東京大学
THE UNIVERSITY OF TOKYO

Microwave Rocket Simulation Code v1.0: User Manual

Table of contents

Author: Florian NGUYEN (M2)

Affiliation: Dpt. Aeronautics & Astronautics, Komurasaki Laboratory

Date: September 28th, 2017.

Introduction

This document is intended for future students who intend to use the MWR code as part of their research at Komurasaki Laboratory. It provides some elements about Microwave Rocket and reed valve theory this, as well as concrete examples of use of this code, applied to several “classic” problems related with the Microwave Rocket topic. The main purpose of the present document is to detail the architecture of this code and to explain how to use the latter, so that future students be able to make use of this code more easily, improve it and produce new results. For additional information, the reader may also refer to (Florian Nguyen, 2017).

Any person who uses this code should understand the underlying theory and the simplifying assumptions that were taken when developing this modelling tool, and realize that this model is not a definitive tool but still requires significant improvements and validation steps before its results can be considered to have good physical relevance and accuracy. Verification and validation of this simulation code being largely limited due to lack of experimental data and of understanding of the physical phenomena at stake in Microwave Rocket, the numerical results of the present tool are only speculative. Therefore, the reader should bear in mind that better understanding and modelling of the microwave discharge and Microwave Supported Detonation/Combustion (labelled as MSD and MSC) are imperative and that the current numerical model only relies on simplified physics of the phenomena observed on ground since 2003, the same way trajectory simulations did in the past.

This manual assumes that the reader has a good understanding of computer programming (C, Python), fluid mechanics and solid mechanics. Details will be given not only about the physics or theory behind the simulation but also about the implementation of numerical methods from the point of view of computer programming and code architecture.

Theory of the fluid model

Euler equations

The flow model assumes a zero-viscosity flow and conservation of mass, momentum and energy so that the stress tensor can be omitted and that the fluid obeys the following equation in a Cartesian frame:

$$\frac{\partial Q}{\partial t}+\frac{\partial F_x}{\partial x}+\frac{\partial G_y}{\partial y}=0,$$

where $Q=\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ E\end{bmatrix}, F_x=\begin{bmatrix}\rho u\\ \rho u^2+p\\ \rho uv\\ u(E+p)\end{bmatrix}$ and $G_y=\begin{bmatrix}\rho v\\ \rho uv\\ \rho v^2+p\\ v(E+p)\end{bmatrix}$.

In an axisymmetric frame, the y-dependency becomes a r-dependency (r stands for radius). The conservation laws have the same expression along the x-axis, but along the y-axis (which becomes the r axis) the following term appears:

$$\frac{1}{r}\frac{\partial G_r}{\partial r}=\frac{1}{r}\frac{\partial}{\partial r}\begin{bmatrix}r\rho v\\ r\rho uv\\ r(\rho v^2+p)\\ rv(E+p)\end{bmatrix}=\frac{1}{r}\begin{bmatrix}\rho v\\ \rho uv\\ \rho v^2+p\\ v(E+p)\end{bmatrix}+\begin{bmatrix}\rho v\\ \rho uv\\ \rho v^2+p\\ v(E+p)\end{bmatrix}.$$

In the above equation's right hand term, a source term H_r appears in addition to the G_y term already obtained in the Cartesian case. Therefore, the final equation solved is:

$$\frac{\partial Q}{\partial t}+\frac{\partial F_x}{\partial x}+\frac{\partial G_y}{\partial y}+H_r=0.$$

In addition, state equations are required to close the problem:

$$p=\rho RT,$$

$$E=\frac{p}{\gamma-1}+\frac{1}{2}\rho(u^2+v^2).$$

In addition, total enthalpy can also be defined for convenience as:

$$H=\frac{(E+p)}{\rho}.$$

Flux Splitting method and MUSCL reconstruction

To solve the equation above, the Flux Splitting method is used so that the equation numerically solved is:

$$\frac{\left(Q_{ij}^{n+1}-Q_{ij}^{n}\right)}{\Delta t}+\frac{S_{i+\frac{1}{2},j}F^n_{i+\frac{1}{2},j}-S_{i-\frac{1}{2},j}F^n_{i-\frac{1}{2},j}}{v_{i,j}}+\frac{S_{i,j+\frac{1}{2}}G^n_{i,j+\frac{1}{2}}-S_{i,j-\frac{1}{2}}G^n_{i,j-\frac{1}{2}}}{v_{i,j}}+H^n_{i,j}=0.$$

