas_y) **4**

Class: Balayage2D_Rotation_VNA_ESP **4**

- ___init___ . . . 4
- select_state_vna(band) . . . 4
- setup_channel_vna(start_freq, stop_freq, points, IFBW) . . . 4
- add_trace_vna(trace_name, trace_number, window) . . . 5
- move(axis, units, movement, absolute, speed, accel, deccel) . . . 5
- define_home(axis) . . . 5
- return_home(axis) . . . 5
- log_error() . . . 5
- balayage_2D(...) . . . 6
- rotation(...) . . . 8

Example usage **10**

- Single-frequency 2D scan, measurement of S21 . . . 10
- Frequency sweep 2D scan between 110GHz and 170GHz, measurements of S12, S21, S11 and S22 . . . 11
- Single-frequency angular scan, measurement of S21 . . . 11
- Recreation of the measurement matrix with frequency sweep data of S12 . . . 11
- Recreation of the measurement matrix with single-frequency scan data of S12 . . . 11

Conclusion **12**

Overview

This document provides a detailed explanation of the Python script `xy_and_angular_scan.py` located in "`C:/Users/Thomas/Documents/Galaad_B/Python_script_NF2FF_2D_and_angular_scan`", used to control a 2D or angular scanning system composed of:

- A Vector Network Analyzer (VNA)
- A 2-axis ESP motion controller and/or an angular 1-axis ESP motion controller

The system performs 2D spatial and angular scans to record RF measurements across a defined area. The script integrates motor positioning with RF data acquisition.

Requirements

- Python 3.6+ or Spyder 3.12+
- `pyvisa` module
- NI-VISA driver installed
- ESP-compatible motion controller
- VNA compatible with SCPI commands (e.g. , Keysight PNA)
- GPIB connections or any other connectors compatible with the VNA and the motion controller

File Structure

The script is structured as a class `Balayage2D_Rotation_VNA_ESP` containing the methods for VNA configuration, motion control, 2D and angular scanning. At the very top of the file, there are four functions useful for recreating the measurement matrix (for the 2D scan) and creating the data files.

Function: matrix(dossier, ligne_cible, colonne_cible)

This function loads and reconstructs 2D field measurement matrices from a folder of line-formatted text files named `Balayage_#.txt`. It:

- Reads the magnitude or phase from each file (specified line and column)
- Automatically determines the scan grid size
- Rebuilds the matrix according to a serpentine scanning pattern

Key parameters:

- dossier (str): path of the directory containing the data files
- ligne_cible (int): line of the frequency that has been chosen by the user

- colonne_cible (int): column of the chosen S-parameter

This function is typically used for near- and far-field data comparison at a chosen frequency.

Function: matrix_single_freq(chemin_fichier, A, B, pas, col)

This function reconstructs a matrix from a single tab-delimited measurement file. It:

- Reads data for a given column index
- Builds a 2D matrix using a boustrophedon (serpentine) scan logic
- Supports custom grid bounds and step size

Key parameters:

- chemin_fichier (str): path of the directory containing the data files (must include the file name at the end)
- A, B (array of int or array of float): start and end coordinates of the scan
- pas (int or float): step size of the scan
- col (int): column of the chosen S-parameter

This function is typically used for near- and far-field data comparison at a fixed frequency.

Function: file_to_array(...)

This function loads and returns concatenated data from a folder of line-formatted text files named `Balayage_#.txt` or `Rotaion_#.txt`.

It opens a file, stores data in the intersection between a given line and a given column, and concatenates it with the other data taken from the other files.

Key parameters:

- dossier (str): path of the directory containing the data files
- ligne_cible (int): line of the frequency that has been chosen by the user
- colonne_cible (int): column of the chosen S-parameter
- rot (bool): if True the functions search for files with the name `Rotation_#.txt`, and if False, it search for files with the name `Balayage_#.txt`

Function: boustrophedon(A, B, pas_x, pas_y)

Generates the list of coordinates (x, y) for a boustrophedon path covering the entire grid defined by points A and B and the given steps. This function is used to calculate the coordinates of each measurement point in order to write them in the data files.

Key parameters:

- A (array of integer or array of floating): start coordinates of the scan.
- B (array of integer or array of floating): end coordinates of the scan.
- pas_x (integer or floating): step size of the x-axis.
- pas_y (integer or floating): step size of the y-axis.

