Puma-EM User's guide

Idesbald van den Bosch

January 5, 2015

Contents

Chapter 1

Introduction

Puma-EM is a code that allows the computation of various electromagnetic quantities when a target is excited by an electromagnetic source. These quantities are the scattered fields and the currents on the surface of the target. It can compute:

- monostatic RCS for a variety of angles or positions (SAR)
- bistatic RCS for a variety of angles
- Antenna pattern
- electric and/or magnetic dipole excitation
- plane-wave excitation
- a combination of the above
- ...

The target is for now a perfect electric conductor (PEC). The method used is the boundary elements method, also commonly known as the method of moments (MOM). The basis functions used are the Rao-Wilton-Glisson triangular rooftops. The integral equation implemented is the combined fields integral equation (CFIE) formulation, which can be degenerated into an electric field integral equation (EFIE) or magnetic field integral equation (MFIE).

Puma-EM also makes use of the Multilevel fast multipole method (MLFMM or MLFMA) for speeding up the computations and also allowing a larger number of basis RWG functions (and hence allowing to solve larger problems) than with the classical MOM.

In addition to the MLFMM, Puma-EM is parallelized and can run on laptop, desktop and homogeneous cluster architectures. The rule of thumb for the number of unknowns Puma-EM can reasonably handle per GByte of memory is between $500,000$ and $2,000,000$ (depending upon the problem specificities and the details of the machine architecture). For example, Puma-EM has solved the monostatic scattering of a cubic target involving 53.9 million of unknowns, on a cluster of 2 machines having each "only" 16 GBytes of memory. It can routinely solve in a few hours problems involving more than 1 million of unkowns of a low-end (dated 2004) 2 GByte laptop.

Chapter 2

Installing

2.1 Linux

For ease of installation, the puma-em directory should be located on an NFS disk and directory which can be seen by all machines on which it is intended to run (for example, your `$HOME` directory). Of course, all machines should have the same Linux distribution! If you install Puma-EM on a single machine, it doesn't matter where it is located.

2.1.1 You have admin rights to the machine

Installing can be tricky, as Puma-EM depends upon numerous libraries, some of which are still under active development. However, to make it easier, some distributions are supported through an automated installation procedure. These are Ubuntu, Fedora, OpenSuse and CentOS. To install on these distributions, simply type

```
puma-em$ ./install.sh
```

and follow the instructions on screen.

If your distribution is not supported out-of-the-box, it is possible that is is a close relative to one of the distributions cited above, for example, Red Hat or Debian. In that case, have a look at the scripts located in the directory

```
installScripts
```

, pick the one that corresponds to the closest relative to your distribution, and run the installation script.

If your distribution is not supported out-of-the-box and is not a close relative to Debian/Red Hat, here is a list of the libraries Puma-EM depends upon:

1. the compiler suite g++ and gfortran (or g77)
2. the python interpreter (included by default in most distributions)
3. open-mpi, an open-source message passing interface library
4. GMSH, an open source CAD and mesh generator
5. blitz++, a C++ array and container library
6. numpy, an array library for python
7. scipy, an open source scientific library for python
8. matplotlib, a powerful plotting tool for python
9. mpi4py, an open-source message passing interface library tuned for Python

10. argparse (optionally, for Python < 2.7 or Python < 3.2)

In order to see how to install these libraries and perform a first run of the code, just have a look at the section 2.1.2.

2.1.2 You are a simple user

Then you'll have to install the libraries locally. This is quite simple, actually. Let's first assume that gcc, g++ and a fortran compiler are installed. Let's also assume that python is installed, and also Open-MPI. Finally, BLAS and ATLAS libraries also must have been installed and their paths correctly defined.

The libraries and binaries we must install are:

1. GMSH
2. blitz++
3. numpy
4. scipy
5. mpi4py
6. matplotlib
7. argparse (optionally, for Python < 2.7 or Python < 3.2)

For the rest of the explanations, we'll assume that you have a `local` folder in your `home` folder (designated by `~/local`), that your login name is `jack`, and that everything is downloaded to a `Downloads` folder in your `home` folder. It is also understood that you use the tcsh shell.