When estimating the derivative along the x-axis, no variations along the r-axis are considered so only state at height j is considered (and vice-versa for the r-axis), which is a simplifying assumption. The surface area of a given face separating two cells is labelled as S, and the volume of a given cell as v. The addition of a source term is not part of the original method but adds no complexity.
While conservative variables are defined at the centre of the cell, fluxes need to be estimated at the cell faces, i.e. at the limit between two cells. The state at cell faces must be determined based on the right (labelled R) and left (labelled L) side of that face. For instance, the AUSMD scheme defines the flux at a cell face as (see original paper on AUDM-DV scheme by Wada et al. for more details):

$$F_{1/2} = \frac{1}{2}(\rho u)_{\frac{1}{2}} \begin{pmatrix} 2 \\ u_R + u_L \\ v_R + v_L \\ H_R + H_L \end{pmatrix} - \left|(\rho u)_{\frac{1}{2}}\right| \begin{pmatrix} 0 \\ u_R - u_L \\ v_R - v_L \\ H_R - H_L \end{pmatrix} + p_{\frac{1}{2}}.$$

The left and right states need to be computed from available variables, i.e. cell centre variables. One way of doing so is the use of MUSCL reconstruction (and of a limiter that must be chosen by the user). Information about reconstruction methods can be found among in large amount in the bibliography of this code's related thesis (Florian Nguyen, 2017).

The flux splitting scheme currently being used is the AUSM-DV scheme established by Wada et al. It is a common choice for high velocity, compressible and high accuracy cases, however in the case of Microwave Rocket no comparison with other schemes has been performed to determine if this scheme is really the most suitable.

MSD model

The Microwave Rocket problem not only involves fluid but also plasma ignited and sustained by microwave beaming. Accurate performance prediction would require a good understanding of the plasma discharge phenomenon, of heating processes and of their dependency to pressure conditions and beam power density. Unfortunately, TRL of Microwave Rocket is still low and microwave discharge has not been successfully theorized yet, so the MSD model employed in the full simulation of Microwave Rocket employs Chapman-Jouguet detonation theory.

The MSD Mach number's calculation is typically calculated assuming an energy conversion coefficient (usually labelled η in previous research by Fukunari et al.). There are several problems involved by the Chapman-Jouguet detonation assumption:
- a value of η of 0.47 is usually taken to match experimentally measured plateau pressure with theoretical Chapman-Jouguet plateau pressure. This results in an inaccurate pressure oscillation inside the detonation tube (one can simply compare numerical results with experimental data from Oda et al.).
- inversely, assuming a unit η results in an accurate pressure oscillation frequency but in a plateau pressure that is far superior to what was found experimentally. The pressure increase in experiment is basically around 50% of the value predicted by Chapman-Jouguet theory.
- the propagation of the ionization front is not modelled. Instead, the distribution of pressure, density and velocity inside the detonation tube at the moment when the Chapman-Jouguet detonation reaches the open

end are assumed at the initial time of computation (according to one-dimensional theory). This results in a discontinuous initial condition and in the impossibility to model more than one cycle of Microwave Rocket.

Far field boundaries

In the Microwave Rocket problem, acoustic waves propagate in the flow domain and induce flow discontinuities. Such discontinuities cannot be absorbed at boundaries, which is why when they reach the limits of the flow domain they tend to reflect and generate numerical problems and non-physical solutions. To prevent this from happening, a very large ambient region is employed so that the waves generated at the open end of the detonation tube can freely propagate during the simulation time. For this reason, longer simulation time also means larger flow domain and higher computational cost. An alternative would be non-reflective boundary conditions but the latter are not implemented in the current version of the Microwave Rocket simulation code.

