Python Antenna solver by Method of Moments

OURO-YENDOU Fahdilou

Laboratoire d'Électronique d'Antenne et de Télécommunications (LEAT)
UNICA - Université côte d'azur
École Doctorale - Science Et Technologie De l'Information et de la Communication

Tutorial for every one who want to simulate antennas through python

April 28, 2025

Outline of presentation

Introduction & Context

Context :

The design and analysis of antennas are essential in modern communication systems. A variety of numerical methods are widely used to solve electromagnetic problems involving radiation and scattering, in order to model and simulate antenna behavior. Among these, the Method of Moments (MoM) plays a particularly significant role.

The Method of Moments is a numerical technique that converts integral equations—derived from the fundamental laws of electromagnetism—into a system of algebraic equations, by projecting these equations onto a set of basis and testing functions.

The development of such numerical methods has only been made possible by the evolution of computers and the continuous growth in their computational power.

Introduction & Context

Motivation : Why MoM? Why Python?

Commercial electromagnetic (EM) solvers are often costly due to licensing fees and usage restrictions. These tools are typically viewed as black boxes by their users, limiting access to the underlying source code and, consequently, restricting a deep understanding of the implemented algorithms and their potential enhancement by third parties.

In contrast, providing an open-source, flexible, and extensible tool fosters a culture of transparency, collaboration, and continuous improvement. In this spirit, we present a Method of Moments (MoM) solver, developed entirely in Python, and integrating other open-source algorithms.

Introduction & Context

Naturally, the implementation of this tool was not built entirely from scratch.

Key references include *Sergey Makarov* [1], whose MATLAB implementation of the MoM served as a foundation, and *Nguyen Tran Quang Khai* [2], whose Python adaptation of Makarov's code strongly inspired this project.

The development of this solver has not only deepened my understanding of the mathematical and physical foundations of antenna simulation, but also highlighted the numerical challenges inherent in such implementations.

[1] Antenna and EM Modeling with MATLAB - ISBN:0-471-21876-6

[2] PhD - His implementation on github

Introduction & Context

Python was chosen not only because it is a language that is easy to understand and implement, but also because we aimed to develop a tool that could be seamlessly integrated with most other software tools used in the design of radio-frequency systems.

Notably, integrated circuit design tools such as KiCad are capable of interfacing with libraries written in Python. This makes it possible to perform the entire design process within a single, unified software environment.

Introduction & Context

Goal :

Develop and validate a Python-based MoM solver for antenna design using RWG basis functions, with a modular design for further extension and research.

Content

Software Architecture

The software code [3] is structured into functional modules, following a clean, layered architecture. The workflow begins with mesh generation, followed by the construction of RWG basis functions over the geometry. The impedance matrix is then computed using the electric field integral equation.

The resulting linear system is solved, and finally, the surface currents and electromagnetic fields are post-processed. This modularity facilitates testing, maintenance, and further development, such as switching solvers or incorporating new basis functions.

[3] https://github.com/FahdilOuro/Antenna_Solver_MoM.git

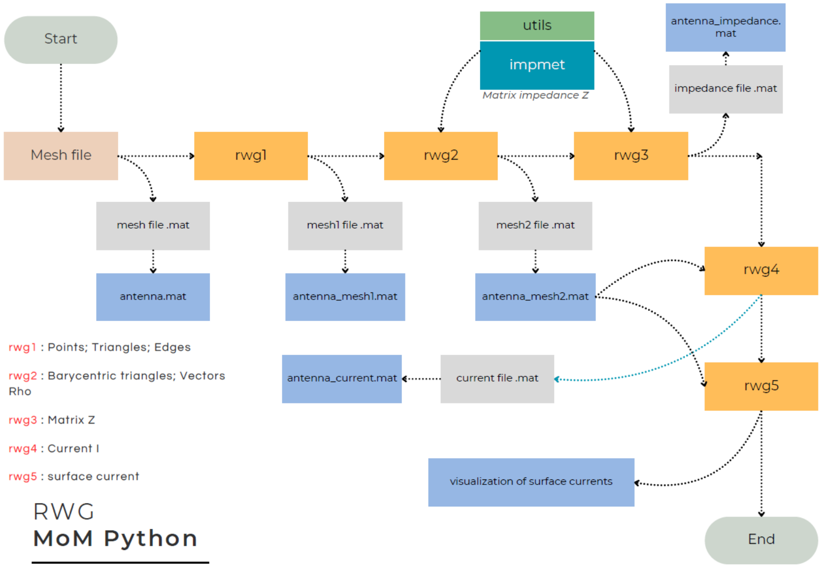
Software Architecture

The entire implementation of the Method of Moments is organized within the *rwg/* directory, which contains Python files named *rwg1.py* through *rwg5.py*. At each stage of processing, a .mat file is saved in the *mat/* folder, helping to offload memory and ensure smooth execution. The final module computes the surface current distribution.