2.1.2.1 Installing GMSH

Download the latest binary from the website `http://geuz.org/gmsh/#Download`. Uncompress the `*.tgz` file, and copy the `gmsh` binary file into a `~/local/bin` folder. Then set up the environment variable such that your system will be able to see GMSH.

In "tcsh" shell, it is done as follows by entering in the shell:

```
set path = ( $path ~/local/bin )
```

or by modifying the `.tcshrc` accordingly. If using the bash shell, enter

```
export PATH=$PATH:/home/jack/local/bin
```

2.1.2.2 Installing blitz++

Download the latest CVS version here: `http://blitz.cvs.sourceforge.net/blitz/blitz/` (click on the "Download gnu tarball" at the bottom of the page). Then uncompress the tarball. Then type in succession:

1. `autoreconf -vif`
2. `./configure --prefix=$HOME/local`
3. `make lib`
4. `make install`

Then the environment variables. There's an easy solution here. In the makefile.inc file of Puma-EM, you have to change the following lines:

```
#INCLUDE_PATH= -I/path/to/include
#LIB_SEARCH_PATH= -L/path/to/lib
```

into

```
INCLUDE_PATH= -I/home/jack/local/include
LIB_SEARCH_PATH= -L/home/jack/local/lib
```

You could also set up a LD_LIBRARY_PATH and other global variables, but the makefile.inc included in puma-em makes this unnecessary.

To test if the installation was successful, go into Puma-EM directory and issue the commands:

```
puma-em$ make clean
puma-em$ make libs
```

If it compiles smoothly with no error messages, you're good to continue!

2.1.2.3 Installing numpy

Download the latest version from `http://numpy.scipy.org/`, say, `numpy-1.6.1`. Then go into `~/Downloads`, untar the package, and `cd` into `numpy-1.6.1`.

First, read the `INSTALL.txt` file. The installation steps are summarized hereafter. But first, make sure that BLAS and ATLAS libraries are installed. Sorry, I won't help you there, ask your administrator.

To build with g77 (on older systems), enter in the shell:

```
python setup.py build --fcompiler=gnu
```

To build with gfortran (on modern systems), enter:

```
python setup.py build --fcompiler=gnu95
```

Now you need to install the libraries. just enter:

```
python setup.py install --prefix=$HOME/local
```

Then set up the environment variable such that your python will be able to import numpy, and also necessary to install SciPy. Type:

```
setenv PYTHONPATH $HOME/local/lib64/python2.5/site-packages
```

or by modifying the `~/.tcshrc` accordingly, so that you don't have to do it each time you log into your machine. In bash shell, you'd type:

```
export PYTHONPATH=$HOME/local/lib64/python2.5/site-packages
```

or modify the `.bashrc` or `.bash_profile` accordingly. (Please note that the example above is for Python 2.5)

To test if all is ok, do the following in your shell:

```
jack@linux-5pa8:~> python
Python 2.7 (r27:82500, Aug 07 2010, 16:54:59) [GCC] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>> numpy.version.version
'1.6.1'
>>>
```

2.1.2.4 Installing SciPy

Download the latest version from `http://www.scipy.org/`, say, `scipy-0.10.0`. Then go into `~/Downloads`, untar the package, and `cd` into `scipy-0.10.0`.

Then follow almost the same steps as for numpy. First type in the shell:

```
python setup.py build
```

To install the libraries. Just enter:

```
python setup.py install --prefix=$HOME/local
```

Then setup the environment variables. Normally the environment variables have been setup when installing numpy (see above).