Mesh convergence

Test of mesh convergence was performed for the very simple problem of the detonation tube (no reed valve, no plenum, just a cylindrical tube with one open end). The number of cells along the tube length (0.5 m) was set to 50, 100, 200 and 400. Same pressure history was obtained for the first pressure oscillation, but with no rigorous convergence was obtained due to oscillations appearing with high cell number. Such a phenomenon could be caused by the analytical initial condition (one-dimensional profile based on Chapman-Jouguet detonation theory) or to the unsteadiness of the modelled case. Therefore, using experimental data as reference cell size was set small enough to have good agreement and above the level for which these oscillations appear (N=200 and over in case of a 0.5 m tube). Mesh convergence tests can be very simply performed by using the "mw_tube" case (change mesh size and time step).

One may note that the heating model of Shimada et al. is preferable for higher accuracy and convergence, since the latter models the microwave heating process and is proven to converge typically for $\Delta x < \lambda/30$. However, the inconvenient of this method is its much larger computational cost, since such a small mesh size is incompatible with the simulation of the whole Microwave Rocket (plenum, intake/outtake region), which is why our model resorts to a simplified but lighter MSD model.

Theory of the solid model

What is FEM? Why use FEM?

The Finite Element Method (FEM) is a very common tool in numerical simulation when applied to the field of mechanics. It consists in describing a solid of random shape by a certain number of elements (and nodes) of given topology (for instance rectangular or triangular in 2D, or tetrahedral or cubic in 3D), which enables to solve the motion of the solid by solving the equation of motion at each of the nodes.

In the past, very simple reed valve shape was assumed (rectangular plate with constant thickness) so analytical models could be used to approximately predict the behaviour of the valve and its vibration frequency. However, more recent work by Kurita et al. and Fukunari et al. has introduced a double-tapered (variable thickness, variable width) for which it is much more difficult to predict tip displacement and vibration frequency. That explains the resort to FEM: even with variable characteristics the double-tapered reed can be modelled by a line of N nodes between which structural properties (cross section, second moment of inertia) and the motion of the plate predicted.

Calibration of the FEM model

The experimental data of Fukunari on the vibration of the reed petal was used for calibration. To obtain the exact same normalized time response, the fixation length had to be calibrated in simulation because in reality due the lack of stiffness of the fixation, it is impossible to determine precisely from where the petal can actually be considered to be fixed. Therefore, three nodes were uniformly displayed over the fixation length and two of them were defined as rigid, which resulted in a satisfying vibration frequency of the reed valve. The number of elements over the reed valve was determined so that for a given reference case (constant pressure load at reed tip) the error in displacement would be within 1%.

Natural damping was assumed based on the Rayleigh damping model and calibrated to fit experimental data, although in reality aerodynamic damping (resulting from fluid-structure interaction) is by far preponderant. To estimate aerodynamic damping, an analytical model was used but since its calibration cannot be generalized to all flight regimes, only ground calibration was used (i.e. the tip displacement history of the reed petal was compared to previous data by Kurita et al. and aerodynamic damping set such that the simulated response would be in a similar range).

Comments

The numerical scheme used to solve the vibration problem of the reed valve is described in related work (Florian Nguyen, 2017) and will not be detailed again here. However, it is worth noticing that this method is only valid if the reed petal's solid properties don't vary in time, which implies that fatigue and temperature related variations are ignored. For instance, taking into account thermal expansion would require to recalculate all the structural matrices (stiffness, mass, damping) at each time step depending on the temperature of the reed petal. This procedure would be very costly from a computational point of view, but is possible in theory.

The FEM model gives excellent agreement with reality for the natural vibration problem (i.e. no fluid effect), but when applied to the Microwave Rocket simulation the latter is impossible to validate and fitted with an empirical aerodynamic damping model. In addition, the simplified nature of the fluid domain (only rectangles and no moving mesh) makes the load calculation at the reed petal much less accurate than a traditional CFD approach with a decent fluid-structure interface.