This codebase enables the simulation of a wide variety of antenna geometries.

The flowchart of the files related to the implementation of the Method of Moments (MoM) is presented below, providing a structured overview of the code architecture.

Software Architecture



rwg1 : Points; Triangles; Edges

rwg2 : Barycentric triangles; Vectors Rho

rwg3 : Matrix Z

rwg4 : Current I

rwg5 : surface current

RWG
MoM Python

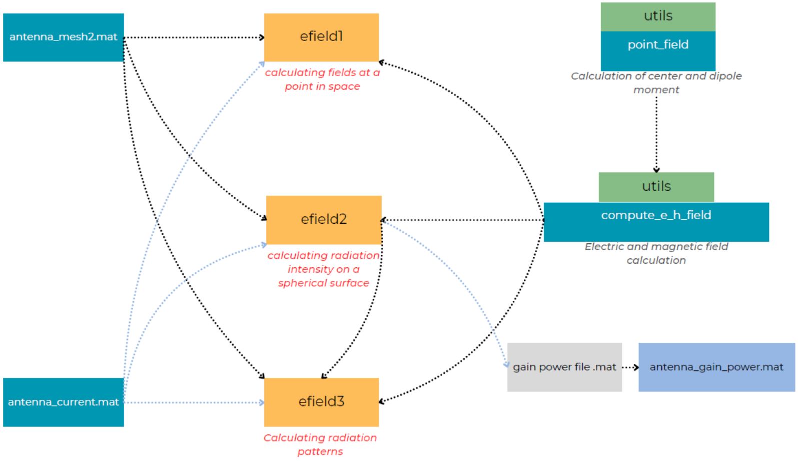
Software Architecture

An antenna can operate in two modes: scattering and radiation. For each mode, a dedicated algorithm computes the surface currents induced on the antenna structure.

The files *efield1.py* to *efield3.py*, located in the *efield/* directory, simulate the radiated fields in both scattering and radiation scenarios.

The structure of the efield module files is illustrated in the diagram below.

Software Architecture


antenna_mesh2.mat
efield1
calculating fields at a point in space
utils
point_field
Calculation of center and dipole moment
efield2
calculating radiation intensity on a spherical surface
utils
compute_e_h_field
Electric and magnetic field calculation
gain power file .mat
antenna_gain_power.mat
antenna_current.mat
efield3
Calculating radiation patterns

EFIELD
MoM Python

Software Architecture

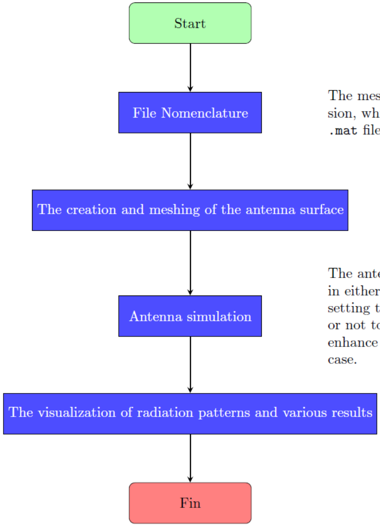
The *utils/* directory contains all the utility functions required for the proper functioning of the entire codebase. For instance, the file empmet.py implements the computation of the impedance matrix Z.

All Python source files are well-commented and documented to ensure clear understanding and to facilitate easy modifications or future extension

To simulate an antenna structure using our solver, five key steps must be followed.

These steps are clearly illustrated in the flowchart below.

Software Architecture


Start
File Nomenclature
The creation and meshing of the antenna surface
Antenna simulation
The visualization of radiation patterns and various results
Fin

The mesh file is saved with the `.msh` extension, while the extracted data is stored in a `.mat` file.

The creation and meshing of the antenna surface are performed using the Gmsh Python API, followed by the invocation of functions to save the `.msh` file and convert it into a `.mat` file.

The antenna simulation can be conducted in either scattering or radiation mode, by setting the feed point and choosing whether or not to use an adaptive mesh in order to enhance the mesh quality in the radiation case.

The visualization of radiation patterns and various results is a crucial step in the analysis and interpretation of the antenna's performance.