To test if all is ok, do the following in your shell:

```
jack@linux-5pa8:~> python
Python 2.7 (r27:82500, Aug 07 2010, 16:54:59) [GCC] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import scipy
>>> scipy.version.version
'0.8.0'
>>>
```

One last thing. SciPy also ships Blitz++ modules, which are used for interfacing Python and C++, but they are a bit buggy. It is not mandatory, but we can replace them by the ones that we downloaded (see section 2.1.2.2). We do this by issuing the following command from the `/home/jack` folder:

```
cp -r local/include/blitz/*
    local/lib64/python2.5/site-packages/scipy/weave/blitz/blitz/
```

2.1.2.5 Installing mpi4py and matplotlib

Same steps as for other python modules. Download and install the latest packages at `http://code.google.com/p/mpi4py/downloads/list` and `http://matplotlib.sourceforge.net/`. Uncompress both packages, and for each do:

```
python setup.py build
```

then

```
python setup.py install --prefix=$HOME/local
```

2.1.2.6 Installing argparse

The Python `argparse` module is needed by Puma-EM. This module is included within Python since Python >= 2.7 and Python >= 3.2. Do not install for these versions of Python, it is not needed. And maybe it's installed on your machine. To check:

```
jack@linux-5pa8:~> python
Python 2.7 (r27:82500, Aug 07 2010, 16:54:59) [GCC] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import argparse
>>> argparse.__version__
'1.1'
>>>
```

Otherwise you'll need to download it from here: `http://code.google.com/p/argparse/downloads/list`. For installing, same steps as for other python modules. Uncompress the package, and do:

```
python setup.py build
```

then

```
python setup.py install --prefix=$HOME/local
```

This module installs itself in the ~/local/lib directory, so we need to make Python aware of that by modifying the variable PYTHONPATH (tcsh shell):

```
setenv PYTHONPATH
$HOME/local/lib64/python2.5/site-packages:$HOME/local/lib/python2.5/site-packages
```

2.1.2.7 Using a vendor-supplied lapack

Puma-EM relies upon an included fortran lapack library for computing the sparse approximate inverse preconditioner. In some cases, the use of a vendor-supplied may yield a faster preconditioner computation. Think of Intel or AMD supplied libraries. To make use of it, you'll have to define the path and library to use in `makefile.inc`:

```
#LIBLAPACK:= -L/usr/lib/ -llapack
```

Uncomment this line in `makefile.inc`, set the correct path and library name, re-link `./code/MoM/compute_SAI_precond` (first erase `./code/MoM/compute_SAI_precond`, and then issue a `make libs` command), and *voilÃă*: if no linking error occured, you are now using the vedor-supplied lapack.

2.1.2.8 First puma-em run

Installation is complete now. Go into Puma-EM directory and issue the following simple command:

```
puma-em$ ./run.sh
```

Yes, that's it, only this command. The code will run on a simple target. If the run fails, something can have been wrong in the above steps, double-check them.

2.2 Mac OS X

If Puma-EM has run on a Mac OS computer, I am not aware of it. However, Mac OS is a BSD system, and ports exist between Linux libraries and programs and Mac OS. See the darwinports website `http://darwinports.com/`. Here are some — possibly outdated — hints:

- g++ and gfortran
- Open-MPI (`http://permalink.gmane.org/gmane.comp.clustering.open-mpi.user/10347` and `http://openmpi.darwinports.com/`)
- GMSH (yes you can: this one works on Mac: `http://geuz.org/gmsh/#Download`)
- Blitz++ (you should be able to install this: `http://blitz.darwinports.com/`)
- mpi4py `http://mpi4py.scipy.org/docs/usrman/appendix.html`
- scipy `http://www.scipy.org/Installing_SciPy/Mac_OS_X`
- matplotlib `http://py-matplotlib.darwinports.com/`

If you succeed, could you post the steps on the Forum? Better yet, if you could provide an installation script, that would be awesome!

2.3 Windows

It is possible to use Puma-EM on Windows... Through a Linux Virtual Machine running on VMWare `http://www.vmware.com/` for example! It might be possible to compile and use Puma-EM on Cygwin `http://www.cygwin.com/` but it has not been reported. Yet.

Chapter 3

Using

3.1 Introduction

Your starting point is the configuration file simulation_parameters.py, located in `./run_in_out`, which is the default folder for simulation inputs and outputs. While it may be repellent at first (no graphical user interface?? Are you freakin' kiddin' me??), it is actually not so difficult to get acquainted with. It contains the parameters that will define the simulation, such as the frequency, the target, the type of result you want (RCS, bistatic RCS, array of dipoles excitation, antenna pattern), and some post-processing possibilities.