Reed flow modelling by source term

Method

The details of the method used to model the reed flow are detailed in associated research work (Florian Nguyen, 2017). As explained there, the tip displacement of the reed petal and the instantaneous pressure ratio between both sides of the reed valve are used to estimate the reed mass flow rate based on previously obtained CFD numerical results (OpenFOAM, 3D calculation).

Of course, this method relies on the simplification of the Microwave Rocket problem by assuming absence of hysteresis and absence of interaction between valves. A source term is used to establish the momentum and energy that are injected during refilling phases inside Microwave Rocket based on the sole knowledge of mass flow rate. While this step is key to the overall model, it induces several numerical issues and must be defined in such a way that results have physical relevance. Whether the current expression of the source term is really suitable requires further investigation and further verification steps.

Remarks of author

The biggest question in the source term is the expression of momentum terms and whether they are compatible with the boundary conditions set in the Microwave Rocket (which are typically slip walls). An unsuitable definition of the momentum terms will result in very different flow behaviours inside Microwave Rocket (and possibly different PFR values) so the user should judge by himself the most suitable source term definition.

Although this modelling method has the merit of allowing the representation of the air-breathing phenomenon inside Microwave Rocket, it is also the major reason for the lack of accuracy of the model and constitutes a notable limit of the latter. In addition, whether the mass flow rate history through reed valves is stable or not when diminishing cell size has not been investigated yet, but since pressure history at the thrust wall is subject to small oscillations at small cell size, it is very unlikely that convergence for reed mass flow can be obtained.

Physics at the reed valve is governed by strong fluid-structure interaction phenomena and therefore the aerodynamic damping factor has a large effect on reed petal vibration and computed reed valve mass flow rate. Currently, this factor is constant but it is very unlikely that this is verified in reality, especially when density and pressure change in large ranges (like when Microwave Rocket is in flight). This constitutes another reason for possible disagreement of this code and reality.

This method is also limited by the fact that the OpenFOAM data has only been estimated for the reed valve geometry defined by Kurita et al. and Fukunari. For any other design, the actual mass flow rate dependency to tip displacement and pressure ratio would have to be estimated again either by experiment or numerical simulation.

Programming methods and environment-related

Running in Linux environment

The overall code is written in Python and C language (respectively for post-processing and pure calculation). It can be used easily in Linux or MacOS environment by simply installing the GCC compiler (Python is usually installed by default).

Sublime Text can be used to edit all files as well as to run them directly from the text editor window. Below are some useful shortcuts to be used in Linux for that purpose:

- "SHIT+CTRL+B" or "CMD+SHIFT+B": Build and run with the desired file (for first time use)
- "CTRL+B" or "CMD+B": Build with the preselected build file for the current file type (faster once build file is selected).

By default, the code will be run in a display box inside Sublime Text and therefore it will be impossible to pause or stop the process. For more convenience, it is recommended to create a customized build file to run the C code (Python procedures can simply be run inside Sublime Text) using OpenMP for parallel computing in a dedicated terminal window. Progress of the simulation will be shown and the user will be free to run other files in the editor in the meantime, while also having the choice of stopping the process (CTRL+C) if needed. Below is a typical example of build file that does as mentioned:

C_parallel.sublime-build

```
{
	"selector": "source.c",
	"cmd": ["bash", "-c", "gcc -L/usr/local/lib/ -fopenmp -std=c99 '${file}' -o '${file_path}/${file_base_name}' -Wall -lgsl -lgslcblas -lm
&& '${file_path}/${file_base_name}'"],
	"file_regex": "^(..[^:]*):([0-9]+):?([0-9]+)?:? (.*)$",
	"working_dir": "${file_path}",


	"variants":
	[
		{
			"name": "Run in terminal",
			"cmd" : ["gnome-terminal -x bash -c \"gcc -L/usr/local/lib/ -fopenmp -std=c99 '${file}' -o
'${file_path}/${file_base_name}' -Wall -lgsl -lgslcblas -lm && '${file_path}/${file_base_name}'; exec bash\""],
			"shell": true,
		}
	]
}
```