Content

How to Use Our Solver

1. **Prepare the Geometry**
   To begin, it is necessary to import the essential modules required for creating the antenna geometry and for simulating its behavior, either in scattering mode or radiation mode.
2. **Mesh the Geometry**
   Use the GMSH Python API to code the antenna geometry and define the feed point for radiation mode simulations.
3. **Set Simulation Parameters**
   Define the simulation parameters, including the frequency range, wave characteristics, and other parameters, all detailed in the documentation provided with the function implementations available on the project's GitHub repository.
4. **Run the Simulation**
   Execute the solver and monitor progress if necessary.
5. **Post-Process the Results**
   Visualize fields, currents, or extract relevant data from the solution.
6. **Validate and Refine**
   Compare with reference results and refine the setup if needed.

A usage example can be found on the project's GitHub repository, in the "antenna_simulation" [4] folder, where several Jupyter notebooks provide a clear overview of the steps to follow.

[4] https://github.com/FahdilOuro/Antenna_Solver_MoM/tree/main/antenna_simulation

Content

Creation and Meshing of the Antenna Surface

Overview of Gmsh [5]

Gmsh is a three-dimensional finite element mesh generator with a built-in CAD engine and post-processor. Its design goal is to provide a fast, light and user-friendly meshing tool with parametric input and flexible visualization capabilities.

Gmsh is built around four modules (geometry, mesh, solver and post-processing), which can be controlled with the graphical user interface (GUI), from the command line, using ‘.geo‘ files, or through the API in C++, C, Python, Julia, and Fortran.[...]

[5] Source : Gmsh documentation, Section "1 Overview of Gmsh"

Creation and Meshing of the Antenna Surface

The geometry meshing process is handled using Gmsh, a powerful mesh generation tool. As mentioned earlier, we use the Gmsh Python API, which provides a flexible and seamless integration within the code. The first step in analyzing an antenna surface is therefore to generate an appropriate mesh using Gmsh.

For more details on how to use Gmsh, readers are encouraged to refer to the official Gmsh documentation[6].

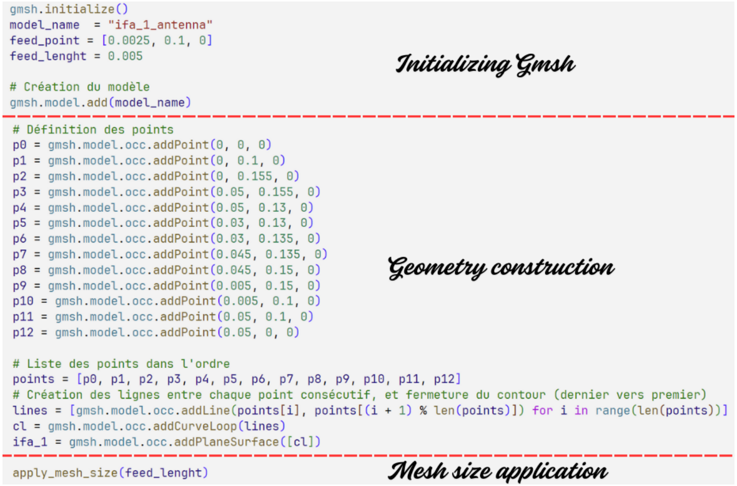
Gmsh's built-in CAD engine, OpenCascade, enables precise modeling and automatic meshing of complex antenna geometries. A dedicated Python script can then extract a `.mat` file containing the data required by the simulation algorithm.

[6] `gmsh.info/doc/texinfo/gmsh.html`

Creation and Meshing of the Antenna Surface

- The antenna geometry is built using the Gmsh Python API with OpenCASCADE.
- Users can define **arbitrary shapes**, add **feed points**, **internaledges**, or **slots**, etc...
- Users can also design their own geometries and apply various operations such as Boolean transformations, translations, rotations, and more—similar to what is available in other geometric modeling tools. Gmsh indeed provides a rich set of features for creating and manipulating complex shapes.
- The mesh is generated automatically, and the element size can be controlled through refinement fields.
- After meshing, the data is saved and converted to `.mat` format for simulation.

Creation and Meshing of the Antenna Surface


```python
gmsh.initialize()
model_name  = "ifa_1_antenna"
feed_point = [0.0025, 0.1, 0]
feed_lenght = 0.005

# Création du modèle
gmsh.model.add(model_name)
```