Once you have modified simulation_parameters.py to suit your needs (see below), simply type:

```
puma-em$ ./run.sh
```

to run the code on the set of parameters that you have defined. The results of the run will be stored in `./run_in_out/result` directory.

To run the code on `X` processes, simply type (with or without a space between `-n` and `X`):

```
puma-em$ ./run.sh -n X
```

The default number of processes is 1.

The code has to perform a lot of disk operations to store and retrieve computation data. The default IO folder for the computations is `./simuDir`. To perform the computation's IO in another folder (for example non-NFS, fast disks), simply type:

```
puma-em$ ./run.sh -nX -s/path/to/local/disks/simulation_dir
```

The code will automatically create the provided simulation folder (as long as you have permissions to write in the parent folder). You can also use this technique on computers which have several disks (for example a spinning and a solid-state disk), and you want to use the fastest one. The folder will be erased at the end of the simulation.

And finally, if you want to setup a use a non-default input-output folder (default is `./run_in_out`):

```
puma-em$ ./run.sh -nX -s /path/to/local/disks/simulation_dir
         -i /use/new/in_out/folder
```

You can also opt for a local folder:

```
puma-em$ ./run.sh -nX -i local_in_out
```

This `local_in_out` folder will be created locally, where `run.sh` is located. The use of different input/output folders is useful for keeping past simulations, and to be able to re-run a simulation by simply passing the folder as an argument to `run.sh`. A last remark on the command syntax: the differences in spacings between the flags and their arguments is voluntary. It just shows it does not matter.

3.2 CAD models and meshes

Puma-EM supports GMSH, and it is strongly advised to use it for your project. Explaining how to use GMSH is beyond the scope of this document. But here are a few facts. Basically, a CAD file comes with a `*.geo` extension, and is a written in a scripting language. A good starting point is to open one of the `geo` files and start playing with it, and visualise the result in GMSH. A mesh file comes with a `*.msh` extension. You don't need to know what's inside (nodes of triangles and nodes coordinates), Puma-EM can read it and interpret it. The good thing is that the density of the mesh will vary with the frequency (to follow the requirement that each RWG function be around $\lambda/10$ in size), so you don't need to worry about setting yourself the parameter that controls mesh size. Well, this is true for the targets shipped with Puma-EM. When designing your own target, you will make sure to parametrize the mesh size (look at the shipped targets to see how it is done).

At the request of users, Puma-EM also supports two other mesh formats from proprietary CAD programs, GiD (mesh extension `*.msh`) and ANSYS (generates 2 mesh files, ELIST.lis and NLIST.lis). Below is explained how to specify the mesh format for Puma-EM. But beforehand, you must know that since these programs are proprietary, no scripting exists that allows parametrization as for GMSH. Therefore, before running Puma-EM, you will have to make sure that the mesh is generated, located in the correct directory (usually `puma-em/geo`), and that the mean RWG size is around $\lambda/10$.

Support could easily be added for other mesh formats. If you want that, just post on the Puma-EM forum and join an example of the mesh format that has to be interpreted.

3.3 Bistatic RCS computation of a target

This section will allow an introduction to the most important parameters, and we will review them separately. Bistatic means that you are primarily interested in knowing the scattered fields due to a given excitation, but not necessarily at the place of the excitation itself.

Open simulation_parameters.py with your favourite text editor. Only the most relevant parameters will be reviewed. Do not modify the other parameters if you are unsure of what they do.

3.3.1 Target

There are two parameters for the name of the target you want to simulate:

```
params_simu.pathToTarget = './geo'
params_simu.targetName = 'cubi'
```

The default geometry files are stored in the folder ./geo. However, if your target file is located in `/somewhere/potato`, feel free to change params_simu.pathToTarget. Also, the name of the target is given without its extension.