Division of full fluid domain in blocks

To be able to remain as general as possible, the current simulation code allows the user to freely define the fluid domain as a sum of conformal rectangles. For each rectangle, the user needs to specify the boundary conditions at each face in the file "parameters.h". Below are examples of simple boundary conditions already available in the current code:

- *"slip":* slip-wall boundary condition (impermeable but not imposing zero velocity at the wall).
- *"wall":* no-slip-wall boundary condition setting 0 velocity at the wall (viscous condition).

- *"conN":* the face is connected to the face of another rectangle. N is the index of the domain connected to the current domain. If this keyword is used for a left face, the right face of domain N will be connected to it (and vice-versa). This applies similarly to the vertical direction. This condition is the most important and insures that all subdomains are connected and form the total fluid domain wanted by the user. Connection is ensured by copying the parameters of the neighbour domain in the ghost cell rows of the current domain.
- *"supI":* a supersonic inlet condition (not able to deal with backflow)
- *"supO":* a supersonic outlet condition (not able to deal with backflow).

parameters.h (not exhaustive)

```
double xstart[] = {0.0,L_TUBE,L_TUBE,-L_INLET_P,0.0,L_TUBE}; // X of most bottom left point
double ystart[] = {0.0,0.0,R0,R0+H_WALL,R0+H_WALL,R0+H_WALL}; // Y of most bottom left point
double xlength[] = {L_TUBE,L_OUT,L_OUT,L_INLET_P,L_TUBE,L_OUT}; // X length of domain
double ylength[] = {R0,R0,H_WALL,R_OUT,R_OUT,R_OUT}; // Y length of domain
char *BOUNDARY_TYPE[][4] = {{"slip","con1","slip","slip"},  // Domain 0
                            {"con0","slip","slip","con2"},  // Domain 1
                            {"slip","slip","con1","con5"},  // Domain 2
                            {"slip","con4","slip","slip"},  // Domain 3
                            {"con3","con5","slip","slip"},  // Domain 4
                            {"con4","slip","con2","slip"}}; // Domain 4
```

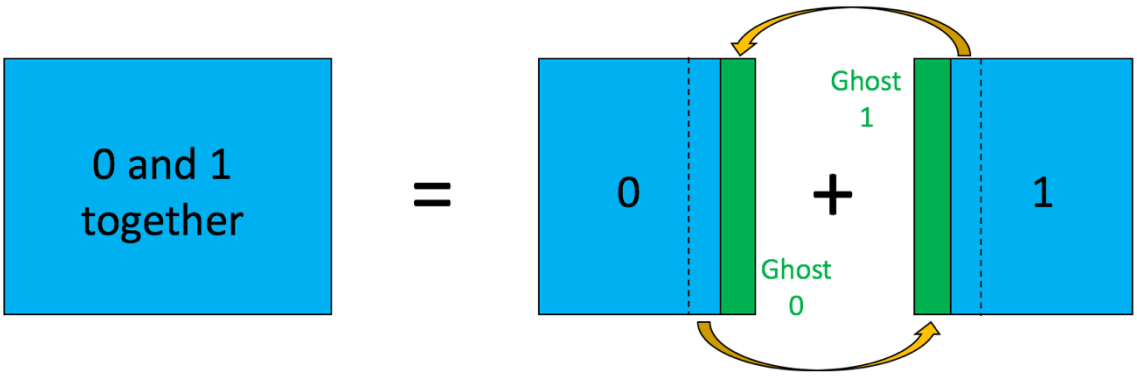

Figure: Illustration of how a domain is cut in several subdomains

Currently, two ghost cells are required at every domain face, so each domain should include more than two cells in each direction to make sure that all the blocks are truly consistent and form the intended fluid domain.

Why use a multi-block strategy?

