Titulació: 
Enginyeria Aeronàutica Superior

Alumne (nom i cognoms): 
Arnau Pons Lorente

Títol PFC: 
Study of Grain Burnback and Performance of Solid Rocket Motors

Director del PFC: 
Dr. Ramon Carreras Planells

Co-Tutor del PFC: 
Dr. Manel Soria Guerrero

Convocatòria de lliurament del PFC:
Quatrimestre de tardor 2012 - Defensa Gener 2013

Contingut d’aquest volum: -ANNEXES-
STUDY OF GRAIN BURNBACK AND PERFORMANCE OF SOLID ROCKET MOTORS

-ANNEXES -

by

ARNAU PONS LORENTE

TUTOR: Dr. Ramon Carreras Planells

CO-TUTOR: Dr. Manel Soria Guerrero

January 2013

A dissertation submitted to the Department of Heat Engines,

ETSEIAT- Universitat Politècnica de Catalunya,

in partial fulfillment of the requirements for the degree of

Aeronautical Engineer
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE OF CONTENTS</td>
<td>1</td>
</tr>
<tr>
<td>ANNEX A. MATLAB® CODE OF THE PROJECT</td>
<td>3</td>
</tr>
<tr>
<td>A.1. CODE OF 2D GRAIN BURNBACK AND 0D UNSTEADY FLOW SIMULATION</td>
<td>4</td>
</tr>
<tr>
<td>A.1.1. MAIN CODE: General_2D_discrete_cart_coord_0D_flow.m</td>
<td>9</td>
</tr>
<tr>
<td>A.1.2. FUNCTIONS</td>
<td>17</td>
</tr>
<tr>
<td>A.1.2.1. geometry_loading.m</td>
<td>17</td>
</tr>
<tr>
<td>A.1.2.2. initial_geometry_organization.m</td>
<td>17</td>
</tr>
<tr>
<td>A.1.2.3. curve_tangent_normal.m</td>
<td>18</td>
</tr>
<tr>
<td>A.1.2.4. cartesian_grid.m</td>
<td>20</td>
</tr>
<tr>
<td>A.1.2.5. minimum_distance_function.m</td>
<td>20</td>
</tr>
<tr>
<td>A.1.2.6. interface_tracker_per.m</td>
<td>21</td>
</tr>
<tr>
<td>A.1.2.7. port_area.m</td>
<td>32</td>
</tr>
<tr>
<td>A.1.2.8. rocket_performance_0D_transient.m</td>
<td>34</td>
</tr>
<tr>
<td>A.1.2.9. phi_one_time_step_evolution.m</td>
<td>36</td>
</tr>
<tr>
<td>A.1.2.10. interface_tracker_in.m</td>
<td>37</td>
</tr>
<tr>
<td>A.1.2.11. interface_tracker.m</td>
<td>51</td>
</tr>
<tr>
<td>A.1.2.12. interface_tracker2.m</td>
<td>62</td>
</tr>
<tr>
<td>A.1.2.13. mach_calculation.m</td>
<td>75</td>
</tr>
<tr>
<td>A.1.2.14. order_vector_points.m</td>
<td>75</td>
</tr>
<tr>
<td>A.1.2.15. perimeter.m</td>
<td>77</td>
</tr>
<tr>
<td>A.1.2.16. units_conversion.m</td>
<td>78</td>
</tr>
<tr>
<td>A.1.2.17. grain_video_evolution.m</td>
<td>78</td>
</tr>
<tr>
<td>A.1.2.18. grain_video_evolution_filled.m</td>
<td>79</td>
</tr>
<tr>
<td>A.2. CODE OF QUASI-3D GRAIN BURNBACK AND 0D UNSTEADY FLOW SIMULATION</td>
<td>81</td>
</tr>
<tr>
<td>A.2.1. MAIN CODE: General_Quasi_3D_discrete_cart_coord_0D_flow.m</td>
<td>87</td>
</tr>
<tr>
<td>A.2.2. FUNCTIONS</td>
<td>95</td>
</tr>
<tr>
<td>A.2.2.1. geometry_loading_3D.m</td>
<td>95</td>
</tr>
<tr>
<td>A.2.2.2. chamber_volume.m</td>
<td>97</td>
</tr>
<tr>
<td>A.2.2.3. curve_tangent_normal_3D.m</td>
<td>98</td>
</tr>
<tr>
<td>A.2.2.4. minimum_distance_function_3D.m</td>
<td>99</td>
</tr>
<tr>
<td>A.2.2.5. grain_cross_sections_pmt.m</td>
<td>100</td>
</tr>
<tr>
<td>A.2.2.6. rocket_performance_0D_transient_quasi_3D.m</td>
<td>102</td>
</tr>
<tr>
<td>A.2.2.7. grain_3D_plot_matrix.m</td>
<td>104</td>
</tr>
<tr>
<td>A.2.2.8. motor_case_generation.m</td>
<td>104</td>
</tr>
<tr>
<td>A.2.2.9. add_tri_matrix.m</td>
<td>105</td>
</tr>
</tbody>
</table>
A.2.2.10. case_radius_int.m ................................................................. 106
A.2.2.11. grain_cross_sections.m ..................................................... 106
A.2.2.12. grain_video_evolution_3D.m ............................................. 108
A.2.2.13. initial_geometry_organization_3D.m ................................. 109
A.2.2.14. interface_tracker_in_P.m ................................................. 110
A.2.2.15. interpolate_cross_section.m ............................................. 124
A.2.2.16. tri_generation.m ............................................................... 125
A.2.2.17. wetted_surface.m .............................................................. 128
A.2.2.18. wetted_surface2.m ............................................................. 128
A.3. CODE OF QUASI-3D GRAIN BURNBACK AND 1D QUASI-STEADY
FLOW SIMULATION .................................................................................. 129
A.3.1. MAIN CODE: General_Quasi_3D_flow_1D_discrete_cart_coord.m
................................................................. 133
A.3.2. FUNCTIONS .................................................................................. 142
A.3.2.1. dif_A_p.m .............................................................................. 142
A.3.2.2. grain_cross_sections_pmt_1D.m ............................................. 142
A.3.2.3. interpolate_bt_cs.m ............................................................... 145
A.3.2.4. nozzle_inlet_mach.m ............................................................. 145
A.3.2.5. p_u_rho_calc.m .................................................................. 145
A.3.2.6. perimeter_cross_section.m .................................................. 146
A.3.2.7. rocket_performance_1D_steady_Q3D.m .................................... 147

ANNEX B. SOME RELEVANT RESULTS OF 2D GRAIN DESIGNS .......... 150
B.1. STAR OF 10 SLOTTED TIPS ............................................................ 151
B.2. TRI-SLOT ....................................................................................... 154
B.3. DOGBONE ..................................................................................... 157
B.4. ANCHOR ...................................................................................... 160
B.5. REFERENCES ................................................................................. 163
**ANNEX A. MATLAB® CODE OF THE PROJECT**

This Annex A contains the Matlab® code developed during the *Study of Grain Burnback and Performance of Solid Rocket Motors*. This code has been used to study the grain burnback as well as the performance of different kinds of Solid Rocket Motors.

In order to be coherent with the Project Report, three main sections are distinguished. Firstly, the code of 2D grain burnback and 0D unsteady flow simulation is presented. Then, the code corresponding to quasi-3D grain burnback and 0D unsteady flow simulation is attached. Lastly, the code of 3D grain burnback and 1D quasi-steady flow simulation is included in this document.

In the introduction of each section a table summarizing the name of the functions and a brief introduction of each function is attached. Additionally, it should be pointed out that in each of the three main-sections another two sub-sections can be identified: the main code and the functions. This way, from the main code the functions are called. Therefore, while running the simulation tools for each of the cases; it is necessary to run the main code in Matlab® (not the functions) as well as to contain the functions in the same folder of the main code.
A.1. **CODE OF 2D GRAIN BURNBACK AND 0D UNSTEADY FLOW SIMULATION**

In the following Table 1 the names of the different Matlab® files used in the implementation of this 2D grain burnback and 0D unsteady flow simulation are identified. It should be pointed out that while there is a unique file corresponding to the main code, there are a total of 18 files used to implement independent functions.

<table>
<thead>
<tr>
<th>Nº</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Main Code</td>
<td>General_2D_discrete_cart_coord_0D_flow.m</td>
</tr>
<tr>
<td>1</td>
<td>Function</td>
<td>geometry_loading.m</td>
</tr>
<tr>
<td>2</td>
<td>Function</td>
<td>initial_geometry_organization.m</td>
</tr>
<tr>
<td>3</td>
<td>Function</td>
<td>curve_tangent_normal.m</td>
</tr>
<tr>
<td>4</td>
<td>Function</td>
<td>cartesian_grid.m</td>
</tr>
<tr>
<td>5</td>
<td>Function</td>
<td>minimum_distance_function.m</td>
</tr>
<tr>
<td>6</td>
<td>Function</td>
<td>interface_tracker_per.m</td>
</tr>
<tr>
<td>7</td>
<td>Function</td>
<td>port_area.m</td>
</tr>
<tr>
<td>8</td>
<td>Function</td>
<td>rocket_performance_0D_transient.m</td>
</tr>
<tr>
<td>9</td>
<td>Function</td>
<td>phi_one_time_step_evolution.m</td>
</tr>
<tr>
<td>10</td>
<td>Function</td>
<td>interface_tracker_in.m</td>
</tr>
<tr>
<td>11</td>
<td>Function</td>
<td>interface_tracker.m</td>
</tr>
<tr>
<td>12</td>
<td>Function</td>
<td>interface_tracker2.m</td>
</tr>
<tr>
<td>13</td>
<td>Function</td>
<td>mach_calculation.m</td>
</tr>
<tr>
<td>14</td>
<td>Function</td>
<td>order_vector_points.m</td>
</tr>
<tr>
<td>15</td>
<td>Function</td>
<td>perimeter.m</td>
</tr>
<tr>
<td>16</td>
<td>Function</td>
<td>units_conversion.m</td>
</tr>
<tr>
<td>17</td>
<td>Function</td>
<td>grain_video_evolution.m</td>
</tr>
<tr>
<td>18</td>
<td>Function</td>
<td>grain_video_evolution_filled.m</td>
</tr>
</tbody>
</table>

Table 1. List of names used to identify the main code and the functions.

Note that in Table 1, the number of the main code is equal to 0 and for each function, a number from 1 to 18 is used. Principally, in the Table 1 the functions have been listed taking into account the order in which they are called in the main code. Additionally, although some of the functions are not “called” directly from
the main code, they are also required in the simulation. That is why they are included in Table 1. Furthermore, due to being Matlab® files, note that each file name ends with ".m". Finally, it should be pointed out that this section is aimed to explain deeply the section 6.4.1. of the project report. Thereby, here a brief description of each function, but not of the main code, is presented as follows. Note that the functions have been presented taking into account the order given in Table 1:

1) geometry_loading.m. This function is used to upload the text files, where the input geometry data is saved. Note that these text files are generated through the discretization done in GID of the CAD designs. This way, the input data is the text file itself and the output data are three vectors containing the x, y and z coordinates of the nodes. The output vectors are called “NodesX”, “NodesY” and “NodesZ” respectively. The code of the function is attached in the section A.1.2.1. geometry_loading.m.

2) initial_geometry_organization.m. This function organizes the geometry data provided by the user in text files; so that, the code can handle properly the information. The input vectors are the “NodesX”, which is a vector of x coordinate of the points of the initial geometry curves, and “NodesY”, which is a vector of y coordinate of the points of the initial geometry curves. The outputs of this function are the matrix “Xi_int” and “Xi_ext”. In case of matrix “Xi_int”, it contains the points of the inner interface curves, ordered in such way that the curve goes from the beginning to the end of the curve. The “Xi_in” matrix first column corresponds to the x coordinate and the second column to the y coordinate. Analogously, the matrix “Xi_ext” contains the points of the outer interface curve (motor case). Its first column also corresponds to the x coordinate and the second column to the y coordinate. In order to observe the code, please, consult the section A.1.2.2. initial_geometry_organization.m.

3) curve_tangent_normal.m. This function calculates the tangent and normal vectors at each point of a given curve. The input required by this function is the “Xi” matrix. This matrix contains the points of the curve ordered. The first column is the x coordinates of the curve points and the second column is the y coordinates of the curve points. The outputs given by this function are the matrix “Tc” and “Nc”. The matrix “Tc” contains the tangent vector at each point of the given curve, being the first column of the matrix the x components and the second column the y components of the tangent vector of the curve at each point. The matrix “Nc” contains the normal vector at each point of the given curve, being the first column of the matrix the x components and the second column the y components of
the normal vector of the curve at each point. In order to see the code, please consult the section A.1.2.3. curve_tangent_normal.m.

4) **cartesian_grid.m.** This function is used to generate the Cartesian grid matrices needed for the calculation. Precisely, it gives as output the matrices of x coordinates and y coordinates of the grid points, known in the function as “X” and “Y”. Additionally, the inputs required by this function are three: “N” which is the discretization number of the grid (i.e. the grid has (N+1)x(N+1) points), “L” which is the size length of the grid (i.e. the grid has LxL size, widthxheight) and “Delta_xy” which refers to the spatial step size of the grid. Note that “Delta_xy” has to be equal in the x and y directions. These three inputs have to be determined by the user before the simulation begins. In order to see the specific coding, please consult section A.1.2.4. cartesian_grid.m.

5) **minimum_distance_function.m.** Through this function the formulation of minimum distance function is implemented. To understand the mathematical background of this function, please consult the section 4.3.2. of the project report. The input parameters required by this function are four: the matrix of the x coordinate values of the Cartesian grid called “Xcart”, the matrix of the y coordinate values of the Cartesian grid called “Ycart”, the matrix of ordered initial grain geometry named as “Xint” and the matrix of normal vectors at each initial grain geometry point called “Nc”. Note that the first column of “Xint” corresponds with the x coordinates of the initial grain geometry and the second one with the y coordinates. The output given by this function is the matrix of values obtained from the application of minimum distance function to each point of the Cartesian grid and named as “Dmin”. In order to see the Matlab® code, please, consult the section A.1.2.5. minimum_distance_function.m.

6) **interface_tracker_per.m** This function calculates the perimeter of the interface curve given the Cartesian grid and the discretized \( \phi \) function on the background Cartesian grid. The mathematical background of this function is explained in the section 4.3.3.1. of the project report. The required inputs of this function are the “X” and “Y” matrixes of the x and y coordinates of the Cartesian grid, the \( \phi \) matrix (“Phi”) and the radii of the motor case (“Rmax”). The output parameter is the value of the perimeter of the interface. Further information in the section A.1.2.6. interface_tracker_per.m.

7) **port_area.m.** This function calculates the value of the gas cross-section area, known as port area. The mathematical background of this function is found in 4.4.3.1. of the project report. The input parameters required in this section are four: the “X” and “Y” matrixes of the x and y coordinates of the Cartesian grid, the \( \phi \) matrix (“Phi”) and the radii of the motor case.
("R_case"). The output given by the function is the value of the port area itself. The code of the function is found in the section A.1.2.7. port_area.m

8) rocket_performance_0D_transient.m. This function calculates, for each time step, the internal ballistics of the solid rocket motor with a 0D unsteady flow model. Therefore, by calculating the internal ballistics the performance of the SRM is obtained. The input parameters of this function are the performance SRM variables calculated at the previous time step and the initial rocket parameters, such as propellant density, parameters of the burning rate law, etc. Then, using these inputs, the function computes the value of the rocket performance variables for the next time step, which are the outputs. For instance, the performance variables are the chamber pressure, chamber temperature, thrust, mass flow, inter-alia. The mathematical background of this function is explained in the section 5.2. of the project report. The code of the function can be found in the section A.1.2.8. rocket_performance_0D_transient.m.

9) phi_one_time_step_evolution.m. This function propagates the discretized $\phi$ function on the background Cartesian grid given the $\phi$ function in the previous time step. The mathematical background of this function is found in the section 4.3.4 of the project report. The code of the function can be found in the section A.1.2.9. phi_one_time_step_evolution.m.

10) interface_tracker_in.m. This function extracts the zero level set curve but limited to the region defined by the motor case. Hence, if the interface curve goes beyond the motor case, the function only plots the part of the interface belonging to the inside region of the motor case. The inputs parameters are the same of the function nº6 aforementioned. The output is the value of the zero level curve. The mathematical background of this function is explained in the section 4.3.3.1. of the project report. The code is found in the section A.1.2.10. interface_tracker_in.m.

11) interface_tracker.m. This function extracts the zero level set curve, which is the interface at a given time step. The output of the function is this curve itself, whether it goes or not beyond the motor case. The inputs of this function are the same as the function nº6 aforementioned. The mathematical background of this function is explained in the section 4.3.3.1 of the project report. The code is found in the section A.1.2.11. interface_tracker.m.

12) interface_tracker2.m. This function extracts three main outputs: the zero level set curve, the zero level set curve but limited to the region limited by the motor case and the perimeter of the zero level set curve at each time step. The input parameters of these functions are the same that mentioned in the function nº6. The mathematical background of this
function is explained in the section 4.3.3.1 of the project report. The code
is found in the section A.1.2.12. interface_tracker2.m.

13) **mach_calculation.m.** This function calculates the number of Mach
(output parameters) given the $\gamma$ of the gases, the area of the throat and
the nozzle exit area (input parameters). The code of the function is
attached in the section A.1.2.13. mach_calculation.m.

14) **order_vector_points.m.** This function orders the points of a given
matrix. Precisely, this matrix has to contain two columns: one for the $x$
coordinates and the other for the $y$ coordinates. The code of the function
is attached in the section A.1.2.14. order_vector_points.m.

15) **perimeter.m.** This function calculates the perimeter of a given close
curve. This perimeter is the output of the function. The inputs are two: a
matrix containing the $x$ and $y$ coordinates of a close curve ("C") and the
radii of the motor curve ($R_{\text{max}}$). The code of the function is attached in
the section A.1.2.15. perimeter.m.

16) **units_conversion.m.** This function enables to convert units among
meters, centimeters and millimeters. The input is the character of the
stated unity ("m", "cm" or "mm") and the output is the conversion factor.
The code of this function is found in the section A.1.2.16. units_conversion.m.

17) **grain_video_evolution.m.** This function enables to create a video of the
level set curve during the grain burnback. While the output is the video
itself, several inputs are required, such as: the $X$ and $Y$ matrices of the
Cartesian grid, the $\phi$ function, number of frames per second, the radii of
the motor case, inter-alia. The code of this function is found in the section
A.1.2.17. grain_video_evolution.m.

18) **grain_video_evolution_filled.m.** This function enables to create a video
showing the grain burnback evolution. For that, the area between the level
set curve and the motor case is filled. Similarly to the case of
grain_video_evolution.m, while the output of the function is the video
itself, the input parameters are the $X$ and $Y$ matrices of the Cartesian grid,
the $\phi$ function, number of frames per second, the radii of the motor case,
inter-alia. The code of this function is found in the section A.1.2.18. grain_video_evolution_filled.m.

As it has been aforementioned, in the following sections A.1.1. and A.1.2. the
Matlab® code used to simulate the 2D grain burnback and 0D unsteady flow is
attached. This way, during the next sections the format given by the Matlab®
program is conserved.
Study of Grain Burnback and Performance of Solid Rocket Motors

A.1.1. MAIN CODE: General_2D_discrete_cart_coord_0D_flow.m

```matlab
clc
clear all
close all

%-------------------------------------DATA INPUT-------------------------------------%

% INPUT PARAMETERS
N=250;        % Number of nodes per side in the cartesian grid.
N_Delta_web=60; % Number of contour lines shown in the Phi plot.
N_bs_plot=16;  % Number of burnt steps shown in grain evolution plot.
Time=7.5;     % Simulation time [s]
n_steps=900;  % Number of simulation time steps.
t_0=0.5;      % Time duration of the first phase of the simulation (heavy transient).
Delta_t0=0.003; % Time step during t_0 phase of the simulation.
Delta_t1=(Time-floor(t_0/Delta_t0)*Delta_t0)/(n_steps-floor(t_0/Delta_t0)); % Time step for the rest of simulation.

% Distance units selection:
units='cm';

% Units conversion:
[K_units] = units_conversion(units);
atm_units=101325; %Conversion factor from Pa to atm.

% Rocket input parameters:
L_g=1.5;        % Propellant grain length [m]
P_c0=2E6;       % Initial chamber pressure [Pa]
a=8e-5;        % a parameter of the propellant burning rate law: r=a*Pc^n
n_exp=0.23;    % n exponent of the propellant burning rate law: r=a*Pc^n
gamma_e=1.19577; % Combustion products gas gamma parameter
gamma=Cp/Cv
R_air=287;     % R of air [J/KgK]
R_e=322.21;    % R of the combustion products [J/KgK]
rho_p=1134.75; % Propellant density [Kg/m^3]
M_mol=0.03875; % Combustion products molar mass [kg/mol]
De=120;       % Nozzle exit diameter [mm]
Dt=25;        % Nozzle throat diameter [mm]
```
$$A_e=(\pi*(D_e*10^{-3})^2)/4; \text{ %Nozzle exit area [m}^2\text{]}$$

$$A_t=(\pi*(D_t*10^{-3})^2)/4; \text{ %Nozzle trhoat area [m}^2\text{]}$$

$$L_{\text{plenum}}=0.1; \text{ %Plenum length [m]}$$

$$[M_e] = \text{mach\_calculation}(A_e,A_t,\gamma_e); \text{ %Nozzle exit Mach number}$$

$$P_a=1e4; \text{ %Atmospheric pressure [Pa]}$$

$$T_{c,v}=[2462.02544,2470.45874,2474.34792,2476.69366,2478.30590,2479.50156]; \text{ %Vector of chamber temperature data relation with chamber pressure [K].}$$

$$F_{c,v}=1:1:6; \text{ %Vector of chamber pressure data [MPa].}$$

$$\text{p_T=polyfit(F_{c,v},T_{c,v},3);} \text{ %Polynomial coefficients of the function adjust of } T_c=f(P_c)\text{.}$$

\%Assignment of the rocket motor input parameters to a data vector:

rocket_parameters=zeros(1,14); \%rocket\_parameters: Vector array of the rocket inputs parameters.

rocket_parameters(1)=L_g; \%Propellant grain length [m]

rocket_parameters(2)=P_c0; \%Initial chamber pressure [Pa]

rocket_parameters(3)=a; \%a parameter of the propellant burning rate law: \( r=a*P_c^n \)

rocket_parameters(4)=n_exp; \%n exponent of the propellant burning rate law: \( r=a*P_c^n \)

rocket_parameters(5)=\gamma_e; \%Combustion products gas gamma parameter \( \gamma=C_p/C_v \)

rocket_parameters(6)=R_{\text{air}}; \%R of air [J/KgK]

rocket_parameters(7)=R_p; \%R of the propellant [J/KgK]

rocket_parameters(8)=\rho_p; \%Propellant density [Kg/m^3]

rocket_parameters(9)=M_{mol}; \%Combustion products molar mass [Kg/mol]

rocket_parameters(10)=A_e; \%Nozzle exit area [m^2]

rocket_parameters(11)=A_t; \%Nozzle trhoat area [m^2]

rocket_parameters(12)=M_e; \%Nozzle exit Mach number

rocket_parameters(13)=P_a; \%Atmospheric pressure [Pa]

\%GEOMETRY LOADING:

name_file='Star_10_vertexs.dat'; \%Name of the geometry file to be loaded.

[NodesX,NodesY,NodesZ] = geometry\_loading(name_file);
% Signed Minimum Distance calculation:
[Dmin] = minimum_distance_function(X,Y,Xi,Nc);

% Initialization of the Phi function as a signed minimum distance function:
Phi=zeros(N+1,N+1,n_steps+1);
Phi(:,:,1)=Dmin;

Delta_t=zeros(1,n_steps); % Time step vector.
V_time=zeros(1,n_steps+1); % Time vector [s].

rocket_parameters(14)=pi*(Rmax*K_units^-1)^2*L_plenum; % Plenum volume [m^3]

% Combustion chamber pressure considering a 0D steady flow model:
P_cm=zeros(n_steps+1,1);
P_cm(1)=P_c0;

% Initialization of rocket performance variables:
M_rocket=zeros(n_steps+1,17); % M_rocket: Matrix array of the rocket performance variables for each time step.

% M_rocket(:,1)=P Burning perimeter [m]
% M_rocket(:,2)=A_b Burning area [m^2]
% M_rocket(:,3)=A_p Port area [m^2]
% M_rocket(:,4)=r_dot Burning rate [m/s]
% M_rocket(:,5)=Web Burnt depth [m]
% M_rocket(:,6)=m_dot_g Mass production of gas due to combustion [kg/s]
% M_rocket(:,7)=m_dot_d Mass flow rate exiting the nozzle [kg/s]
% M_rocket(:,8)=P_c Combustion chamber pressure [Pa]
% M_rocket(:,9)=Tc Combustion chamber temperature [K]
% M_rocket(:,10)=Vc Combustion chamber volume [m^3]
% M_rocket(:,11)=rho_c Combustion chamber density [kg/m^3]
% M_rocket(:,12)=P_e Nozzle exit pressure [Pa]
% M_rocket(:,13)=V_e Nozzle exit velocity [m/s]
% M_rocket(:,14)=F Thrust [N]
% M_rocket(:,15)=m_p Propellant mass [kg]
% M_rocket(:,16)=c_star Characteristic velocity [m/s]
% M_rocket(:,17)=C_f Thrust coefficient []

% Assignation of initial values of the rocket performance variables:
M\_rocket(1,1)=interface\_tracker\_per(X,Y,Phi(:,1,1),Rmax)*K\_units\_1;\%Initial burning perimeter [m]
M\_rocket(1,3)=port\_area(X,Y,Phi(:,1,1),Rmax)*K\_units\_2;\%Initial port area [m^2]
M\_rocket(1,2)=M\_rocket(1,1)*rocket\_parameters(1)+2*(\pi*(Rmax*K\_units\_1)^2-M\_rocket(1,3));\%Initial burning area [m^2]
M\_rocket(1,4)=P\_c0^n\_exp;\%Initial burning rate [m/s]
M\_rocket(1,5)=0;\%Initial burnt depth [m]
M\_rocket(1,6)=M\_rocket(1,2)*rho\_p*M\_rocket(1,4);\%Initial mass production of gas due to combustion [kg/s]
M\_rocket(1,7)=M\_rocket(1,6);\%Initial mass flow rate exiting the nozzle [kg/s]
M\_rocket(1,8)=P\_c0;\%Initial combustion chamber pressure [Pa]
M\_rocket(1,9)=polyval(p\_T,P\_c0*1e-6);\%Initial combustion chamber temperature [K]
M\_rocket(1,10)=L\_g*M\_rocket(1,3)+rocket\_parameters(14);\%Initial combustion chamber volume [m^3]
M\_rocket(1,11)=0;\%Initial mass production of gas due to combustion [kg/s]
M\_rocket(1,12)=P\_c0/(1+(gamma\_e-1)/2*M\_e^2)*(gamma\_e/(gamma\_e-1));\%Initial nozzle exit pressure [Pa]
M\_rocket(1,13)=2*(gamma\_e*R\_e*M\_rocket(1,9)/(gamma\_e-1))*(1-(M\_rocket(1,12)/P\_c0)^((gamma\_e-1)/gamma\_e));\%Initial nozzle exit velocity [m/s]
M\_rocket(1,14)=M\_rocket(1,7)*M\_rocket(1,13)+(M\_rocket(1,12)-P\_a)*A\_e;\%Initial thrust [N]
M\_rocket(1,15)=rho\_p*L\_g*(\pi*(Rmax*K\_units\_1)^2-M\_rocket(1,3));\%Initial propellant mass [kg]
M\_rocket(1,16)=sqrt(rocket\_parameters(7)*M\_rocket(1,9))/(sqrt(rocket\_parameters(5)))*(2/(rocket\_parameters(5)+1))^((rocket\_parameters(5)+1)/(2*(rocket\_parameters(5)-1))));\%Initial characteristic velocity [m/s]
M\_rocket(1,17)=sqrt(((2*(rocket\_parameters(5)^2)/(rocket\_parameters(5)-1))^((2/(rocket\_parameters(5)+1))^((rocket\_parameters(5)-1)/rocket\_parameters(11)))+((M\_rocket(1,12)-rocket\_parameters(13))/M\_rocket(1,8))*(rocket\_parameters(10)/rocket\_parameters(11)));

%Calculation of the coupled evolution of the Phi function and the rocket performance:
for n=1:n\_steps \%Time loop
    if V\_time(n)<t\_0
        Delta\_t(n)=Delta\_t0;
    else
        Delta\_t(n)=Delta\_t1;
    end
    V\_time(n+1)=V\_time(n)+Delta\_t(n);

%Rocket Performance Calculation:
%M\_rocket(n+1,:)=rocket\_performance\_0D\_transient(M\_rocket(n,:),rocket\_parameters,Rmax,\_units,Delta\_t(n));
% Assignment of V burning rate for the Phi evolution:
V = M_rocket(n+1,4)*K_units;
% Phi evolution for the next time step:
[Phi(:, :, n+1)] = phi_one_time_step_evolution(Phi(:, :, n), N, Delta_xy, Delta_t(n), V);
% Determination of the interface perimeter, port area, burning area, m_d and chamber volume for each time step:
[M_rocket(n+1,1)] =
    interface_tracker_per(X,Y,Phi(:, :, n+1),Rmax)*K_units^-1;
[M_rocket(n+1,3)] = port_area(X,Y,Phi(:, :, n+1),Rmax)*K_units^-2;
M_rocket(n+1,2) = M_rocket(n+1,1)*(rocket_parameters(1)-2*M_rocket(n+1,5)*K_units^-1);% Determination of the interface perimeter, port area, burning area, m_d and chamber volume for each time step:
M_rocket(n+1,6) = rocket_parameters(8)*M_rocket(n+1,2)*M_rocket(n+1,4);
M_rocket(n+1,10) = (rocket_parameters(1)-2*M_rocket(n+1,5)*K_units^-1)*M_rocket(n+1,3)+2*pi*(Rmax*K_units^-1)^2+rocket_parameters(14);
% Combustion chamber pressure considering a 0D steady flow model:
P_cm(n+1) = ((M_rocket(n+1,2)/rocket_parameters(11))*rocket_parameters(3)*rocket_parameters(8)*M_rocket(n+1,16))^(1/(1-rocket_parameters(4)));
end
%
-------------------PLOTS-------------------
figure(1);
Phi_max = max(max(Phi(:, :, n_steps+1)));
Phi_min = min(min(Phi(:, :, n_steps+1)));
[C] = interface_tracker(X,Y,Phi(:, :, n_steps+1));
contourf(X,Y,Phi(:, :, n_steps+1),[Phi_min:Delta_web:Phi_max]);
hold on;
axis('equal')
plot(Xi(:, 1), Xi(:, 2), 'color', [1 1 1])
hold on;
plot(C(:, 1), C(:, 2), 'color', [1 1 1])
hold on;
plot(Xi_Ext(:, 1), Xi_Ext(:, 2), 'color', [1 1 1])
hold on;
xlim([-1.1*L/2 1.1*L/2])
ylim([-1.1*L/2 1.1*L/2])
text1 = ['x [' ' units '] ']
text2 = ['y [' ' units '] ']
xlabel(text1)
ylabel(text2)
title('Phi evolution')
colorbar

figure(2);
Delta_web2 = (max(M_rocket(:, 5)) - min(M_rocket(:, 5)))/(N_bs_plot+1);
i=1;
for n=2:1:n_steps+1
Web1 = Web1*M_rocket(n, 5);
if S1<0
    [Cin] = interface_tracker_in(X,Y,Phi(:,:,n),Rmax);
    plot(Cin(:,1),Cin(:,2),'color',[0 0 1])
    hold on;
    i=i+1;
end

plot(Xi(:,1),Xi(:,2),'LineWidth',2,'color',[0 0 0])
hold on;
plot(Xi_Ext(:,1),Xi_Ext(:,2),'LineWidth',2,'color',[0 0 0])
hold on;
axis('equal')
text1=['x [' units ']'];
text2=['y [' units ']'];
xlabel(text1)
ylabel(text2)
title('Grain evolution during rocket burning')
xlim([-0.75*L/2 0.75*L/2])
ylim([-0.75*L/2 0.75*L/2])

figure(3);
set(3,'Name','Time evolution of the rocket performance 1');

subplot(2,2,1);
plot(V_time,M_rocket(:,2),'color',[0 0 1])
hold on
xlabel('t [s]')
ylabel('A_b [m^2]')
title('Burning area vs. time')

subplot(2,2,2);
plot(V_time,M_rocket(:,6),'color',[0 0 1])
hold on
xlabel('t [s]')
ylabel('m_d_o_t_g [kg/s]')
title('Mass flow rate due to propellant combustion vs. time ')

subplot(2,2,3);
plot(V_time,M_rocket(:,8)/atm_units,'color',[0 0 1])
hold on
xlabel('t [s]')
ylabel('P_c [atm]')
title('Combustion chamber pressure vs. time')

subplot(2,2,4);
plot(V_time,M_rocket(:,7),'color',[0 0 1])
hold on
xlabel('t [s]')
ylabel('m_d_o_t_d [kg/s]')
title('Mass flow rate exiting the nozzle vs. time')

figure(4);
set(4,'Name','Time evolution of the rocket performance 2');

subplot(2,2,1);
plot(V_time,1000*M_rocket(:,4),'color',[0 0 1])
hold on
xlabel('t [s]')
ylabel('r_d_o_t [mm/s]')
title('Burning rate vs. time')

subplot(2,2,2);
plot(V_time,M_rocket(:,1)*K_units,'color',[0 0 1])
hold on
xlabel('t [s]')
text1=['Perimeter [' units ']'];
ylabel(text1)
title('Burning perimeter versus time')

subplot(2,2,3);
plot(V_time,M_rocket(:,3)*K_units^2,'color',[0 0 1])
hold on
xlabel('t [s]')
text1=['A_p [' units '^2']'];
ylabel(text1)
title('Port area vs. time')

subplot(2,2,4);
plot(V_time,M_rocket(:,9), 'color',[0 0 1])
hold on
xlabel('t [s]')
ylabel('T_c [K]')
title('Combustion chamber temperature vs. time')

figure(5);
set(5,'Name','Time evolution of the rocket performance 3');

subplot(2,2,1);
plot(V_time,M_rocket(:,14), 'color',[0 0 1])
hold on
xlabel('t [s]')
ylabel('F [N]')
title('Thrust vs. time')

subplot(2,2,2);
plot(V_time,M_rocket(:,13), 'color',[0 0 1])
hold on
xlabel('t [s]')
ylabel('V_e_x_i_t [m/s]')
title('Nozzle exit velocity vs. time')

subplot(2,2,3);
plot(V_time,M_rocket(:,11), 'color',[0 0 1])
hold on
xlabel('t [s]')
ylabel('\rho_c [kg/m^3]')
title('Combustion chamber density vs. time')

subplot(2,2,4);
plot(V_time,M_rocket(:,15), 'color',[0 0 1])
hold on
xlabel('t [s]')
ylabel('M_p_r_o_p_e_l_l_a_n_t [kg]')
title('Propellant mass vs. time')
figure(6);
set(6,'Name','Time evolution of the rocket performance 4');

subplot(2,2,1);
plot(V_time,M_rocket(:,16),'color',[0 0 1])
hold on
xlabel('t [s]')
ylabel('c* [m/s]')
title('Characteristic velocity vs. time')

subplot(2,2,2);
plot(V_time,M_rocket(:,17),'color',[0 0 1])
hold on
xlabel('t [s]')
ylabel('C_f')
title('Thrust coefficient vs. time')

subplot(2,2,3);
plot(M_rocket(:,5),M_rocket(:,1)*K_units,'color',[0 0 1])
hold on
text2=['Web [' units ']];
xlabel(text2)
text1=['Perimeter [' units ']];
ylabel(text1)
title('Burning perimeter vs. burnt depth')

subplot(2,2,4);
plot(V_time,M_rocket(:,12)/atm_units,'color',[0 0 1])
hold on
xlabel('t [s]')
ylabel('P_e_x_i_t [atm]')
title('Nozzle exit pressure vs. time')

figure(7);
plot(V_time,P_cm/atm_units,'color',[0 0 1])
hold on
xlabel('t [s]')
ylabel('P_c_m [atm]')
title('Combustion chamber pressure (0D steady flow) versus time ')

figure(8);
plot(Xi(:,1),Xi(:,2),'color',[0 0 1])
hold on;
plot(Xi_Ext(:,1),Xi_Ext(:,2),'LineWidth',2,'color',[0 0 0])
hold on
axis('equal')
text1=['x [' units ']];
text2=['y [' units ']];
xlabel(text1)
ylabel(text2)
legend('Initial grain geometry','Motor case')
title('Grain initial geometry')
xlim([-0.75*L/2 0.75*L/2])
ylim([-0.75*L/2 0.75*L/2])
A.1.2. FUNCTIONS

The functions presented in this section are listed considering the order given in the Table 1.