3.3.2 Frequency and target size

Next, the frequency of the source (in Hz units):

```
params_simu.f = 2.12e9
```

Next, the lc (characteristic length) factor. It will multiply lambda (the wavelength) to obtain the average edge length (lc) of the mesh. Usually: lc = lambda/10.

```
params_simu.lc_factor = 1.0/10.0
```

Then, you can define the dimensions of the target (in meters). It is only applicable to targets which have parameterized dimensions (see sphere.geo for example); undefined parameters will not impact the simulation. Unless you find a way to script your favourite mesher, it does not apply to another mesher than GMSH either.

```
params_simu.lx = .2
params_simu.ly = .2
params_simu.lz = .2
```

The dimensions can also be given in terms of wavelength:

```
params_simu.lx = 20.0 * c/params_simu.f
```

3.3.3 Mesh format

You have to choose the mesh format. Default is GMSH.

```
params_simu.meshFormat = MESH_FORMAT[0]
```

3.3.4 Type of simulation

Fast-forward to the parameters that define which type of computation—bistatic—we are going to perform:

```
params_simu.BISTATIC = 1
params_simu.MONOSTATIC_RCS = 0
params_simu.MONOSTATIC_SAR = 0
```

In fact, bistatic defines a whole lot of possibilities, but for now we concentrate on the simple one electric dipole source.

3.3.5 Bistatic simulation settings: dipoles excitation

We define:

```
params_simu.BISTATIC_EXCITATION_DIPOLES = 1
params_simu.BISTATIC_EXCITATION_PLANE_WAVE = 0
```

If both parameters are set to 1, the excitation will be a combined one: plane wave + dipole.

The excitation is defined from a file located in the input/output folder (by default `run_in_out`, see 3.1). Here is the thing:

```
....
params_simu.BISTATIC_EXCITATION_DIPOLES = 1
....
# now the details of each excitation
# the name (with path) of the user-supplied excitation file. Set to "" if empty
params_simu.BISTATIC_EXCITATION_J_DIPOLES_FILENAME = "J_dipoles.txt"
params_simu.BISTATIC_EXCITATION_M_DIPOLES_FILENAME = ""
# the structure of the excitation file MUST BE AS FOLLOWS:
# 1 line per dipole, as many lines as there are dipoles
# each line has 9 columns that must be arranged as follows:
#
# real(J_x) imag(J_x) real(J_y) imag(J_y) real(J_z) imag(J_z) r_x r_y r_z
#
# where J = [J_x J_y J_z] is the dipole and r = [r_x r_y r_z] its origin.
```

The excitation file included by default defines a dipole $\mathbf{J}_1 = [1,0,0]$ located at $\mathbf{r}_1 = [0.1,0.1,20]$. Dimensions are in meters. Each line defines a dipole with its amplitude and location. So adding lines adds dipoles. A similar excitation file can be created for magnetic dipoles.

3.3.6 Bistatic simulation settings: plane wave excitation

Now we want just the plane wave. So, everything else being equal, we set the following parameters:

```
params_simu.BISTATIC_EXCITATION_DIPOLES = 0
params_simu.BISTATIC_EXCITATION_PLANE_WAVE = 1
```

Remember: if both parameters are set to 1, the excitation will be a combined one: plane wave + dipole. Then, to define the plane wave, one has to set:

```
if params_simu.BISTATIC_EXCITATION_PLANE_WAVE == 1:
    # origin, strength, phase and polarization of the plane wave
    params_simu.theta_inc = pi/2.0
    params_simu.phi_inc = pi/2.0
    params_simu.E_inc_theta = 1.0+0.j # the theta component
    params_simu.E_inc_phi = 0.0+0.j # the phi component
```

It defines a wave that will be coming from the defined direction.