If the fluid domain is just a rectangle, it is very simple to apply boundary conditions since only four are needed at each of the faces. When the domain is more complex, several boundary conditions must be set at random locations so the latter must be specified by the user for each new case, which is very inconvenient. As an alternative, the Microwave Rocket simulation code can model any kind of fluid domain that can be defined as the union of conformal rectangles (similarly to the *BlockMesh* tool of OpenFOAM). In that case, each domain is a rectangle and it becomes very easy to apply boundary conditions.

However, this strategy is not without disadvantages. First, a direct consequence is higher programming complexity since all domains have different sizes and must yet be stored in a way that allows the user to easily manipulate them. In the code, resort to C pointers enables a notation of the following form:

$$u(k,i,j),$$

where u is the axial velocity component (respectively p for pressure, ρ for density, v for the radial velocity component, etc.). In the notation above, k represents the domain's index, while i and j represent the column and row indexes of that domain. It is very important to understand that the structure thereby defined is not a (NxMxL) matrix but an array of matrices of size (N_k, M_k). To define such structures in C, pointers and memory allocation in the heap memory (as opposed to stack by default) are employed.

List of files and short description

The C code is divided in a main file and several headers. Each header regroups files that have similar use or purpose in the code. Header files are all included in the main C file. The user may rename these files and add/remove/edit contents but should keep back up versions "just in case". Each of these files' contents is briefly summarized below:

ausm.h: Functions relative to the AUSM-DV scheme (Hanel scheme, entropy fix, etc.).

bound_cond.h: All boundary conditions are regrouped there along with the generalized function used in the main that automatically attributes ghost cell values based on the boundary keyword set by the user. New boundary conditions should be defined there.

export.h: Functions that aim at exporting simulation data in .DAT format, appending new data to existing files, creating folders and so on. Changing the output format of the simulation data can be done by editing this file and creating new exporting methods.

fem_model.h: Regroups all the functions relative to the FEM model (definition of reed petal mesh and structural matrices, numerical schemes, load calculation, etc.).

functions.h: Several functions that do not fit in other header files.

initial.h: Relative to initial conditions for each new case. If the user creates new cases, he should also define new initial conditions in this file.

main.c: The main file in which all other header files are called. It contains initialization of all structures and variables (with memory allocation), computation of FEM matrices, time loop and export routines.

microwave.h: Functions relative to the calculation of MSD Mach number and initial conditions inside Microwave Rocket.

muscl.h: Functions relative to the MUSCL reconstruction method, including flux limiter functions.

parameters.h: All simulation parameters that need to be specified by the user.

displayResults.py: A Python routine to plot the computed solution at any of the exported time steps for any of the saved simulations. The file should be customized by the user.

exportAnimation.py: A Python routine that generates an animation based on exported data. The animation is for one field (density, pressure for instance). The minimal and maximal values of axes should be customized by the user for the animation to show more colors.

showMesh.py: A simple Python routine that can be used to check the mesh and different subdomains of the currently run case.

Data export

Below are the different files currently written during simulation and their contents:

- **FEM.dat:** Nodal structural data of the first reed stage stored in columns: x position (in global reference frame), petal width at x, petal thickness at x, cross section at x, second moment of inertia at x.
- **M.dat, C.dat, K.dat:** Mass, structural damping and stiffness matrices.
- **mfr_intake.dat:** mass flow rate at Microwave Rocket intake (should be configured by user)
- **mfr_plenum.dat:** mass flow rate at plenum entry (should be configured by user)
- **mfr_stage.dat:** mass flow rate at each reed stage for each time step.
- **mfr_total.dat:** total mass flow rate through all valves at all time steps.
- **parameters.dat:** parameter file used as input for Python procedures.
- **p_drag.dat:** Pressure at the plenum wall.
- **pfr.dat:** PFR at every time step (definition of Oda-san).
- **plenum_density.dat:** Average density in plenum at all time steps (should be configured by user).
- **plenum_pressure.dat:** Average pressure in plenum at all time steps (should be configured by user).
- **p_ratio.dat:** Pressure ratio at all reed stages at all times.
- **P_tube_mean.dat:** Average pressure in Microwave Rocket (i.e. detonation tube) at all time steps.
- **p_wall.dat:** Wall pressure at the Microwave Rocket thrust wall.
- **RHO_tube_mean.dat:** Average density in Microwave Rocket at all time steps.
- **y_tip.dat:** tip displacement of all reed stages at all time steps.