A.1.2.1. geometry_loading.m

```matlab
function [NodesX,NodesY,NodesZ] = geometry_loading(name_file)

%UNTITLED Summary of this function goes here
% Detailed explanation goes here

M=load(name_file);

NodesX=M(:,1);
NodesY=M(:,2);
NodesZ=M(:,3);

end
```

A.1.2.2. initial_geometry_organization.m

```matlab
function [Xi_int,Xi_ext,Rmax] = initial_geometry_organization(NodesX,NodesY)

%initial_geometry_organization: this function organizes the geometry data
%given by the user in a .dat file so that the code can handle properly
%this information.

%The inputs of this function are:
%NodesX: vector of x coordinate of the points of the initial geometry
%NodesY: vector of y coordinate of the points of the initial geometry

%The outputs of this function are:
%Xi_int: the points of the inner interface curve. The points are
%ordered, so that the curves goes from the beginning to the end.
%The first column is the x coordinate of the ith curve point and
%the second column is the y coordinate of the ith curve point.
%Xi_ext: the points of the outer interface curve, which corresponds
%to the outer limit of the propellant, the motor case. The points
%are ordered, so that the curves goes from the beginning to the
%end. The first column is the x coordinate of the ith curve point
%and the second column is the y coordinate of the ith curve point.
```
%Rmax: Rmax corresponds to the motor case radii i.e. the outer limit of the propellant.

Determination of the upper limit of the initial geometry points (Rmax):
Rmax=0;
for i=1:1:size(NodesX,1)
  R=sqrt(NodesX(i)^2+NodesY(i)^2);
  if R>Rmax
    Rmax=R;
  end
end

Calculation of Xi_int and Xi_ext:
k=1;
l=1;
for i=1:1:size(NodesX,1)
  X1=NodesX(i);
  Y1=NodesY(i);
  R1=sqrt(X1^2+Y1^2);
  if R1<0.95*Rmax
    Xi_int(k,:)= [X1 Y1];
    k=k+1;
  else
    Xi_ext(l,:)= [X1 Y1];
    l=l+1;
  end
end

[Xi_int] = order_vector_points(Xi_int);
[Xi_ext] = order_vector_points(Xi_ext);
%Check that both curves close themself. If any of the curves does not close itself, then close itself:
if Xi_int(1,:)-Xi_int(size(Xi_int,1),:)~= [0 0]
  Xi_int=[Xi_int; Xi_int(1,:)];
end
if Xi_ext(1,:)-Xi_ext(size(Xi_ext,1),:)~= [0 0]
  Xi_ext=[Xi_ext; Xi_ext(1,:)];
end

A.1.2.3. curve_tangent_normal.m

function [Tc,Nc] = curve_tangent_normal(Xi)
%curve_tangent_normal: this function calculates the tangent and normal vectors of a given curve at each curve point.

%The inputs of this function are:
%Xi: the points of the curve, ordered. The first column is the x coordinate of the ith curve point and the second column is the y coordinate of the ith curve point.
%The outputs of this function are:
Study of Grain Burnback and Performance of Solid Rocket Motors

%Tc: the tangent vector at each point of 'Xi'. The first column is the %x component of the tangent vector to the curve at the ith point, %and the second column is the y component. %Nc: the normal vector at each point of 'Xi'. The first column is the %x component of the normal vector to the curve at the ith point, %and the second column is the y component.

N_int=size(Xi,1);
%Initialization of Tc and Nc.
Tc=zeros(N_int,2); %Tangent vector at each point of the curve.
Nc=zeros(N_int,2); %Normal vector at each point of the curve.
if Xi(1,:)-Xi(N_int,:)==[0 0]
    cond=1;
else
    cond=0;
end
for i=1:1:N_int
    if i==1 && cond==1
        Delta_S=sqrt((Xi(i,1)-Xi(i+1,1))^2+(Xi(i,2)-Xi(i+1,2))^2);
        Xi_m=[Xi(N_int-1,1) Xi(N_int-1,2)];
        Xi_p=[Xi(i+1,1) Xi(i+1,2)];
    elseif i==1 && cond==0
        Delta_S=sqrt((Xi(i,1)-Xi(N_int,1))^2+(Xi(i,2)-Xi(N_int,2))^2);
        Xi_m=[Xi(N_int,1) Xi(N_int,2)];
        Xi_p=[Xi(i+1,1) Xi(i+1,2)];
    elseif i==N_int && cond==1
        Delta_S=sqrt((Xi(i,1)-Xi(i-1,1))^2+(Xi(i,2)-Xi(i-1,2))^2);
        Xi_m=[Xi(i-1,1) Xi(i-1,2)];
        Xi_p=[Xi(2,1) Xi(2,2)];
    elseif i==N_int && cond==0
        Delta_S=sqrt((Xi(i,1)-Xi(i-1,1))^2+(Xi(i,2)-Xi(i-1,2))^2);
        Xi_m=[Xi(i,1) Xi(i,2)];
        Xi_p=[Xi(1,1) Xi(1,2)];
    else
        Delta_S=sqrt((Xi(i,1)-Xi(i-1,1))^2+(Xi(i,2)-Xi(i-1,2))^2);
        Xi_m=[Xi(i-1,1) Xi(i-1,2)];
        Xi_p=[Xi(i+1,1) Xi(i+1,2)];
    end
end
%Calculation of tangent and normal vectors of the interface curve
Tc(i,:)=(1/(2*Delta_S))*(Xi_p-Xi_m);
Tcnorm=norm(Tc(i,:));
%Normalization of the tangent vector
Tc(i,:)=Tc(i,:)/Tcnorm;
Nc(i,:)=Tc(i,:)*Tc(i,:);
Ncnorm=norm(Nc(i,:));
%Normalization of the normal vector
Nc(i,:)=Nc(i,:)/Ncnorm;
A.1.2.4. cartesian_grid.m

function [X,Y] = cartesian_grid(N,L,Delta_xy)
%cartesian_grid: this function generates the matrices X and Y of the
cartesian grid needed for the code calculations.

%The inputs of this function are:
% N: discretization number of the grid i.e. the grid has
% (N+1)x(N+1) points.
% L: size (length) of the grid i.e. the grid has LxL size
% WIDTHxHEIGHT.
% Delta_xy: spatial step size of the grid. Equal in x and y
direction.
%The outputs of this function are:
% X: matrix of the x coordinates of the grid points.
% Y: matrix of the y coordinates of the grid points.

%Intialization of X and Y matrices.
X=zeros(N+1); % X matrix.
Y=zeros(N+1); % Y matrix.

%X and Y Matrices calculation:
for i=1:1:N+1
    for j=1:1:N+1
        X(i,j)=Delta_xy*(j-1)-L/2;
        Y(i,j)=Delta_xy*(i-1)-L/2;
    end
end

A.1.2.5. minimum_distance_function.m

function [Dmin] = minimum_distance_function(Xcart,Ycart,Xint,Nc)
%UNTITLED4 Summary of this function goes here
% Detailed explanation goes here

N=size(Xcart,1)-1;
%Intialization of Dmin matrix.
Dmin=zeros(N+1,N+1);

for i=1:1:N+1
    %y direction loop of the cartesian grid
    for j=1:1:N+1
        %x direction loop of the cartesian grid
        Dmin(i,j)=1e6;
        for k=1:1:size(Xint,1)
            %loop for all the interface curve points
            D=sqrt((Xcart(i,j)-Xint(k,1))^2+(Ycart(i,j)-Xint(k,2))^2);
            if D<Dmin(i,j)
                Dmin(i,j)=D;
                Pos=k;
            end
        end
        Vcart=[Xcart(i,j) Ycart(i,j)];
        condicio=dot(Vcart-Xint(Pos,:),Nc(Pos,:));
        if condicio<0
            Dmin(i,j)=-Dmin(i,j);
        end
    end
end
function [P] = interface_tracker_per(X,Y,Phi,Rmax)
%interface_tracker_per: this function calculates the perimeter of the
%interface curve given the cartesian grid and the Phi matrix.
%   Detailed explanation goes here

%%
dx=diff(Y(1:2,1));
Delta_xy=dx;
N=size(X,1);
epsilon=1e-6;

%% %First step: find a interface point and set it as the first point of the curve:
%Vertical scan from the y=-L/2 and x=0 to y=L/2 and x=0.
j_cc=floor((N+1)/2);
for i=1:1:N-1
    Phi1=Phi(i,j_cc);
    Phi2=Phi(i+1,j_cc);
    Phi3=Phi(i+1,j_cc+1);
    Phi4=Phi(i,j_cc+1);
    X1=X(i,j_cc);
    Y1=Y(i,j_cc);
    X2=X(i+1,j_cc);
    Y2=Y(i+1,j_cc);
    X3=X(i+1,j_cc+1);
    Y3=Y(i+1,j_cc+1);
    X4=X(i,j_cc+1);
    Y4=Y(i,j_cc+1);
    if Phi1*Phi4<0 && Phi2*Phi3<0 %CASE 1
        Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
        Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
        Ya_0=Y1;
        Yb_0=Y2;
        i_0=i;
        j_0=j_cc;
        case_id=1;
        break;
    elseif Phi1*Phi2<0 && Phi1*Phi4<0 %CASE 2
        Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
        Xb_0=X1;
        Ya_0=Y1;
        Yb_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
        i_0=i;
        j_0=j_cc;
        case_id=2;
        break;
    elseif Phi1*Phi4<0 && Phi3*Phi4<0 %CASE 3
        Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
        Xb_0=X4;
        Ya_0=Y1;
        Yb_0=Y1;
Yb_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
i_0=i;
j_0=j_cc;
case_id=3;
break;
elseif Phi1*Phi2<0 && Phi3*Phi4<0  %CASE 4
Xa_0=X1;
Xb_0=X4;
Ya_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
Yb_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
i_0=i;
j_0=j_cc;
case_id=4;
break;
elseif Phi1*Phi2<0 && Phi2*Phi3<0  %CASE 5
Xa_0=X1;
Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
Ya_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
Yb_0=Y2;
i_0=i;
j_0=j_cc;
case_id=5;
break;
elseif Phi2*Phi3<0 && Phi3*Phi4<0  %CASE 6
Xa_0=X4;
Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
Ya_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi2)))*Delta_xy;
Yb_0=Y2;
i_0=i;
j_0=j_cc;
case_id=6;
break;
else
Xa_0=0;
Xb_0=0;
Ya_0=0;
Yb_0=0;
case_id=0;
end
end

%%
%If the vertical scan does not find the interface, then a horizontal scan
%shall be done:
if case_id==0
i_cc=floor((N+1)/2);
for j=1:1:N-1
Phi1=Phi(i_cc,j);
Phi2=Phi(i_cc+1,j);
Phi3=Phi(i_cc+1,j+1);
Phi4=Phi(i_cc,j+1);
X1=X(i_cc,j);
Y1=Y(i_cc,j);
X2=X(i_cc+1,j);
Y2=Y(i_cc+1,j);
X3=X(i_cc+1,j+1);
Y3=Y(i_cc+1,j+1);
X4=X(i_cc,j+1);


```plaintext
Y4 = Y(i_cc, j+1);
if Phi1*Phi4<0 && Phi2*Phi3<0  % CASE 1
  Xa_0 = X1 + (abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
  Xb_0 = X2 + (abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
  Ya_0 = Y1;
  Yb_0 = Y2;
  i_0 = i_cc;
  j_0 = j;
  case_id = 1;
  break;
elseif Phi1*Phi2<0 && Phi1*Phi4<0  % CASE 2
  Xa_0 = X1 + (abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
  Xb_0 = X1;
  Ya_0 = Y1;
  Yb_0 = Y1 + (abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
  i_0 = i_cc;
  j_0 = j;
  case_id = 2;
  break;
elseif Phi1*Phi4<0 && Phi3*Phi4<0  % CASE 3
  Xa_0 = X1 + (abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
  Xb_0 = X4;
  Ya_0 = Y1;
  Yb_0 = Y4 + (abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
  i_0 = i_cc;
  j_0 = j;
  case_id = 3;
  break;
elseif Phi1*Phi2<0 && Phi3*Phi4<0  % CASE 4
  Xa_0 = X1;
  Xb_0 = X4;
  Ya_0 = Y1 + (abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
  Yb_0 = Y4 + (abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
  i_0 = i_cc;
  j_0 = j;
  case_id = 4;
  break;
elseif Phi1*Phi2<0 && Phi2*Phi3<0  % CASE 5
  Xa_0 = X1;
  Xb_0 = X2 + (abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
  Ya_0 = Y1 + (abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
  Yb_0 = Y2;
  i_0 = i_cc;
  j_0 = j;
  case_id = 5;
  break;
elseif Phi2*Phi3<0 && Phi3*Phi4<0  % CASE 6
  Xa_0 = X4;
  Xb_0 = X2 + (abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
  Ya_0 = Y4 + (abs(Phi4)/(abs(Phi4)+abs(Phi2)))*Delta_xy;
  Yb_0 = Y2;
  i_0 = i_cc;
  j_0 = j;
  case_id = 6;
  break;
else
  Xa_0 = 0;
  Xb_0 = 0;
  Ya_0 = 0;
```
Yb_0=0;
case_id=-1;
end
dend

%%
% If the vertical and horizontal scan do not find the interface, 
% then an 
% overall scan shall be done:
if case_id==-1
for i=1:1:N-1  % y-direction loop
    for j=1:1:N-1 % x-direction loop
        Phi1=Phi(i,j);
        Phi2=Phi(i+1,j);
        Phi3=Phi(i+1,j+1);
        Phi4=Phi(i,j+1);
        X1=X(i,j);
        Y1=Y(i,j);
        X2=X(i+1,j);
        Y2=Y(i+1,j);
        X3=X(i+1,j+1);
        Y3=Y(i+1,j+1);
        X4=X(i,j+1);
        Y4=Y(i,j+1);
        if Phi1*Phi4<0 && Phi2*Phi3<0 % CASE 1
            Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
            Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
            Ya_0=Y1;
            Yb_0=Y2;
            i_0=i;
            j_0=j;
            case_id=1;
            break;
        elseif Phi1*Phi2<0 && Phi1*Phi4<0 % CASE 2
            Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
            Xb_0=X1;
            Ya_0=Y1;
            Yb_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
            i_0=i;
            j_0=j;
            case_id=2;
            break;
        elseif Phi1*Phi4<0 && Phi3*Phi4<0 % CASE 3
            Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
            Xb_0=X4;
            Ya_0=Y1;
            Yb_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
            i_0=i;
            j_0=j;
            case_id=3;
            break;
        elseif Phi1*Phi2<0 && Phi3*Phi4<0 % CASE 4
            Xa_0=X1;
            Xb_0=X4;
            Ya_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
            Yb_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
            i_0=i;
            j_0=j;
case_id=4;
break;
elseif Phi1*Phi2<0 && Phi2*Phi3<0 %CASE 5
Xa_0=X1;
Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
Ya_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
Yb_0=Y2;
i_0=i;
j_0=j;
case_id=5;
break;
elseif Phi2*Phi3<0 && Phi3*Phi4<0 %CASE 6
Xa_0=X4;
Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
Ya_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi2)))*Delta_xy;
Yb_0=Y2;
i_0=i;
j_0=j;
case_id=6;
break;
else
Xa_0=0;
Xb_0=0;
Ya_0=0;
Yb_0=0;
i_0=i;
j_0=j;
case_id=0;
end
end
end

%%% INTERFACE SCANNING AND DETERMINATION
closed=0;
k=1;
P=0;
while closed==0 %The condition closed means the closing of the
interface curve over itself.
%When closed is equal to 1 then the curve has closed and
%the while loop can be stopped.
%Assignment of the initial point of the interface curve.
if k=1
i_x=i_0;
j_x=j_0;
switch case_id
  case {1,3}
case_id_A=1;
C(1,1)=Xa_0;
C(1,2)=Ya_0;
  case {4,5}
case_id_A=2;
C(1,1)=Xa_0;
C(1,2)=Ya_0;
end
end
end
case 2
  case_id_A=2;
  C(1,1)=Xb_0;
  C(1,2)=Yb_0;
end

case 6
  case_id_A=3;
  C(1,1)=Xb_0;
  C(1,2)=Yb_0;
end

otherwise
  case_id_A=0;
  C(1,1)=0;
  C(1,2)=0;
end

% Assignment of the A point as the B point calculated in the
% previous iteration (A(k)=B(k-1)).
Xa=C(k,1);
Ya=C(k,2);
i=i_x;
% Node 1 i component of the square to be
j=j_x;
% Node 1 j component of the square to be
scanned.
X1=X(i,j);
% X position of the Node 1 of the square
to be scanned.
Y1=Y(i,j);
% Y position of the Node 1 of the square
to be scanned.
X2=X(i+1,j);
% X position of the Node 2 of the square
to be scanned.
Y2=Y(i+1,j);
% Y position of the Node 2 of the square
to be scanned.
X3=X(i+1,j+1);
% X position of the Node 3 of the square
to be scanned.
Y3=Y(i+1,j+1);
% Y position of the Node 3 of the square
to be scanned.
X4=X(i,j+1);
% X position of the Node 4 of the square
to be scanned.
Y4=Y(i,j+1);
% Y position of the Node 4 of the square
to be scanned.

Phi1=Phi(i,j);
% Phi value at the Node 1 of the square
to be scanned.
Phi2=Phi(i+1,j);
% Phi value at the Node 2 of the square
to be scanned.
Phi3=Phi(i+1,j+1);
% Phi value at the Node 3 of the square
to be scanned.
Phi4=Phi(i,j+1);
% Phi value at the Node 4 of the square
to be scanned.

% Identification of a square which has the interface passing
% through it
% twice:
if Phi1*Phi2<0 && Phi2*Phi3<0 && Phi3*Phi4<0 && Phi1*Phi4<0
  % Condition of interface double crossing the square.

V=C(k,:)-C(k-1,:);  % Vector of the interface ongoing
direction in the k point.
switch case_id_A  %Location of the square edge where the A point is.
    case 1  %A is located in the lower edge of the square.
        Xd_2=X1;
        Yd_2=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
        Xd_4=X4;
        Yd_4=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
        a=[(Xd_2-Xa) (Yd_2-Ya)];
        b=[(Xd_4-Xa) (Yd_4-Ya)];
        Va=dot(V,a);
        Vb=dot(V,b);
        if Va>Vb  %B is located in the left edge of the square.
            Xb=Xd_2;
            Yb=Yd_2;
            case_v=4;
            %Location of the next iteration point A (4=right edge of the square).
            i_x=i;
            %Node 1 i component of the square to be scanned in the next step.
            j_x=j-1;
            %Node 1 j component of the square to be scanned in the next step.
            else  %B is located in the right edge of the square.
                Xb=Xd_4;
                Yb=Yd_4;
                case_v=2;
                %Location of the next iteration point A (2=left edge of the square).
                i_x=i;
                %Node 1 i component of the square to be scanned in the next step.
                j_x=j+1;
                %Node 1 j component of the square to be scanned in the next step.
        end
    case 2  %A is located in the left edge of the square.
        Xd_1=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
        Yd_1=Y1;
        Xd_3=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
        Yd_3=Y2;
        a=[(Xd_1-Xa) (Yd_1-Ya)];
        b=[(Xd_3-Xa) (Yd_3-Ya)];
        Va=dot(V,a);
        Vb=dot(V,b);
        if Va>Vb  %B is located in the lower edge of the square.
            Xb=Xd_1;
            Yb=Yd_1;
            case_v=3;
            %Location of the next iteration point A (3=upper edge of the square).
            i_x=i-1;
            %Node 1 i component of the square to be scanned in the next step.
            j_x=j;
            %Node 1 j component of the square to be scanned in the next step.
            else  %B is located in the upper edge of the square.
                Xb=Xd_3;
                Yb=Yd_3;
                case_v=1;
                %Location of the next iteration point A (1=lower edge of the square).
\begin{verbatim}
    i_x=i+1; %Node 1 i component of the square to be scanned in the next step.
    j_x=j;  %Node 1 j component of the square to be scanned in the next step.
end

    case 3 %A is located in the upper edge of the square.
    Xd_2=X1;
    Yd_2=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
    Xd_4=X4;
    Yd_4=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
    a=([Xd_2-Xa) (Yd_2-Ya)];
    b=([Xd_4-Xa) (Yd_4-Ya)];
    Va=dot(V,a);
    Vb=dot(V,b);
    if Va>Vb %B is located in the left edge of the square.
        Xb=Xd_2;
        Yb=Yd_2;
        case_v=4; %Location of the next iteration point A (4=right edge of the square).
    end

    else %B is located in the right edge of the square.
        Xb=Xd_4;
        Yb=Yd_4;
        case_v=2; %Location of the next iteration point A (2=left edge of the square).
    end

    case 4 %A is located in the right edge of the square.
    Xd_1=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
    Yd_1=Y1;
    Xd_3=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
    Yd_3=Y2;
    a=([Xd_1-Xa) (Yd_1-Ya)];
    b=([Xd_3-Xa) (Yd_3-Ya)];
    Va=dot(V,a);
    Vb=dot(V,b);
    if Va>Vb %B is located in the lower edge of the square.
        Xb=Xd_1;
        Yb=Yd_1;
        case_v=3; %Location of the next iteration point A (3=upper edge of the square).
    end

    else %B is located in the upper edge of the square.
        Xb=Xd_3;
        Yb=Yd_3;
    end
\end{verbatim}
case_v=1; %Location of the next iteration point A (1=lower edge of the square).
i_x=i+1; %Node  i component of the square to be scanned in the next step.
end
otherwise
Xb=0;
Yb=0;
case_v=-1; %Location of the next iteration point A (-1=error).
i_x=0; %Node  i component of the square to be scanned in the next step (0=error).
j_x=0; %Node  j component of the square to be scanned in the next step (0=error).
end
else %The interface crosses the square just once.
switch case_id_A %Location of the square edge where the A point is.
case 1 %A is located in the lower edge of the square.
if Phi1*Phi2<0 %B is located in the left edge of the square.
Xb=X1;
Yb=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
case_v=4; %Location of the next iteration point A (4=right edge of the square).
i_x=i; %Node  i component of the square to be scanned in the next step.
j_x=j-1; %Node  j component of the square to be scanned in the next step.
elseif Phi2*Phi3<0 %B is located in the upper edge of the square.
Xb=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
Yb=Y2;
case_v=1; %Location of the next iteration point A (1=lower edge of the square).
i_x=i+1; %Node  i component of the square to be scanned in the next step.
j_x=j; %Node  j component of the square to be scanned in the next step.
elseif Phi3*Phi4<0 %B is located in the right edge of the square.
Xb=X4;
Yb=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
case_v=2; %Location of the next iteration point A (2=left edge of the square).
i_x=i; %Node  i component of the square to be scanned in the next step.
j_x=j+1; %Node  j component of the square to be scanned in the next step.
end
case 2 %A is located in the left edge of the square.
if Phi1*Phi4<0 %B is located in the lower edge of the square.
\[
X_b = X_1 + \frac{\text{abs}(\Phi_1)}{\text{abs}(\Phi_1) + \text{abs}(\Phi_4)} \cdot \Delta_{xy};
Y_b = Y_1;
\]

\text{case v=3; \%Location of the next iteration point A (3=upper edge of the square).}
\text{i_x=i-1; \%Node 1 i component of the square to be scanned in the next step.}
\text{j_x=j; \%Node 1 j component of the square to be scanned in the next step.}
\text{elseif } \Phi_2 \cdot \Phi_3 < 0 \text{ \%B is located in the upper edge of the square.}
\text{X_b=X_2 + \frac{\text{abs}(\Phi_2)}{\text{abs}(\Phi_2) + \text{abs}(\Phi_3)} \cdot \Delta_{xy};}
\text{Y_b=Y_2;}
\text{case v=1; \%Location of the next iteration point A (1=lower edge of the square).}
\text{i_x=i+1; \%Node 1 i component of the square to be scanned in the next step.}
\text{j_x=j; \%Node 1 j component of the square to be scanned in the next step.}
\text{elseif } \Phi_3 \cdot \Phi_4 < 0 \text{ \%B is located in the right edge of the square.}
\text{X_b=X_4;}
\text{Y_b=Y_4 + \frac{\text{abs}(\Phi_4)}{\text{abs}(\Phi_4) + \text{abs}(\Phi_3)} \cdot \Delta_{xy};}
\text{case v=2; \%Location of the next iteration point A (2=left edge of the square).}
\text{i_x=i; \%Node 1 i component of the square to be scanned in the next step.}
\text{j_x=j+1; \%Node 1 j component of the square to be scanned in the next step.}
\text{end}
\text{case 3 \%A is located in the upper edge of the square.}
\text{if } \Phi_1 \cdot \Phi_4 < 0 \text{ \%B is located in the lower edge of the square.}
\text{X_b=X_1 + \frac{\text{abs}(\Phi_1)}{\text{abs}(\Phi_1) + \text{abs}(\Phi_4)} \cdot \Delta_{xy};}
\text{Y_b=Y_1;}
\text{case v=3; \%Location of the next iteration point A (3=upper edge of the square).}
\text{i_x=i-1; \%Node 1 i component of the square to be scanned in the next step.}
\text{j_x=j; \%Node 1 j component of the square to be scanned in the next step.}
\text{elseif } \Phi_1 \cdot \Phi_2 < 0 \text{ \%B is located in the left edge of the square.}
\text{X_b=X_1;}
\text{Y_b=Y_1 + \frac{\text{abs}(\Phi_1)}{\text{abs}(\Phi_1) + \text{abs}(\Phi_2)} \cdot \Delta_{xy};}
\text{case v=4; \%Location of the next iteration point A (4=right edge of the square).}
\text{i_x=i; \%Node 1 i component of the square to be scanned in the next step.}
\text{j_x=j-1; \%Node 1 j component of the square to be scanned in the next step.}
\text{elseif } \Phi_3 \cdot \Phi_4 < 0 \text{ \%B is located in the right edge of the square.}
\text{X_b=X_4;}
\text{Y_b=Y_4 + \frac{\text{abs}(\Phi_4)}{\text{abs}(\Phi_4) + \text{abs}(\Phi_3)} \cdot \Delta_{xy};}
\text{case v=2; \%Location of the next iteration point A (2=left edge of the square).}
\text{i_x=i; \%Node 1 i component of the square to be scanned in the next step.}
\text{endif}
j_x=j+1; %Node 1 j component of the square to be scanned in the next step.
end

case 4 %A is located in the right edge of the square.
  if Phi1*Phi4<0 %B is located in the lower edge of the square.
    Xb=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
    Yb=Y1;
    case_v=3; %Location of the next iteration point A (3=upper edge of the square).
    i_x=i-1; %Node 1 i component of the square to be scanned in the next step.
    j_x=j; %Node 1 j component of the square to be scanned in the next step.
  elseif Phi1*Phi2<0 %B is located in the left edge of the square.
    Xb=X1;
    Yb=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
    case_v=4; %Location of the next iteration point A (4=right edge of the square).
    i_x=i;
    j_x=j-1; %Node 1 j component of the square to be scanned in the next step.
  elseif Phi2*Phi3<0 %B is located in the upper edge of the square.
    Xb=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
    Yb=Y2;
    case_v=1; %Location of the next iteration point A (1=lower edge of the square).
    i_x=i+1;
    j_x=j; %Node 1 j component of the square to be scanned in the next step.
  end
otherwise
  Xb=0;
  Yb=0;
  case_v=-1; %Location of the next iteration point A (-1=error).
  i_x=0;
  j_x=0; %Node 1 j component of the square to be scanned in the next step (0=error).
end
end

%Assignment of the B point to the interface curve.
C(k+1,1)=Xb;
C(k+1,2)=Yb;
case_id_A=case_v; %Assignment of the case_id_A for the next iteration.
dif=sqrt((Xb-C(1,1))^2+(Yb-C(1,2))^2);

%Perimeter calculation:
Ra=sqrt(Xa^2+Ya^2);
Rb=sqrt(Xb^2+Yb^2);
if Ra>=Rmax && Rb>=Rmax
di=0;
elseif Ra<Rmax && Rb<Rmax
di=sqrt((Xa-Xb)^2+(Ya-Yb)^2);
elseif Ra<Rmax && Rb>Rmax || Ra>Rmax && Rb<Rmax
a=(Xb-Xa)^2+(Yb-Ya)^2;
b=2*Xa*(Xb-Xa)+2*Ya*(Yb-Ya);
c=Xa^2+Ya^2-Rmax^2;
Disc=b^2-4*a*c;
if Disc>=0
    s1=(-b+sqrt(Disc))/(2*a);
s2=(-b-sqrt(Disc))/(2*a);
    if s1>=0 && s1<=1
        s=s1;
x=Xa+(Xb-Xa)*s;
y=Ya+(Yb-Ya)*s;
    elseif Ra<Rmax && Rb>Rmax
        di=sqrt((x-Xa)^2+(y-Ya)^2);
    elseif Ra>Rmax && Rb<Rmax
        di=sqrt((x-Xb)^2+(y-Yb)^2);
    end
elseif s2>=0 && s2<=1
    s=s2;
x=Xa+(Xb-Xa)*s;
y=Ya+(Yb-Ya)*s;
    if Ra<Rmax && Rb>Rmax
        di=sqrt((x-Xa)^2+(y-Ya)^2);
    elseif Ra>Rmax && Rb<Rmax
        di=sqrt((x-Xb)^2+(y-Yb)^2);
    end
else
    di=0;
end
P=di+P;
if dif==0
closed=1;
elseif dif<epsilon
closed=2;
else
closed=0;
end
k=k+1;
end

A.1.2.7. port_area.m

function [A_p] = port_area(X,Y,Phi,R_case)
% UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

N=size(X,1);
dx=diff(Y(1:2,1));
Delta_xy=dx;
A_p=0;
for i=1:1:N-1
    % y direction loop
    for j=1:1:N-1
        % x direction loop
        Phi1=Phi(i,j);
        % Phi value at the Node 1 of the square to be scanned.
        Phi2=Phi(i+1,j);
        % Phi value at the Node 2 of the square to be scanned.
        Phi3=Phi(i+1,j+1);
        % Phi value at the Node 3 of the square to be scanned.
        Phi4=Phi(i,j+1);
        % Phi value at the Node 4 of the square to be scanned.
        R1=sqrt((X(i,j))^2+(Y(i,j))^2);
        % Distance from the origin of the node 1 of the square to be scanned.
        Rm=R1+(Delta_xy*(X(i,j)+Y(i,j)))/(2*R1);
        % Mean radius (distance from the origin) of the four nodes of the square to be scanned.