3.3.7 Bistatic simulation settings: EM fields sampling

3.3.7.1 Arbitrary cartesian sampling of E-field

The following parameter allows one to sample the scattered electric field in arbitrary, cartesian-defined positions. It is to be defined through a file whose construction is then done by the user.

```
# sampling points: sampling of the resulting field at user-specified points in space.
# It will be used only for BISTATIC
params_simu.BISTATIC_R_OBS = 1
# the name (with path) of the user-supplied r_obs file. Set to "" if empty
params_simu.BISTATIC_R_OBS_FILENAME = "r_obs.txt"
# the structure of the r_obs file MUST BE AS FOLLOWS:
# 1 line per observation point, as many lines as there are points
# each line has 3 columns that must be arranged as follows:
#
# r_obs_x r_obs_y r_obs_z
```

In the default r_obs.txt file, the field is sampled at $\mathbf{r_1} = [0.1, 0.1, 20.0]$ and $\mathbf{r_2} = [0.1, 0.1, 20.1]$. The file r_obs.txt can be constructed by a simple program, allowing to visualize the field along lines, curves, planes, volumes, etc. As it is an input file, it should be located in the input/output folder.

3.3.7.2 User-defined far-field sampling of E-field

The following parameter allows one to sample the scattered electric far field in arbitrary, angle-defined positions. It is to be defined through a file whose construction is then done by the user.

```
# we can also have sampling angles from a user-input file
params_simu.BISTATIC_ANGLES_OBS = 1
# the name (with path) of the user-supplied r_obs file. Set to "" if empty
params_simu.BISTATIC_ANGLES_OBS_FILENAME = "bistatic_angles_obs.txt"
# the structure of the bistatic _angles_obs file MUST BE AS FOLLOWS:
# 1 line per observation angle, as many lines as there are angles
# each line has 2 columns which are the angles in degrees (easier for human reading).
# We must have: 0 <= theta <= 180 degrees (0 is the z axis, 180 is -z).
# likewise, 0 <= phi <= 360 degrees. For theta = 90 degrees, phi = 0 is the x axis,
# 180 is -x, 360 is x.
#
# theta_obs phi_obs
```

To be sure that you use the same spherical system as in the code, please bear in mind that:

- x-axis corresponds to $\theta = 90, \phi = 0$
- y-axis corresponds to $\theta = 90, \phi = 90$
- z-axis corresponds to $\theta = 0$.

3.3.7.3 Automatic far-field grid sampling of E-field

This is controlled by the following lines:

```
# thetas
params_simu.START_THETA = pi/2.0
params_simu.STOP_THETA = pi
params_simu.AUTOMATIC_THETAS = False
params_simu.USER_DEFINED_NB_THETA = 1
# phis
params_simu.START_PHI = 0.0
params_simu.STOP_PHI = 2.0 * pi
params_simu.AUTOMATIC_PHIS = True
params_simu.USER_DEFINED_NB_PHI = 200
```

Normally the code provides "best angles of observation", best from a "field spatial information for minimal sampling size" point of view. If you want the program to choose the best points for θ for example, set params_simu.AUTOMATIC_THETAS to `True`. If you want to provide your own sampling points (because you need less/more angles, for example), set params_simu.AUTOMATIC_THETAS to `False`. If you chose 1 point only, and params_simu.AUTOMATIC_THETAS is `False`, then the point will correspond to params_simu.START_THETA for θ. The same reasoning holds for ϕ parameters

3.4 Monostatic RCS computation of a target

First Let's take a look at the following parameters:

```
params_simu.BISTATIC = 0
params_simu.MONOSTATIC_RCS = 1
params_simu.MONOSTATIC_SAR = 0
```

There are two possibilities for doing monostatic computations: classic and SAR. We can do computations for the antenna polarization in vertical mode at emission and reception (VV following the θ angle), or horizontal mode at emission and reception (HH following the ϕ angle), or mixed (HV):

```
params_simu.COMPUTE_RCS_HH = 1
params_simu.COMPUTE_RCS_VV = 1
params_simu.COMPUTE_RCS_HV = 0
```