The user is free to define new variables to export or to modify the existing code so as to store output data in a different and more convenient manner (for instance, all parameters in a common file, perhaps even directly written in a format readable by Excel or LibreOffice).

Annex 1: Simulation procedure for a detonation tube case

The detonation tube case simply models a detonation tube and an ambient region. This case employs only three domains and is labelled as "mw_tube" in the Microwave Rocket simulation code. All walls are slip-walls, and far-field regions are also treated as slip (which can also have the meaning of symmetry axis).

Below are the steps that the user should follow to run such a case:
- check that no folder is called "OUTPUT" in the path directory. If there is, delete this folder or rename it depending on whether the user wants to keep this data or not.
- in "parameters.h", the user should put the SIM_CASE variable to the value "mw_tube";" and comment other similar lines.
- in the same file, the user should choose geometry parameters (L and D), time-related parameters (time step, total duration of computation), initial ambient conditions (microwave power, ambient pressure and density).
- before running the full simulation, always run a few iterations and check the initial mesh, flow conditions, different domains, etc. (by setting a very low computation time). To run the code, go to the "main.c" file and press "CTRL+SHIFT+B" or "CTRL+B" (select "C parallel – run in terminal). A terminal window should open and display information.
- if no problem is observed, run the same procedure for a longer computation time until the progress in terminal shows 100%. During the simulation, it is possible to go to "displayResults.py" and to display real time results by setting the case name to "OUTPUT", adapting the number of domains (NSTART, NEND) and the time step to display. By default, first time step is 0 and 200 time steps are saved for one simulation. The last time step can be displayed by choosing N_ITER_PLOT=-1. Trying to plot time steps not exported yet will result in an error. It is recommended to regularly check results during the simulation to make sure that no unexpected problem occurs.
- rename the "OUTPUT" folder and use the .DAT files created inside for post processing.

Annex 2: Simulation procedure for a simulation with plenum

A simulation with plenum is more complex than a simulation without plenum and requires several simulations, including to pre-initialize the flow solution in the plenum. The overall procedure can be summarized as follows:
- check that there is no "OUTPUT" folder already existing.
- in "parameters.h", put the SIM_CASE variable to **"sup_plen_rocket"**. This case is a 5-domain case which only models a plenum region in supersonic flow. The supersonic conditions should be determined by the user (shock relations, assumption of altitude, barometric law, etc.).
- when a converged solution is obtained (no more pressure oscillation inside the plenum region), the simulation can be stopped and the folder corresponding to the last time step should be copied and pasted into the folder called "PRERUN". This is the default name of the folder containing initial solutions that can directly be imported in other cases. Rename the pasted folder and input the corresponding name to the variable PRERUN_FOLDER in "parameters.h".

- rename or delete the existing “OUTPUT” folder and put the SIM_CASE variable to **“plenum_rocket”**. This case will use the solution stored in the folder whose name corresponds to PRERUN_FOLDER and use it as initial solution for the plenum and outside supersonic field. The average conditions inside this initial solution are used to determine the MSD Mach number and the initial conditions inside the detonation tube.
- run this case as explained in the previous annex.

In this procedure, obtaining the convergence of the plenum solution is rather difficult and requires many interventions from the user and a long computation time. A reliable procedure ensuring that no oscillation is remaining after a given time should be found. To this day, no general procedure could be found.

Bibliography

Florian Nguyen. (2017). *Estimation of the Refilling Performance through Reed Valves and an Inlet Plenum in Microwave Rocket using CFD.* Tokyo: The University of Tokyo.