Initializing Gmsh

```python
# Définition des points
p0 = gmsh.model.occ.addPoint(0, 0, 0)
p1 = gmsh.model.occ.addPoint(0, 0.1, 0)
p2 = gmsh.model.occ.addPoint(0, 0.155, 0)
p3 = gmsh.model.occ.addPoint(0.05, 0.155, 0)
p4 = gmsh.model.occ.addPoint(0.05, 0.13, 0)
p5 = gmsh.model.occ.addPoint(0.03, 0.13, 0)
p6 = gmsh.model.occ.addPoint(0.03, 0.135, 0)
p7 = gmsh.model.occ.addPoint(0.045, 0.135, 0)
p8 = gmsh.model.occ.addPoint(0.045, 0.15, 0)
p9 = gmsh.model.occ.addPoint(0.005, 0.15, 0)
p10 = gmsh.model.occ.addPoint(0.005, 0.1, 0)
p11 = gmsh.model.occ.addPoint(0.05, 0.1, 0)
p12 = gmsh.model.occ.addPoint(0.05, 0, 0)

# Liste des points dans l'ordre
points = [p0, p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, p11, p12]
# Création des lignes entre chaque point consécutif, et fermeture du contour (dernier vers premier)
lines = [gmsh.model.occ.addLine(points[i], points[(i + 1) % len(points)]) for i in range(len(points))]
cl = gmsh.model.occ.addCurveLoop(lines)
ifa_1 = gmsh.model.occ.addPlaneSurface([cl])
```

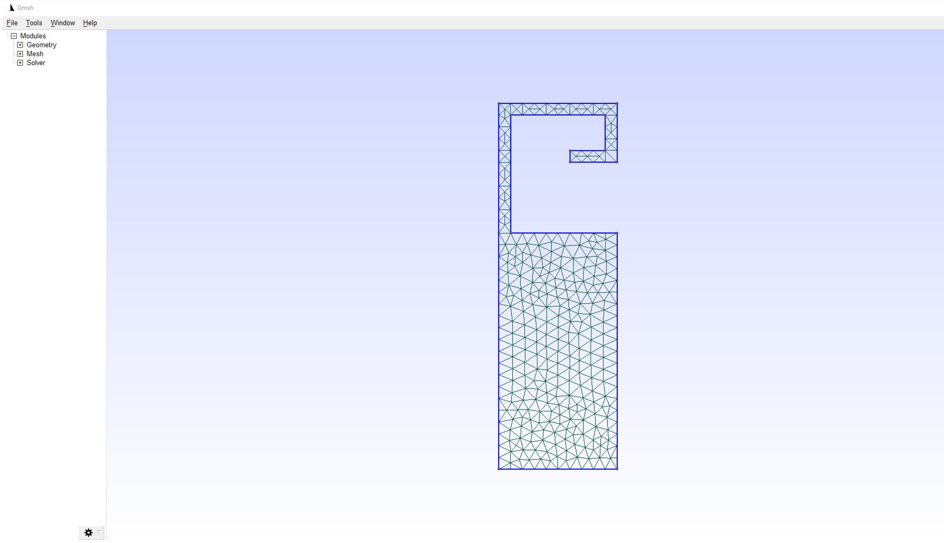
Geometry construction

```python
apply_mesh_size(feed_lenght)
```

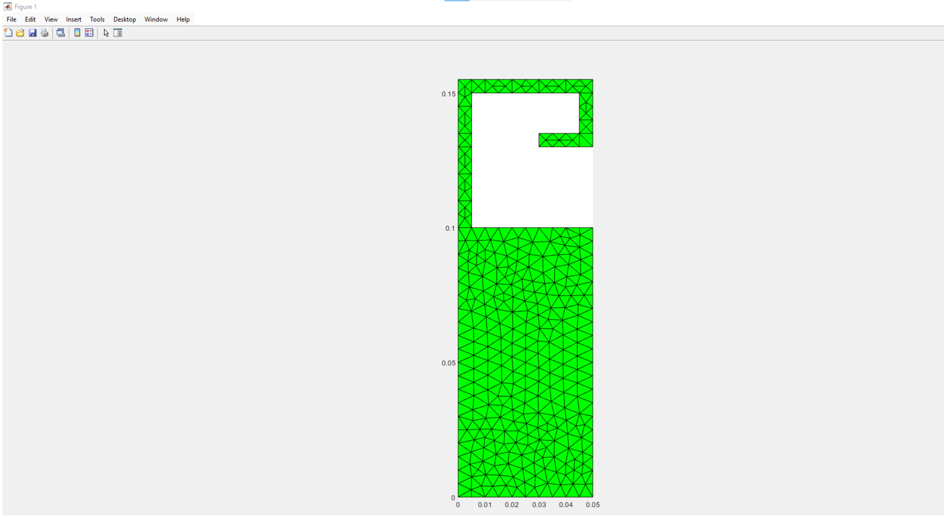
Mesh size application

The results obtained from running this Python code are illustrated by the images on the following page.