        if Rm>R_case
            % The square evaluated is beyond the motor case.
            A_pi=0;
        else
            % The square evaluated is inside the motor case.

            if Phi1<0 && Phi2<0 && Phi3<0 && Phi4<0
                % The whole square evaluated is inside the interface.
                A_pi=Delta_xy^2;
            elseif Phi1*Phi4<0 && Phi2*Phi3<0
                % CASE 1
                A_ps=(1/2)*Delta_xy^2*((abs(Phi1)/(abs(Phi1)+abs(Phi4)))+abs(Phi2)/(abs(Phi2)+abs(Phi3)));
                if Phi1<0
                    A_pi=A_ps;
                else
                    A_pi=Delta_xy^2-A_ps;
                end
            elseif Phi1*Phi2<0 && Phi1*Phi4<0
                % CASE 2
                A_ps=(1/2)*Delta_xy^2*((abs(Phi1)/(abs(Phi1)+abs(Phi4)))*(abs(Phi1)/(abs(Phi1)+abs(Phi2))));
                if Phi1<0
                    A_pi=A_ps;
                else
                    A_pi=Delta_xy^2-A_ps;
                end
            elseif Phi1*Phi4<0 && Phi3*Phi4<0
                % CASE 3
                A_ps=(1/2)*Delta_xy^2*((abs(Phi4)/(abs(Phi1)+abs(Phi4)))*abs(Phi1)/(abs(Phi1)+abs(Phi2)));
                if Phi1<0
                    A_pi=A_ps;
                else
                    A_pi=Delta_xy^2-A_ps;
                end
            elseif Phi1*Phi2<0 && Phi3*Phi4<0
                % CASE 4
                A_ps=(1/2)*Delta_xy^2*((abs(Phi4)/(abs(Phi1)+abs(Phi4)))*abs(Phi4)/(abs(Phi3)+abs(Phi4)));
                if Phi4<0
                    A_pi=A_ps;
                else
                    A_pi=Delta_xy^2-A_ps;
                end
            end
        end
    end
end
A_{ps}=(1/2)\Delta_{xy}^2((\text{abs}(\Phi_1)/(\text{abs}(\Phi_1)+\text{abs}(\Phi_2))+(\text{abs}(\Phi_4)/(\text{abs}(\Phi_3)+\text{abs}(\Phi_4))));

\begin{align*}
  \text{if } \Phi_1<0 \\
  A_{pi}=A_{ps}; \\
  \text{else } \\
  A_{pi}=\Delta_{xy}^2-A_{ps}; \\
  \text{end}
\end{align*}

\text{elseif } \Phi_1*\Phi_2<0 \&\& \Phi_2*\Phi_3<0 \text{ \%CASE 5}

A_{ps}=(1/2)\Delta_{xy}^2((\text{abs}(\Phi_2)/(\text{abs}(\Phi_2)+\text{abs}(\Phi_3)))(\text{abs}(\Phi_2)/(\text{abs}(\Phi_1)+\text{abs}(\Phi_2))));

\begin{align*}
  \text{if } \Phi_2<0 \\
  A_{pi}=A_{ps}; \\
  \text{else } \\
  A_{pi}=\Delta_{xy}^2-A_{ps}; \\
  \text{end}
\end{align*}

\text{elseif } \Phi_2*\Phi_3<0 \&\& \Phi_3*\Phi_4<0 \text{ \%CASE 6}

A_{ps}=(1/2)\Delta_{xy}^2((\text{abs}(\Phi_3)/(\text{abs}(\Phi_2)+\text{abs}(\Phi_3)))(\text{abs}(\Phi_3)/(\text{abs}(\Phi_3)+\text{abs}(\Phi_4))));

\begin{align*}
  \text{if } \Phi_3<0 \\
  A_{pi}=A_{ps}; \\
  \text{else } \\
  A_{pi}=\Delta_{xy}^2-A_{ps}; \\
  \text{end}
\end{align*}

\text{else} \\
A_{pi}=0; \\
\text{end}

A_p=A_p+A_{pi};

end

end

end

A.1.2.8. rocket_performance_0D_transient.m

\textbf{function} \texttt{[M}_\texttt{rocket}_\texttt{next]} = \texttt{rocket\_performance\_0D\_transient(M}\_\texttt{rocket}_\texttt{n},\texttt{rocket\_parameters,R}}\_\texttt{case,p_T,K}}\_\texttt{units,Delta}}\_\texttt{t})
\texttt{\%UNTITLED Summary of this function goes here}
\texttt{\% Detailed explanation goes here}

\% INPUT PARAMETERS:
\%
\%M\_rocket\_n: Vector array of the rocket performance variables at the n time step.
\%M\_rocket\_n(1)=P \quad \text{Burning perimeter [m]}
\%M\_rocket\_n(2)=A\_b \quad \text{Burning area [m}^2]\text{]}
\%M\_rocket\_n(3)=A\_p \quad \text{Port area [m}^2]\text{]}
\%M\_rocket\_n(4)=r\_dot \quad \text{Burning rate [m/s]}
\%M\_rocket\_n(5)=Web \quad \text{Burnt depth [m]}
\%M\_rocket\_n(6)=m\_dot\_g \quad \text{Mass production of gas due to combustion [kg/s]}
\%M\_rocket\_n(7)=m\_dot\_d \quad \text{Mass flow rate exiting the nozzle [kg/s]}
\%M\_rocket\_n(8)=P\_c \quad \text{Combustion chamber pressure [Pa]}

34
% $M_{rocket \_ n}(9)=T_c$ Combustion chamber temperature [K]
% $M_{rocket \_ n}(10)=V_c$ Combustion chamber volume [m$^3$]
% $M_{rocket \_ n}(11)=\rho_c$ Combustion chamber density [kg/m$^3$]
% $M_{rocket \_ n}(12)=P_e$ Nozzle exit pressure [Pa]
% $M_{rocket \_ n}(13)=V_e$ Nozzle exit velocity [m/s]
% $M_{rocket \_ n}(14)=F$ Thrust [N]
% $M_{rocket \_ n}(15)=m_p$ Propellant mass [kg]
% $M_{rocket \_ n}(16)=c_{\_ star}$ Characteristic velocity [m/s]
% $M_{rocket \_ n}(17)=C_f$ Thrust coefficient []

%rocket_parameters: Vector array of the rocket inputs parameters.
%rocket_parameters(1)=L_g Propellant grain length [m]
%rocket_parameters(2)=P_c0 Initial chamber pressure [Pa]
%rocket_parameters(3)=a a parameter of the propellant burning rate law: $r=a\cdot P_{c}^n$
%rocket_parameters(4)=n_exp an exponent of the propellant burning rate law: $r=a\cdot P_{c}^n$
gamma parameter $\gamma=c_{p}/c_{v}$
%rocket_parameters(5)=gamma_e Combustion products gas gamma parameter $\gamma=C_{p}/C_{v}$
%rocket_parameters(6)=R_air $R$ of air [J/KgK]
%rocket_parameters(7)=R_e $R$ of the propellant [J/KgK]
%rocket_parameters(8)=rho_p Propellant density [Kg/m$^3$]
%rocket_parameters(9)=M_mol Combustion products molar mass [kg/mol]
%rocket_parameters(10)=A_e Nozzle exit area [m$^2$]
%rocket_parameters(11)=A_t Nozzle throat area [m$^2$]
%rocket_parameters(12)=M_e Nozzle exit Mach number
%rocket_parameters(13)=P_a Atmospheric pressure [Pa]
%rocket_parameters(14)=V_plenum Plenum volume [m$^3$]

%Initialization of M_rocket_next vector:
M_rocket_next=zeros(1,17);

%Calculation of the rocket performance variables:
Gamma_C=sqrt(rocket_parameters(5))*(2/(rocket_parameters(5)+1))^{((rocket_parameters(5)+1)/(2*(rocket_parameters(5)-1))))};
c_n=sqrt(rocket_parameters(7)*M_rocket_n(9))/Gamma_C;

%The interface perimeter, burning area and port area are calculated later because those depend on the Phi matrix evolution:
M_rocket_next(1)=M_rocket_n(1);
M_rocket_next(2)=M_rocket_n(2);
M_rocket_next(3)=M_rocket_n(3);

%Calculation of the combustion chamber pressure:
M_rocket_next(8)=M_rocket_n(8)+((Gamma_C^2*c_n^2)/M_rocket_n(10))*((rocket_parameters(8)-M_rocket_n(11))*M_rocket_n(2)*M_rocket_n(4)-((rocket_parameters(11)*M_rocket_n(8))/c_n))*Delta_t;

%Calculation of the next performance variables derivated from the combustion chamber pressure P_c:
M_rocket_next(4)=rocket_parameters(3)*M_rocket_next(8)^rocket_parameters(4);
M_rocket_next(5)=M_rocket_next(4)*K_units*Delta_t+M_rocket_n(5);
M_rocket_next(6)=rocket_parameters(8)*M_rocket_next(2)*M_rocket_next(4);
M_rocket_next(9)=polyval(p_T,M_rocket_next(8)*1e-6);
M_rocket_next(16)=sqrt(rocket_parameters(7)*M_rocket_next(9))/Gamma_C;
M_rocket_next(7)=(M_rocket_next(8)*rocket_parameters(11))/M_rocket_next(16);
M_rocket_next(10)=(rocket_parameters(1)-2*M_rocket_next(5)*K_units-1)*M_rocket_next(3)+2*M_rocket_next(5)*K_units-1*pi*(R_case*K_units-1)^2+rocket_parameters(14);
M_rocket_next(11)=M_rocket_next(8)/(M_rocket_next(16)^2*Gamma_C^2);
M_rocket_next(12)=M_rocket_next(8)/(1+(rocket_parameters(5)-1)/2*rocket_parameters(12)^2)^rocket_parameters(5)/(rocket_parameters(5)-1);
M_rocket_next(13)=(((2*rocket_parameters(5)*rocket_parameters(7)*M_rocket_next(9))/(rocket_parameters(5)-1))*(1-(M_rocket_next(12)/M_rocket_next(8))^((rocket_parameters(5)-1)/rocket_parameters(5))))^((rocket_parameters(5)-1)/rocket_parameters(5)))^((1/2);
M_rocket_next(17)=sqrt(((2*rocket_parameters(5)^2)/(rocket_parameters(5)-1))*((2/(rocket_parameters(5)+1))^((rocket_parameters(5)+1)/(rocket_parameters(5)-1))))*((M_rocket_next(12)/M_rocket_next(8))^((rocket_parameters(5)-1)/rocket_parameters(5))))+((M_rocket_next(12)/M_rocket_next(8))^((rocket_parameters(5)-1)/rocket_parameters(11)));% End of function

A.1.2.9. phi_one_time_step_evolution.m

function [Phi_next] =
phi_one_time_step_evolution(Phi_n,N,Delta_xy,Delta_t,V)
%UNTITLED Summary of this function goes here
% Detailed explanation goes here

%Initialization of Phi_n1 matrix:
Phi_next=zeros(N+1,N+1);
for i=1:1:N+1     %y direction loop
  for j=1:1:N+1     %x direction loop
    if i==1 & j==1
      DPhi_mX=0;
      DPhi_pX=(Phi_n(i+1,j)-Phi_n(i,j))/Delta_xy;
      DPhi_mY=0;
      DPhi_pY=(Phi_n(i+1,j)-Phi_n(i,j))/Delta_xy;
    elseif i==1 & j<N+1
      DPhi_mX=(Phi_n(i,j)-Phi_n(i,j-1))/Delta_xy;
      DPhi_pX=(Phi_n(i,j+1)-Phi_n(i,j))/Delta_xy;
      DPhi_mY=0;
      DPhi_pY=(Phi_n(i+1,j)-Phi_n(i,j))/Delta_xy;
    elseif j==1 & i<N+1
      DPhi_mX=(Phi_n(i,j)-Phi_n(i-1,j))/Delta_xy;
      DPhi_mY=0;
      DPhi_pX=(Phi_n(i+1,j)-Phi_n(i,j))/Delta_xy;
      DPhi_pY=(Phi_n(i,j)-Phi_n(i,j-1))/Delta_xy;
    elseif j==N+1 & i<N+1
      DPhi_mX=(Phi_n(i,j)-Phi_n(i-1,j+1))/Delta_xy;
      DPhi_pX=(Phi_n(i,j+1)-Phi_n(i,j))/Delta_xy;
      DPhi_mY=0;
      DPhi_pY=(Phi_n(i,j+1)-Phi_n(i,j-1))/Delta_xy;
    else
      if (i==1 && j==1) || (i==1 && j==N+1) || (j==1 && i==1) || (j==N+1 && i==1)
        DPhi_mX=0;
        DPhi_pX=(Phi_n(i,j)-Phi_n(i-1,j))/Delta_xy;
        DPhi_mY=0;
        DPhi_pY=(Phi_n(i,j)-Phi_n(i,j-1))/Delta_xy;
      elseif (i==1 && j==1) || (i==1 && j==N+1) || (j==1 && i==1) || (j==N+1 && i==1)
        DPhi_mX=(Phi_n(i,j)-Phi_n(i-1,j))/Delta_xy;
        DPhi_mY=0;
        DPhi_pX=(Phi_n(i,j)-Phi_n(i,j-1))/Delta_xy;
        DPhi_pY=(Phi_n(i+1,j)-Phi_n(i,j))/Delta_xy;
      elseif (i==1 && j==1) || (i==1 && j==N+1) || (j==1 && i==1) || (j==N+1 && i==1)
        DPhi_mX=(Phi_n(i,j)-Phi_n(i-1,j-1))/Delta_xy;
        DPhi_pX=(Phi_n(i,j)-Phi_n(i,j-1))/Delta_xy;
        DPhi_mY=0;
        DPhi_pY=(Phi_n(i+1,j)-Phi_n(i,j))/Delta_xy;
      elseif (i==1 && j==1) || (i==1 && j==N+1) || (j==1 && i==1) || (j==N+1 && i==1)
        DPhi_mX=(Phi_n(i,j)-Phi_n(i-1,j+1))/Delta_xy;
        DPhi_mY=0;
        DPhi_pX=(Phi_n(i,j)-Phi_n(i,j-1))/Delta_xy;
        DPhi_pY=(Phi_n(i+1,j)-Phi_n(i,j))/Delta_xy;
      end
    end
  end
end
\[ \text{DPhi}_pY = (\Phi_{n(i+1,j)} - \Phi_{n(i,j)}) / \Delta_{xy}; \]
\[ \text{elseif } i==1 && j==N+1 \]
\[ \text{DPhi}_mX = (\Phi_{n(i,j)} - \Phi_{n(i,j-1)}) / \Delta_{xy}; \]
\[ \text{DPhi}_mY = 0; \]
\[ \text{DPhi}_pX = 0; \]
\[ \text{DPhi}_pY = (\Phi_{n(i+1,j)} - \Phi_{n(i,j)}) / \Delta_{xy}; \]
\[ \text{elseif } i<N+1 && j==1 \]
\[ \text{DPhi}_mX = 0; \]
\[ \text{DPhi}_pX = (\Phi_{n(i,j+1)} - \Phi_{n(i,j)}) / \Delta_{xy}; \]
\[ \text{DPhi}_mY = (\Phi_{n(i,j)} - \Phi_{n(i-1,j)}) / \Delta_{xy}; \]
\[ \text{DPhi}_pY = (\Phi_{n(i+1,j)} - \Phi_{n(i,j)}) / \Delta_{xy}; \]
\[ \text{elseif } i<N+1 && j==N+1 \]
\[ \text{DPhi}_mX = (\Phi_{n(i,j)} - \Phi_{n(i,j-1)}) / \Delta_{xy}; \]
\[ \text{DPhi}_pX = 0; \]
\[ \text{DPhi}_mY = (\Phi_{n(i,j)} - \Phi_{n(i-1,j)}) / \Delta_{xy}; \]
\[ \text{DPhi}_pY = 0; \]
\[ \text{elseif } i==N+1 && j==1 \]
\[ \text{DPhi}_mX = 0; \]
\[ \text{DPhi}_pX = (\Phi_{n(i,j+1)} - \Phi_{n(i,j)}) / \Delta_{xy}; \]
\[ \text{DPhi}_mY = (\Phi_{n(i,j)} - \Phi_{n(i-1,j)}) / \Delta_{xy}; \]
\[ \text{DPhi}_pY = 0; \]
\[ \text{elseif } i==N+1 && j<N+1 \]
\[ \text{DPhi}_mX = (\Phi_{n(i,j)} - \Phi_{n(i,j-1)}) / \Delta_{xy}; \]
\[ \text{DPhi}_pX = (\Phi_{n(i,j+1)} - \Phi_{n(i,j)}) / \Delta_{xy}; \]
\[ \text{DPhi}_mY = (\Phi_{n(i,j)} - \Phi_{n(i-1,j)}) / \Delta_{xy}; \]
\[ \text{DPhi}_pY = 0; \]
\[ \text{elseif } i==N+1 && j==N+1 \]
\[ \text{DPhi}_mX = (\Phi_{n(i,j)} - \Phi_{n(i,j-1)}) / \Delta_{xy}; \]
\[ \text{DPhi}_pX = 0; \]
\[ \text{DPhi}_mY = (\Phi_{n(i,j)} - \Phi_{n(i-1,j)}) / \Delta_{xy}; \]
\[ \text{DPhi}_pY = 0; \]
\[ \text{else} \]
\[ \text{DPhi}_mX = (\Phi_{n(i,j)} - \Phi_{n(i,j-1)}) / \Delta_{xy}; \]
\[ \text{DPhi}_pX = (\Phi_{n(i,j+1)} - \Phi_{n(i,j)}) / \Delta_{xy}; \]
\[ \text{DPhi}_mY = (\Phi_{n(i,j)} - \Phi_{n(i-1,j)}) / \Delta_{xy}; \]
\[ \text{DPhi}_pY = (\Phi_{n(i+1,j)} - \Phi_{n(i,j)}) / \Delta_{xy}; \]
\[ \text{end} \]

\[ \text{Nabla}_p = (\text{max}(\text{DPhi}_mX,0)^2 + \text{min}(\text{DPhi}_pX,0)^2 + \text{max}(\text{DPhi}_mY,0)^2 + \text{min}(\text{DPhi}_pY,0)^2)^{1/2}; \]

\[ \text{Nabla}_m = (\text{min}(\text{DPhi}_mX,0)^2 + \text{max}(\text{DPhi}_pX,0)^2 + \text{min}(\text{DPhi}_mY,0)^2 + \text{max}(\text{DPhi}_pY,0)^2)^{1/2}; \]

\[ \text{Phi}_\text{next}(i,j) = \Phi_n(i,j) - \Delta_t \cdot (\text{max}(V,0) \cdot \text{Nabla}_p \cdot \text{min}(V,0) \cdot \text{Nabla}_m); \]

\[ \text{end} \]

\[ \text{end} \]

**A.1.2.10. interface_tracker_in.m**

```matlab
function \[Cin\] = interface_tracker_in(X, Y, Phi, Rcase)
% Interface tracker2: this function calculates the points of the interface
% given the cartesian grid and the Phi matrix.
% Detailed explanation goes here
```
\[
dx = \text{diff}(Y(1:2,1));
\]
\[
\text{Delta}_x = dx;
\]
\[
N = \text{size}(X,1);
\]
\[
\epsilon = 1e-6;
\]

%%
% First step: find a interface point and set it as the first point of the curve:
% Vertical scan from the y=-L/2 and x=0 to y=L/2 and x=0.
% \text{j} = \text{floor}((N+1)/2);
\begin{verbatim}
for i=1:1:N-1
    Phi1=Phi(i,j);
    Phi2=Phi(i+1,j);
    Phi3=Phi(i+1,j+1);
    Phi4=Phi(i,j+1);
    X1=X(i,j);
    Y1=Y(i,j);
    X2=X(i+1,j);
    Y2=Y(i+1,j);
    X3=X(i+1,j+1);
    Y3=Y(i+1,j+1);
    X4=X(i,j+1);
    Y4=Y(i,j+1);
    if Phi1*Phi4<0 && Phi2*Phi3<0
        \text{CASE 1}
        Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_x;
        XB_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_x;
        Ya_0=Y1;
        YB_0=Y2;
        i_0=i;
        j_0=j;
        case_id=1;
        break;
    elseif Phi1*Phi2<0 && Phi1*Phi4<0
        \text{CASE 2}
        Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_x;
        XB_0=X1;
        Ya_0=Y1;
        YB_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_x;
        i_0=i;
        j_0=j;
        case_id=2;
        break;
    elseif Phi1*Phi4<0 && Phi3*Phi4<0
        \text{CASE 3}
        Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_x;
        XB_0=X4;
        Ya_0=Y1;
        YB_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_x;
        i_0=i;
        j_0=j;
        case_id=3;
        break;
    elseif Phi1*Phi2<0 && Phi3*Phi4<0
        \text{CASE 4}
        Xa_0=X1;
        XB_0=X4;
        Ya_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_x;
        YB_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_x;
        i_0=i;
        j_0=j;
        case_id=4;
    \end{verbatim}
break;
elseif Phi1*Phi2<0 && Phi2*Phi3<0  %CASE 5
  Xa_0=X1;
  Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
  Ya_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
  Yb_0=Y2;
  i_0=i;
  j_0=j_cc;
  case_id=5;
  break;
elseif Phi2*Phi3<0 && Phi3*Phi4<0  %CASE 6
  Xa_0=X4;
  Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
  Ya_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi2)))*Delta_xy;
  Yb_0=Y2;
  i_0=i;
  j_0=j_cc;
  case_id=6;
  break;
else
  Xa_0=0;
  Xb_0=0;
  Ya_0=0;
  Yb_0=0;
  case_id=0;
end
end

%%
%If the vertical scan does not find the interface, then a 
%horizontal scan 
%shall be done:
if case_id==0
  i_cc=floor((N+1)/2);
  for j=1:i:N-1
    Phi1=Phi(i_cc,j);
    Phi2=Phi(i_cc+1,j);
    Phi3=Phi(i_cc+1,j+1);
    Phi4=Phi(i_cc,j+1);
    X1=X(i_cc,j);
    Y1=Y(i_cc,j);
    X2=X(i_cc+1,j);
    Y2=Y(i_cc+1,j);
    X3=X(i_cc+1,j+1);
    Y3=Y(i_cc+1,j+1);
    X4=X(i_cc,j+1);
    Y4=Y(i_cc,j+1);
    if Phi1*Phi4<0 && Phi2*Phi3<0  %CASE 1
      Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
      Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
      Ya_0=Y1;
      Yb_0=Y2;
      i_0=i_cc;
      j_0=j;
      case_id=1;
      break;
    elseif Phi1*Phi2<0 && Phi1*Phi4<0  %CASE 2
      Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
      Xb_0=X1;
      Ya_0=Y1;
      Yb_0=Y2;
      i_0=i_cc;
      j_0=j;
      case_id=1;
      break;
    elseif Phi2*Phi4<0 && Phi2*Phi3<0  %CASE 3
      Xa_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
      Xb_0=X1;
      Ya_0=Y1;
      Yb_0=Y2;
      i_0=i_cc;
      j_0=j;
      case_id=1;
      break;
    elseif Phi3*Phi4<0 && Phi3*Phi2<0  %CASE 4
      Xa_0=X2+(abs(Phi3)/(abs(Phi3)+abs(Phi4)))*Delta_xy;
      Xb_0=X1;
      Ya_0=Y1;
      Yb_0=Y2;
      i_0=i_cc;
      j_0=j;
      case_id=1;
      break;
    else
...
Ya_0=Y1;  
Yb_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;  
i_0=i_cc;  
j_0=j;  
case_id=2;  
break;  
elseif Phi1*Phi4<0 && Phi3*Phi4<0  
  \%CASE 3  
  Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;  
  Xb_0=X4;  
  Ya_0=Y1;  
  Yb_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;  
i_0=i_cc;  
j_0=j;  
case_id=3;  
break;  
elseif Phi1*Phi2<0 && Phi3*Phi4<0  
  \%CASE 4  
  Xa_0=X1;  
  Xb_0=X4;  
  Ya_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;  
  Yb_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;  
i_0=i_cc;  
j_0=j;  
case_id=4;  
break;  
elseif Phi1*Phi2<0 && Phi2*Phi3<0  
  \%CASE 5  
  Xa_0=X1;  
  Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;  
  Ya_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;  
  Yb_0=Y2;  
i_0=i_cc;  
j_0=j;  
case_id=5;  
break;  
elseif Phi2*Phi3<0 && Phi3*Phi4<0  
  \%CASE 6  
  Xa_0=X4;  
  Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;  
  Ya_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi2)))*Delta_xy;  
  Yb_0=Y2;  
i_0=i_cc;  
j_0=j;  
case_id=6;  
break;  
else  
  Xa_0=0;  
  Xb_0=0;  
  Ya_0=0;  
  Yb_0=0;  
case_id=-1;  
end  
end  
end  

%%  
%If the vertical and horizontal scan do not find the interface, then a  
%overall scan shall be done:  
if case_id==-1  
  for i=1:1:N-1 \%y-direction loop  
      for j=1:1:N-1 \%x-direction loop  
          
  end  
  end  
end
\begin{verbatim}
Phi1=Phi(i,j);
Phi2=Phi(i+1,j);
Phi3=Phi(i+1,j+1);
Phi4=Phi(i,j+1);
X1=X(i,j);
Y1=Y(i,j);
X2=X(i+1,j);
Y2=Y(i+1,j);
X3=X(i+1,j+1);
Y3=Y(i+1,j+1);
X4=X(i,j+1);
Y4=Y(i,j+1);

if Phi1*Phi4<0 && Phi2*Phi3<0 %CASE 1
    Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
    Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
    Ya_0=Y1;
    Yb_0=Y2;
    i_0=i;
    j_0=j;
    case_id=1;
    break;
elseif Phi1*Phi2<0 && Phi1*Phi4<0 %CASE 2
    Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
    Xb_0=X1;
    Ya_0=Y1;
    Yb_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
    i_0=i;
    j_0=j;
    case_id=2;
    break;
elseif Phi1*Phi4<0 && Phi3*Phi4<0 %CASE 3
    Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
    Xb_0=X4;
    Ya_0=Y1;
    Yb_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
    i_0=i;
    j_0=j;
    case_id=3;
    break;
elseif Phi1*Phi2<0 && Phi3*Phi4<0 %CASE 4
    Xa_0=X1;
    Xb_0=X4;
    Ya_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
    Yb_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
    i_0=i;
    j_0=j;
    case_id=4;
    break;
elseif Phi1*Phi2<0 && Phi2*Phi3<0 %CASE 5
    Xa_0=X1;
    Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
    Ya_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
    Yb_0=Y2;
    i_0=i;
    j_0=j;
    case_id=5;
    break;
elseif Phi2*Phi3<0 && Phi3*Phi4<0 %CASE 6
    Xa_0=X4;
\end{verbatim}
Study of Grain Burnback and Performance of Solid Rocket Motors

\[ Xb_0 = X_2 + \frac{(\text{abs}(\Phi_2) / (\text{abs}(\Phi_2) + \text{abs}(\Phi_3))) \cdot \Delta_{xy}}{}; \]
\[ Ya_0 = Y_4 + \frac{(\text{abs}(\Phi_4) / (\text{abs}(\Phi_4) + \text{abs}(\Phi_2))) \cdot \Delta_{xy}}{}; \]
\[ Yb_0 = Y_2; \]
\[ i_0 = i; \]
\[ j_0 = j; \]
\[ \text{case_id} = 6; \]
\[ \text{break}; \]
\[ \text{else} \]
\[ Xa_0 = 0; \]
\[ Xb_0 = 0; \]
\[ Ya_0 = 0; \]
\[ Yb_0 = 0; \]
\[ i_0 = i; \]
\[ j_0 = j; \]
\[ \text{case_id} = 0; \]
\[ \text{end} \]
\[ \text{end} \]
\[ \text{end} \]

%%
% INTERFACE SCANNING AND DETERMINATION
closed = 0;
k = 1;
l = 2;
while closed == 0  % The condition closed means the closing of the
% interface curve over itself.
% When closed is equal to 1 then the curve has
% the while loop can be stopped.
% Assignment of the intial point of the interface curve.
if k == 1
i_x = i_0;
j_x = j_0;
switch case_id
\[ \text{case} \{1, 3\} \]
case_id = 1;
C(1, 1) = Xa_0;
C(1, 2) = Ya_0;

\[ \text{case} \{4, 5\} \]
case_id = 2;
C(1, 1) = Xa_0;
C(1, 2) = Ya_0;

\[ \text{case} 2 \]
C(1, 1) = Xb_0;
C(1, 2) = Yb_0;

\[ \text{case} 6 \]
C(1, 1) = Xb_0;
C(1, 2) = Yb_0;

\[ \text{otherwise} \]
\[ \text{case_id} = 0; \]
C(1, 1) = 0;
C(1, 2) = 0;
end
C2(1,1)=C(1,1);
C2(1,2)=C(1,2);
C2(1,3)=0;
end

%Assignment of the A point as the B point calculated in the previous iteration (A(k)=B(k-1)).
Xa=C(k,1);
Ya=C(k,2);
i=i_x;
%Node 1 i component of the square to be scanned.
j=j_x;
%Node 1 j component of the square to be scanned.
X1=X(i,j);
%X position of the Node 1 of the square to be scanned.
Y1=Y(i,j);
%Y position of the Node 1 of the square to be scanned.
X2=X(i+1,j);
%X position of the Node 2 of the square to be scanned.
Y2=Y(i+1,j);
%Y position of the Node 2 of the square to be scanned.
X3=X(i+1,j+1);
%X position of the Node 3 of the square to be scanned.
Y3=Y(i+1,j+1);
%Y position of the Node 3 of the square to be scanned.
X4=X(i,j+1);
%X position of the Node 4 of the square to be scanned.
Y4=Y(i,j+1);
%Y position of the Node 4 of the square to be scanned.
Phi1=Phi(i,j);
%Phi value at the Node 1 of the square to be scanned.
Phi2=Phi(i+1,j);
%Phi value at the Node 2 of the square to be scanned.
Phi3=Phi(i+1,j+1);
%Phi value at the Node 3 of the square to be scanned.
Phi4=Phi(i,j+1);
%Phi value at the Node 4 of the square to be scanned.