Returns :

- A list of tuples (x, y) representing the positions to be traversed in boustrophedon order, covering the entire grid.

Class: Balayage2D_Rotation_VNA_ESP

`__init__`

Establishes communication with the VNA and ESP motion controller using VISA addresses. It also prints the identification strings of each device.

`select_state_vna(band)`

Loads a saved VNA measurement state.
Be sure to use only measurement state files named `"WR.<x>_Galaad.csa"` (for example: WR6.5_Galaad.csa)

List of available frequency ranges:

- WR10 ==> 75GHz - 110GHz
- WR6.5 ==> 110GHz - 170GHz
- WR4.3 ==> 170GHz - 260GHz
- WR3.4 ==> 220GHZ - 330GHz
- WR2.2 ==> 390GHz - 410GHz

`setup_channel_vna(start_freq, stop_freq, points, IFBW)`

Configures a VNA channel:

- Start and stop frequency (Hz) (int or float)
- Number of points (int)
- IF bandwidth (Hz) (int or float)
- Disables continuous measurement

`add_trace_vna(trace_name, trace_number, window)`

Adds and displays a new measurement trace on the VNA.

Key parameters:

- trace_name (str): name of the scattering parameter (e.g. "S11")
- trace_number (int): number of the trace to be displayed (trace_number $\in$ [1,4])
- window (int): number of the window you want the trace to be displayed in (window $\in$ [1,4])

`move(axis, units, movement, absolute, speed, accel, deccel)`

Moves a given axis with defined parameters.

Key parameters:

- axis (int): ESP axis numbers (e.g. 2)
- units (int): the unit of the movement (0 = encoder count, 1 = motor step, 2 = millimeter, 3 = micrometer, 4 = inches, 5 = milli-inches, 6 = micro-inches, 7 = degree, 8 = gradient, 9 = radian, 10 = milliradian, 11 = microradian)
- movement (int or float): the distance the given axis must move
- absolute (bool): if True, the movement will be absolute; if False, it will be relative
- speed (int): speed mode of the axis (1 = slow, 2 = medium speed, 3 = fast)
- accel (int or float): acceleration of the movement (m.s^{-2})
- deccel (int or float): deceleration of the movement (m.s^{-2})

`define_home(axis)`

Defines the current position as the zero reference for the given axis.

`return_home(axis)`

Returns the axis to its defined home position.

`log_error()`

Prints the current system error from the VNA (useful for debugging).

`balayage_2D(...)`

Performs a full 2D scan between two spatial points `A` and `B`.
The scan will begin at point A and end at point B. It will take measures at every step, and the global trajectory of the probe can be schematized as:

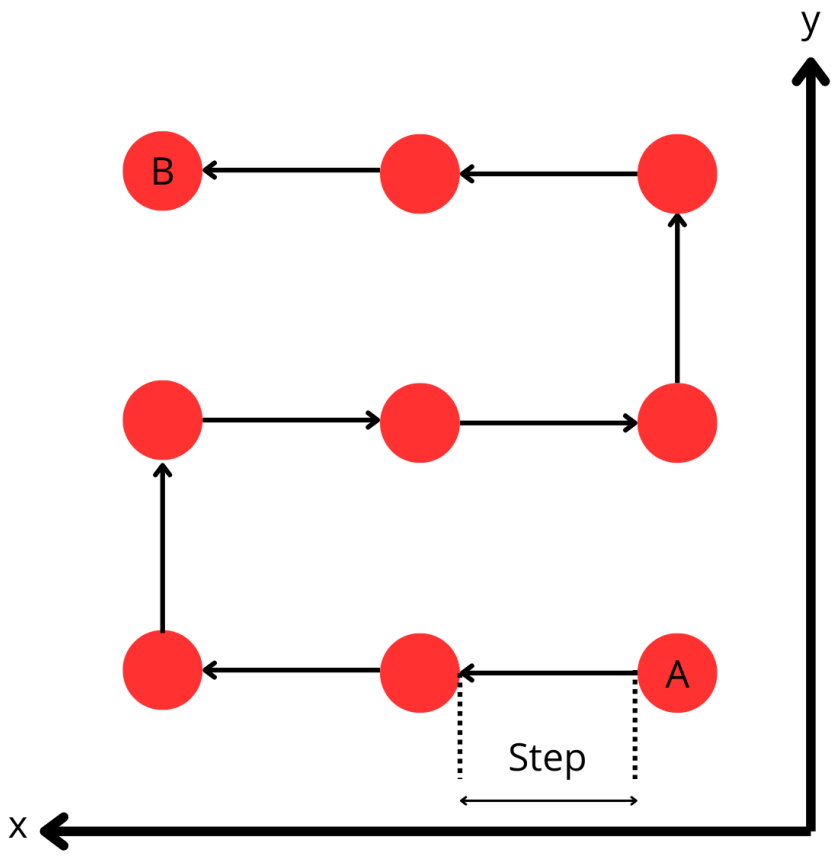

Figure 1: Diagram of the probe trajectory during a 2D scan (the red points represent measurement points and the black arrows represent the displacement of the probe).

Supports both:

- Single-frequency scans
- Frequency sweeps

Key parameters:

- `trace_name` (array of str): list of S-parameters (e.g. ["S21"])
- `axis` (array of int): ESP axis numbers (e.g. [2, 3])
- units (int): set the unit of the step and the scan coordinates (0 = encoder count, 1 = motor step, 2 = millimeter, 3 = micrometer, 4 = inches, 5 = milli-inches, 6 = micro-inches, 7 = degree, 8 = gradient, 9 = radian, 10 = milliradian, 11 = microradian)
- `A, B` (array of int or array of float): start and end coordinates (e.g. [-5, -5], [5, 5])
- `pas_axe1, pas_axe2` (int or float): step size for each axis

- `state_avg, count_avg` (bool and int): measurement averaging options
- `save_path, note` (str): output directory and optional annotation
- `File_name` (str): name of the final files returned by the script

How does it work ?
If you define A as (a_1, a_2) and B as (b_1, b_2) with steps s_x and s_y, then the script will take $N = \left\lceil (\frac{b_1 - a_1}{s_x} + 1) * (\frac{b_2 - a_2}{s_y} + 1) \right\rceil = \lceil N_x * N_y \rceil$ measures. Where N_x and N_y are respectively the number of points of measures on the x- and y-axis.
If $a_1 = a_2 = a$, $b_1 = b_2 = b$ and $s_x = s_y = s$ then N can be simplified with $N = \left\lceil (\frac{b-a}{s} + 1)^2 \right\rceil$.

When executed, both of the axes will go to the given coordinate of the point A, the script will create the parameter file, and take the first measure. It will then travel a distance equal to $b_1 - a_1$ on the x-axis in its positive direction, stopping at every step s_x to take a measure and save the data in a file. After doing N_x steps (i.e. after the x-axis reaches $b_1 - a_1$) it will travel a distance equal to s_y on the y-axis (in its positive direction).
After that, it will travel a distance equal to $b_1 - a_1$ on the x-axis but in its negative direction. At the end, the trajectory will be the same as shown in figure 1 as an example. In this case, $N_x = N_y = 3$ and therefore $N = N_x * N_y = 9$.

The time taken by the script to take a measure can be variable; if you average your data and have a wide frequency range with many points, each measure could last between 15 and 20 seconds (or more..). If you do not average your data and if you are doing a single frequency scan, each measure could last between 6 and 12 seconds.

Usage instructions
To properly use this function, you need to align your setup (i.e. to align the probe with the center of your antenna) and you need to define the zero of both axes when aligned (use define_home(<axis>)).

When done, you should define A and B accordingly to your goal :
For example, if you have a Horn antenna with a diameter of 1 cm and aligned with the probe, and if you want to scan the entirety of the antenna. You should define A as [-30 mm, -30 mm] and B as [30 mm, 30 mm] (or any other preferred scan dimension).

Another example would be if you have an antenna with a diameter of 10 cm and aligned with the probe, and if you only want to scan a fourth of the total area. You should define A as [0, 0] and B as [80 mm, 80 mm] (or any other preferred scan dimension).

You should also define the step parameters as $s \leq \lambda/2$ to avoid any aliasing (with λ the operating wavelength).

If you want to recreate the measurement matrix to visualize it (with plt.imshow() for example), you should use `matrix_single_freq` located in the same file but outside of the class `Balayage2D_Rotation_VNA_ESP`.

This script also supports 2D scan over a rectangular area (i.e. B[0] $\neq$ B[1]). In this case, the measurement matrix is harder to reconstruct and cannot be recreated with `matrix` or `matrix_single_freq`.