Gmsh Python API result



Matlab file result



Antenna Simulation

- The simulation core solves an integral equation using the Method of Moments (MoM) with RWG basis functions.
- Surfaces are discretized into triangles, and the system impedance matrix is constructed.
- Two modes are supported:
  - **Radiation mode [7]:** a voltage feed is defined at a specific location.
  - **Scattering mode [8]:** the antenna is excited by an incident field.
- Adaptive mesh refinement can be applied based on current distribution to improve accuracy.

[7] radiation_algorithm
[8] scattering_algorithm

Content

Validation & Results

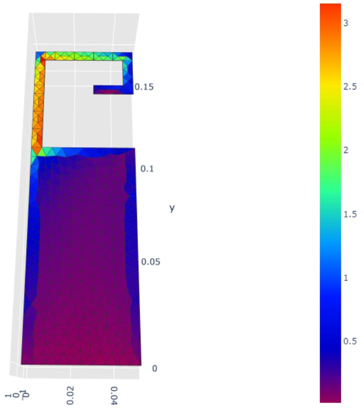
- After solving, various outputs can be visualized:
  - Surface current distribution (magnitude, phase, vectors).
  - 2D or 3D radiation patterns.
  - Input impedance, gain, directivity.
- Results can be exported in `.mat` format or displayed using Python libraries such as Matplotlib [9] or Plotly[10].
- This step is essential to validate the design and compare configurations.

[9] Matplotlib: Visualization with Python

[10] Plotly: Data Apps for Production

Validation & Results

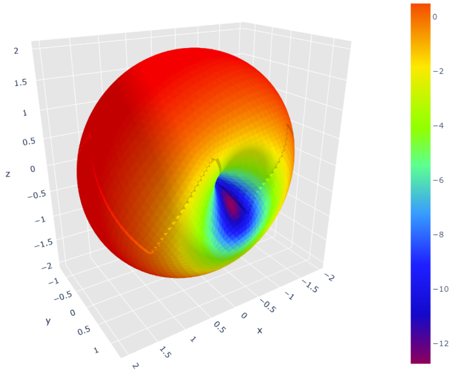
Simulation of an **Inverted-F antenna** in **radiation mode** at `605.9 MHz`. The colors displayed on the antenna surface represent the surface current distribution.



This result is obtained using the radiation algorithm, which can be found in the Python file radiation_algorithm.py. This file relies on various RWG functions that implement the Method of Moments.

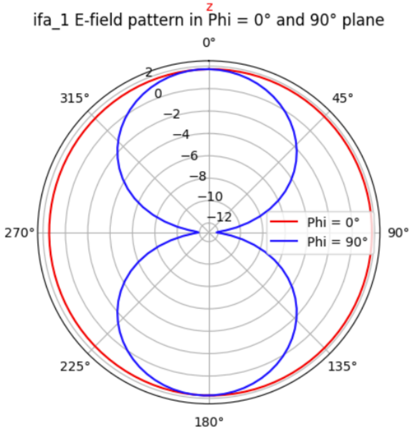
Validation & Results

This function performs the electromagnetic field calculations required to visualize the radiation intensity and gain distribution on the surface of a sphere surrounding the **Inverted-F antenna**.



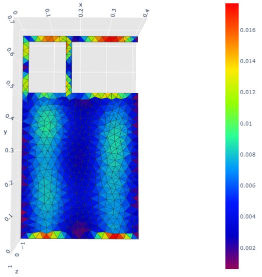
Validation & Results

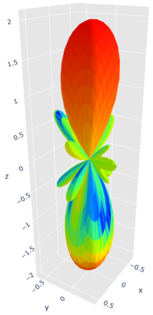
Visualization of the directivity pattern of an antenna in the $\phi = 0°$ and $\phi = 90°$ planes of the Inverted-F antenna.

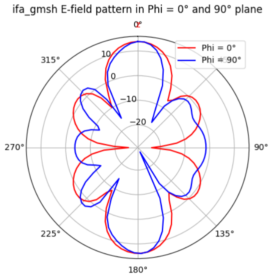


Validation & Results

Simulation results of the Inverted-F antenna (IFA) antenna in scattering mode at the frequency `1.3 Ghz`, including the surface current distribution, radiation pattern, and directivity pattern







Content

Conclusion

This modular solver provides a complete workflow for antenna simulation using Python and Gmsh.

It allows fast prototyping, testing of geometries, and integration of adaptive strategies.

Content

Future Work

Future work will focus on increasing the code execution speed, adding support for dielectric materials, calculating the quality factor, and implementing various types of antennas.

Content

Q&A / Backup Slides

Thank you

github link : FahdilOuro/Antenna_Solver_MoM