%Identification of a square which has the interface passing through it twice:
if Phi1*Phi2<0 && Phi2*Phi3<0 && Phi3*Phi4<0 && Phi1*Phi4<0
%Condition of interface double crossing the square.
V=C(k,:)-C(k-1,:); %Vector of the interface ongoing direction in the k point.
switch case_id_A %Location of the square edge where the A point is.
case 1 %A is located in the lower edge of the square.
Xd_2=X1;
Yd_2=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
Xd_4=X4;
Yd_4=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
a=([Xd_2-Xa] (Yd_2-Ya));
b=([Xd_4-Xa] (Yd_4-Ya));
Va=dot(V,a);
Vb=dot(V,b);            %B is located in the left edge of
if Va>Vb                 %Location of the next iteration
    Xb=Xd_2;            point A (4=right edge of the square).
    Yb=Yd_2;            i_x=i;                 %Node 1 i component of the square
    case_v=4;          to be scanned in the next step.  
    j_x=j-1;           %Node 1 j component of the square
to be scanned in the next step.  
else                     %B is located in the right edge of
    Xb=Xd_4;            %Location of the next iteration
    Yb=Yd_4;            point A (2=left edge of the square).
    case_v=2;          i_x=i;                 %Node 1 i component of the square
    %Location of the next iteration
to be scanned in the next step.  
    j_x=j+1;           %Node 1 j component of the square
to be scanned in the next step.
end

case 2  %A is located in the left edge of the square. 
    Xd_1=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
    Yd_1=Y1;
    Xd_3=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
    Yd_3=Y2;
    a=[(Xd_1-Xa) (Yd_1-Ya)];
    b=[(Xd_3-Xa) (Yd_3-Ya)];
    Va=dot(V,a);
    Vb=dot(V,b);
    if Va>Vb               %B is located in the lower edge of
        Xb=Xd_1;            %Location of the next iteration
        Yb=Yd_1;            point A (3=upper edge of the square).
        case_v=3;          i_x=i-1;                 %Node 1 i component of the square
        %Location of the next iteration
to be scanned in the next step.  
        j_x=j;            %Node 1 j component of the square
to be scanned in the next step.  
else
    Xb=Xd_3;            %Location of the next iteration
    Yb=Yd_3;            point A (1=lower edge of the square).
    case_v=1;          i_x=i+1;                 %Node 1 i component of the square
to be scanned in the next step.  
    j_x=j;            %Node 1 j component of the square
to be scanned in the next step.
end

case 3  %A is located in the upper edge of the square. 
    Xd_2=X1;
    Yd_2=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
    Xd_4=X4;
    Yd_4=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
    a=[(Xd_2-Xa) (Yd_2-Ya)];
    b=[(Xd_4-Xa) (Yd_4-Ya)];
Study of Grain Burnback and Performance of Solid Rocket Motors

\[ \text{Va} = \text{dot}(V, a); \]
\[ \text{Vb} = \text{dot}(V, b); \]
\[ \text{if Va} > \text{Vb} \]
\[ \text{Xb} = X_{d, 2}; \]
\[ \text{Yb} = Y_{d, 2}; \]
\[ \text{case} \_v = 4; \]
\[ \text{if} \]
\[ \text{Va} > \text{Vb} \]
\[ \% \text{B is located in the left edge of the square.} \]
\[ \text{Xb} = X_{d, 2}; \]
\[ \text{Yb} = Y_{d, 2}; \]
\[ \text{case} \_v = 4; \]
\[ \% \text{Location of the next iteration point A (4=right edge of the square).} \]
\[ \text{i} = i; \]
\[ \text{j} = j - 1; \]
\[ \% \text{Node 1 i component of the square to be scanned in the next step.} \]
\[ \% \text{Node 1 j component of the square to be scanned in the next step.} \]
\[ \text{else} \]
\[ \% \text{B is located in the right edge of the square.} \]
\[ \text{Xb} = X_{d, 4}; \]
\[ \text{Yb} = Y_{d, 4}; \]
\[ \text{case} \_v = 2; \]
\[ \% \text{Location of the next iteration point A (2=left edge of the square).} \]
\[ \text{i} = i; \]
\[ \text{j} = j + 1; \]
\[ \% \text{Node 1 i component of the square to be scanned in the next step.} \]
\[ \% \text{Node 1 j component of the square to be scanned in the next step.} \]
\[ \text{end} \]
\[ \text{case 4} \% \text{A is located in the right edge of the square.} \]
\[ X_{d, 1} = X_1 + (\text{abs}(\text{Phi}1)/(\text{abs}(\text{Phi}1) + \text{abs}(\text{Phi}4))) \times \Delta_{xy}; \]
\[ Y_{d, 1} = Y_1; \]
\[ X_{d, 3} = X_2 + (\text{abs}(\text{Phi}2)/(\text{abs}(\text{Phi}2) + \text{abs}(\text{Phi}3))) \times \Delta_{xy}; \]
\[ Y_{d, 3} = Y_2; \]
\[ a = \{(X_{d, 1} - Xa) \} \{Y_{d, 1} - Ya\}; \]
\[ b = \{(X_{d, 3} - Xa) \} \{Y_{d, 3} - Ya\}; \]
\[ \text{Va} = \text{dot}(V, a); \]
\[ \text{Vb} = \text{dot}(V, b); \]
\[ \text{if Va} > \text{Vb} \]
\[ \% \text{B is located in the lower edge of the square.} \]
\[ \text{i} = i - 1; \]
\[ \text{j} = j; \]
\[ \% \text{Node 1 i component of the square to be scanned in the next step.} \]
\[ \% \text{Node 1 j component of the square to be scanned in the next step.} \]
\[ \text{else} \]
\[ \% \text{B is located in the upper edge of the square.} \]
\[ \text{i} = i + 1; \]
\[ \text{j} = j; \]
\[ \% \text{Node 1 i component of the square to be scanned in the next step.} \]
\[ \% \text{Node 1 j component of the square to be scanned in the next step.} \]
\[ \text{end} \]
\[ \text{otherwise} \]
\[ \text{Xb} = 0; \]
\[ \text{Yb} = 0; \]
\[ \text{case} \_v = 1; \]
\[ \% \text{Location of the next iteration point A (1=lower edge of the square).} \]

\[ \text{Va} = \text{dot}(V, a); \]
\[ \text{Vb} = \text{dot}(V, b); \]
\[ \text{if Va} > \text{Vb} \]
\[ \% \text{B is located in the lower edge of the square.} \]
\[ \text{i} = i - 1; \]
\[ \text{j} = j; \]
\[ \% \text{Node 1 i component of the square to be scanned in the next step.} \]
\[ \% \text{Node 1 j component of the square to be scanned in the next step.} \]
\[ \text{else} \]
\[ \% \text{B is located in the upper edge of the square.} \]
\[ \text{i} = i + 1; \]
\[ \text{j} = j; \]
\[ \% \text{Node 1 i component of the square to be scanned in the next step.} \]
\[ \% \text{Node 1 j component of the square to be scanned in the next step.} \]
\[ \text{end} \]
\[ \text{otherwise} \]
\[ \text{Xb} = 0; \]
\[ \text{Yb} = 0; \]
\[ \text{case} \_v = 1; \]
\[ \% \text{Location of the next iteration point A (-1=error).} \]
i_x=0; % Node 1 i component of the square to be scanned in the next step (0=error).
j_x=0; % Node 1 j component of the square to be scanned in the next step (0=error).

else % The interface crosses the square just once.

    switch case_id_A % Location of the square edge where the A point is.
        case 1 % A is located in the lower edge of the square.
            if Phi1*Phi2<0 % B is located in the left edge of the square.
                Xb=X1;
                Yb=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
                case_v=4; % Location of the next iteration point A (4=right edge of the square).
                i_x=i; % Node 1 i component of the square to be scanned in the next step.
                j_x=j-1; % Node 1 j component of the square to be scanned in the next step.
            elseif Phi2*Phi3<0 % B is located in the upper edge of the square.
                Xb=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
                Yb=Y2;
                case_v=1; % Location of the next iteration point A (1=lower edge of the square).
                i_x=i+1; % Node 1 i component of the square to be scanned in the next step.
                j_x=j; % Node 1 j component of the square to be scanned in the next step.
            elseif Phi3*Phi4<0 % B is located in the right edge of the square.
                Xb=X4;
                Yb=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
                case_v=2; % Location of the next iteration point A (2=left edge of the square).
                i_x=i-1; % Node 1 i component of the square to be scanned in the next step.
                j_x=j+1; % Node 1 j component of the square to be scanned in the next step.
            end
        case 2 % A is located in the lower edge of the square.
            if Phi1*Phi4<0 % B is located in the lower edge of the square.
                Xb=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
                Yb=Y1;
                case_v=3; % Location of the next iteration point A (3=upper edge of the square).
                i_x=i-1; % Node 1 i component of the square to be scanned in the next step.
                j_x=j; % Node 1 j component of the square to be scanned in the next step.
            elseif Phi2*Phi3<0 % B is located in the upper edge of the square.
                Xb=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
                Yb=Y2;

        % The interface crosses the square just once.
    end

end
case_v=1;  %Location of the next iteration point A (1=lower edge of the square).
  i_x=i+1;  %Node 1 i component of the square to be scanned in the next step.
  j_x=j;  %Node 1 j component of the square to be scanned in the next step.
elseif Phi3*Phi4<0  %B is located in the right edge of the square.
  Xb=X4;
  Yb=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
  case_v=2;  %Location of the next iteration point A (2=left edge of the square).
  i_x=i;  %Node 1 i component of the square to be scanned in the next step.
  j_x=j+1;  %Node 1 j component of the square to be scanned in the next step.
end

  case 3  %A is located in the upper edge of the square.
      if Phi1*Phi4<0  %B is located in the lower edge of the square.
         Xb=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
         Yb=Y1;
         case_v=3;  %Location of the next iteration point A (3=upper edge of the square).
         i_x=i-1;  %Node 1 i component of the square to be scanned in the next step.
         j_x=j;  %Node 1 j component of the square to be scanned in the next step.
      elseif Phi1*Phi2<0  %B is located in the left edge of the square.
         Xb=X1;
         Yb=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
         case_v=4;  %Location of the next iteration point A (4=right edge of the square).
         i_x=i-1;  %Node 1 i component of the square to be scanned in the next step.
         j_x=j-1;  %Node 1 j component of the square to be scanned in the next step.
      elseif Phi3*Phi4<0  %B is located in the right edge of the square.
         Xb=X4;
         Yb=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
         case_v=2;  %Location of the next iteration point A (2=left edge of the square).
         i_x=i;  %Node 1 i component of the square to be scanned in the next step.
         j_x=j+1;  %Node 1 j component of the square to be scanned in the next step.
end

  case 4  %A is located in the right edge of the square.
      if Phi1*Phi4<0  %B is located in the lower edge of the square.
         Xb=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
         Yb=Y1;
         case_v=3;  %Location of the next iteration point A (3=upper edge of the square).
         i_x=i-1;  %Node 1 i component of the square to be scanned in the next step.
Study of Grain Burnback and Performance of Solid Rocket Motors

j_x=j; %Node 1 j component of the square to be scanned in the next step.

elseif Phi1*Phi2<0 %B is located in the left edge of the square.
    Xb=X1;
    Yb=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
    case_v=4; %Location of the next iteration point A (4=right edge of the square).
    i_x=i; %Node 1 i component of the square to be scanned in the next step.
    j_x=j-1; %Node 1 j component of the square to be scanned in the next step.
end

elseif Phi2*Phi3<0 %B is located in the upper edge of the square.
    Xb=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
    Yb=Y2;
    case_v=1; %Location of the next iteration point A (1=lower edge of the square).
    i_x=i+1; %Node 1 i component of the square to be scanned in the next step.
    j_x=j; %Node 1 j component of the square to be scanned in the next step.
end

otherwise
    Xb=0;
    Yb=0;
    case_v=-1; %Location of the next iteration point A (-1=error).
    i_x=0; %Node 1 i component of the square to be scanned in the next step (0=error).
    j_x=0; %Node 1 j component of the square to be scanned in the next step (0=error).
end

%Assignment of the B point to the interface curve.
C(k+1,1)=Xb;
C(k+1,2)=Yb;
case_id_A=case_v; %Assignment of the case_id_A for the next iteration.
dif=sqrt((Xb-C(1,1))^2+(Yb-C(1,2))^2);

%Assignment of only the points inside or frontier of the motor case:
Ra=sqrt(Xa^2+Ya^2);
Rb=sqrt(Xb^2+Yb^2);
if Ra<Rcase && Rb<Rcase %The interface is inside the motor case.
    C2(l,1)=C(k+1,1);
    C2(l,2)=C(k+1,2);
    C2(l,3)=0;
    l=l+1;
elseif Ra<Rcase && Rb>Rcase || Ra>Rcase && Rb<Rcase %The interface is in the frontier of the motor case.
a=(Xb-Xa)^2+(Yb-Ya)^2;
b=2*Xa*(Xb-Xa)+2*Ya*(Yb-Ya);
c=Xa^2+Ya^2-Rcase^2;
Disc=b^2-4*a*c;
  if Disc>=0
    s1=(-b+sqrt(Disc))/(2*a);
    s2=(-b-sqrt(Disc))/(2*a);
    if s1>=0 && s1<=1
      s=s1;
      x=Xa+(Xb-Xa)*s;
      y=Ya+(Yb-Ya)*s;
    elseif s2>=0 && s2<=1
      s=s2;
      x=Xa+(Xb-Xa)*s;
      y=Ya+(Yb-Ya)*s;
    else
      x=0;
      y=0;
    end
  C2(l,1)=x;
  C2(l,2)=y;
  C2(l,3)=1;  \%Identification of the intersection point of
  the interface with the motor case.
  l=l+1;
end

if dif==0
  closed=1;
elseif dif<epsilon
  closed=2;
else
  closed=0;
end
k=k+1;
end

%Check whether the first point of C2 is inside the motor case or not:
if norm(C2(1,:))>Rcase
  C2(1,:)=[];
end

%%
%Generation of the interface curve accounting only the points inside
%motor case:
k=1;
Na=100;  \%Number of nodes of the segment in the motor case
between interface curve escape and return.
for i=1:size(C2,1)
  if i<size(C2,1) && C2(i,3)==1 && C2(i+1,3)==1
    Theta_max=atan2(C2(i+1,2),C2(i+1,1));
    Theta_min=atan2(C2(i,2),C2(i,1));
    r_max=[C2(i+1,1)  C2(i+1,2) 0];
    r_min=[C2(i,1)  C2(i,2) 0];
if cross(r_min,r_max)*[0 0 1]'>0 %The interface goes in counter clockwise sense.
    if Theta_max<Theta_min
        Theta=2*pi-abs(Theta_max)-abs(Theta_min);
    else
        Theta=abs(Theta_max-Theta_min);
    end
else %The interface goes in clockwise sense.
    if Theta_max>Theta_min
        Theta=abs(Theta_max)+abs(Theta_min)-2*pi;
    else
        Theta=Theta_max-Theta_min;
    end
end
Delta_theta=Theta/Na;
for j=1:1:Na+1
    Theta_j=Theta_min+(j-1)*Delta_theta;
    Cin(k,1)=Rcase*cos(Theta_j);
    Cin(k,2)=Rcase*sin(Theta_j);
    k=k+1;
end
elseif i==size(C2,1) && C2(i,3)==1 && C2(1,3)==1
    Theta_max=atan2(C2(1,2),C2(1,1));
    Theta_min=atan2(C2(i,2),C2(i,1));
    r_max=[C2(1,1) C2(1,2) 0];
    r_min=[C2(i,1) C2(i,2) 0];
    if cross(r_min,r_max)*[0 0 1]'>0 %The interface goes in counter clockwise sense.
        if Theta_max<Theta_min
            Theta=2*pi-abs(Theta_max)-abs(Theta_min);
        else
            Theta=abs(Theta_max-Theta_min);
        end
    else %The interface goes in clockwise sense.
        if Theta_max>Theta_min
            Theta=abs(Theta_max)+abs(Theta_min)-2*pi;
        else
            Theta=Theta_max-Theta_min;
        end
    end
    Delta_theta=Theta/Na;
    for j=1:1:Na+1
        Theta_j=Theta_min+(j-1)*Delta_theta;
        Cin(k,1)=Rcase*cos(Theta_j);
        Cin(k,2)=Rcase*sin(Theta_j);
        k=k+1;
    end
    Cin(k,1)=C2(i,1);
    Cin(k,2)=C2(i,2);
    k=k+1;
end
end
A.1.2.11. interface_tracker.m

function [C] = interface_tracker(X,Y,Phi)
%Interface follower: this function calculates the points of the interface
%given the cartesian grid and the Phi matrix.
% Detailed explanation goes here

%%
dx=diff(Y(1:2,1));
Delta_xy=dx;
N=size(X,1);
epsilon=1e-6;

%%
%First step: find a interface point and set it as the first point of the curve:
%Vertical scan from the y=-l/2 and x=0 to y=L/2 and x=0.
j_cc=floor((N+1)/2);
for i=1:1:N-1
    Phi1=Phi(i,j_cc);
    Phi2=Phi(i+1,j_cc);
    Phi3=Phi(i+1,j_cc+1);
    Phi4=Phi(i,j_cc+1);
    X1=X(i,j_cc);
    Y1=Y(i,j_cc);
    X2=X(i+1,j_cc);
    Y2=Y(i+1,j_cc);
    X3=X(i+1,j_cc+1);
    Y3=Y(i+1,j_cc+1);
    X4=X(i,j_cc+1);
    Y4=Y(i,j_cc+1);
    if Phi1*Phi4<0 && Phi2*Phi3<0
        %CASE 1
        Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
        Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
        Ya_0=Y1;
        Yb_0=Y2;
        i_0=i;
        j_0=j_cc;
        case_id=1;
        break;
    elseif Phi1*Phi2<0 && Phi1*Phi4<0
        %CASE 2
        Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
        Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
        Ya_0=Y1;
        Yb_0=Y2;
        i_0=i;
        j_0=j_cc;
        case_id=2;
        break;
    elseif Phi1*Phi4<0 && Phi3*Phi4<0
        %CASE 3
        Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
        Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
        Ya_0=Y1;
        Yb_0=Y2;
        i_0=i;
        j_0=j_cc;
        case_id=3;
        break;
    end
end
break;
elseif Phi1*Phi2<0 && Phi3*Phi4<0  %CASE 4
Xa_0=X1;
Xb_0=X4;
Ya_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
Yb_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
i_0=i;
j_0=j_cc;
case_id=4;
break;
elseif Phi1*Phi2<0 && Phi2*Phi3<0  %CASE 5
Xa_0=X1;
Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
Ya_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
Yb_0=Y2;
i_0=i;
j_0=j_cc;
case_id=5;
break;
elseif Phi2*Phi3<0 && Phi3*Phi4<0  %CASE 6
Xa_0=X4;
Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
Ya_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi2)))*Delta_xy;
Yb_0=Y2;
i_0=i;
j_0=j_cc;
case_id=6;
break;
else
Xa_0=0;
Xb_0=0;
Ya_0=0;
Yb_0=0;
case_id=0;
end
end

%%
%If the vertical scan does not find the interface, then a
%horizontal scan
%shall be done:
if case_id==0
i_cc=floor((N+1)/2);
for j=1:1:N-1
    Phi1=Phi(i_cc,j);
    Phi2=Phi(i_cc+1,j);
    Phi3=Phi(i_cc+1,j+1);
    Phi4=Phi(i_cc,j+1);
    X1=X(i_cc,j);
    Y1=Y(i_cc,j);
    X2=X(i_cc+1,j);
    Y2=Y(i_cc+1,j);
    X3=X(i_cc+1,j+1);
    Y3=Y(i_cc+1,j+1);
    X4=X(i_cc,j+1);
    Y4=Y(i_cc,j+1);
    if Phi1*Phi4<0 && Phi2*Phi3<0  %CASE 1
        Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
        Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
    end
end
end
Ya_0=Y1;
Yb_0=Y2;
i_0=i_{cc};
j_0=j;
case_id=1;
break;
elseif \Phi_1*\Phi_2<0 \&\& \Phi_1*\Phi_4<0 \hspace{1em} %CASE 2
Xa_0=X1+(abs(\Phi_1)/(abs(\Phi_1)+abs(\Phi_4)))\times\Delta_{xy};
Xb_0=X1;
Ya_0=Y1;
Yb_0=Y1+(abs(\Phi_1)/(abs(\Phi_1)+abs(\Phi_2)))\times\Delta_{xy};
i_0=i_{cc};
j_0=j;
case_id=2;
break;
elseif \Phi_1*\Phi_4<0 \&\& \Phi_3*\Phi_4<0 \hspace{1em} %CASE 3
Xa_0=X1+(abs(\Phi_1)/(abs(\Phi_1)+abs(\Phi_4)))\times\Delta_{xy};
Xb_0=X4;
Ya_0=Y1;
Yb_0=Y4+(abs(\Phi_4)/(abs(\Phi_4)+abs(\Phi_3)))\times\Delta_{xy};
i_0=i_{cc};
j_0=j;
case_id=3;
break;
elseif \Phi_1*\Phi_2<0 \&\& \Phi_3*\Phi_4<0 \hspace{1em} %CASE 4
Xa_0=X1;
Xb_0=X4;
Ya_0=Y1+(abs(\Phi_1)/(abs(\Phi_1)+abs(\Phi_2)))\times\Delta_{xy};
Yb_0=Y4+(abs(\Phi_4)/(abs(\Phi_4)+abs(\Phi_3)))\times\Delta_{xy};
i_0=i_{cc};
j_0=j;
case_id=4;
break;
elseif \Phi_1*\Phi_2<0 \&\& \Phi_2*\Phi_3<0 \hspace{1em} %CASE 5
Xa_0=X1;
Xb_0=X2+(abs(\Phi_2)/(abs(\Phi_2)+abs(\Phi_3)))\times\Delta_{xy};
Ya_0=Y1+(abs(\Phi_1)/(abs(\Phi_1)+abs(\Phi_2)))\times\Delta_{xy};
Yb_0=Y2;
i_0=i_{cc};
j_0=j;
case_id=5;
break;
elseif \Phi_2*\Phi_3<0 \&\& \Phi_3*\Phi_4<0 \hspace{1em} %CASE 6
Xa_0=X4;
Xb_0=X2+(abs(\Phi_2)/(abs(\Phi_2)+abs(\Phi_3)))\times\Delta_{xy};
Ya_0=Y4+(abs(\Phi_4)/(abs(\Phi_4)+abs(\Phi_3)))\times\Delta_{xy};
Yb_0=Y2;
i_0=i_{cc};
j_0=j;
case_id=6;
break;
else
Xa_0=0;
Xb_0=0;
Ya_0=0;
Yb_0=0;
case_id=-1;
end
end
%% If the vertical and horizontal scan do not find the interface, then a
%% overall scan shall be done:
if case_id==-1
    for i=1:1:N-1
        for j=1:1:N-1
            Phi1=Phi(i,j);
            Phi2=Phi(i+1,j);
            Phi3=Phi(i+1,j+1);
            Phi4=Phi(i,j+1);
            X1=X(i,j);
            Y1=Y(i,j);
            X2=X(i+1,j);
            Y2=Y(i+1,j);
            X3=X(i+1,j+1);
            Y3=Y(i+1,j+1);
            X4=X(i,j+1);
            Y4=Y(i,j+1);
            if Phi1*Phi4<0 && Phi2*Phi3<0
                Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
                Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
                Ya_0=Y1;
                Yb_0=Y2;
                i_0=i;
                j_0=j;
                case_id=1;
                break;
            elseif Phi1*Phi2<0 && Phi1*Phi4<0
                Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
                Xb_0=X1;
                Ya_0=Y1;
                Yb_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
                i_0=i;
                j_0=j;
                case_id=2;
                break;
            elseif Phi1*Phi4<0 && Phi3*Phi4<0
                Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
                Xb_0=X4;
                Ya_0=Y1;
                Yb_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
                i_0=i;
                j_0=j;
                case_id=3;
                break;
            elseif Phi1*Phi2<0 && Phi3*Phi4<0
                Xa_0=X1;
                Xb_0=X4;
                Ya_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
                Yb_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
                i_0=i;
                j_0=j;
                case_id=4;
                break;
            elseif Phi1*Phi2<0 && Phi2*Phi3<0
                Xa_0=X1;
    end
Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
Ya_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
Yb_0=Y2;
i_0=i;
j_0=j;
case_id=5;
break;
elseif Phi2*Phi3<0 && Phi3*Phi4<0  %CASE 6
Xa_0=X4;
Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
Ya_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi2)))*Delta_xy;
Yb_0=Y2;
i_0=i;
j_0=j;
case_id=6;
break;
else
Xa_0=0;
Xb_0=0;
Ya_0=0;
Yb_0=0;
i_0=i;
j_0=j;
case_id=0;
end
end
end

%%
%INTERFACE SCANNING AND DETERMINATION
closed=0;
k=1;
while closed==0  %The condition closed means the closing of the
%interface curve over itself.
%When closed is equal to 1 then the curve has
%the while loop can be stopped.

%Assignment of the intial point of the interface curve.
if k==1
i_x=i_0;
j_x=j_0;
switch case_id
  case {1,3}
    case_id_A=1;
    C(1,1)=Xa_0;
    C(1,2)=Ya_0;
  case {4,5}
    case_id_A=2;
    C(1,1)=Xa_0;
    C(1,2)=Ya_0;
  case 2
    case_id_A=2;
    C(1,1)=Xb_0;
    C(1,2)=Yb_0;
  case 6
case_id_A=3;
C(1,1)=Xb_0;
C(1,2)=Yb_0;
otherwise
  case_id_A=0;
  C(1,1)=0;
  C(1,2)=0;
end
end

%Assignment of the A point as the point calculated in the
previous iteration (A(k)=B(k-1)).
Xa=C(k,1);
Ya=C(k,2);
i=i_x;
%Node 1 i component of the square to be
scanned.
j=j_x;
%Node 1 j component of the square to be
scanned.
X1=X(i,j);
%X position of the Node 1 of the square
to be scanned.
Y1=Y(i,j);
%Y position of the Node 1 of the square
to be scanned.
X2=X(i+1,j);
%X position of the Node 2 of the square
to be scanned.
Y2=Y(i+1,j);
%Y position of the Node 2 of the square
to be scanned.
X3=X(i+1,j+1);
%X position of the Node 3 of the square
to be scanned.
Y3=Y(i+1,j+1);
%Y position of the Node 3 of the square
to be scanned.
X4=X(i,j+1);
%X position of the Node 4 of the square
to be scanned.
Y4=Y(i,j+1);
%Y position of the Node 4 of the square
to be scanned.
Phi1=Phi(i,j);
%Phi value at the Node 1 of the square
to be scanned.
Phi2=Phi(i+1,j);
%Phi value at the Node 2 of the square
to be scanned.
Phi3=Phi(i+1,j+1);
%Phi value at the Node 3 of the square
to be scanned.
Phi4=Phi(i,j+1);
%Phi value at the Node 4 of the square
to be scanned.

%Identification of a square which has the interface pasing
through it
%twice:
if Phi1*Phi2<0 && Phi2*Phi3<0 && Phi3*Phi4<0 && Phi1*Phi4<0
%Condition of interface double crossing the square.
V=C(k,:)-C(k-1,:);  %Vector of the interface ongoing
direction in the k point.
switch case_id_A
  case 1  %A is located in the lower edge of the square.
    Xd_2=X1;
\begin{equation}
Yd_2 = Y1 + \frac{(\text{abs(Phi1)})}{(\text{abs(Phi1)} + \text{abs(Phi2)})}\Delta_{xy};
\end{equation}
\begin{equation}
Xd_4 = X4;
\end{equation}
\begin{equation}
Yd_4 = Y4 + \frac{(\text{abs(Phi4)})}{(\text{abs(Phi4)} + \text{abs(Phi3)})}\Delta_{xy};
\end{equation}
\begin{equation}
a = [(Xd_2 - Xa) \ (Yd_2 - Ya)];
\end{equation}
\begin{equation}
b = [(Xd_4 - Xa) \ (Yd_4 - Ya)];
\end{equation}
\begin{equation}
Va = \text{dot}(V, a);
\end{equation}
\begin{equation}
Vb = \text{dot}(V, b);
\end{equation}
\begin{equation}
\text{if } Va > Vb \quad \% \text{B is located in the left edge of the square.}
\end{equation}
\begin{equation}
Xb = Xd_2;
\end{equation}
\begin{equation}
Yb = Yd_2;
\end{equation}
\begin{equation}
\text{case}_v = 4; \quad \% \text{Location of the next iteration point A (4=right edge of the square).}
\end{equation}
\begin{equation}
i_x = i;
\end{equation}
\begin{equation}
j_x = j - 1; \quad \% \text{Node 1 i component of the square to be scanned in the next step.}
\end{equation}
\begin{equation}
j_x = j + 1; \quad \% \text{Node 1 j component of the square to be scanned in the next step.}
\end{equation}
\begin{equation}
\text{else} \quad \% \text{B is located in the right edge of the square.}
\end{equation}
\begin{equation}
Xb = Xd_4;
\end{equation}
\begin{equation}
Yb = Yd_4;
\end{equation}
\begin{equation}
\text{case}_v = 2; \quad \% \text{Location of the next iteration point A (2=left edge of the square).}
\end{equation}
\begin{equation}
i_x = i;
\end{equation}
\begin{equation}
j_x = j;
\end{equation}
\begin{equation}
\text{end}
\end{equation}
\begin{equation}
\text{case} 2 \quad \% \text{A is located in the left edge of the square.}
\end{equation}
\begin{equation}
Xd_1 = X1 + \frac{(\text{abs(Phi1)})}{(\text{abs(Phi1)} + \text{abs(Phi4)})}\Delta_{xy};
\end{equation}
\begin{equation}
Yd_1 = Y1;
\end{equation}
\begin{equation}
Xd_3 = X2 + \frac{(\text{abs(Phi2)})}{(\text{abs(Phi2)} + \text{abs(Phi3)})}\Delta_{xy};
\end{equation}
\begin{equation}
Yd_3 = Y2;
\end{equation}
\begin{equation}
a = [(Xd_1 - Xa) \ (Yd_1 - Ya)];
\end{equation}
\begin{equation}
b = [(Xd_3 - Xa) \ (Yd_3 - Ya)];
\end{equation}
\begin{equation}
Va = \text{dot}(V, a);
\end{equation}
\begin{equation}
Vb = \text{dot}(V, b);
\end{equation}
\begin{equation}
\text{if } Va > Vb \quad \% \text{B is located in the lower edge of the square.}
\end{equation}
\begin{equation}
Xb = Xd_1;
\end{equation}
\begin{equation}
Yb = Yd_1;
\end{equation}
\begin{equation}
\text{case}_v = 3; \quad \% \text{Location of the next iteration point A (3=upper edge of the square).}
\end{equation}
\begin{equation}
i_x = i - 1; \quad \% \text{Node 1 i component of the square to be scanned in the next step.}
\end{equation}
\begin{equation}
j_x = j; \quad \% \text{Node 1 j component of the square to be scanned in the next step.}
\end{equation}
\begin{equation}
\text{else} \quad \% \text{B is located in the upper edge of the square.}
\end{equation}
\begin{equation}
Xb = Xd_3;
\end{equation}
\begin{equation}
Yb = Yd_3;
\end{equation}
\begin{equation}
\text{case}_v = 1; \quad \% \text{Location of the next iteration point A (1=lower edge of the square).}
\end{equation}
\begin{equation}
i_x = i + 1; \quad \% \text{Node 1 i component of the square to be scanned in the next step.}
\end{equation}
\begin{equation}
j_x = j; \quad \% \text{Node 1 j component of the square to be scanned in the next step.}
\end{equation}
\begin{equation}
\text{end}
\end{equation}
\begin{equation}
\text{case} 3 \quad \% \text{A is located in the upper edge of the square.}
\[ Xd_2 = X_1; \]
\[ Yd_2 = Y_1 + \frac{(\text{abs}(\Phi_1)/(\text{abs}(\Phi_1)+\text{abs}(\Phi_2)))}{\Delta_{xy}}; \]
\[ Xd_4 = X_4; \]
\[ Yd_4 = Y_4 + \frac{(\text{abs}(\Phi_4)/(\text{abs}(\Phi_4)+\text{abs}(\Phi_3)))}{\Delta_{xy}}; \]
\[ a = \{(Xd_2 - X_a) (Yd_2 - Y_a)\}; \]
\[ b = \{(Xd_4 - X_a) (Yd_4 - Y_a)\}; \]
\[ Va = \text{dot}(V, a); \]
\[ Vb = \text{dot}(V, b); \]

\text{if Va > Vb} \\
\text{end} \\
% B is located in the left edge of the square.
\[ Xb = Xd_2; \]
\[ Yb = Yd_2; \]
% Location of the next iteration point A (4=right edge of the square).
\[ i_x = i; \]
% Node 1 i component of the square to be scanned in the next step.
\[ j_x = j - 1; \]
% Node 1 j component of the square to be scanned in the next step.
\text{end} \\
% A is located in the right edge of the square.
\[ Xd_1 = X_1 + \frac{(\text{abs}(\Phi_1)/(\text{abs}(\Phi_1)+\text{abs}(\Phi_4)))}{\Delta_{xy}}; \]
\[ Yd_1 = Y_1; \]
\[ Xd_3 = X_2 + \frac{(\text{abs}(\Phi_2)/(\text{abs}(\Phi_2)+\text{abs}(\Phi_3)))}{\Delta_{xy}}; \]
\[ Yd_3 = Y_2; \]
\[ a = \{(Xd_1 - X_a) (Yd_1 - Y_a)\}; \]
\[ b = \{(Xd_3 - X_a) (Yd_3 - Y_a)\}; \]
\[ Va = \text{dot}(V, a); \]
\[ Vb = \text{dot}(V, b); \]
\text{if Va > Vb} \\
% B is located in the lower edge of the square.
\[ Xb = Xd_1; \]
\[ Yb = Yd_1; \]
% Location of the next iteration point A (3=upper edge of the square).
\[ i_x = i - 1; \]
% Node 1 i component of the square to be scanned in the next step.
\[ j_x = j; \]
% Node 1 j component of the square to be scanned in the next step.
\text{end} \\
% A is located in the upper edge of the square.
\[ Xd_1 = X_1; \]
\[ Yd_1 = Y_1; \]
% Location of the next iteration point A (1=lower edge of the square).
\[ i_x = i + 1; \]
% Node 1 i component of the square to be scanned in the next step.
\[ j_x = j; \]
% Node 1 j component of the square to be scanned in the next step.
otherwise
    Xb=0;
    Yb=0;
    case_v=-1;  \%Location of the next iteration point A (-1=error).
    i_x=0;  \%Node 1 i component of the square to be scanned in the next step (0=error).
    j_x=0;  \%Node 1 j component of the square to be scanned in the next step (0=error).
end
else  \%The interface crosses the square just once.