To perform a scan, open with Spyder and execute the Python file named `exe_xy_and_angular_scan.py` (located in the same directory) in which all of the functions are already properly called.

Returns :
During the scan, the script writes N files `Balayage_<i>.txt` with $i \in \mathbb{N}$ (i=1, 2, 3, ..., N) and where N is the number of measurement points.
If there is a power outage or any other problem that might interrupt the script, those files can be used to recover some data.

When the scan is completed, it deletes all temporary scan files (e.g. `Balayage_<i>.txt`) and generates final output files, which are P files `{File_name}_<freq>.txt` where `File_name` is chosen by the user and where P is the number of frequencies in the sweep. Those files are single frequency data compilations and are structured with:

- x (float)
- y (float)
- Magnitude (dB) and phase (degree) for each trace (float)
- Optional metadata and user notes (str)

The variable `freq` is the frequency of the given file.

The script also writes a file `{File_name}_parameters.txt` in which is listed :

- the lower limit frequency of the sweep (in Hz) (float)
- the upper limit frequency of the sweep (in Hz) (float)
- the number of points between the lower and upper frequency limit (int)
- the average count (if it is 0 it means no average) (int)
- step of the x axis (int or float)
- step of the y axis (int or float)
- the minimal value of x (float)
- the maximal value of x (float)
- the minimal value of y (float)
- the maximal value of y (float)
- the unit in which the step and the extrema value of x and y are expressed in (str)

`rotation(...)`

Performs a full angular scan between two values of the θ angle θ_{min} and θ_{max}.
The scan will begin at θ_{min} and end at θ_{max}. It will take measures at every step.

Supports both:

- Single-frequency scans
- Frequency sweeps

Key parameters:

- `trace_name` (array of str): list of S-parameters (e.g. ["S21"])
- `axis` (int): ESP axis numbers (e.g. 1)
- units (int): set the unit of the step and the scan coordinates (0 = encoder count, 1 = motor step, 2 = millimeter, 3 = micrometer, 4 = inches, 5 = milli-inches, 6 = micro-inches, 7 = degree, 8 = gradient, 9 = radian, 10 = milliradian, 11 = microradian)
- `theta_min, theta_max` (int or float): start and end coordinates (e.g. 0, 10)
- `sens` ("-to+" or "+to-"): set the direction of the rotation of the motorized axis
- `pas` (int or float): step size for each axis
- `state_avg, count_avg` (bool and int): measurement averaging options
- `save_path, note` (str): output directory and optional annotation
- `File_name` (str): name of the final files returned by the script

How does it work ?
The script will take $N = \left\lceil \frac{\theta_{max} - \theta_{min}}{s} + 1 \right\rceil$ measures, with "s" as the angular step defined by the user.

When executed, the axis will go to the given value of θ_{min} and create the parameter file. It will then take the first measure and travel an angular distance equal to $\theta_{max} - \theta_{min}$, stopping at each step to take a measure and create the corresponding file.

Each measure can typically take between 5 and 12 seconds (or more if you have a wide frequency range with many points).

Usage instructions
To properly use this function, you need to align your setup (i.e. to align the probe with the center of your antenna) and you need to define the zero of the ESP when aligned (use define_home(<axis>)).

To perform a scan, open with Spyder and execute the Python file named `exe_xy_and_angular_scan.py` (located in the same directory) in which all of the functions are already properly called.

Returns :
During the scan, the script writes M files `Rotation_<i>.txt` with $i \in \mathbb{N}$ (i=1, 2, 3, ..., M) and where M is the number of measurement points.
If there is a power outage or any other problem that might interrupt the script, those files can be used to recover some data.

When the scan is completed, it deletes all temporary scan files (e.g. , `Rotation_<i>.txt`)

and generates final output files, which are P files `{File_name}_<freq>.txt` where `File_name` is chosen by the user and where P is the number of frequencies in the sweep. Those files are single frequency data compilations and are structured with:

- θ (float)
- Magnitude (dB) and phase (degree) for each trace (float)
- Optional metadata and user notes (float)

The variable <freq> is the frequency of the given file.