3.4.1 Classic RCS

We then define the angles we want to study. This is controlled by the following lines:

```
# thetas
params_simu.START_THETA = pi/2.0
params_simu.STOP_THETA = pi
params_simu.AUTOMATIC_THETAS = False
params_simu.USER_DEFINED_NB_THETA = 1
# phis
params_simu.START_PHI = 0.0
params_simu.STOP_PHI = 2.0 * pi
params_simu.AUTOMATIC_PHIS = True
params_simu.USER_DEFINED_NB_PHI = 200
```

Normally the code provides "best angles of observation", best from a "field spatial information for minimal sampling size" point of view. If you want the program to choose the best points for θ for example, set params_simu.AUTOMATIC_THETAS to `True`. If you want to provide your own sampling points (because you need less/more angles, for example), set params_simu.AUTOMATIC_THETAS to `False`. If you chose 1 point only, and params_simu.AUTOMATIC_THETAS is `False`, then the point will correspond to params_simu.START_THETA for θ. The same reasoning holds for ϕ parameters

Finally, for visualization, we can decide:

```
params_simu.SHOW_FIGURE = 1
```

3.4.2 Monostatic SAR

Instead of a classic, along θ or ϕ, monostatic evaluation of a target, it is possible to gather SAR data by virtually “flying” an antenna in a plane and sample the fields at given points. For this we do:

```
params_simu.BISTATIC = 0
params_simu.MONOSTATIC_RCS = 0
params_simu.MONOSTATIC_SAR = 1
```

We then decide at which locations we want to sample the monostatic fields:

```
if params_simu.MONOSTATIC_SAR==1:
    # dipole antenna will "fly" in a plane defined by local x and y axis
    # we also need an origin for the plane,
    # and a distribution of observation points
    params_simu.SAR_local_x_hat = [-1.0, 0.0, 0.0]
    params_simu.SAR_local_y_hat = [0.0, 0.0, 1.0]
    params_simu.SAR_plane_origin = [0.0, 50.0, 0.0]
    # span of the scanning rectangle (in meters)
    params_simu.SAR_x_span = 200.
    params_simu.SAR_y_span = 200.
    # offset of the scanning rectangle wrt its origin
    params_simu.SAR_x_span_offset = -100.
    params_simu.SAR_y_span_offset = -100.
    # the number of points in each direction
    params_simu.SAR_N_x_points = 8
    params_simu.SAR_N_y_points = 3
```

This is self explanatory.

3.5 Antenna radiation pattern

Thanks to the help of Jean-Pierre Adam (of IEEA: http://www.ieea.fr/) and a bit of sweat on my part, antenna radiation pattern is now available, with 3D plotting of the power radiated by the antenna. Values are normalized, and presentation in dB units is still to be done, but otherwise it works quite well for horn antennas. Patch antennas have not been tested yet. For it to work, the following parameters must be set:

```
...
params_simu.BISTATIC = 1
...
params_simu.ANTENNA_DIAGRAM = 1
```

```
...
# thetas
params_simu.START_THETA = 0.0
params_simu.STOP_THETA = pi
params_simu.AUTOMATIC_THETAS = True
params_simu.USER_DEFINED_NB_THETA = 1
# phis
params_simu.START_PHI = 0.0
params_simu.STOP_PHI = 2.0 * pi
params_simu.AUTOMATIC_PHIS = True
params_simu.USER_DEFINED_NB_PHI = 200
...
params_simu.SHOW_FIGURE = 1
```

An antenna system comprises in general an excitation feed, a waveguiding structure and a radiation part (horn). The excitation source (a dipole) should be located within the structure. Typically, an electric dipole located at $\lambda/4$ of the end of a waveguide will provide a very acceptable model of a dipole feed. I refer you to section 3.3.5 for how to excite the structure with a dipole.

The `params_simu.SHOW_FIGURE` can be set to zero, but then no image will be displayed. It is also possible to show only half of the pattern, or even a quarter. For this, set the far angles as follows:

```
...
params_simu.START_PHI = 0.0
params_simu.STOP_PHI = pi/2.0
...
params_simu.SHOW_FIGURE = 1
```

and the pattern for $0 \leq \theta \leq \pi$, $0 \leq \phi \leq \frac{\pi}{2}$ will be displayed.

Bibliography