switch case_id_A  \%Location of the square edge where the A point is.
    case 1  \%A is located in the lower edge of the square.
        if Phi1*Phi2<0  \%B is located in the left edge of the square.
            Xb=X1;
            Yb=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
            case_v=4;  \%Location of the next iteration point A (4=right edge of the square).
            i_x=i;  \%Node 1 i component of the square to be scanned in the next step.
            j_x=j-1;  \%Node 1 j component of the square to be scanned in the next step.
        elseif Phi2*Phi3<0  \%B is located in the upper edge of the square.
            Xb=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
            Yb=Y2;
            case_v=1;  \%Location of the next iteration point A (1=lower edge of the square).
            i_x=i+1;  \%Node 1 i component of the square to be scanned in the next step.
            j_x=j;  \%Node 1 j component of the square to be scanned in the next step.
        elseif Phi3*Phi4<0  \%B is located in the right edge of the square.
            Xb=X4;
            Yb=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
            case_v=2;  \%Location of the next iteration point A (2=left edge of the square).
            i_x=i;  \%Node 1 i component of the square to be scanned in the next step.
            j_x=j+1;  \%Node 1 j component of the square to be scanned in the next step.
        end
    case 2  \%A is located in the left edge of the square.
        if Phi1*Phi4<0  \%B is located in the lower edge of the square.
            Xb=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
            Yb=Y1;
            case_v=3;  \%Location of the next iteration point A (3=upper edge of the square).
            i_x=i-1;  \%Node 1 i component of the square to be scanned in the next step.
\[ j_x = j; \] % Node 1 j component of the square to be scanned in the next step.
else if \( \Phi_2 \Phi_3 < 0 \) % B is located in the upper edge of the square.
  \[ X_b = X_2 + \left( \frac{\text{abs}(\Phi_2)}{\text{abs}(\Phi_2) + \text{abs}(\Phi_3)} \right) \times \Delta_{xy}; \]
  \[ Y_b = Y_2; \]
  case_v = 1; % Location of the next iteration point A (1=lower edge of the square). i_x = i+1;
  \[ j_x = j; \] % Node 1 j component of the square to be scanned in the next step.
  \[ X_b = X_4; \]
  \[ Y_b = Y_4 + \left( \frac{\text{abs}(\Phi_4)}{\text{abs}(\Phi_4) + \text{abs}(\Phi_3)} \right) \times \Delta_{xy}; \]
end % Location of the next iteration point A (2=left edge of the square). i_x = i;
  \[ j_x = j; \] % Node 1 j component of the square to be scanned in the next step.
  \[ X_b = X_1; \]
  \[ Y_b = Y_1 + \left( \frac{\text{abs}(\Phi_1)}{\text{abs}(\Phi_1) + \text{abs}(\Phi_2)} \right) \times \Delta_{xy}; \]
end % Location of the next iteration point A (3=upper edge of the square). i_x = i-1;
  \[ j_x = j; \] % Node 1 j component of the square to be scanned in the next step.
  \[ X_b = X_1; \]
  \[ Y_b = Y_1 + \left( \frac{\text{abs}(\Phi_1)}{\text{abs}(\Phi_1) + \text{abs}(\Phi_2)} \right) \times \Delta_{xy}; \]
end % Location of the next iteration point A (4=right edge of the square). i_x = i;
  \[ j_x = j-1; \] % Node 1 j component of the square to be scanned in the next step.
  \[ X_b = X_4; \]
  \[ Y_b = Y_4 + \left( \frac{\text{abs}(\Phi_4)}{\text{abs}(\Phi_4) + \text{abs}(\Phi_3)} \right) \times \Delta_{xy}; \]
end % Location of the next iteration point A (2=left edge of the square). i_x = i;
  \[ j_x = j+1; \] % Node 1 j component of the square to be scanned in the next step.
end % Location of the next iteration point A (4=right edge of the square). i_x = i;
  \[ j_x = j; \] % Node 1 j component of the square to be scanned in the next step.
Xb=Y1;
case_v=3;
\%Location of the next iteration point A (3=upper edge of the square).
i_x=i-1; \%Node 1 i component of the square to be scanned in the next step.
j_x=j; \%Node 1 j component of the square to be scanned in the next step.
elseif Phi1*Phi2<0
\%B is located in the left edge of the square.
Xb=X1;
Yb=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
case_v=4;
\%Location of the next iteration point A (4=right edge of the square).
i_x=i; \%Node 1 i component of the square to be scanned in the next step.
j_x=j-1; \%Node 1 j component of the square to be scanned in the next step.
elseif Phi2*Phi3<0
\%B is located in the upper edge of the square.
Xb=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
Yb=Y2;
case_v=1;
\%Location of the next iteration point A (1=lower edge of the square).
i_x=i+1; \%Node 1 i component of the square to be scanned in the next step.
j_x=j; \%Node 1 j component of the square to be scanned in the next step.
end
otherwise
Xb=0;
Yb=0;
case_v=-1;
\%Location of the next iteration point A (-1=error).
i_x=0; \%Node 1 i component of the square to be scanned in the next step (0=error).
j_x=0; \%Node 1 j component of the square to be scanned in the next step (0=error).
end
end

\%Assignment of the B point to the interface curve.
C(k+1,1)=Xb;
C(k+1,2)=Yb;
case_id_A=case_v; \%Assignment of the case_id_A for the next iteration.
dif=sqrt((Xb-C(1,1))^2+(Yb-C(1,2))^2);
if dif==0
closed=1;
elseif dif<epsilon
closed=2;
else
closed=0;
end
k=k+1;
end
end
A.1.2.12. interface_tracker2.m

function [C,C2,Cin] = interface_tracker2(X,Y,Phi,Rcase)
%Interface tracker2: this function calculates the points of the interface
%given the cartesian grid and the Phi matrix.
% Detailed explanation goes here

%% dx=diff(Y(1:2,1));
Delta_xy=dx;
N=size(X,1);
epsilon=1e-6;

%%
%First step: find a interface point and set it as the first point of the curve:
%Vertical scan from the y=-L/2 and x=0 to y=L/2 and x=0.
j_cc=floor((N+1)/2);
for i=1:1:N-1
    Phi1=Phi(i,j_cc);
    Phi2=Phi(i+1,j_cc);
    Phi3=Phi(i+1,j_cc+1);
    Phi4=Phi(i,j_cc+1);
    X1=X(i,j_cc);
    Y1=Y(i,j_cc);
    X2=X(i+1,j_cc);
    Y2=Y(i+1,j_cc);
    X3=X(i+1,j_cc+1);
    Y3=Y(i+1,j_cc+1);
    X4=X(i,j_cc+1);
    Y4=Y(i,j_cc+1);
    if Phi1*Phi4<0 && Phi2*Phi3<0         %CASE 1
        Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
        Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
        Ya_0=Y1;
        Yb_0=Y2;
        i_0=i;
        j_0=j_cc;
        case_id=1;
        break;
    elseif Phi1*Phi2<0 && Phi1*Phi4<0      %CASE 2
        Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
        Xb_0=X1;
        Ya_0=Y1;
        Yb_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
        i_0=i;
        j_0=j_cc;
        case_id=2;
        break;
    elseif Phi1*Phi4<0 && Phi3*Phi4<0      %CASE 3
        Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
        Xb_0=X4;
        Ya_0=Y1;
        Yb_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
        i_0=i;
        j_0=j_cc;
        case_id=3;
    end
end
% Study of Grain Burnback and Performance of Solid Rocket Motors

break;
elseif Phi1*Phi2<0 && Phi3*Phi4<0 %CASE 4
    Xa_0=X1;
    Xb_0=X4;
    Ya_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
    Yb_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
    i_0=i;
    j_0=j_cc;
    case_id=4;
    break;
elseif Phi1*Phi2<0 && Phi2*Phi3<0 %CASE 5
    Xa_0=X1;
    Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
    Ya_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
    Yb_0=Y2;
    i_0=i;
    j_0=j_cc;
    case_id=5;
    break;
elseif Phi2*Phi3<0 && Phi3*Phi4<0 %CASE 6
    Xa_0=X4;
    Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
    Ya_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi2)))*Delta_xy;
    Yb_0=Y2;
    i_0=i;
    j_0=j_cc;
    case_id=6;
    break;
else
    Xa_0=0;
    Xb_0=0;
    Ya_0=0;
    Yb_0=0;
    case_id=0;
end

%%
% If the vertical scan does not find the interface, then a horizontal scan
% shall be done:
if case_id==0
    i_cc=floor((N+1)/2);
    for j=1:1:N-1
        Phi1=Phi(i_cc,j);
        Phi2=Phi(i_cc+1,j);
        Phi3=Phi(i_cc+1,j+1);
        Phi4=Phi(i_cc,j+1);
        X1=X(i_cc,j);
        Y1=Y(i_cc,j);
        X2=X(i_cc+1,j);
        Y2=Y(i_cc+1,j);
        X3=X(i_cc+1,j+1);
        Y3=Y(i_cc+1,j+1);
        X4=X(i_cc,j+1);
        Y4=Y(i_cc,j+1);
        if Phi1*Phi4<0 && Phi2*Phi3<0 %CASE 1
            Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
            Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
        end
    end
end
Ya_0=Y1;
Yb_0=Y2;
i_0=i_cc;
j_0=j;
case_id=1;
bbreak;
elseif PHI1*PHI2<0 && PHI1*PHI4<0 %CASE 2
Xa_0=X1+(abs(PHI1)/(abs(PHI1)+abs(PHI4)))*Delta_xy;
Xb_0=X1;
Ya_0=Y1;
Yb_0=Y1+(abs(PHI1)/(abs(PHI1)+abs(PHI2)))*Delta_xy;
i_0=i_cc;
j_0=j;
case_id=2;
bbreak;
elseif PHI1*PHI4<0 && PHI3*PHI4<0 %CASE 3
Xa_0=X1+(abs(PHI1)/(abs(PHI1)+abs(PHI4)))*Delta_xy;
Xb_0=X4;
Ya_0=Y1;
Yb_0=Y4+(abs(PHI4)/(abs(PHI4)+abs(PHI3)))*Delta_xy;
i_0=i_cc;
j_0=j;
case_id=3;
bbreak;
elseif PHI1*PHI2<0 && PHI3*PHI4<0 %CASE 4
Xa_0=X1;
Xb_0=X4;
Ya_0=Y1+(abs(PHI1)/(abs(PHI1)+abs(PHI2)))*Delta_xy;
Yb_0=Y4+(abs(PHI4)/(abs(PHI4)+abs(PHI3)))*Delta_xy;
i_0=i_cc;
j_0=j;
case_id=4;
bbreak;
elseif PHI1*PHI2<0 && PHI2*PHI3<0 %CASE 5
Xa_0=X1;
Xb_0=X2+(abs(PHI2)/(abs(PHI2)+abs(PHI3)))*Delta_xy;
Ya_0=Y1+(abs(PHI1)/(abs(PHI1)+abs(PHI2)))*Delta_xy;
Yb_0=Y2;
i_0=i_cc;
j_0=j;
case_id=5;
bbreak;
elseif PHI2*PHI3<0 && PHI3*PHI4<0 %CASE 6
Xa_0=X4;
Xb_0=X2+(abs(PHI2)/(abs(PHI2)+abs(PHI3)))*Delta_xy;
Ya_0=Y4+(abs(PHI4)/(abs(PHI4)+abs(PHI2)))*Delta_xy;
Yb_0=Y2;
i_0=i_cc;
j_0=j;
case_id=6;
bbreak;
else
Xa_0=0;
Xb_0=0;
Ya_0=0;
Yb_0=0;
case_id=-1;
end
end
end

%%
%If the vertical and horizontal scan do not find the interface, then a
%overall scan shall be done:
if case_id== -1
  for i=1:1:N-1 %y-direction loop
    for j=1:1:N-1 %x-direction loop
      Phi1=Phi(i,j);
      Phi2=Phi(i+1,j);
      Phi3=Phi(i+1,j+1);
      Phi4=Phi(i,j+1);
      X1=X(i,j);
      Y1=Y(i,j);
      X2=X(i+1,j);
      Y2=Y(i+1,j);
      X3=X(i+1,j+1);
      Y3=Y(i+1,j+1);
      X4=X(i,j+1);
      Y4=Y(i,j+1);
      if Phi1*Phi4<0 && Phi2*Phi3<0 %CASE 1
        Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
        Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
        Ya_0=Y1;
        Yb_0=Y2;
        i_0=i;
        j_0=j;
        case_id=1;
        break;
      elseif Phi1*Phi2<0 && Phi1*Phi4<0 %CASE 2
        Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
        Xb_0=X1;
        Ya_0=Y1;
        Yb_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
        i_0=i;
        j_0=j;
        case_id=2;
        break;
      elseif Phi1*Phi4<0 && Phi3*Phi4<0 %CASE 3
        Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
        Xb_0=X4;
        Ya_0=Y1;
        Yb_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
        i_0=i;
        j_0=j;
        case_id=3;
        break;
      elseif Phi1*Phi2<0 && Phi3*Phi4<0 %CASE 4
        Xa_0=X1;
        Xb_0=X4;
        Ya_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
        Yb_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
        i_0=i;
        j_0=j;
        case_id=4;
        break;
      elseif Phi1*Phi2<0 && Phi2*Phi3<0 %CASE 5
        Xa_0=X1;
Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
Ya_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
Yb_0=Y2;
i_0=i;
j_0=j;
case_id=5;
break;
elseif Phi2*Phi3<0 && Phi3*Phi4<0  %CASE 6
Xa_0=X4;
Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
Ya_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi2)))*Delta_xy;
Yb_0=Y2;
i_0=i;
j_0=j;
case_id=6;
break;
else
Xa_0=0;
Xb_0=0;
Ya_0=0;
Yb_0=0;
i_0=i;
j_0=j;
case_id=0;
end
end
end

%%
%INTERFACE SCANNING AND DETERMINATION
closed=0;
k=1;
l=2;
while closed==0  %The condition closed means the closing of the
%When closed is equal to 1 then the curve has
%the while loop can be stopped.
end

%Assignment of the intial point of the interface curve.
if k==1
i_x=i_0;
j_x=j_0;
switch case_id
  case {1,3}
    case_id_A=1;
    C(1,1)=Xa_0;
    C(1,2)=Ya_0;
  case {4,5}
    case_id_A=2;
    C(1,1)=Xa_0;
    C(1,2)=Ya_0;
  case 2
    case_id_A=2;
    C(1,1)=Xb_0;
    C(1,2)=Yb_0;
end

end
case 6
  case_id A=3;
  C(1,1)=Xb_0;
  C(1,2)=Yb_0;
  otherwise
  case_id A=0;
  C(1,1)=0;
  C(1,2)=0;
end
C2(1,1)=C(1,1);
C2(1,2)=C(1,2);
C2(1,3)=0;
end

%Assignment of the A point as the B point calculated in the previous iteration (A(k)=B(k-1)).
Xa=C(k,1);
Ya=C(k,2);
i=i_x;
%Node 1 i component of the square to be scanned.
j=j_x;
%Node 1 j component of the square to be scanned.
X1=X(i,j);
%X position of the Node 1 of the square to be scanned.
Y1=Y(i,j);
%Y position of the Node 1 of the square to be scanned.
X2=X(i+1,j);
%X position of the Node 2 of the square to be scanned.
Y2=Y(i+1,j);
%Y position of the Node 2 of the square to be scanned.
X3=X(i+1,j+1);
%X position of the Node 3 of the square to be scanned.
Y3=Y(i+1,j+1);
%Y position of the Node 3 of the square to be scanned.
X4=X(i,j+1);
%X position of the Node 4 of the square to be scanned.
Y4=Y(i,j+1);
%Y position of the Node 4 of the square to be scanned.
Phi1=Phi(i,j);
%Phi value at the Node 1 of the square to be scanned.
Phi2=Phi(i+1,j);
%Phi value at the Node 2 of the square to be scanned.
Phi3=Phi(i+1,j+1);
%Phi value at the Node 3 of the square to be scanned.
Phi4=Phi(i,j+1);
%Phi value at the Node 4 of the square to be scanned.

%Identification of a square which has the interface passing through it twice:
if Phi1*Phi2<0 && Phi2*Phi3<0 && Phi3*Phi4<0 && Phi1*Phi4<0
%Condition of interface double crossing the square.
V=C(k,:)-C(k-1,:);  %Vector of the interface ongoing direction in the k point.
switch case_id_A  %Location of the square edge where the
A point is.
    case 1  %A is located in the lower edge of the square.
        Xd_2=X1;
        Yd_2=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
        Xd_4=X4;
        Yd_4=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
        a=[(Xd_2-Xa) (Yd_2-Ya)];
        b=[(Xd_4-Xa) (Yd_4-Ya)];
        Va=dot(V,a);
        Vb=dot(V,b);
        if Va>Vb  %B is located in the left edge of
        the square.
            Xb=Xd_2;
            Yb=Yd_2;
            case_v=4;  %Location of the next iteration
        point A (4=right edge of the square).
            i_x=i;
            %Node 1 i component of the square
to be scanned in the next step.
            j_x=j-1;
            %Node 1 j component of the square
to be scanned in the next step.
        else
            Xb=Xd_4;
            Yb=Yd_4;
            case_v=2;  %Location of the next iteration
        point A (2=left edge of the square).
            i_x=i;
            %Node 1 i component of the square
to be scanned in the next step.
            j_x=j+1;
            %Node 1 j component of the square
to be scanned in the next step.
        end
    case 2  %A is located in the left edge of the square.
        Xd_1=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
        Yd_1=Y1;
        Xd_3=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
        Yd_3=Y2;
        a=[(Xd_1-Xa) (Yd_1-Ya)];
        b=[(Xd_3-Xa) (Yd_3-Ya)];
        Va=dot(V,a);
        Vb=dot(V,b);
        if Va>Vb  %B is located in the lower edge of
the square.
            Xb=Xd_1;
            Yb=Yd_1;
            case_v=3;  %Location of the next iteration
        point A (3=upper edge of the square).
            i_x=i-1;
            %Node 1 i component of the square
to be scanned in the next step.
            j_x=j;
            %Node 1 j component of the square
to be scanned in the next step.
        else
            Xb=Xd_3;
            Yb=Yd_3;
            case_v=1;  %Location of the next iteration
        point A (1=lower edge of the square).
            i_x=i+1;
            %Node 1 i component of the square
to be scanned in the next step.
\(j_x=j\); \(\%\) Node 1 \(j\) component of the square to be scanned in the next step.
\end
\end
\end
\begin{case}3\end{case} \(\%\) A is located in the upper edge of the square.
\begin{align*}
X_d_2 &= X_1; \\
Y_d_2 &= Y_1 + (\text{abs(Phi1)} / (\text{abs(Phi1)} + \text{abs(Phi2)})) \times \Delta_{xy}; \\
X_d_4 &= X_4; \\
Y_d_4 &= Y_4 + (\text{abs(Phi4)} / (\text{abs(Phi4)} + \text{abs(Phi3)})) \times \Delta_{xy}; \\
a &= [(X_d_2 - X_a) \ (Y_d_2 - Y_a)]; \\
b &= [(X_d_4 - X_a) \ (Y_d_4 - Y_a)]; \\
V_a &= \text{dot}(V, a); \\
V_b &= \text{dot}(V, b); \\
\text{if} \ V_a > V_b \text{ \(\%\) B is located in the left edge of the square.}
\begin{align*}
X_b &= X_d_2; \\
Y_b &= Y_d_2; \\
\text{case} \_v &= 4; \\
\text{\%Location of the next iteration point A (4=right edge of the square).}
\end{align*}
\end{case}
\begin{case}4\end{case} \(\%\) A is located in the right edge of the square.
\begin{align*}
X_d_1 &= X_1 + (\text{abs(Phi1)} / (\text{abs(Phi1)} + \text{abs(Phi4)})) \times \Delta_{xy}; \\
Y_d_1 &= Y_1; \\
X_d_3 &= X_2 + (\text{abs(Phi2)} / (\text{abs(Phi2)} + \text{abs(Phi3)})) \times \Delta_{xy}; \\
Y_d_3 &= Y_2; \\
a &= [(X_d_1 - X_a) \ (Y_d_1 - Y_a)]; \\
b &= [(X_d_3 - X_a) \ (Y_d_3 - Y_a)]; \\
V_a &= \text{dot}(V, a); \\
V_b &= \text{dot}(V, b); \\
\text{if} \ V_a > V_b \text{ \(\%\) B is located in the lower edge of the square.}
\begin{align*}
X_b &= X_d_1; \\
Y_b &= Y_d_1; \\
\text{case} \_v &= 3; \\
\text{\%Location of the next iteration point A (3=upper edge of the square).}
\end{align*}
\end{case}
\begin{case}2\end{case} \(\%\) A is located in the left edge of the square.
\begin{align*}
X_d_2 &= X_2; \\
Y_d_2 &= Y_2; \\
\text{case} \_v &= 2; \\
\text{\%Location of the next iteration point A (2=left edge of the square).}
\end{align*}
\end{case}
\begin{case}1\end{case} \(\%\) A is located in the lower edge of the square.
\begin{align*}
X_d_3 &= X_3; \\
Y_d_3 &= Y_3; \\
\text{case} \_v &= 1; \\
\text{\%Location of the next iteration point A (1=lower edge of the square).}
\end{align*}
\end{case}
\begin{verbatim}
    i_x=i+1; %Node 1 i component of the square to be scanned in the next step.
    j_x=j;  %Node 1 j component of the square to be scanned in the next step.
end

otherwise
    Xb=0;
    Yb=0;
    case_v=-1;
    i_x=i+1; %Node 1 i component of the square to be scanned in the next step.
    j_x=j;  %Node 1 j component of the square to be scanned in the next step.
end

else %The interface crosses the square just once.

    switch case_id_A %Location of the square edge where the A point is.
        case 1 %A is located in the lower edge of the square.
            if Phi1*Phi2<0 %B is located in the left edge of the square.
                case_v=4; %Location of the next iteration point A (4=right edge of the square).
                i_x=i; %Node 1 i component of the square to be scanned in the next step.
                j_x=j-1; %Node 1 j component of the square to be scanned in the next step.
            elseif Phi2*Phi3<0 %B is located in the upper edge of the square.
                Xb=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
                Yb=Y2;
                case_v=1; %Location of the next iteration point A (1=lower edge of the square).
                i_x=i+1; %Node 1 i component of the square to be scanned in the next step.
                j_x=j;  %Node 1 j component of the square to be scanned in the next step.
            elseif Phi3*Phi4<0 %B is located in the right edge of the square.
                Xb=X4;
                Yb=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
                case_v=2; %Location of the next iteration point A (2=left edge of the square).
                i_x=i; %Node 1 i component of the square to be scanned in the next step.
                j_x=j+1; %Node 1 j component of the square to be scanned in the next step.
            end
        case 2 %A is located in the left edge of the square.
            if Phi1*Phi4<0 %B is located in the lower edge of the square.
                case_v=1;
                Xb=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
                Yb=Y1;
            end
        end
\end{verbatim}
Study of Grain Burnback and Performance of Solid Rocket Motors

case_v=3; %Location of the next iteration point A (3=upper edge of the square).
i_x=i-1; %Node 1 i component of the square to be scanned in the next step.
j_x=j; %Node 1 j component of the square to be scanned in the next step.
elseif Phi2*Phi3<0 %B is located in the upper edge of the square.
    Xb=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
    Yb=Y2;
    case_v=1; %Location of the next iteration point A (1=lower edge of the square).
i_x=i+1; %Node 1 i component of the square to be scanned in the next step.
j_x=j; %Node 1 j component of the square to be scanned in the next step.
end

case_v=1; %Location of the next iteration point A (1=lower edge of the square).
i_x=i+1; %Node 1 i component of the square to be scanned in the next step.
j_x=j; %Node 1 j component of the square to be scanned in the next step.
elseif Phi3*Phi4<0 %B is located in the right edge of the square.
    Xb=X4;
    Yb=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
    case_v=2; %Location of the next iteration point A (2=left edge of the square).
i_x=i; %Node 1 i component of the square to be scanned in the next step.
j_x=j+1; %Node 1 j component of the square to be scanned in the next step.
end

case 3 %A is located in the upper edge of the square.
if Phi1*Phi4<0 %B is located in the lower edge of the square.
    Xb=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
    Yb=Y1;
    case_v=3; %Location of the next iteration point A (3=upper edge of the square).
i_x=i-1; %Node 1 i component of the square to be scanned in the next step.
j_x=j; %Node 1 j component of the square to be scanned in the next step.
elseif Phi1*Phi2<0 %B is located in the left edge of the square.
    Xb=X1;
    Yb=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
    case_v=4; %Location of the next iteration point A (4=right edge of the square).
i_x=i; %Node 1 i component of the square to be scanned in the next step.
j_x=j-1; %Node 1 j component of the square to be scanned in the next step.
elseif Phi3*Phi4<0 %B is located in the right edge of the square.
    Xb=X4;
    Yb=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
    case_v=2; %Location of the next iteration point A (2=left edge of the square).
i_x=i; %Node 1 i component of the square to be scanned in the next step.
j_x=j+1; %Node 1 j component of the square to be scanned in the next step.
end
Study of Grain Burnback and Performance of Solid Rocket Motors

```matlab
% Assignment of the B point to the interface curve.
C(k+1,1)=Xb;
C(k+1,2)=Yb;
case_id_A=case_v;         % Assignment of the case_id_A for the next iteration.
dif=sqrt((Xb-C(1,1))^2+(Yb-C(1,2))^2);

% Assignment of only the points inside or frontier of the motor case:
Ra=sqrt(Xa^2+Ya^2);
Rb=sqrt(Xb^2+Yb^2);
if Ra<Rcase && Rb<Rcase  % The interface is inside the motor case.
```

C2(1,1)=C(k+1,1);
C2(1,2)=C(k+1,2);
C2(1,3)=0;
l=l+1;
elseif Ra<Rcase && Rb>Rcase || Ra>Rcase && Rb<Rcase
%The interface is in the frontier of the motor case.
a=(Xb-Xa)^2+(Yb-Ya)^2;
b=2*Xa*(Xb-Xa)+2*Ya*(Yb-Ya);
c=Xa^2+Ya^2-Rcase^2;
Disc=b^2-4*a*c;
if Disc>=0
s1=(-b+sqrt(Disc))/(2*a);
s2=(-b-sqrt(Disc))/(2*a);
if s1>=0 && s1<=1
s=s1;
x=Xa+(Xb-Xa)*s;
y=Ya+(Yb-Ya)*s;
elseif s2>=0 && s2<=1
s=s2;
x=Xa+(Xb-Xa)*s;
y=Ya+(Yb-Ya)*s;
else
x=0;
y=0;
end
else
x=0;
y=0;
end
C2(1,1)=x;
C2(1,2)=y;
C2(1,3)=1;
%Identification of the intersection point of the interface with the motor case.
l=l+1;
end

if dif==0
closed=1;
elseif dif<epsilon
closed=2;
else
closed=0;
end

k=k+1;
end

%Check whether the first point of C2 is inside the motor case or not:
if norm(C2(1,:))>Rcase
C2(1,:)=[];
end

% %Generation of the interface curve accounting only the points inside the motor case:
k=1; %Number of nodes of the segment in the motor case
Na=100; %Number of nodes of the segment in the motor case
between interface curve escape and return.
for i=1:size(C2,1)
    if i<size(C2,1) && C2(i,3)==1 && C2(i+1,3)==1
        Theta_max=atan2(C2(i+1,2),C2(i+1,1));
        Theta_min=atan2(C2(i,2),C2(i,1));
        r_max=[C2(i+1,1) C2(i+1,2) 0];
        r_min=[C2(i,1) C2(i,2) 0];
        if cross(r_min,r_max)*[0 0 1]'>0 %The interface goes in
counter clockwise sense.
            if Theta_max<Theta_min
                Theta=2*pi-abs(Theta_max)-abs(Theta_min);
            else
                Theta=abs(Theta_max-Theta_min);
            end
        elseif i==size(C2,1) && C2(i,3)==1 && C2(1,3)==1
            Theta_max=atan2(C2(1,2),C2(1,1));
            Theta_min=atan2(C2(1,2),C2(1,1));
            r_max=[C2(1,1) C2(1,2) 0];
            r_min=[C2(1,1) C2(1,2) 0];
            if cross(r_min,r_max)*[0 0 1]'>0 %The interface goes in
counter clockwise sense.
                if Theta_max<Theta_min
                    Theta=2*pi-abs(Theta_max)-abs(Theta_min);
                else
                    Theta=abs(Theta_max-Theta_min);
                end
            end
        end
        Delta_theta=Theta/Na;
        for j=1:1:Na+1
            Theta_j=Theta_min+(j-1)*Delta_theta;
            Cin(k,1)=Rcase*cos(Theta_j);
            Cin(k,2)=Rcase*sin(Theta_j);
            k=k+1;
        end
        elseif i=size(C2,1) && C2(i,3)==1 && C2(1,3)==1
            Theta_max=atan2(C2(1,2),C2(1,1));
            Theta_min=atan2(C2(1,2),C2(1,1));
            r_max=[C2(1,1) C2(1,2) 0];
            r_min=[C2(1,1) C2(1,2) 0];
            if cross(r_min,r_max)*[0 0 1]'>0 %The interface goes in
            clockwise sense.
                if Theta_max>Theta_min
                    Theta=abs(Theta_max)+abs(Theta_min)-2*pi;
                else
                    Theta=Theta_max-Theta_min;
                end
            elseif i==size(C2,1) && C2(i,3)==1 && C2(1,3)==1
                Theta_max=atan2(C2(1,2),C2(1,1));
                Theta_min=atan2(C2(1,2),C2(1,1));
                r_max=[C2(1,1) C2(1,2) 0];
                r_min=[C2(1,1) C2(1,2) 0];
                if cross(r_min,r_max)*[0 0 1]'>0 %The interface goes in
            clockwise sense.
                    if Theta_max>Theta_min
                        Theta=abs(Theta_max)+abs(Theta_min)-2*pi;
                    else
                        Theta=Theta_max-Theta_min;
                    end
                end
        end
        Delta_theta=Theta/Na;
        for j=1:1:Na+1
            Theta_j=Theta_min+(j-1)*Delta_theta;
            Cin(k,1)=Rcase*cos(Theta_j);
            Cin(k,2)=Rcase*sin(Theta_j);
            k=k+1;
        end
        else
            % Do something else
        end
    end
end
Cin(k,1)=C2(i,1);
Cin(k,2)=C2(i,2);
k=k+1;
end
end

A.1.2.13. mach_calculation.m

function [M] = mach_calculation(A_e,A_t,Gamma)
%UNTITLED Summary of this function goes here
% Detailed explanation goes here
delta=1e-6; %Numeric precision
fr=0.5; %Relaxation factor for the calculation
aux=(2*(Gamma-1))/(Gamma+1);

M=1;
dif=2*delta;

while dif>delta
x=M;
M=sqrt((2/(Gamma-1))*(((Gamma+1)/2)*(((M*A_e)/A_t).^aux)-1));
dif=abs(M-x);
M=x+fr*(M-x);
end

A.1.2.14. order_vector_points.m

function [X_ord] = order_vector_points(Xi)
%UNTITLED Summary of this function goes here
% Detailed explanation goes here

N=size(Xi,1);

Rep1=0;
for i=1:1:N
    for j=1:1:N
        if i~=j
            A=Xi(i,:)-Xi(j,:);
            if A==[0 0]
                Rep1=i;
                Rep2=j;
            end
        end
    end
end

B=zeros(N,1);
X_ord=zeros(N,2);
if Rep1==0
    X_ord(1,:)=Xi(1,:);
    B(1)=1;
    N2=N-1;
else
    X_ord(1,:)=Xi(Rep1,:);
    B(Rep1)=1;
    X_ord(N,:)=Xi(Rep2,:);
    B(Rep2)=1;
    N2=N-2;
end

Dmin0=1e6;
for k=1:1:N
    D=sqrt((X_ord(1,1)-Xi(k,1))^2+(X_ord(1,2)-Xi(k,2))^2);
    if D<Dmin0 && B(k)==0
        Dmin0=D;
        Pos0=k;
    end
end

X_ord(2,:)=Xi(Pos0,:);
B(Pos0)=1;
for i=2:1:N2
    Dmin=1e6;
    for j=1:1:N
        D=sqrt((X_ord(i,1)-Xi(j,1))^2+(X_ord(i,2)-Xi(j,2))^2);
        if D<Dmin && B(j)==0
            Dmin=D;
            Pos=j;
        end
    end
    X_ord(i+1,:)=Xi(Pos,:);
    B(Pos)=1;
end

Smax=0;
for i=1:1:N
    S=X_ord(i,1)+X_ord(i,2);
    if S>Smax;
        Smax=S;
        PosS=i;
    end
end

Xp=X_ord(PosS+1,:)-X_ord(PosS,:);
Xm=X_ord(PosS+1,:)-X_ord(PosS,:);
Np=[Xp(2) -Xp(1)];
Nm=[Xm(2) -Xm(1)];
Xa=X_ord(PosS,:)+Np;
Sa=Xa(1)+Xa(2);
Xb=X_ord(PosS,:)+Nm;
Sb=Xb(1)+Xb(2);
if \( Sa < S_{\text{max}} \) && \( Sb < S_{\text{max}} \) \%The curve goes in clockwise sense, so the sense
k=N; \%has to be turned, so that it is counter clockwise.
X_ord2=zeros(N,2);
for \( i=1:1:N \)
    \( X_{\text{ord2}}(i,:) \)=X_ord(k,:);
    k=k-1;
end
X_ord=X_ord2;
end

\[ A.1.2.15. \text{perimeter.m} \]

\begin{verbatim}
    function [P] = perimeter(C,Rmax)
    %UNTITLED Summary of this function goes here
    % Detailed explanation goes here
    C(:,1)=[]; \%Unassignment of first column, due to is general information
    %and not a X or Y position.
P=0;
    for \( i=1:1:\text{size(C,2)} \)
        if \( i=\text{size(C,2)} \)
            Xa=C(1,i);
            Ya=C(2,i);
            Xb=C(1,1);
            Yb=C(2,1);
        else
            Xa=C(1,i);
            Ya=C(2,i);
            Xb=C(1,i+1);
            Yb=C(2,i+1);
        end
        Ra=sqrt(Xa^2+Ya^2);
        Rb=sqrt(Xb^2+Yb^2);
        if \( Ra \geq R_{\text{max}} \) && \( Rb \geq R_{\text{max}} \)
            di=0;
        elseif \( Ra < R_{\text{max}} \) && \( Rb < R_{\text{max}} \)
            di=sqrt((Xa-Xb)^2+(Ya-Yb)^2);
        elseif \( Ra < R_{\text{max}} \) && \( Rb > R_{\text{max}} \) || \( Ra > R_{\text{max}} \) && \( Rb < R_{\text{max}} \)
            a=(Xb-Xa)^2+(Yb-Ya)^2;
            b=2*Xa*(Xb-Xa)+2*Ya*(Yb-Ya);
            c=Xa^2+Ya^2-R_{\text{max}}^2;
            Disc=b^2-4*a*c;
            if Disc>=0
                s1=(-b+sqrt(Disc))/(2*a);
                s2=(-b-sqrt(Disc))/(2*a);
                if s1>=0 && s1<=1
                    s=s1;
                    x=Xa+(Xb-Xa)*s;
                    y=Ya+(Yb-Ya)*s;
                    if \( Ra < R_{\text{max}} \) && \( Rb > R_{\text{max}} \)
                        di=sqrt((x-Xa)^2+(y-Ya)^2);
                    elseif \( Ra > R_{\text{max}} \) && \( Rb < R_{\text{max}} \)
                        di=sqrt((x-Xb)^2+(y-Yb)^2);
                end
            end
    end
end
\end{verbatim}
end
elseif s2>=0 && s2<=1
s=s2;
x=Xa+(Xb-Xa)*s;
y=Ya+(Yb-Ya)*s;
if Ra<Rmax && Rb>Rmax
di=sqrt((x-Xa)^2+(y-Ya)^2);
elseif Ra>Rmax && Rb<Rmax
di=sqrt((x-Xb)^2+(y-Yb)^2);
end
else
di=0;
end
else
di=0;
end
end
P=di+P;
end

A.1.2.16. units_conversion.m

function [K_units] = units_conversion(units)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
switch units
    case 'm'
        K_units=1;
    case 'cm'
        K_units=100;
    case 'mm'
        K_units=1000;
end

A.1.2.17. grain_video_evolution.m

function [M] = grain_video_evolution(X,Y,Phi,Xi,Xi_Ext,L,Rmax,n_frames,n_times,fps,name_file_a,units)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
M=moviein(n_frames);
set(gca,'nextplot','replacechildren');
n_steps=size(Phi,3);
for i=1:n_frames
delta_i=floor(n_steps/n_frames);
[C,C2,Cin] = interface_tracker2(X,Y,Phi(:,,:,delta_i*i),Rmax);
hold off;
plot(Cin(:,1),Cin(:,2),'color',[0 0 1]);
end
hold on;
plot(Xi(:,1),Xi(:,2), 'LineWidth', 2, 'color', [0 0 0])
hold on;
plot(Xi_Ext(:,1),Xi_Ext(:,2), 'LineWidth', 2, 'color', [0 0 0])
axis('equal')
text1=['x [' units ']'];
text2=['y [' units ']'];
xlabel(text1)
ylabel(text2)
title('Grain evolution during rocket burning')
xlim([-0.75*L/2 0.75*L/2])
ylim([-0.75*L/2 0.75*L/2])
M(:,i)=getframe(gcf);
end

if n_times>1
    M1=[M M M];  %Save the movie frames 3 times in order to repeat the
else  %movie 3 times.
    M1=M;
end

movie(M,n_times,fps);
name_file_movie=[name_file_a '.avi'];
movie2avi(M1,name_file_movie,'fps',fps,'compression','None');
end

A.1.2.18. grain_video_evolution_filled.m

function [M] =
grain_video_evolution_filled(X,Y,Phi,Xi,Xi_Ext,L,Rmax,n_frames,n_times,fps,name_file_a,units)

M=moviein(n_frames);

set(gca,'nextplot','replacechildren');
n_steps=size(Phi,3);

for i=1:n_frames
    delta_i=floor(n_steps/n_frames);
    [C,C2,Cin] = interface_tracker2(X,Y,Phi(:,:,delta_i*i),Rmax);
    hold off;
    plot(Xi_Ext(:,1),Xi_Ext(:,2), 'LineWidth', 2, 'color', [0 0 0])
    hold on;
    fill(Xi_Ext(:,1),Xi_Ext(:,2),[0 0 1]);
    hold on;
    fill(Cin(:,1),Cin(:,2),[1 1 1]);
    axis('equal')
    text1=['x [' units ']'];
    text2=['y [' units ']'];
    xlabel(text1)
    ylabel(text2)
    title('Grain evolution during rocket burning')
    xlim([-0.75*L/2 0.75*L/2])
end
ylim([-0.75*L/2 0.75*L/2])
M(:,i)=getframe(gcf);
end

if n_times>1
M1=[M M M];  %Save the movie frames 3 times in order to repeat the
else            %movie 3 times.
M1=M;
end
movie(M,n_times,fps);
name_file_movie=[name_file_a '.avi'];
movie2avi(M1,name_file_movie,'fps',fps,'compression','None');
end
A.2. Code of Quasi-3D Grain Burnback and 0D Unsteady Flow Simulation

The main aim of this section is to explain deeply the section 6.4.2 of the project report. This way, in the following Table 2 the names of the different Matlab® files used in the implementation of this Quasi-3D grain burnback and 0D unsteady flow simulation are identified. It should be pointed out that while there is a unique file corresponding to the main code, there are a total of 25 files used to implement independent functions. Furthermore, in Table 2 the main code has also been identified with number 0 and the functions have been numbered from 1 to 25. Note, that not all of the functions are directly called from the main code; nevertheless, they are also necessary to perform the simulation. That is why they are included in the Table 2 too. Finally, it is remarkable that all of the file’s names end with “.m” because they are Matlab® files.