The script also writes a file `{File_name}_parameters.txt` in which is listed :

- the lower limit frequency of the sweep (in Hz) (float)
- the upper limit frequency of the sweep (in Hz) (float)
- the number of points between the lower and upper frequency limit (int)
- the average count (if it is 0 it means no average) (int)
- step of the axis (int or float)
- the minimal value of θ (float)
- the maximal value of θ (float)
- the unit in which the step and the extrema value of θ are expressed in (str)

Example usage

Single-frequency 2D scan, measurement of S21

```
import xy_and_angular_scan as sc
meas = sc.Balayage2D_Rotation_VNA_ESP("GPIB1::16::INSTR", "GPIB1::1::INSTR")
meas.select_state_vna("WR6.5_Galaad.csa")
meas.setup_channel_vna(start_freq=170E9, stop_freq=170E9, points=1,
IFBW=1000)
meas.add_trace_vna("S21", trace_number=1, window=1)
meas.balayage_2D(trace_name=["S21"], axis=[2,3], A=[-10,-10], B=[10,10],
pas_axe1=0.75, pas_axe2=0.75, state_avg=True,
count_avg=20, save_path="C:\\...\\vna_data",
File_name="2D_single_freq_S21")
```

The above script performs a 2D single-frequency scan of S21. The measurements are averaged by 20 (state_avg=True & count_avg=20).

Frequency sweep 2D scan between 110GHz and 170GHz, measurements of S12, S21, S11 and S22

```
import xy_and_angular_scan as sc
meas = sc.Balayage2D_Rotation_VNA_ESP("GPIB1::16::INSTR", "GPIB1::1::INSTR")
meas.select_state_vna(band="WR6.5_Galaad.csa")
meas.setup_channel_vna(start_freq=110E9, stop_freq=170E9, points=201,
IFBW=1000)
meas.add_trace_vna("S12", trace_number=1, window=1)
meas.add_trace_vna("S21", trace_number=2, window=2)
meas.add_trace_vna("S11", trace_number=3, window=3)
meas.add_trace_vna("S22", trace_number=4, window=4)
meas.balayage_2D(trace_name=["S12", "S21", "S11", "S22"], axis=[2,3], units=2,
A=[0,0], B=[10,10], pas_axe1=0.75, pas_axe2=0.75, state_avg=False, count_avg=20,
save_path="C:\\...\\vna_data", File_name="2D_sweep_all_S")
```

The above script performs a 2D frequency sweep scan of S12, S21, S11 and S22. The measurements are not averaged (state_avg=False).

Single-frequency angular scan, measurement of S21

```
import xy_and_angular_scan as sc
meas = sc.Balayage2D_Rotation_VNA_ESP("GPIB1::16::INSTR", "GPIB1::1::INSTR")
meas.select_state_vna(band="WR6.5_Galaad.csa")
meas.setup_channel_vna(start_freq=170E9, stop_freq=170E9, points=201,
IFBW=1000)
meas.add_trace_vna("S21", trace_number=1, window=1)
meas.rotation(trace_name=["S21"], axis=1, units=7, theta_max=5, theta_min=-5,
sens="-to+", pas=1, state_avg=True, count_avg=20, save_path="C:\\...\\vna_data",
File_name="angular_single_freq_S21")
```

The above script performs an angular single-frequency scan of S21. The measurements are averaged by 20 (state_avg=True & count_avg=20).

Recreation of the measurement matrix with frequency sweep data of S12

```
import xy_and_angular_scan as sc
path_data_S12 = os.path.abspath("C:\\...\\vna_data_S12")
mag_S12 = sc.matrix(dossier=path_data_S12, ligne_cible=1, colonne_cible=2)
phase_S12 = sc.matrix(dossier=path_data_S12, ligne_cible=1, colonne_cible=3)
```

Recreation of the measurement matrix with single-frequency scan data of S12

```
import xy_and_angular_scan as sc
path_data_S12 = os.path.abspath("C:\\...\\vna_data_S12\\meas_S12")
mag_S12 = sc.matrix_single_freq(chemin_fichier=path_data_S12, A=[-5,-5], B=[5,5],
pas=1, col=3)
```

```
phase_S12 = sc.matrix_single_freq(chemin_fichier=path_data_S12, A=[-5,-5], B=[5,5],
pas=1, col=4)
```

Conclusion

This Python tool provides a robust framework for automated 2D and angular RF field measurements using a VNA and programmable motion systems. It supports flexible averaging, trace selection, and result export for post-processing and visualization.