<table>
<thead>
<tr>
<th>Nº</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Main Code</td>
<td>General_Quali_3D_discrete_cart_coord_0D_flow.m</td>
</tr>
<tr>
<td>1</td>
<td>Function</td>
<td>geometry_loading_3D.m</td>
</tr>
<tr>
<td>2</td>
<td>Function</td>
<td>chamber_volume.m</td>
</tr>
<tr>
<td>3</td>
<td>Function</td>
<td>curve_tangent_normal_3D.m</td>
</tr>
<tr>
<td>4</td>
<td>Function</td>
<td>minimum_distance_function_3D.m</td>
</tr>
<tr>
<td>5</td>
<td>Function</td>
<td>grain_cross_sections_pmt.m</td>
</tr>
<tr>
<td>6</td>
<td>Function</td>
<td>rocket_performance_0D_transient_quasi_3D.m</td>
</tr>
<tr>
<td>7</td>
<td>Function</td>
<td>grain_3D_plot_matrix.m</td>
</tr>
<tr>
<td>8</td>
<td>Function</td>
<td>motor_case_generation.m</td>
</tr>
<tr>
<td>9</td>
<td>Function</td>
<td>add_tri_matrix.m</td>
</tr>
<tr>
<td>10</td>
<td>Function</td>
<td>case_radius_int.m</td>
</tr>
<tr>
<td>11</td>
<td>Function</td>
<td>grain_cross_sections.m</td>
</tr>
<tr>
<td>12</td>
<td>Function</td>
<td>grain_video_evolution_3D.m</td>
</tr>
<tr>
<td>13</td>
<td>Function</td>
<td>initial_geometry_organization_3D.m</td>
</tr>
<tr>
<td>14</td>
<td>Function</td>
<td>interface_tracker_in_P.m</td>
</tr>
<tr>
<td>15</td>
<td>Function</td>
<td>interpolate_cross_section.m</td>
</tr>
<tr>
<td>16</td>
<td>Function</td>
<td>tri_generation.m</td>
</tr>
<tr>
<td>17</td>
<td>Function</td>
<td>wetted_surface.m</td>
</tr>
</tbody>
</table>
function Names

<table>
<thead>
<tr>
<th></th>
<th>Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>wetted_surface2.m</td>
</tr>
<tr>
<td>19*</td>
<td>units_conversion.m</td>
</tr>
<tr>
<td>20*</td>
<td>mach_calculation.m</td>
</tr>
<tr>
<td>21*</td>
<td>phi_one_time_step_evolution.m</td>
</tr>
<tr>
<td>22*</td>
<td>cartesian_grid.m</td>
</tr>
<tr>
<td>23*</td>
<td>curve_tangent_normal.m</td>
</tr>
<tr>
<td>24*</td>
<td>order_vector_points.m</td>
</tr>
<tr>
<td>25*</td>
<td>port_area.m</td>
</tr>
</tbody>
</table>

Table 2. List of names used to identify the main code and the functions.

Some of the functions of the Table 2 are repeated in the Table 1; precisely, all of the functions that have a star ("*") in the number are repeated (functions located just at the end of the Table 2). This way, in the introduction of the section A.1 a brief description of these repeated functions is enclosed. Additionally, in the section A.1.2.16. units_conversion.m is found the code of the function nº19 (repeated function). Analogously, the function nº20 is found in the section A.1.2.13. mach_calculation.m; the function nº21 in the section A.1.2.9. phi_one_time_step_evolution.m; the function nº22 in the section A.1.2.4. cartesian_grid.m; the nº23 in the section A.1.2.3. curve_tangent_normal.m; the function nº24 in the section A.1.2.14. order_vector_points.m and finally, the nº25 in the section A.1.2.7. port_area.m. Consequently, in this introduction a brief description of only not repeated functions is presented as follows:

1) **geometry_loading_3D.m.** This function is used to upload the text files, where the input geometry data is saved. Note that these text files are generated through the discretization done in GID of the CAD designs. In this case, two kinds of text files are uploaded: the text files containing the points of the grain cross-sections and the text files and the text file containing the motor case profile. These text files are the inputs of the functions. With respect to the outputs, several parameters are given, such as the x, y and z coordinates of the cross-sections saved in the vectors “Xgr”, “Ygr” and “Zgr”, a beeper to localize the longitudinal position of the cross-section called “G_loc”, the radius of the motor case called “R_case”, etc. The Matlab® code is found in the section A.2.2.1. geometry_loading_3D.m.

2) **chamber_volume.m.** This function calculates the volume of the combustion chamber. Note that the combustion chamber volume goes from the front end to the nozzle throat. The output of the function is the volume of the combustion chamber itself. The inputs are the values of the
z coordinate and radius of the motor case profile ("Zcp" and "Rcp" respectively). The code of this function is attached in the section A.2.2.2. chamber_volume.m.

3) **curve_tangent_normal_3D.m.** This function calculates the tangent and normal vectors at each point of a given curve. The inputs required by this function are two: the “X_grain” matrix and the “G_loc”. The former is a matrix containing the x, y and z of the cross-sections of the Quasi-3D grain. The latter is a beeper aimed to indicate when begins and ends a cross-section. “G_loc” contains the beginning and ending index of each cross-section. The outputs given by this function are the matrix “Tc” and “Nc”. The matrix “Tc” contains the tangent vector at each point of the given curve, being the first column of the matrix the x components, the second the y components and the third the z components of the tangent vector of the curve at each point. The matrix “Nc” contains the normal vector at each point of the given curve, being the first column of the matrix the x components, the second the y components and the third the z components of the normal vector of the curve at each point. In order to see the code, please consult the section A.2.2.3. curve_tangent_normal_3D.m.

4) **minimum_distance_function_3D.m.** This function implements the formulation of minimum distance function. To understand the mathematical background of this function, please consult the section 4.3.2 of the project report. The input parameters required by this function in this case are five: the matrix of the x coordinate values of the Cartesian grid called “Xcart”, the matrix of the y coordinate values of the Cartesian grid called “Ycart”, the aforementioned beeper the cross-section named as “X_grain”, the matrix of normal vectors at each cross-section point called “Nc”. The output given by this function is the matrix of values obtained from the application of minimum distance function to each point of the Cartesian grid and named as “Dmin”. In order to see the Matlab® code, please, consult the section A.2.2.4. minimum_distance_function_3D.m.

5) **grain_cross_sections_pmt.m.** This function extracts the interface curve from the φ function for each grain cross-sectional. Consequently, for each grain cross-section there is a φ function. So that, this function extracts the interface curve for each time step and calculates the grain geometric parameters; i.e. the burning are, the propellant volume, etc. This way, the input parameters required by this function are the “X” and “Y” matrices of x and y coordinates of the Cartesian grid, the matrix of the φ function called “Phi” and the number of cross-sections called “N_cs”, inter-alia. The outputs given by this function are four: the matrix of the points of
cross-sections called “Xg_cs”, the beeper to localize the position of cross-sections called “Gr_loc”, the burning area known as “A_b” and the propellant volume known as “V”. The mathematical background of this function can be found in the section 4.4.2.3. of the project report. In order to see the Matlab® code, please, consult the section A.2.2.5. grain_cross_sections_pmt.m.

6) rocket_performance_0D_transient_quasi_3D.m. This function gives, for each time step, the internal ballistics of the solid rocket motor with a 0D unsteady flow model but considering the Quasi-3D grain geometry. Therefore, by calculating the internal ballistics the performance of the SRM is obtained. The input parameters of this function are the performance SRM variables calculated at the previous time step and the initial rocket parameters, such as propellant density, parameters of the burning rate law, etc. Then, using these inputs, the function computes the value of the rocket performance variables for the next time step, which are the outputs. For instance, the performance variables are the chamber pressure, chamber temperature, thrust, mass flow, inter-alia. The difference between this function and the analogous for the 2D lies on the way in which the grain geometric parameters are introduced. The mathematical background of this function is explained in the section 5.2. of the project report, although some considerations due to Quasi-3D grain burnback are taken into account. The code of the function can be found in the section A.2.2.6. rocket_performance_0D_transient_quasi_3D.m.

7) grain_3D_plot_matrix.m. This function has been created in order to ease the plotting of the Quasi-3D grain. This function, basically, creates a unique matrix where all of the cross-sections are saved. Additionally, the triangulation between different matrices is performed. For that the add_tri_matrix.m is called. This way, the inputs required by this function are three: the matrix of the points of cross-sections called “Xgrain_cs”, the beeper to localize the position of cross-sections called “Gr_loc” and the number of cross-sections called “N_cs”. The output given by this function are four: the matrix containing all of the cross-sections called “TRI_g” and the corresponding three vectors of x, y and z coordinates called as “Xg”, “Yg” and “Zg”. The code of this function can be found in the section A.2.2.7. grain_3D_plot_matrix.m.

8) motor_case_generation.m. This function generates the 3D surface matrices of the motor case. Then, these surface matrices are used to plot the motor case. For that, three main inputs are introduced in the function: the minimum revolution angle called “theta_min_case”, the maximum revolution angle called “theta_max_case”, the values of the z coordinate called “Zcp” and the radius of the motor case profile called “Rcp”. The
outputs given are the matrices “X_c”, “Y_c” and “Z_c”, which are matrices used to do surf. The code of this function is attached in the section A.2.2.8. motor_case_generation.m.

9) **add_tri_matrix.m.** This function is called inside the function grain_3D_plot_matrix.m. It is used to save in a unique big matrix and three vectors all of the triangles connections and the coordinates of the points of each cross-section. This means that it saves the connection triangles generated by the function tri_generation.m. So, there are three four main outputs: “TRI_g”, “Xg”, “Yg” and “Zg”. The “TRI_g” is big matrix containing the triangle connections and “Xg”, “Yg” and “Zg,” are the vectors where x, y and z coordinates of the points of each cross-section are saved. The inputs required by this function are the “tri” matrix and “X”, “Y” and “Z” vectors given by the tri_generation.m function and the x,y and z points of the two cross-sections studied and saved in the matrices “XA” and “XB”. The code of this function can be found in the section A.2.2.9. add_tri_matrix.m.

10) **case_radius_int.m.** This function interpolates the radius of the motor case when longitudinal z value is given. This way, the inputs parameters required by this function are the values of z coordinate and radius of the motor case profile “Zcp” and “Rcp” respectively, as well as the z coordinate value where it is desired to evaluate the radius named as “z”. The output given is the value of the interpolated radius at the “z” longitudinal position. The code of this function can be found in the section A.2.2.10. case_radius_int.m.

11) **grain_cross_sections.m.** This function is almost the same as the function nº5 grain_cross_sections_pmt.m. In the similar way that the previous one, this function also extracts the interface curve from the φ function for each grain cross-sectional for each time step. The input parameters are also the same as the required in case of the function grain_cross_sections_pmt.m. The only difference between these functions lies on the fact that nº11 function does not give as output the burning area and the propellant volume. Nevertheless, it gives the matrix Xg_cs” and the vector “Gr_loc”. In addition, the mathematical background of this function can be also found in the section 4.4.2. of the project report and the Matlab® code in the section A.2.2.11. grain_cross_sections.m.

12) **grain_video_evolution_3D.m.** This function enables to create a 3D video of the grain burnback. While the output is the video itself, several inputs are required, such as: the matrix of the points of cross-sections called “Xgrain_cs”, the beeper to localize the position of cross-sections called “Gr_loc”, the number of cross-sections called “N_cs”, the geometry
of the motor case, inter-alia. The code of this function is found in the section A.2.2.12. grain_video_evolution_3D.m.

13) **initial_geometry_organization_3D.m.** This function organizes the geometry data provided by the user in text files; so that, the code can handle properly the information. The input vectors are the “Xgr”, which is a vector of x coordinate of the points of the initial geometry curves, “Ygr”, which is a vector of y coordinate of the points of the initial geometry curves and “Zgr”, which is a vector of z coordinate of the points of the initial geometry curves. The outputs of this function are two: the matrix “X_grain” and vector “Z_cs”. In case of matrix “X_grain”, it contains the points of the propellant grain geometry. The points are ordered, so that the curve goes from the beginning to the end. The “X_grain” matrix first column corresponds to the x coordinate, the second column to the y coordinate and the third with the z coordinate. Finally, the vector “Z_cs” contains the z coordinate values of each cross-section. In order to observe the code, please, consult the section A.2.2.13. initial_geometry_organization_3D.m.

14) **interface_tracker_in_P.** This function extracts the zero level set curve but limited to the region defined by the motor case. Hence, if the interface curve goes beyond the motor case, the function only plots the part of the interface belonging to the inside region of the motor case. This way, the outputs given by this function are two: limited zero level set curve called as “Cin” and the perimeter of this curve, called “P”. The required inputs of this function are the “X” and “Y” matrixes of the x and y coordinates of the Cartesian grid, the matrix (“Phi”) and the radius of the motor case (“Rcase”). The mathematical background of this function is explained in the section 4.2. of the project report. In the section A.2.2.14. interface_tracker_in_P.m is presented the code.

15) **interpolate_cross_section.m.** This function generates an intermedium cross-section curve between given cross-sections. Generally, these given two cross-sections are the ones obtained from the CAD design after having discretized them through the GID. So, the output obtained from this function is the “X_cs” matrix which contains the x, y and z coordinates of the intermedium cross-section. Meanwhile, the inputs required by this function are the vectors containing the x, y and z coordinates of both cross-sections (namely, “Xa”, “Ya”, “Za”, “Xb”, “Yb” and “Zb”) and the z coordinate plane where the interpolated cross-section has to be calculates (known as “z_cs”). Note that “z_cs” shall be between the z coordinate of the section A and B. The mathematical background of this function can be found in the 4.4.2.2. of the project report. The Matlab® code is enclosed in the section A.2.2.15. interpolate_cross_section.m.
16) *tri_generation.m*. Given two cross-sections located at a certain distance in the longitudinal axes of the motor, this function generates the triangulation between them. For that the mathematical background explained in the section 4.4.2.2. of the project report should be considered. This way, the inputs required by this function are the x, y and z coordinates of the cross-sections. Note that while the coordinates of the first cross-section are identified by a sub-index “a” (“Xa”, “Ya” and “Za”), for the second cross-section the letter “b” is used (“Xb”, “Yb” and “Zb”). The outputs given are four: the matrix containing the triangles generated and named as “tri” and the vectors of x, y and z coordinates named as “X”, “Y” and “Z”. The code of this function can be found in the section A.2.2.16. *tri_generation.m*.

17) *wetted_surface.m*. This function calculates the wetted surface, which is the surface exposed to the hot gas between two studied cross-sections. This function performs a cross-product to estimate the value of the wetted surface. This function inputs the matrices of x, y and z coordinates of the studied two cross-sections (matrix “Xa” for the first one and “Xb” for the second one). The output given is the value of the wetted area. The code of this function is found in the section A.2.2.17. *wetted_surface.m*.

18) *wetted_surface2.m*. This function calculates the wetted surface, which is the surface exposed to the hot gas between two studied cross-sections. For that, this function performs an arithmetic mean between the perimeters of the two studied cross-sections multiplied by the z distance. This function inputs the matrices of x, y and z coordinates of the studied two cross-sections (matrix “Xa” for the first one and “Xb” for the second one). The output given is the value of the wetted area. The code of this function is found in the section A.2.2.18. *wetted_surface2.m*.

As it has been aforementioned, in the section A.2.2. the Matlab® the codes of the functions here described are included.

**A.2.1. MAIN CODE: General_Quasi_3D_discrete_cart_coord_0D_flow.m**

% STUDY OF GRAIN BURNBACK AND PERFORMANCE OF SOLID ROCKET MOTORS--
%%
% AUTHOR: ARNAU PONS LORENTE
% TUTOR: Dr. RAMON CARRERAS PLANELLS
% CO-TUTOR: Dr. MANEL SORIA GUERRERO
% UNIVERSITY: ETSEIAT-UNIVERSITAT POLITÈCNICA DE CATALUNYA-
% BARCELONA TECH.
% DEFENSE: JANUARY 2013
%%
% QUASI 3D SOLID ROCKET MOTOR GRAIN BURNBACK ANALYSIS AND 0D UNSTEADY FLOW MODEL PERFORMANCE.
clc
clear all
close all

%-------------------------------------------DATA INPUT------------------------------------------
% %
% INPUT PARAMETERS
N=250; %Number of nodes per side in the cartesian grid.
N_Delta_web=60; %Number of contour lines shown in the Phi plot.
N_bs_plot=16; %Number of burnt steps shown in grain evolution plot.
N_cs_i=1; %Number of grain cross-sections interpolated between each reference cross-section.
Time=7.5; %Simulation time [s]
n_steps=900; %Number of simulation time steps.
t_0=0.5; %Time duration of the first phase of the simulation (heavy transient).
Delta_t0=0.005; %Time step during t_0 phase of the simulation.
Delta_t1=(Time-floor(t_0/Delta_t0)*Delta_t0)/(n_steps-floor(t_0/Delta_t0)); %Time step for the rest of simulation.
K_size=8; %Parameter for the size of Xgrain_cs.

%Distance units selection:
units='mm';
%Units conversion:
[K_units] = units_conversion(units);
atm_units=101325; %Conversion factor from Pa to atm.

%Rocket input parameters:
P_c0=1.1E6; %Initial chamber pressure [Pa]
a=2.34015e-5; %a parameter of the propellant burning rate law: \( r = a * P_c^n \)
n_exp=0.36; %n exponent of the propellant burning rate law: \( r = a * P_c^n \)
gamma_e=1.19577; %Combustion products gas gamma parameter
gamma=Cp/Cv
R_air=287; %R of air [J/KgK]
R_e=322.21; %R of the combustion products [J/KgK]
rho_p=1800; %Propellant density [Kg/m^3]
M_mol=0.03875; %Combustion products molar mass [kg/mol]
theta_min_case=90; %Initial angle for the plot of the motor case [º]
theta_max_case=270; %Final angle for the plot of the motor case [º]
P_a=101325; %Atmospheric pressure [Pa]

T_c_v=[2970 2980 2986.15]; %Vector of chamber temperature data relation with chamber pressure [K].
P_c_v=[1 3 6.9]; %Vector of chamber pressure data [MPa].
p_T=polyfit(P_c_v,T_c_v,2); %Polynomial coefficients of the function adjust of \( T_c = f(P_c) \)

%GEOMETRY LOADING:
name_file_grain='NAWC_motor_n6_grain.dat'; %Name of the propellant grain geometry file to be loaded.
name_file_case='NAWC_6_motor_case_profile.dat';  %Name of the motor case geometry file to be loaded.
[Xgr,Ygr,Zgr,G_loc,R_case,A_e,A_t,Zcp,Rcp] =
gallery_loader_3D(name_file_grain,name_file_case);
name_file_v=name_file_grain(1:idx-1);  %File name of the grain burnback video evolution.

[M_e] = mach_calculation(A_e,A_t,gamma_e);   %Nozzle exit Mach number
[Vc_0] = chamber_volume(Zcp,Rcp)*K_units^-3;  %Volume of the combustion chamber in empty configuration (no propellant in) [m^3]

%Assignment of the rocket motor input parameters to a data vector:
rocket_parameters=zeros(1,13);  %rocket_parameters: Vector array of the rocket inputs parameters.
rocket_parameters(1)=P_c0;  %Initial chamber pressure [Pa]
rocket_parameters(2)=a;  %a parameter of the propellant burning rate law: r=a*Pc^n
rocket_parameters(3)=n_exp;  %n exponent of the propellant burning rate law: r=a*Pc^n
rocket_parameters(4)=gamma_e;  %Combustion products gas gamma parameter gamma=Cp/Cv
rocket_parameters(5)=R_air;  %R of air [J/KgK]
rocket_parameters(6)=R_e;  %R of the propellant [J/KgK]
rocket_parameters(7)=rho_p;  %Propellant density [Kg/m^3]
rocket_parameters(8)=M_mol;  %Combustion products molar mass [kg/mol]
rocket_parameters(9)=A_e*K_units^-2;  %Nozzle exit area [m^2]
rocket_parameters(10)=A_t*K_units^-2;  %Nozzle throat area [m^2]
rocket_parameters(11)=M_e;  %Nozzle exit Mach number
rocket_parameters(12)=P_a;  %Atmospheric pressure [Pa]
rocket_parameters(13)=Vc_0;  %Volume of the combustion chamber in empty configuration (no propellant in) [m^3]

%-----------------------------GEOMETRICAL CALCULATIONS-------------------------------------------

%Organization of the initial geometry data given by the user:
[X_grain,Z_cs] =
inital_geometry_organization_3D(Xgr,Ygr,Zgr,G_loc);
N_cs=size(Z_cs,2)+(size(Z_cs,2)-1)*N_cs_i;  %Total number of cross-sections.
%Calculation of tangent and normal vectors of the interface curve:
[Tc,Nc] = curve_tangent_normal_3D(X_grain,G_loc);
%Cartesian square grid generation:
L=1.75*2*R_case;  %Width and height of the cartesian grid square.
Delta_xy=L/N;  %Step size Delta_x=Delta_y=Delta_xy, due to the grid is cartesian.
[X,Y] = cartesian_grid(N,L,Delta_xy);
Delta_web=R_case/N Delta_web;  %Step size of the plotting contours of Phi function.
%------------------------------------CALCULATIONS-----------------------------------%

% Signed Minimum Distance calculation for each grain cross-section:
[Dmin] = minimum_distance_function_3D(X,Y,X_grain,G_loc,Nc);

% Initialization of the Phi function as a signed minimum distance function:
Phi=zeros(N+1,N+1,size(G_loc,2),n_steps+1);
Phi(:,:,1)=Dmin;

% Calculation of the initial grain cross-sections, burning area and propellant volume:
[Xgrain_0,Gr_loc_0,A_b0,V_prop_0] =
grain_cross_sections_pmt(X,Y,Phi(:,:,1),Z_cs,N_cs_i,N_cs,R_case,
K_units);

% Initialization of the Xgrain_cs matrix, which has the points of all the grain cross-sections for each time step:
Xgrain_cs=zeros(K_size*size(X,1)*N_cs,3,n_steps+1);
Gr_loc=zeros(N_cs,n_steps+1);
X_grain_cs(:,:,1)=Xgrain_0;
Gr_loc(:,1)=Gr_loc_0;

Delta_t=zeros(1,n_steps); % Time step vector.
V_time=zeros(1,n_steps+1); % Time vector [s].

% Combustion chamber pressure considering a 0D steady flow model:
P_cm=zeros(n_steps+1,1);
P_cm(1)=P_c0;

% Initialization of rocket performance variables:
M_rocket=zeros(n_steps+1,16); % M_rocket: Matrix array of the rocket performance variables for each time step.

% M_rocket_n(1)=A_b Burning area [m^2]
% M_rocket_n(2)=V_prop Propellant volume [m^3]
% M_rocket_n(3)=r_dot Burning rate [m/s]
% M_rocket_n(4)=Web Burnt depth [m]
% M_rocket_n(5)=m_dot_g Mass production of gas due to combustion [kg/s]
% M_rocket_n(6)=m_dot_d Mass flow rate exiting the nozzle [kg/s]
% M_rocket_n(7)=P_c Combustion chamber pressure [Pa]
% M_rocket_n(8)=Tc Combustion chamber temperature [K]
% M_rocket_n(9)=Vc Combustion chamber volume [m^3]
% M_rocket_n(10)=rho_c Combustion chamber density [kg/m^3]
% M_rocket_n(11)=P_e Nozzle exit pressure [Pa]
% M_rocket_n(12)=V_e Nozzle exit velocity [m/s]
% M_rocket_n(13)=F Thrust [N]
% Assignment of initial values of the rocket performance variables:
M_rocket(1,1)=A_b0;
% Initial burning area [m^2]
M_rocket(1,2)=V_prop_0;
% Initial propellant volume [m^3]
M_rocket(1,3)=a*P_c0^n_exp;
% Initial burning rate [m/s]
M_rocket(1,4)=0;
% Initial burnt depth [m]
M_rocket(1,5)=M_rocket(1,1)*rho_p*M_rocket(1,3);
% Initial mass production of gas due to combustion [kg/s]
M_rocket(1,6)=M_rocket(1,5);
% Initial mass flow rate exiting the nozzle [kg/s]
M_rocket(1,7)=P_c0;
% Initial combustion chamber pressure [Pa]
M_rocket(1,8)=polyval(p_T,P_c0*1e-6);
% Initial combustion chamber temperature [K]
M_rocket(1,9)=rocket_parameters(13)-V_prop_0;
% Initial combustion chamber volume [m^3]
M_rocket(1,10)=0;
% Initial combustion chamber density [kg/m^3]
M_rocket(1,11)=P_c0/(1+(gamma_e-1)/2*M_e^2)^(gamma_e/(gamma_e-1));
% Initial nozzle exit pressure [Pa]
M_rocket(1,12)=(2*gamma_e*R_e*M_rocket(1,8)/(gamma_e-1)*(1-(M_rocket(1,11)/P_c0)^(gamma_e-1)/gamma_e))^((1/2));
% Initial nozzle exit velocity [m/s];
M_rocket(1,13)=M_rocket(1,6)*M_rocket(1,12)+(M_rocket(1,11)-P_a)*rocket_parameters(9);
% Initial thrust [N];
M_rocket(1,14)=rho_p*V_prop_0;
% Initial propellant mass [kg]
M_rocket(1,15)=sqrt(rocket_parameters(6)/M_rocket(1,8))/sqrt(rocket_parameters(4)+1)*(rocket_parameters(4)+1)/(2*(rocket_parameters(4)-1)));
% Initial characteristic velocity [m/s];
M_rocket(1,16)=sqrt(((2*rocket_parameters(4)^2)/(rocket_parameters(4)-1))*((2/(rocket_parameters(4)+1))^((rocket_parameters(4)+1)/(2*(rocket_parameters(4)-1)))));
% Initial thrust coefficient []
for n=1:1:n_steps
% Time loop
if V_time(n)<t_0
\[
\Delta t(n) = \begin{cases} 
\Delta t_0; \\
\Delta t_1; 
\end{cases} 
\]

\[
V_{\text{time}}(n+1) = V_{\text{time}}(n) + \Delta t(n); 
\]

% Rocket Performance Calculation:
\[
[M_{\text{rocket}}(n+1,:)] = 
\text{rocket_performance_0D_transient_quasi_3D}(M_{\text{rocket}}(n,:),rocket \text{ parameters},p_T,K_{\text{units}},\Delta t(n)); 
\]

% Assignment of V burning rate for the Phi evolution:
\[
V = M_{\text{rocket}}(n+1,3) * K_{\text{units}}; 
\]

% Phi evolution for the next time step:
\[
\text{for } k=1:1:\text{size}(G_{\text{loc}},2) \text{ %Reference cross-sections loop} 
\[
[\Phi(:,:,k,n+1)] = 
\text{phi_one_time_step_evolution}(\Phi(:,:,k,n),N,\Delta xy,\Delta t(n),V); 
\]
\end{\text{for}} 
\]

% Calculation of the grain cross-sections and updating the burning area and propellant volume:
\[
[X_{\text{grain}},G_{\text{loc}},M_{\text{rocket}}(n+1,1),M_{\text{rocket}}(n+1,2)] = 
\text{grain_cross_sections_pmt}(X,Y,Phi(:,:,n+1),Z_{\text{cs}},N_{\text{cs}},R_{\text{case}},K_{\text{units}}); 
\]

% Updating the chamber volume, nozzle mass flow:
\[
M_{\text{rocket}}(n+1,9) = \text{rocket_parameters}(13) - M_{\text{rocket}}(n+1,2); 
M_{\text{rocket}}(n+1,6) = \text{rocket_parameters}(7) * M_{\text{rocket}}(n+1,1) * M_{\text{rocket}}(n+1,3); 
\]

% Combustion chamber pressure considering a 0D steady flow model:
\[
P_{\text{cm}}(n+1) = \left(\frac{M_{\text{rocket}}(n+1,1)}{\text{rocket\_parameters}(10)} \right) * \text{rocket\_parameters}(2) * \text{rocket\_parameters}(7) * M_{\text{rocket}}(n+1,15)^{1/(1-\text{rocket\_parameters}(3))}; 
\]

\%
\%
\text{figure}(1); 
Az=50; 
El=20; 
[TRI_g,Xg,Yg,Zg] = 
grain_3D_plot_matrix(X_{\text{grain}},G_{\text{loc}},N_{\text{cs}}); 
trisurf(TRI_g,Yg,Zg,Xg,... 
'\text{FaceColor}', 'interp',... 
'\text{FaceLighting}', 'phong') 
hold on; 
[TRI_g0,Xg0,Yg0,Zg0] = 
grain_3D_plot_matrix(X_{\text{grain}},G_{\text{loc}},N_{\text{cs}}); 
trisurf(TRI_g0,Yg0,Zg0,Xg0,... 
'\text{FaceColor}', 'green',... 
'\text{EdgeColor}', 'none',... 
'\text{FaceLighting}', 'phong') 
hold on; 
[X_c,Y_c,Z_c] = 
motor_case_generation(theta_{\text{min\_case}},theta_{\text{max\_case}},Z_{\text{cp}},R_{\text{cp}}); 
surf(X_c,Z_c,Y_c,... 
'\text{FaceColor}', 'blue',...
'FaceLighting','phong')
hold on;
camlight left
colormap(hot)
text1=['y [' units ']'];
text2=['z [' units ']'];
text3=['x [' units ']'];
xlabel(text1)
ylabel(text2)
zlabel(text3)
view([Az El])
figure(2);
set(2,'Name','Time evolution of A_b, m_dot_g, P_c, m_dot_d');

subplot(2,2,1);
plot(V_time,M_rocket(:,1),'color',[0 0 1])
hold on
xlim([V_time(1) V_time(n_steps+1)]);
xlabel('t [s]')
ylabel('A_b [m^2]')
title('Burning area vs. time')

subplot(2,2,2);
plot(V_time,M_rocket(:,5),'color',[0 0 1])
hold on
xlim([V_time(1) V_time(n_steps+1)]);
xlabel('t [s]')
ylabel('m_d_o_t_g [kg/s]')
title('Mass flow rate due to propellant combustion vs. time')

subplot(2,2,3);
plot(V_time,M_rocket(:,7)/atm_units,'color',[0 0 1])
hold on
xlim([V_time(1) V_time(n_steps+1)]);
xlabel('t [s]')
ylabel('P_c [atm]')
title('Combustion chamber pressure vs. time')

subplot(2,2,4);
plot(V_time,M_rocket(:,6),'color',[0 0 1])
hold on
xlim([V_time(1) V_time(n_steps+1)]);
xlabel('t [s]')
ylabel('m_d_o_t_d [kg/s]')
title('Mass flow rate exiting the nozzle vs. time')

figure(3);
set(3,'Name','Time evolution of r_dot, V_prop, P_cm, T_c');

subplot(2,2,1);
plot(V_time,1000*M_rocket(:,3),'color',[0 0 1])
hold on
xlim([V_time(1) V_time(n_steps+1)]);
xlabel('t [s]')
ylabel('r_d_o_t [mm/s]')
Study of Grain Burnback and Performance of Solid Rocket Motors

title('Burning rate vs. time')

subplot(2,2,1);
plot(V_time,M_rocket(:,2)*1000,'color',[0 0 1])
hold on
xlim([V_time(1) V_time(n_steps+1)]);
xlabel('t [s]')
ylabel('V_p_r_o_p_e_l_l_a_n [L]')
title('Propellant volume versus time ')

subplot(2,2,2);
plot(V_time,P_cm/atm_units,'color',[0 0 1])
hold on
xlim([V_time(1) V_time(n_steps+1)]);
xlabel('t [s]')
ylabel('P_c_m [atm]')
title('Combustion chamber pressure (0D steady flow) versus time ')

subplot(2,2,3);
plot(V_time,M_rocket(:,8),'color',[0 0 1])
hold on
xlim([V_time(1) V_time(n_steps+1)]);
xlabel('t [s]')
ylabel('T_c [K]')
title('Combustion chamber temperature vs. time')

figure(4);
set(4,'Name','Time evolution of T, V_e, rho_c, M_prop')

subplot(2,2,1);
plot(V_time,M_rocket(:,13),'color',[0 0 1])
hold on
xlim([V_time(1) V_time(n_steps+1)]);
xlabel('t [s]')
ylabel('F [N]')
title('Thrust vs. time')

subplot(2,2,2);
plot(V_time,M_rocket(:,12),'color',[0 0 1])
hold on
xlim([V_time(1) V_time(n_steps+1)]);
xlabel('t [s]')
ylabel('V_e_x_i_t [m/s]')
title('Nozzle exit velocity vs. time')

subplot(2,2,3);
plot(V_time,rho_p*M_rocket(:,2),'color',[0 0 1])
hold on
xlim([V_time(1) V_time(n_steps+1)]);
xlabel('t [s]')
ylabel('M_p_r_o_p_e_l_l_a_n_t [kg]')
title('Propellant mass vs. time')

figure(5);
set(5,'Name','Time evolution of c*, C_f, A_b, P_e');

subplot(2,2,1);
plot(V_time,M_rocket(:,15),'color',[0 0 1])
hold on
xlim([V_time(1) V_time(n_steps+1)]);
xlabel('t [s]')
ylabel('c* [m/s]')
c_max=max(M_rocket(:,15));
ylim([0.995*c_max 1.005*c_max])
title('Characteristic velocity vs. time')

subplot(2,2,2);
plot(V_time,M_rocket(:,16),'color',[0 0 1])
hold on
xlim([V_time(1) V_time(n_steps+1)]);
xlabel('t [s]')
ylabel('C_f')
title('Thrust coefficient vs. time')

subplot(2,2,3);
plot(M_rocket(:,4),M_rocket(:,1),'color',[0 0 1])
hold on
xlim([M_rocket(1,4) M_rocket(n_steps+1,4)]);
text2=['Web [ ' units ']'];
xlabel(text2)
ylabel('A_b [m^2]')
title('Burning area vs. burnt depth')

subplot(2,2,4);
plot(V_time,M_rocket(:,11)/atm_units,'color',[0 0 1])
hold on
xlim([V_time(1) V_time(n_steps+1)]);
xlabel('t [s]')
ylabel('P_e_x_i_t [atm]')
title('Nozzle exit pressure vs. time')

A.2.2. FUNCTIONS

The functions presented in this section are listed considering the order given in the Table 2. Additionally, it should be pointed out that the repeated functions of Table 2 are not included again in this section.

A.2.2.1. geometry_loading_3D.m

function [Xgr,Ygr,Zgr,G_loc,R_case,A_e,A_t,Zcp,Rcp] = geometry_loading_3D(name_file_grain,name_file_case)

%UNTITLED Summary of this function goes here
% Detailed explanation goes here
%------------------------PROPELLANT GRAIN GEOMETRY------------------------%
%Loading of the propellant grain geometry:
Mg=load('name_file_grain');
Ng=size(Mg,1);

%X-Coordinate of the propellant grain geometry points.
Xgr=zeros(Ng,1);
%Y-Coordinate of the propellant grain geometry points.
Ygr=zeros(Ng,1);
%Z-Coordinate of the propellant grain geometry points.
Zgr=zeros(Ng,1);

%G_loc: is a vector containing the index number of the Xgr, Ygr and Zgr vectors %where each cross-section ends. It is later used as a reference to know %where one cross-section begins and ends in the Xgr, Ygr and Zgr vectors.

%Fill of the Xgr, Ygr, Zgr and G_loc vector with the data from the propellant %grain geometry file:
i=1;
j=1;
k=1;
while i<Ng
    if Mg(i,1)==0 && Mg(i,2)==0 && Mg(i,3)==0 && Mg(i+1,1)==0 && Mg(i+1,2)==0 && Mg(i+1,3)==0
        G_loc(k)=j-1;
i=i+2;
k=k+1;
    else
        Xgr(j)=Mg(i,1);
        Ygr(j)=Mg(i,2);
        Zgr(j)=Mg(i,3);
i=i+1;
j=j+1;
    end
end

%Unassignment of last rows of the vectors Xgr, Ygr and Zgr which are [0 0 0]:
Cond1=0;
i=Ng;
while Cond1==0
    if Xgr(i)==0 && Ygr(i)==0 && Zgr(i)==0
        Xgr(i)=[];
        Ygr(i)=[];
        Zgr(i)=[];
        Cond1=1;
    else
        Cond1=1;
    end
i=i-1;
end

%--------------------------MOTOR CASE GEOMETRY---------------------------%
%Loading of the motor case geometry:
Mc=load(name_file_case);
N_columns_Mc=size(Mc,2);
switch N_columns_Mc
    case 2
        Zcp=Mc(:,1); %Vector with the z-coordinate of each motor case cross-section.
        Rcp=Mc(:,2); %Vector with the r-coordinate (radius) of each motor case cross-section.
        R_case=Rcp(1);
    case 3
        Zcp=Mc(:,3); %Vector with the z-coordinate of each motor case cross-section.
        Rcp=sqrt(Mc(:,1).^2+Mc(:,2).^2); %Vector with the r-coordinate (radius) of each motor case cross-section.
        R_case=Rcp(1);
end

%Calculation of the area of the nozzle exit and throat:
R_t=1e6;
for i=1:1:size(Rcp,1)
    if Rcp(i)<R_t
        R_t=Rcp(i);
    end
end
A_t=pi*R_t^2;
A_e=pi*Rcp(size(Rcp,1))^2;
end

A.2.2.2. chamber_volume.m

function [Vc] = chamber_volume(Zcp,Rcp)
%UNTITLED Summary of this function goes here
% Detailed explanation goes here
%The combustion chamber volume goes from the front head end to the nozzle %throat.

%Indentify the beginning of the nozzle and the nozzle throat position:
R_t=1e6;
for i=1:1:size(Rcp,1)
    if Rcp(i)==Rcp(1)
        i_0=i;
    end
    if Rcp(i)<R_t
        R_t=Rcp(i);
        i_n=i;
    end
end

Lc=Zcp(i_0)-Zcp(1); %Length of the cylindrical part of the motor %case.
Study of Grain Burnback and Performance of Solid Rocket Motors

\[ V_{c_c} = \pi R_{cp}(1)^2 L_c; \]
\[
\text{Volume of the cylindrical part of the motor}
\]

\[ L_{nc} = Z_{cp}(i_n) - Z_{cp}(i_0); \]
\[
\text{Length of the nozzle convergent part (from the beginning of the nozzle to the throat).}
\]

\[ N_z = 200; \]
\[
\text{Number of divisions in the z-direction.}
\]

\[ \Delta z = L_{nc}/N_z; \]
\[
\text{z spatial step.}
\]

\[ V_{nc} = 0; \]
\[
\text{Volume of the nozzle convergent part (from the beginning of the nozzle to the throat).}
\]

if \( L_{nc} \neq 0 \)
\[
\text{for } i = 1:1:N_z
\]
\[
\text{zi}_m = Z_{cp}(i_0) + (i-1/2) \cdot \Delta z; \]
\[
\text{ri} = \text{case_radius_int}(Z_{cp}, R_{cp}, zi_m); \]
\[
V_{nc} = V_{nc} + \pi \cdot ri^2 \cdot \Delta z;
\]
end
end

\[ V_{c} = V_{c_c} + V_{nc}; \]
end

A.2.2.2. curve_tangent_normal_3D.m

function \([Tc, Nc] = \text{curve_tangent_normal}_3D(X_{grain}, G_{loc})\)
\[
\text{curve_tangent_normal: this function calculates the tangent and normal vectors of a given curve at each curve point.}
\]

\%
\text{The inputs of this function is:}
\%
\text{Xi: the points of the initial grain geometry cross-sections, ordered.}
\%
\text{The first column is the x coordinate of the ith curve point, the}
\%
\text{second column is the y coordinate of the ith curve point, and the}
\%
\text{third column is the z component.}
\%
\text{The outputs of this function are:}
\%
\text{Tc: the tangent vector at each point of 'Xi'. The first column is the}
\%
\text{x component of the tangent vector to the curve at the ith point, the}
\%
\text{second column is the y component and the third column is the}
\%
\text{z component.}
\%
\text{Nc: the normal vector at each point of 'Xi'. The first column is the}
\%
\text{x component of the tangent vector to the curve at the ith point, the}
\%
\text{second column is the y component and the third column is the}
\%
\text{z component.}
\%
N = \text{size}(X_{grain}, 1);
% Initialization of Tc and Nc.
Tc=zeros(N,3); % Tangent vector at each point of the curve.
Nc=zeros(N,3); % Normal vector at each point of the curve.

% Filling of X_grain:
for j=1:1:size(G_loc,2)
    if j==1
        i_0=1;
    else
        i_0=G_loc(j-1)+1;
    end
    Xi=[X_grain(i_0:G_loc(j),1) X_grain(i_0:G_loc(j),2)];
    % Calculation of the tangent and normal vectors at each curve point:
    [Tci,Nci] = curve_tangent_normal(Xi);
    Vz=zeros(size(Xi,1),1);
    Tc(i_0:G_loc(j),:)=Tci;
    Nc(i_0:G_loc(j),:)=Nci;
end

A.2.2.4. minimum_distance_function_3D.m

function [Dmin] = minimum_distance_function_3D(Xcart,Ycart,X_grain,G_loc,Nc)
% UNTITLED4 Summary of this function goes here
% Detailed explanation goes here

N=size(Xcart,1)-1;
% Initialization of Dmin matrix.
Dmin=zeros(N+1,N+1,size(G_loc,2));

for l=1:1:size(G_loc,2) % loop for all the grain cross-sections
    if l==1
        i_0=1;
    else
        i_0=G_loc(l-1)+1;
    end
    for i=1:1:N+1 % y direction loop of the cartesian grid
        for j=1:1:N+1 % x direction loop of the cartesian grid
            Dmin(i,j,l)=1e6;
            for k=i_0:1:G_loc(l) % loop for all the interface curves points
                D=sqrt((Xcart(i,j)-X_grain(k,1))^2+(Ycart(i,j)-X_grain(k,2))^2);
                if D<Dmin(i,j,l)
                    Dmin(i,j,l)=D;
                    Pos=k;
                end
            end
            Vcart=[Xcart(i,j) Ycart(i,j)];
            condicio=dot(Vcart-X_grain(Pos,1:2),Nc(Pos,1:2));
            if condicio<0
                Dmin(i,j,l)=-Dmin(i,j,l);
            end
        end
    end
end

99
end
end
end

A.2.2.5. grain_cross_sections_pmt.m

function [Xg_cs,Gr_loc,A_b,V_prop] =
grain_cross_sections_pmt(X,Y,Phi,Z_cs,N_cs_i,N_cs,R_case,K_units)
%UNTITLED Summary of this function goes here
% Detailed explanation goes here

Gr_loc=zeros(N_cs,1);
N_Xg_cs=8*size(X,1)*N_cs;
Xg_cs=zeros(N_Xg_cs,3); %Matrix with [x y z] coordinates
for the points
%of all the interpolated cross-
sections.

%Filling of the Xg_cs matrix with the reference cross-sections of
%grain given by the Phi function, and the interpolated cross-
sections
%between each reference cross-section. Also, calculation of the
%burning
%area A_b and the propellant volume V_p_in:

A_b=0;
V_p_in=0;
N_in_t=0;
k=1;
for i=1:size(Z_cs,2)-1 %i-th reference cross-section:
    [Cin1,P_1] = interface_tracker_in_P(X,Y,Phi(:,:,i),R_case);
    [A_p1] = port_area(X,Y,Phi(:,:,i),R_case);
    N_in1=size(Cin1,1);
    Vz_1=Z_cs(i)*ones(N_in1,1);
    %i+1-th reference cross-section:
    [Cin2,P_2] = interface_tracker_in_P(X,Y,Phi(:,:,i+1),R_case);
    [A_p2] = port_area(X,Y,Phi(:,:,i+1),R_case);
    N_in2=size(Cin2,1);
    Vz_2=Z_cs(i+1)*ones(N_in2,1);
    %Filling the Xg_cs matrix with the points of the i-th reference
%cross-section:
    i_0=N_in_t+1;
    N_in_t=N_in1+N_in_t;
    Xg_cs(i_0:N_in_t,:)=[Cin1 Vz_1];
    Gr_loc(k)=N_in_t;
    k=k+1;
    Lz=Z_cs(i+1)-Z_cs(i);
    Delta_z=Lz/(N_cs_i+1);
    %Calculation of the wet surface between the previous
interpolated
    %cross-section and the cross-section A:
    if i>1
        if P_1==0 && P_csi~=0
            delta_z2=Z_cs(i)-Z_cs_ant;
            A_b=P_csi*delta_z2+A_b; %Burning area.
elseif P_csi==0 && P_1~=0
delta_z2=Z_cs(i)-z_cs_ant;
A_b=P_1*delta_z2+A_b;  % Burning area.
else
   P_m=(P_csi+P_1)/2;    % Mean perimeter
delta_z2=Z_cs(i)-z_cs_ant;
A_b=P_m*delta_z2+A_b;  % Burning area.
end

end

% Calculation of the port volume between the cross-sections A and B:
A_pm=(A_p1+A_p2)/2;    % Mean port area, between the cross-sections A and B
V_p_in=A_pm*Lz+V_p_in;
for j=1:1:N_cs_i
   % Calculation of the z-coordinate of the interpolated section:
   z_cs=Z_cs(i)+j*Delta_z;
   % Interpolation of a cross-section between the cross-sections A and B:
   [X_csi] = interpolate_cross_section(Cin1(:,1),Cin1(:,2),Vz_1,Cin2(:,1),Cin2(:,2),Vz_2,z_cs);
P_csi=perimeter_cross_section(X_csi(:,1:2),R_case);
   % Calculation of the wet surface between the two considered cross-sections:
   if P_1==0 && P_csi~=0
      delta_z2=z_cs-z_cs_ant;
      A_b=P_csi*delta_z2+A_b;  % Burning area.
   elseif P_csi==0 && P_1~=0
      delta_z2=z_cs-z_cs_ant;
      A_b=P_1*delta_z2+A_b;  % Burning area.
   else
      P_m=(P_csi+P_1)/2;    % Mean perimeter
delta_z2=z_cs-z_cs_ant;
      A_b=P_m*delta_z2+A_b;  % Burning area.
   end
   z_cs_ant=z_cs;
P_1=P_csi;
end

% Assignment of the last cross-section:
if i==size(Z_cs,2)-1
   i_0=N_in_t+1;
   N_in_t=N_in_t+N_csi;
   Xg_cs(i_0:N_in_t,:)=X_csi;
   Gr_loc(k)=N_in_t;
k=k+1;
end
delta_z2=Z_cs(size(Z_cs,2))-z_cs_ant;
A_b=P_1*delta_z2+A_b;
elseif P_1<2*pi*R_case && P_2<2*pi*R_case
P_m=(P_1+P_2)/2;  %Mean perimeter
delta_z2=Z_cs(size(Z_cs,2))-z_cs_ant;
A_b=P_m*delta_z2+A_b;
end
end
end

%Convert the units of A_b and V_p_in so that A_b is in [m^2] and
V_p_in is
%[m^3]:
A_b=A_b*K_units^2;
V_prop=(pi*R_case^2*(Z_cs(size(Z_cs,2))-Z_cs(1))-V_p_in)*K_units^-3;
end

A.2.2.6. rocket_performance_0D_transient_quasi_3D.m

function [M_rocket_next] = rocket_performance_0D_transient_quasi_3D(M_rocket_n,rocket_parameters,p_T,K_units,Delta_t)
%UNTITLED Summary of this function goes here
% Detailed explanation goes here

%%
%INPUT PARAMETERS:
%M_rocket_n: Vector array of the rocket performance variables at
%the n time
%step.
%M_rocket_n(1)=A_b          Burning area [m^2]
%M_rocket_n(2)=V_prop       Propellant volume [m^3]
%M_rocket_n(3)=r_dot        Burning rate [m/s]
%M_rocket_n(4)=Web          Burnt depth [m]
%M_rocket_n(5)=m_dot_g      Mass production of gas due
combustion [kg/s]
%M_rocket_n(6)=m_dot_d      Mass flow rate exiting the nozzle[kg/s]
%M_rocket_n(7)=P_c          Combustion chamber pressure [Pa]
%M_rocket_n(8)=Tc           Combustion chamber temperature [K]
%M_rocket_n(9)=Vc           Combustion chamber volume [m^3]
%M_rocket_n(10)=rho_c       Combustion chamber density [kg/m^3]
%M_rocket_n(11)=P_e         Nozzle exit pressure [Pa]
%M_rocket_n(12)=V_e         Nozzle exit velocity [m/s]
%M_rocket_n(13)=F           Thrust [N]
%M_rocket_n(14)=m_p         Propellant mass [kg]
%M_rocket_n(15)=c_star      Characteristic velocity [m/s]
%M_rocket_n(16)=C_f         Thrust coefficient []

%rocket_parameters: Vector array of the rocket inputs parameters.
%rocket_parameters(1)=P_c0;       %Initial chamber pressure [Pa]
%rocket_parameters(2)=a;           %a parameter of the
propellant burning rate law: r=a*Pc^n
%rocket_parameters(3)=n_exp; %n exponent of the propellant burning rate law: \( r = a \cdot P_c^n \)
%rocket_parameters(4)=gamma_e; %Combustion products gas gamma parameter gamma=Cp/Cv
%rocket_parameters(5)=R_air; %R of air [J/KgK]
%rocket_parameters(6)=R_e; %R of the propellant [J/KgK]
%rocket_parameters(7)=rho_p; %Propellant density [Kg/m^3]
%rocket_parameters(8)=M_mol; %Combustion products molar mass [kg/mol]
%rocket_parameters(9)=A_e; %Nozzle exit area [m^2]
%rocket_parameters(10)=A_t; %Nozzle throat area [m^2]
%rocket_parameters(11)=M_e; %Nozzle exit Mach number
%rocket_parameters(12)=P_a; %Atmospheric pressure [Pa]
%rocket_parameters(13)=Vc_0; %Volume of the combustion chamber in empty configuration (no propellant in) [m^3]

%Initialization of M_rocket_next vector:
M_rocket_next=zeros(1,16);

%Calculation of the rocket performance variables:
Gamma_C=sqrt(rocket_parameters(4))*(2/(rocket_parameters(4)+1))^{(rocket_parameters(4)+1)/(2*(rocket_parameters(4)-1))};
c_n=sqrt(rocket_parameters(6)*M_rocket_n(8))/Gamma_C;
%The interface burning area and propellant volume are calculated later
%because those depend on the Phi matrix evolution:
M_rocket_next(1)=M_rocket_n(1);
M_rocket_next(2)=M_rocket_n(2);

%Calculation of the combustion chamber pressure:
M_rocket_next(7)=M_rocket_n(7)+((Gamma_C^2*c_n^2)/M_rocket_n(9))*((rocket_parameters(7)-M_rocket_n(10))*M_rocket_n(1)*M_rocket_n(3)-((rocket_parameters(10)*M_rocket_n(7))/c_n))*Delta_t;

%Calculation of the next performance variables derivated from the combustion chamber pressure P_c:
M_rocket_next(3)=rocket_parameters(2)*M_rocket_next(7)^rocket_parameters(3);
M_rocket_next(4)=M_rocket_next(3)*K_units*Delta_t+M_rocket_n(4);
M_rocket_next(5)=rocket_parameters(7)*M_rocket_n(1)*M_rocket_next(3);
M_rocket_next(8)=polyval(p_T,M_rocket_next(7)*1e-6);
M_rocket_next(15)=sqrt(rocket_parameters(6)*M_rocket_next(8))/Gamma_C;
M_rocket_next(6)=(M_rocket_next(7)*rocket_parameters(10))/M_rocket_next(15);
M_rocket_next(9)=rocket_parameters(13)-M_rocket_n(2);
M_rocket_next(10)=M_rocket_next(7)/(M_rocket_next(15)^2*Gamma_C^2);
M_rocket_next(11)=M_rocket_next(7)/(1+(rocket_parameters(4)-1)/2*rocket_parameters(11)^2*(rocket_parameters(4)/(rocket_parameters(4)-1)));
M_rocket_next(12)=((2*rocket_parameters(4)*rocket_parameters(6)*M_rocket_next(8))/(rocket_parameters(4)-1))*(1-
\[(M_{\text{rocket\_next}}(11)/M_{\text{rocket\_next}}(7))^{((\text{rocket\_parameters}(4)-1)/\text{rocket\_parameters}(4)))^{(1/2)};\]  
\[M_{\text{rocket\_next}}(16)=\sqrt{((2*\text{rocket\_parameters}(4)^2)/(\text{rocket\_parameters}(4)-1))*((2/(\text{rocket\_parameters}(4)+1))^((\text{rocket\_parameters}(4)+1)/(\text{rocket\_parameters}(4)-1)))*(1-(M_{\text{rocket\_next}}(11)/M_{\text{rocket\_next}}(7))^{((\text{rocket\_parameters}(4)-1)/\text{rocket\_parameters}(4))})+(M_{\text{rocket\_next}}(11)-\text{rocket\_parameters}(12))/M_{\text{rocket\_next}}(7))*(\text{rocket\_parameters}(9)/\text{rocket\_parameters}(10));\]  
\[M_{\text{rocket\_next}}(13)=M_{\text{rocket\_next}}(6)*M_{\text{rocket\_next}}(15)*M_{\text{rocket\_next}}(16);\]  
\[M_{\text{rocket\_next}}(14)=M_{\text{rocket\_n}}(14)-M_{\text{rocket\_n}}(5)*\Delta t;\]

\[\]

A.2.2.7. grain_3D_plot_matrix.m

\[\]
Study of Grain Burnback and Performance of Solid Rocket Motors

\[
t = (\theta_{\text{min\_case}} \times \pi / 180):0.05:(\theta_{\text{max\_case}} \times \pi / 180); \\
N_t = \text{size}(t, 2); \\
N_c = \text{size}(Zcp, 1); \\
X_c = \text{zeros}(N_c, N_t); \\
Y_c = \text{zeros}(N_c, N_t); \\
Z_c = \text{zeros}(N_c, N_t); \\
\]

for \( i = 1:1:N_c \)
  for \( j = 1:1:N_t \)
    \( X_c(i, j) = Rcp(i) \times \cos(t(j)); \)
    \( Y_c(i, j) = Rcp(i) \times \sin(t(j)); \)
    \( Z_c(i, j) = Zcp(i); \)
  end
end

\section*{A.2.2.9. \texttt{add\_tri\_matrix.m}}

\begin{verbatim}
function [TRI_g,Xg,Yg,Zg]=add_tri_matrix(tri,TRI_gin,Xg_in,Yg_in,Zg_in,XA,XB)
%UNTITLED Summary of this function goes here
% Detailed explanation goes here
if TRI_gin==0
  %This case happens for the first adding.
  TRI_g=tri;
  Na=size(XA,1);
  Nb=size(XB,1);
  Xg=zeros(Na+Nb,1);
  Yg=zeros(Na+Nb,1);
  Zg=zeros(Na+Nb,1);
  Xg(1:Na)=XA(:,1);
  Xg(Na+1:Na+Nb)=XB(:,1);
  Yg(1:Na)=XA(:,2);
  Yg(Na+1:Na+Nb)=XB(:,2);
  Zg(1:Na)=XA(:,3);
  Zg(Na+1:Na+Nb)=XB(:,3);
else
  %This case happens for the rest of the addings.
  Na=size(XA,1);
  Nb=size(XB,1);
  N_Xgin=size(Xg_in,1);
  N_TRI_gin=size(TRI_gin,1);
  N_tri=size(tri,1);
  Xg=zeros(N_Xgin+Nb,1);
  Yg=zeros(N_Xgin+Nb,1);
  Zg=zeros(N_Xgin+Nb,1);
  Xg(1:N_Xgin)=Xg_in;
  Xg(N_Xgin+1:N_Xgin+Nb)=XB(:,1);
  Yg(1:N_Xgin)=Yg_in;
  Yg(N_Xgin+1:N_Xgin+Nb)=XB(:,2);
  Zg(1:N_Xgin)=Zg_in;
  Zg(N_Xgin+1:N_Xgin+Nb)=XB(:,3);
  TRI_g=zeros(N_TRI_gin+N_tri,3);
  TRI_g(1:N_TRI_gin,:)=TRI_gin;
  V_N1=(N_Xgin-Na)*ones(size(tri,1),3);
end
\end{verbatim}
TRI_g(N_TRI_gin+1:N_TRI_gin+N_tri,:) = tri+V_N1;
end

A.2.2.10. case_radius_int.m

function [r_case] = case_radius_int(Zcp, Rcp, z)
%UNTITLED2 Summary of this function goes here
% Detailed explanation goes here
for i=1:size(Zcp,1)
  if z<=Zcp(i)
    i_0=i;
    break;
  end
end

if i_0==1
  i_0=2;
end

if Rcp(i_0)==Rcp(i_0-1) && Rcp(i_0)==Rcp(i_0+1)
  r_case=Rcp(i_0);
elseif Rcp(i_0)==Rcp(i_0-1) && Rcp(i_0)==Rcp(i_0+1) &&
  Zcp(i_0)==Zcp(i_0+1)
  A=[Zcp(i_0-2) Zcp(i_0-1) 1; Zcp(i_0-1)^2 Zcp(i_0+1) 1; Zcp(i_0+2)^2 Zcp(i_0+2) 1];
  b=[Rcp(i_0-2) Rcp(i_0-1) Rcp(i_0)];
  p2=inv(A)*b;
  r_case=p2(1)*z^2+p2(2)*z+p2(3);
elseif Rcp(i_0)==Rcp(i_0-1) && Rcp(i_0)==Rcp(i_0+1) &&
  Zcp(i_0)==Zcp(i_0-1)
  r_case=Rcp(i_0);
elseif i_0==size(Zcp,1)
  A=[Zcp(i_0-2)^2 Zcp(i_0-1) 1; Zcp(i_0-1)^2 Zcp(i_0-1) 1; Zcp(i_0)^2 Zcp(i_0) 1];
  b=[Rcp(i_0-2) Rcp(i_0-1) Rcp(i_0)];
  p2=inv(A)*b;
  r_case=p2(1)*z^2+p2(2)*z+p2(3);
else
  A=[Zcp(i_0-1)^2 Zcp(i_0-1) 1; Zcp(i_0)^2 Zcp(i_0) 1; Zcp(i_0+1)^2 Zcp(i_0+1) 1];
  b=[Rcp(i_0-1) Rcp(i_0) Rcp(i_0+1)];
  p2=inv(A)*b;
  r_case=p2(1)*z^2+p2(2)*z+p2(3);
end
end

A.2.2.11. grain_cross_sections.m

function [Xg_cs, Gr_loc] =
grain_cross_sections(X, Y, Phi, Z_cs, N_cs_i, N_cs, R_case)
%UNTITLED Summary of this function goes here
% Detailed explanation goes here
Gr_loc=zeros(1,N_cs);
N_Xg_cs=8*size(X,1)*N_cs;
Xg_cs=zeros(N_Xg_cs,3); %Matrix with [x y z] coordinates
for the points %of all the interpolated cross-sections.
%Filling of the Xg_cs matrix with the reference cross-sections of the
%grain given by the Phi function, and the interpolated cross-sections
%between each reference cross-section:
N_in_t=0;
k=1;
for i=1:size(Z_cs,2)-1 %i-th reference cross-section:
    Cin1 = interface_tracker_in(X,Y,Phi(:,:,i),R_case);
    N_in1=size(Cin1,1);
    Vz_1=Z_cs(i)*ones(N_in1,1);
    %i+1-th reference cross-section:
    Cin2 = interface_tracker_in(X,Y,Phi(:,:,i+1),R_case);
    N_in2=size(Cin2,1);
    Vz_2=Z_cs(i+1)*ones(N_in2,1);
    %Filling the X_g_s matrix with the points of the i-th reference
    %cross-section: i_0=N_in_t+1;
    N_in_t=N_in1+N_in_t;
    Xg_cs(i_0:N_in_t,:)=Cin1 Vz_1;
    Gr_loc(k)=N_in_t;
    k=k+1;
    Lz=Z_cs(i+1)-Z_cs(i);
    if N_cs_i>0
        Delta_z=Lz/(N_cs_i+1);
        for j=1:1:N_cs_i %Calculation of the z-coordinate of the interpolated section:
            z_cs=Z_cs(i)+j*Delta_z;
            %Interpolation of a cross-section between the cross-sections A and B:
            X_csi =
            interpolate_cross_section(Cin1(:,1),Cin1(:,2),Vz_1,Cin2(:,1),Cin2(:,2),Vz_2,z_cs);
            N_csi=size(X_csi,1);
            i_0=N_in_t+1;
            N_in_t=N_csi+N_in_t;
            %Filling the X_g_s matrix with the points of the j-th interpolated
            %cross-section between the cross-sections A and B:
            Xg_cs(i_0:N_in_t,:)=X_csi;
            Gr_loc(k)=N_in_t;
            k=k+1;
        end
    end
end %Assignment of the last cross-section:
if i=size(Z_cs,2)-1
    i_0=N_in_t+1;
    N_in_t=N_in2+N_in_t;
    Xg_cs(i_0:N_in_t,:)=Cin2 Vz_2;
    Gr_loc(k)=N_in_t;
end
%Unassignment of last rows of the matrix Xg_cs which are [0 0 0]:
Cond1=0;
i=N_Xg_cs;
while Cond1==0
    if Xg_cs(i,1)==0 && Xg_cs(i,2)==0 && Xg_cs(i,3)==0
        Xg_cs(i,:)=[];
    else
        Cond1=1;
    end
    i=i-1;
end
end

A.2.2.12. grain_video_evolution_3D.m

function [M] = grain_video_evolution_3D(Xgrain_cs,Gr_loc,N_cs,theta_min_case,theta_max_case,L,Zcp,Rcp,n_frames,n_times,fps,name_file_v,units,Az,El,axis_style)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
M=moviein(n_frames);

set(gca,'nextplot','replacechildren');
n_steps=size(Gr_loc,2);
delta_i=floor(n_steps/n_frames);
[X_c,Y_c,Z_c] =motor_case_generation(theta_min_case,theta_max_case,Zcp,Rcp);
for i=1:n_frames+1
    hold off;
    [TRI_g,Xg,Yg,Zg] = grain_3D_plot_matrix(Xgrain_cs(:,,(i-1)*delta_i+1),Gr_loc(:,(i-1)*delta_i+1),N_cs);
    trisurf(TRI_g,Yg,Zg,Xg,
            'FaceColor','interp',
            'FaceLighting','phong')
    hold on;
camlight left
colormap(hot)
surf(X_c,Z_c,Y_c,
     'FaceColor','blue',
     'EdgeColor','black',
     'FaceLighting','phong')
    hold on;
text1=['y [' units ']'];
text2=['z [' units ']'];
text3=['x [' units ']'];
xlabel(text1)
ylabel(text2)
zlabel(text3)
axis(axis_style)
axis([-0.75*L/2 0.75*L/2 0 1.05*Zcp(size(Zcp,1)) -0.75*L/2 0.75*L/2]);
title('Grain evolution during rocket operation')
view([Az El])
M(:,i)=getframe(gcf);
end

if n_times>1
M1=[M M M]; %Save the movie frames 3 times in order to repeat
else %movie 3 times.
M1=M;
end
movie(M,n_times,fps);
name_file_movie=[name_file_v '.avi'];
movie2avi(M1,name_file_movie,'fps',fps,'compression','None');
end

A.2.2.13. initial_geometry_organization_3D.m

function [X_grain,Z_cs] =
initial_geometry_organization_3D(Xgr,Ygr,Zgr,G_loc)
%initial_geometry_organization: this function organizes the
geometry data
%given by the user in a .dat file so that the code can handle
properly
%this information.

%X_grain: the points of the propellant grain geometry. The
points are
%ordered, so that the curve goes from the beginning to the
end. 
%The first column is the x coordinate of the ith curve
point,
%the second column is the y coordinate of the ith curve
point and
%the third column is the z coordinate of the points.

%Initialization of the X_grain matrix:
X_grain=zeros(size(Xgr,1),3);
Z_cs=zeros(1,size(G_loc,2));

%Filling of X_grain:
k=1;
for j=1:size(G_loc,2)
    if j==1
        i_0=1;
    else
        i_0=G_loc(j-1)+1;
    end
\[
X_i = [X_{gr(i_0:G_loc(j))}; Y_{gr(i_0:G_loc(j))}];
\]

% Applying the `order_vector_points` function in order to assure that the
% cross-sections points are correctly ordered:
[Xord] = order_vector_points(Xi);

X_grain(i_0:G_loc(j),:) = [Xord Z_{gr(i_0:G_loc(j))}];

Z_cs(k) = Z_{gr(i_0)};

k = k + 1;

end
end

A.2.2.14. interface_tracker_in_P.m

```
function [Cin, P] = interface_tracker_in_P(X, Y, Phi, Rcase)
% Interface tracker2: this function calculates the points of the
% interface given the cartesian grid and the Phi matrix.
% Detailed explanation goes here

dx = diff(Y(1:2, 1));
Delta_xy = dx;
N = size(X, 1);
epsilon = 1e-6;

%%
% First step: find a interface point and set it as the first point
% of the curve:
% Vertical scan from the y=-L/2 and x=0 to y=L/2 and x=0.

j_cc = floor((N + 1) / 2);
for i = 1:N-1
    Phi_1 = Phi(i, j_cc);
    Phi_2 = Phi(i + 1, j_cc);
    Phi_3 = Phi(i + 1, j_cc + 1);
    Phi_4 = Phi(i, j_cc + 1);
    X_1 = X(i, j_cc);
    Y_1 = Y(i, j_cc);
    X_2 = X(i + 1, j_cc);
    Y_2 = Y(i + 1, j_cc);
    X_3 = X(i + 1, j_cc + 1);
    Y_3 = Y(i + 1, j_cc + 1);
    X_4 = X(i, j_cc + 1);
    Y_4 = Y(i, j_cc + 1);

    if Phi_1 * Phi_4 < 0 && Phi_2 * Phi_3 < 0
        % CASE 1
        Xa_0 = X_1 + (abs(Phi_1) / (abs(Phi_1) + abs(Phi_4))) * Delta_xy;
        Xb_0 = X_2 + (abs(Phi_2) / (abs(Phi_2) + abs(Phi_3))) * Delta_xy;
        Ya_0 = Y_1;
        Yb_0 = Y_2;
        i_0 = i;
        j_0 = j_cc;
        case_id = 1;
        break;
    elseif Phi_1 * Phi_2 < 0 && Phi_1 * Phi_4 < 0
        % CASE 2
        Xa_0 = X_1 + (abs(Phi_1) / (abs(Phi_1) + abs(Phi_4))) * Delta_xy;
        Xb_0 = X_1;
        Ya_0 = Y_1;
        Yb_0 = Y_1 + (abs(Phi_1) / (abs(Phi_1) + abs(Phi_4))) * Delta_xy;
        i_0 = i;
```

---

110
\( j_0 = j_{cc}; \)
\( \text{case id} = 2; \)
\( \text{break}; \)
\( \text{elseif } \Phi_1*\Phi_4 < 0 \&\& \Phi_3*\Phi_4 < 0 \quad \% \text{CASE 3} \)
\( X_0 = X_1 + \frac{\text{abs}(\Phi_1)}{\text{abs}(\Phi_1)+\text{abs}(\Phi_4)}*\Delta_{xy}; \)
\( X_b = X_4; \)
\( Y_0 = Y_1; \)
\( Y_b = Y_4 + \frac{\text{abs}(\Phi_4)}{\text{abs}(\Phi_4)+\text{abs}(\Phi_3)}*\Delta_{xy}; \)
\( i_0 = i; \)
\( j_0 = j_{cc}; \)
\( \text{case id} = 3; \)
\( \text{break}; \)
\( \text{elseif } \Phi_1*\Phi_2 < 0 \&\& \Phi_3*\Phi_4 < 0 \quad \% \text{CASE 4} \)
\( X_a = X_1; \)
\( X_b = X_4; \)
\( Y_a = Y_1 + \frac{\text{abs}(\Phi_1)}{\text{abs}(\Phi_1)+\text{abs}(\Phi_2)}*\Delta_{xy}; \)
\( Y_b = Y_4 + \frac{\text{abs}(\Phi_4)}{\text{abs}(\Phi_4)+\text{abs}(\Phi_3)}*\Delta_{xy}; \)
\( i_0 = i; \)
\( j_0 = j_{cc}; \)
\( \text{case id} = 4; \)
\( \text{break}; \)
\( \text{elseif } \Phi_1*\Phi_2 < 0 \&\& \Phi_2*\Phi_3 < 0 \quad \% \text{CASE 5} \)
\( X_a = X_1; \)
\( X_b = X_2 + \frac{\text{abs}(\Phi_2)}{\text{abs}(\Phi_2)+\text{abs}(\Phi_3)}*\Delta_{xy}; \)
\( Y_a = Y_1 + \frac{\text{abs}(\Phi_1)}{\text{abs}(\Phi_1)+\text{abs}(\Phi_2)}*\Delta_{xy}; \)
\( Y_b = Y_2; \)
\( i_0 = i; \)
\( j_0 = j_{cc}; \)
\( \text{case id} = 5; \)
\( \text{break}; \)
\( \text{elseif } \Phi_2*\Phi_3 < 0 \&\& \Phi_3*\Phi_4 < 0 \quad \% \text{CASE 6} \)
\( X_a = X_4; \)
\( X_b = X_2 + \frac{\text{abs}(\Phi_2)}{\text{abs}(\Phi_2)+\text{abs}(\Phi_3)}*\Delta_{xy}; \)
\( Y_a = Y_4 + \frac{\text{abs}(\Phi_4)}{\text{abs}(\Phi_4)+\text{abs}(\Phi_2)}*\Delta_{xy}; \)
\( Y_b = Y_2; \)
\( i_0 = i; \)
\( j_0 = j_{cc}; \)
\( \text{case id} = 6; \)
\( \text{break}; \)
\( \text{else} \)
\( X_a = 0; \)
\( X_b = 0; \)
\( Y_a = 0; \)
\( Y_b = 0; \)
\( \text{case id} = 0; \)
\( \text{end} \)

%%
\% If the vertical scan does not find the interface, then a horizontal scan
\% shall be done:
\if case id==0
\( i_{cc} = \text{floor}((N+1)/2); \)
\for j=1:N-1
\( \Phi_1=\Phi(i_{cc},j); \)
\( \Phi_2=\Phi(i_{cc}+1,j); \)
\( \Phi_3=\Phi(i_{cc}+1,j+1); \)
\( \Phi_4=\Phi(i_{cc},j+1); \)
\endfor
\fi
\begin{verbatim}
X1=X(i_cc,j);
Y1=Y(i_cc,j);
X2=X(i_cc+1,j);
Y2=Y(i_cc+1,j);
X3=X(i_cc+1,j+1);
Y3=Y(i_cc+1,j+1);
X4=X(i_cc,j+1);
Y4=Y(i_cc,j+1);
if Phi1*Phi4<0 && Phi2*Phi3<0 %CASE 1
  Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
  Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
  Ya_0=Y1;
  Yb_0=Y2;
  i_0=i_cc;
  j_0=j;
  case_id=1;
end

elseif Phi1*Phi2<0 && Phi1*Phi4<0 %CASE 2
  Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
  Xb_0=X1;
  Ya_0=Y1;
  Yb_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
  i_0=i_cc;
  j_0=j;
  case_id=2;
end

elseif Phi1*Phi4<0 && Phi3*Phi4<0 %CASE 3
  Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
  Xb_0=X4;
  Ya_0=Y1;
  Yb_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
  i_0=i_cc;
  j_0=j;
  case_id=3;
end

elseif Phi1*Phi2<0 && Phi3*Phi4<0 %CASE 4
  Xa_0=X1;
  Xb_0=X4;
  Ya_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
  Yb_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
  i_0=i_cc;
  j_0=j;
  case_id=4;
end

elseif Phi2*Phi3<0 && Phi3*Phi4<0 %CASE 5
  Xa_0=X1;
  Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
  Ya_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
  Yb_0=Y2;
  i_0=i_cc;
  j_0=j;
  case_id=5;
end

elseif Phi2*Phi3<0 && Phi3*Phi4<0 %CASE 6
  Xa_0=X4;
  Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
  Ya_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi2)))*Delta_xy;
  Yb_0=Y2;
  i_0=i_cc;
end
\end{verbatim}
j_0=j;
case_id=6;
break;
else
Xa_0=0;
Xb_0=0;
Ya_0=0;
Yb_0=0;
case_id=-1;
end
end

%%
%If the vertical and horizontal scan do not find the interface, then a
%overall scan shall be done:
if case_id== -1
for i=1:1:N-1
for j=1:1:N-1
Phi1=Phi(i,j);
Phi2=Phi(i+1,j);
Phi3=Phi(i+1,j+1);
Phi4=Phi(i,j+1);
X1=X(i,j);
Y1=Y(i,j);
X2=X(i+1,j);
Y2=Y(i+1,j);
X3=X(i+1,j+1);
Y3=Y(i+1,j+1);
X4=X(i,j+1);
Y4=Y(i,j+1);
if Phi1*Phi4<0  &&  Phi2*Phi3<0  %CASE 1
Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
Ya_0=Y1;
Yb_0=Y2;
i_0=i;
j_0=j;
case_id=1;
break;
elseif Phi1*Phi2<0  &&  Phi1*Phi4<0  %CASE 2
Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
Xb_0=X1;
Ya_0=Y1;
Yb_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
i_0=i;
j_0=j;
case_id=2;
break;
elseif Phi1*Phi4<0  &&  Phi3*Phi4<0  %CASE 3
Xa_0=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi4)))*Delta_xy;
Xb_0=X4;
Ya_0=Y1;
Yb_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
i_0=i;
j_0=j;
case_id=3;
break;
else
Xa_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
Xb_0=X2;
Ya_0=Y2;
Yb_0=Y2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
i_0=i;
j_0=j;
case_id=2;
break;
end
end
end
end
end
elseif Phi1*Phi2<0 && Phi3*Phi4<0  %CASE 4
Xa_0=X1;
Xb_0=X4;
Ya_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
Yb_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
i_0=i;
j_0=j;
case_id=4;
break);
elseif Phi1*Phi2<0 && Phi2*Phi3<0  %CASE 5
Xa_0=X1;
Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
Ya_0=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
Yb_0=Y4;
i_0=i;
j_0=j;
case_id=5;
break);
elseif Phi2*Phi3<0 && Phi3*Phi4<0  %CASE 6
Xa_0=X4;
Xb_0=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
Ya_0=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi2)))*Delta_xy;
Yb_0=Y2;
i_0=i;
j_0=j;
case_id=6;
break);
else
Xa_0=0;
Xb_0=0;
Ya_0=0;
Yb_0=0;
i_0=i;
j_0=j;
case_id=0;
end
end
end

%%
%INTERFACE SCANNING AND DETERMINATION
closed=0;
k=1;
P=0;
l=2;
while closed==0  %The condition closed means the closing of the
%interface curve over itself.
%When closed is equal to 1 then the curve has
%the while loop can be stopped.
%Assignment of the intial point of the interface curve.
if k=1
  i_x=i_0;
j_x=j_0;
switch case_id

case {1,3}
case_id_A=1;
C(1,1)=Xa_0;
C(1,2)=Ya_0;
case {4,5}
case_id_A=2;
C(1,1)=Xa_0;
C(1,2)=Ya_0;
case 2
case_id_A=2;
C(1,1)=Xb_0;
C(1,2)=Yb_0;
case 6
case_id_A=3;
C(1,1)=Xb_0;
C(1,2)=Yb_0;
otherwise
case_id_A=0;
C(1,1)=0;
C(1,2)=0;
end
C2(1,1)=C(1,1);
C2(1,2)=C(1,2);
C2(1,3)=0;
end

%Assignment of the A point as the B point calculated in the previous iteration (A(k)=B(k-1)).
Xa=C(k,1);
Ya=C(k,2);
i=i_x;
scanned.
j=j_x;
scanned.
X1=X(i,j);
to be scanned.
Y1=Y(i,j);
to be scanned.
X2=X(i+1,j);
to be scanned.
Y2=Y(i+1,j);
to be scanned.
X3=X(i+1,j+1);
to be scanned.
Y3=Y(i+1,j+1);
to be scanned.
X4=X(i,j+1);
to be scanned.
Y4=Y(i,j+1);
to be scanned.
Phi1=Phi(i,j);
to be scanned.
Phi2=Phi(i+1,j);
to be scanned.
Phi3=Phi(i+1,j+1);
to be scanned.
Phi4=Phi(i,j+1);
to be scanned.
%Identification of a square which has the interface passing through it
%twice:
if Phi1*Phi2<0 && Phi2*Phi3<0 && Phi3*Phi4<0 && Phi1*Phi4<0
%Condition of interface double crossing the square.
V=C(k,:)-C(k-1,:); %Vector of the interface ongoing direction in the k point.

switch case_id_A
    case 1 %A is located in the lower edge of the square.
        Xd_2=X1;
        Yd_2=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
        Xd_4=X4;
        Yd_4=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
        a=[(Xd_2-Xa) (Yd_2-Ya)];
        b=[(Xd_4-Xa) (Yd_4-Ya)];
        Va=dot(V,a);
        Vb=dot(V,b);
        if Va>Vb %B is located in the left edge of the square.
            Xb=Xd_2;
            Yb=Yd_2;
            case_v=4; %Location of the next iteration point A (4=right edge of the square).
            i_x=i;
            j_x=j-1;
            %Node 1 i component of the square to be scanned in the next step.
            %Node 1 j component of the square to be scanned in the next step.
        else
            Xb=Xd_4;
            Yb=Yd_4;
            case_v=2; %Location of the next iteration point A (2=left edge of the square).
            i_x=i;
            j_x=j+1;
            %Node 1 i component of the square to be scanned in the next step.
            %Node 1 j component of the square to be scanned in the next step.
    end
    case 2 %A is located in the left edge of the square.
        Xd_1=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
        Yd_1=Y1;
        Xd_3=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
        Yd_3=Y2;
        a=[(Xd_1-Xa) (Yd_1-Ya)];
        b=[(Xd_3-Xa) (Yd_3-Ya)];
        Va=dot(V,a);
        Vb=dot(V,b);
        if Va>Vb %B is located in the lower edge of the square.
            Xb=Xd_1;
            Yb=Yd_1;
            case_v=3; %Location of the next iteration point A (3=upper edge of the square).
i_x=i-1; %Node 1 i component of the square to be scanned in the next step.  
j_x=j; %Node 1 j component of the square to be scanned in the next step.
else

Xb=Xd_3;  
Yb=Yd_3;  
case v=1; %Location of the next iteration point A (1=lower edge of the square).
i_x=i+1; %Node 1 i component of the square to be scanned in the next step.  
j_x=j; %Node 1 j component of the square to be scanned in the next step.
end

case 3 %A is located in the upper edge of the square.
Xd_2=X1;  
Yd_2=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;  
Xd_4=X4;  
Yd_4=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;  
a=[(Xd_2-Xa) (Yd_2-Ya)];  
b=[(Xd_4-Xa) (Yd_4-Ya)];  
Va=dot(V,a);  
Vb=dot(V,b);  
if Va>Vb %B is located in the left edge of the square.
Xb=Xd_2;  
Yb=Yd_2;  
case v=4; %Location of the next iteration point A (4=right edge of the square).
i_x=i; %Node 1 i component of the square to be scanned in the next step.  
j_x=j-1; %Node 1 j component of the square to be scanned in the next step.
else
Xb=Xd_4;  
Yb=Yd_4;  
case v=2; %Location of the next iteration point A (2=left edge of the square).
i_x=i; %Node 1 i component of the square to be scanned in the next step.  
j_x=j+1; %Node 1 j component of the square to be scanned in the next step.
end

case 4 %A is located in the right edge of the square.
Xd_1=X1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;  
Yd_1=Y1;  
Xd_3=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;  
Yd_3=Y2;  
a=[(Xd_1-Xa) (Yd_1-Ya)];  
b=[(Xd_3-Xa) (Yd_3-Ya)];  
Va=dot(V,a);  
Vb=dot(V,b);  
if Va>Vb %B is located in the lower edge of the square.
Xb=Xd_1;  
Yb=Yd_1;
Study of Grain Burnback and Performance of Solid Rocket Motors

```matlab
% Location of the next iteration point A (3=upper edge of the square).
case_v=3;
i_x=i-1;
j_x=j;

% Location of the next iteration point A (3=upper edge of the square).
end

% Location of the next iteration point A (1=lower edge of the square).
else
    Xb=Xd_3;
    Yb=Yd_3;
    case_v=1;

% Location of the next iteration point A (1=lower edge of the square).
end

otherwise
    Xb=0;
    Yb=0;
    case_v=-1;

% Location of the next iteration point A (-1=error).
end
else
    switch case_id_A
% Location of the square edge where the A point is.
    case 1
% A is located in the lower edge of the square.
        if Phi1*Phi2<0
% B is located in the left edge of the square.
            Xb=X1;
            Yb=Y1+(abs(Phi1)/(abs(Phi1)+abs(Phi2)))*Delta_xy;
            case_v=4;
        end

% Location of the square edge where the A point is.
    elseif Phi2*Phi3<0
% B is located in the upper edge of the square.
            Xb=X2+(abs(Phi2)/(abs(Phi2)+abs(Phi3)))*Delta_xy;
            Yb=Y2;
            case_v=1;

% Location of the square edge where the A point is.
    elseif Phi3*Phi4<0
% B is located in the right edge of the square.
            Xb=X4;
            Yb=Y4+(abs(Phi4)/(abs(Phi4)+abs(Phi3)))*Delta_xy;
    end
```
case_v=2; %Location of the next iteration point A (2=left edge of the square).
i_x=i; %Node 1 i component of the square to be scanned in the next step.
\[ j_x=j+1; \] %Node 1 j component of the square to be scanned in the next step.
end

case 2 %A is located in the left edge of the square.
if Phi1*Phi4<0 %B is located in the lower edge of the square.
\[ X_b=X_1+(abs(\Phi_1)/(abs(\Phi_1)+abs(\Phi_4)))*\Delta_{xy}; \]
\[ Y_b=Y_1; \]
case_v=3; %Location of the next iteration point A (3=upper edge of the square).
i_x=i-1; %Node 1 i component of the square to be scanned in the next step.
\[ j_x=j; \] %Node 1 j component of the square to be scanned in the next step.
elseif Phi2*Phi3<0 %B is located in the upper edge of the square.
\[ X_b=X_2+(abs(\Phi_2)/(abs(\Phi_2)+abs(\Phi_3)))*\Delta_{xy}; \]
\[ Y_b=Y_2; \]
case_v=1; %Location of the next iteration point A (1=lower edge of the square).
i_x=i+1; %Node 1 i component of the square to be scanned in the next step.
\[ j_x=j; \] %Node 1 j component of the square to be scanned in the next step.
elseif Phi3*Phi4<0 %B is located in the right edge of the square.
\[ X_b=X_4; \]
\[ Y_b=Y_4+(abs(\Phi_4)/(abs(\Phi_4)+abs(\Phi_3)))*\Delta_{xy}; \]
case_v=2; %Location of the next iteration point A (2=left edge of the square).
i_x=i; %Node 1 i component of the square to be scanned in the next step.
\[ j_x=j+1; \] %Node 1 j component of the square to be scanned in the next step.
end

case 3 %A is located in the upper edge of the square.
if Phi1*Phi4<0 %B is located in the lower edge of the square.
\[ X_b=X_1+(abs(\Phi_1)/(abs(\Phi_1)+abs(\Phi_4)))*\Delta_{xy}; \]
\[ Y_b=Y_1; \]
case_v=3; %Location of the next iteration point A (3=upper edge of the square).
i_x=i-1; %Node 1 i component of the square to be scanned in the next step.
\[ j_x=j; \] %Node 1 j component of the square to be scanned in the next step.
elseif Phi1*Phi2<0 %B is located in the left edge of the square.
\[ X_b=X_1; \]
\[ Y_b=Y_1+(abs(\Phi_1)/(abs(\Phi_1)+abs(\Phi_2)))*\Delta_{xy}; \]
case_v=4; %Location of the next iteration point A (4=right edge of the square).
i_x=i; %Node 1 i component of the square to be scanned in the next step.
\[ j_x = j - 1; \]  % Node 1 j component of the square to be scanned in the next step.

\text{elseif} \ \Phi_i^3 * \Phi_i^4 < 0 \quad \% B is located in the right edge of the square.
\begin{align*}
X_b &= X_4; \\
Y_b &= Y_4 + (\frac{\text{abs}(\Phi_i^4)}{\text{abs}(\Phi_i^4) + \text{abs}(\Phi_i^3)}) \times \Delta_{xy}; \\
\text{case} \ v &= 2; \\
\text{end}
\end{align*}

\text{point A (2=left edge of the square).}
\begin{align*}
i_x &= i; \\
j_x &= j + 1; \\
\text{case} \ v &= 2; \\
\text{end}
\end{align*}

\text{Location of the next iteration point A (2=left edge of the square).}
\begin{align*}
i_x &= i - 1; \\
j_x &= j; \\
\text{else-if} \ \Phi_i^1 * \Phi_i^4 < 0 \quad \% B is located in the lower edge of the square.
\begin{align*}
X_b &= X_1 + (\frac{\text{abs}(\Phi_i^1)}{\text{abs}(\Phi_i^1) + \text{abs}(\Phi_i^4)}) \times \Delta_{xy}; \\
Y_b &= Y_1; \\
\text{case} \ v &= 3; \\
\text{end}
\end{align*}

\text{point A (3=upper edge of the square).}
\begin{align*}
i_x &= i; \\
j_x &= j - 1; \\
\text{else-if} \ \Phi_i^1 * \Phi_i^2 < 0 \quad \% B is located in the upper edge of the square.
\begin{align*}
X_b &= X_2 + (\frac{\text{abs}(\Phi_i^2)}{\text{abs}(\Phi_i^2) + \text{abs}(\Phi_i^3)}) \times \Delta_{xy}; \\
Y_b &= Y_2; \\
\text{case} \ v &= 1; \\
\text{end}
\end{align*}

\text{point A (1=lower edge of the square).}
\begin{align*}
i_x &= i + 1; \\
j_x &= j; \\
\text{else-if} \ \Phi_i^2 * \Phi_i^3 < 0 \quad \% B is located in the right edge of the square.
\begin{align*}
X_b &= X_1; \\
Y_b &= Y_1 + (\frac{\text{abs}(\Phi_i^1)}{\text{abs}(\Phi_i^1) + \text{abs}(\Phi_i^4)}) \times \Delta_{xy}; \\
\text{case} \ v &= 4; \\
\text{end}
\end{align*}

\text{point A (4=right edge of the square).}
\begin{align*}
i_x &= i; \\
j_x &= j - 1; \\
\text{else-if} \ \Phi_i^1 * \Phi_i^2 < 0 \quad \% B is located in the upper edge of the square.
\begin{align*}
X_b &= X_2; \\
Y_b &= Y_2 + (\frac{\text{abs}(\Phi_i^2)}{\text{abs}(\Phi_i^2) + \text{abs}(\Phi_i^3)}) \times \Delta_{xy}; \\
\text{case} \ v &= 1; \\
\text{end}
\end{align*}

\text{point A (1=lower edge of the square).}
\begin{align*}
i_x &= i + 1; \\
j_x &= j; \\
\text{else-if} \ \Phi_i^2 * \Phi_i^3 < 0 \quad \% B is located in the right edge of the square.
\begin{align*}
X_b &= X_1; \\
Y_b &= Y_1; \\
\text{case} \ v &= -1; \\
\text{end}
\end{align*}

\text{point A (-1=error).}
\begin{align*}
i_x &= 0; \\
j_x &= 0; \\
\text{end}
\end{align*}

\text{end end}
%Assignment of the B point to the interface curve.
C(k+1,1)=Xb;
C(k+1,2)=Yb;
case_id_A=case_v; %Assignment of the case_id_A for the
next iteration.
dif=sqrt((Xb-C(1,1))^2+(Yb-C(1,2))^2);

%Assignment of only the points inside or frontier of the motor
case:
Ra=sqrt(Xa^2+Ya^2);
Rb=sqrt(Xb^2+Yb^2);
if Ra>=Rcase && Rb>=Rcase
di=0;
elseif Ra<Rcase && Rb<Rcase %The interface is inside the
motor case.
di=sqrt((Xa-Xb)^2+(Ya-Yb)^2);
C2(1,1)=C(k+1,1);
C2(1,2)=C(k+1,2);
C2(1,3)=0;
l=l+1;
elseif Ra<Rcase && Rb>Rcase || Ra>Rcase && Rb<Rcase %The
interface is in the frontier of the motor case.
a=(Xb-Xa)^2+(Yb-Ya)^2;
b=2*Xa*(Xb-Xa)+2*Ya*(Yb-Ya);
c=Xa^2+Ya^2-Rcase^2;
Disc=b^2-4*a*c;
if Disc>=0
s1=(-b+sqrt(Disc))/(2*a);
s2=(-b-sqrt(Disc))/(2*a);
if s1>=0 && s1<=1
s=s1;
x=Xa+(Xb-Xa)*s;
y=Ya+(Yb-Ya)*s;
if Ra<Rcase && Rb>Rcase
di=sqrt((x-Xa)^2+(y-Ya)^2);
elseif Ra>Rcase && Rb<Rcase
di=sqrt((x-Xb)^2+(y-Yb)^2);
end
elseif s2>=0 && s2<=1
s=s2;
x=Xa+(Xb-Xa)*s;
y=Ya+(Yb-Ya)*s;
if Ra<Rcase && Rb>Rcase
di=sqrt((x-Xa)^2+(y-Ya)^2);
elseif Ra>Rcase && Rb<Rcase
di=sqrt((x-Xb)^2+(y-Yb)^2);
end
else
x=0;
y=0;
di=0;
end
C2(l,1)=x;
C2(l,2)=y;
%Identification of the intersection point of the interface with the motor case.
l=1+1;
end
P=di+P;

if dif==0
closed=1;
elseif dif<epsilon
closed=2;
else
closed=0;
end

k=k+1;
end

%Check whether the first point of C2 is inside the motor case or not:
if norm(C2(1,:))>Rcase
C2(1,:)=[];
end

%%
%Generation of the interface curve accounting only the points inside the motor case:
if size(C2,1)<3 %The interface is totally outside the motor case.
%Perimeter:
P=0; %The perimeter is then 0, because all the propellant of this section has burnt.
%Interface points:
Na=500;
Delta_theta=(2*pi)/Na;
k=1;
for j=1:1:Na+1
    Theta_j=(j-1)*Delta_theta;
    Cin(k,1)=Rcase*cos(Theta_j);
    Cin(k,2)=Rcase*sin(Theta_j);
k=k+1;
end
else %The interface is partially or totally inside the motor case.
    k=1;
    Na=100; %Number of nodes of the segment in the motor case between interface curve escape and return.
    for i=1:1:size(C2,1)
        if i<size(C2,1) && C2(i,3)==1 && C2(i+1,3)==1
            Theta_max=atan2(C2(i+1,2),C2(i+1,1));
            Theta_min=atan2(C2(i,2),C2(i,1));
            r_max=[C2(i+1,1) C2(i+1,2) 0];
            r_min=[C2(i,1) C2(i,2) 0];
            if cross(r_min,r_max)*[0 0 1]'>0 %The interface goes in counter clockwise sense.
                if Theta_max<Theta_min
                    Theta=2*pi-abs(Theta_max)-abs(Theta_min);
                end
            end
        end
    end
end
else
    Theta=abs(Theta_max-Theta_min);
end
else
    Theta=Theta_max-Theta_min;
end
%The interface goes in clockwise sense.
if Theta_max>Theta_min
    Theta=abs(Theta_max)+abs(Theta_min)-2*pi;
else
    Theta=Theta_max-Theta_min;
end
Delta_theta=Theta/Na;
for \(j=1:1:Na+1\)
    Theta_j=Theta_min+(j-1)*Delta_theta;
    Cin(k,1)=Rcase*cos(Theta_j);
    Cin(k,2)=Rcase*sin(Theta_j);
    k=k+1;
end
elseif i==size(C2,1) && C2(i,3)==1 && C2(1,3)==1
    Theta_max=atan2(C2(1,2),C2(1,1));
    Theta_min=atan2(C2(i,2),C2(i,1));
    r_max=[C2(1,1) C2(1,2) 0];
    r_min=[C2(i,1) C2(i,2) 0];
    if cross(r_min,r_max)*[0 0 1]'>0
        %The interface goes in counter clockwise sense.
        if Theta_max<Theta_min
            Theta=2*pi-abs(Theta_max)-abs(Theta_min);
        else
            Theta=abs(Theta_max-Theta_min);
        end
    else
        %The interface goes in clockwise sense.
        if Theta_max>Theta_min
            Theta=abs(Theta_max)+abs(Theta_min)-2*pi;
        else
            Theta=Theta_max-Theta_min;
        end
    end
end
Delta_theta=Theta/Na;
for \(j=1:1:Na+1\)
    Theta_j=Theta_min+(j-1)*Delta_theta;
    Cin(k,1)=Rcase*cos(Theta_j);
    Cin(k,2)=Rcase*sin(Theta_j);
    k=k+1;
end
else
    Cin(k,1)=C2(i,1);
    Cin(k,2)=C2(i,2);
    k=k+1;
end
end
end


**A.2.2.15. interpolate_cross_section.m**

```matlab
function [X_cs] =
    interpolate_cross_section(Xa,Ya,Za,Xb,Yb,Zb,z_cs)
%UNTITLED Summary of this function goes here
% Detailed explanation goes here

%z_cs is the z-coordinate plane where the interpolated cross-
%section has to
%calculated. z_cs is between the z coordinate of the section A and
%B.

X1=[Xa Ya];
X2=[Xb Yb];
N1=size(X1,1);
N2=size(X2,1);
Vo=zeros(N1,2);
Cond1=0;
if N1==N2
    if X1-X2==Vo
        %Case in which the two cross-section
        %geometries are equal.
        Cond1=1;
        Vz=z_cs*ones(N1,1);
        X_cs=[X1 Vz];
    end
end
if Cond1==0
    %Case in which the two cross-section geometries
    %are different.
    %First, the triangulation between the two cross-sections is
    %computed:
    [tri,Xt,Yt,Zt] = tri_generation(Xa,Ya,Za,Xb,Yb,Zb);
    Na=size(Xa,1)-1;
    X_cs=zeros(size(tri,1),3);
    for i=1:1:size(tri,1)
        a=tri(i,1);
        b=tri(i,2);
        c=tri(i,3);
        if a<=Na && b>Na && c<=Na
            a=tri(i,1);
            b=tri(i,2);
            c=tri(i,3);
        elseif a>Na && b>Na && c<=Na
            a=tri(i,3);
            b=tri(i,1);
            c=tri(i,2);
        elseif a<=Na && b>Na && c>Na
            a=tri(i,1);
            b=tri(i,2);
            c=tri(i,3);
        end
        Xia=[Xt(a) Yt(a) Zt(a)];
        Xib=[Xt(b) Yt(b) Zt(b)];
    end
    X_cs=[Xa Vz; Xb Vz];
end
```

---

124
\[ s = \frac{z_{cs} - X_{ia}(3)}{X_{ib}(3) - X_{ia}(3)}; \]
\[ X_{cs}(i,:) = X_{ia} + (X_{ib} - X_{ia}) \times s; \]

\section*{A.2.2.16. \textit{tri\_generation.m}}

\begin{verbatim}
function [tri,X,Y,Z] = tri_generation(Xa,Ya,Za,Xb,Yb,Zb)
%UNTITLED Summary of this function goes here
% Detailed explanation goes here
N_a=size(Xa,1);
N_b=size(Xb,1);
Xa(N_a)=[]; %Unassign the last value because it is repeated
           % (it is equal to the start point).
Ya(N_a)=[]; %Unassign the last value because it is repeated
           % (it is equal to the start point).
Za(N_a)=[]; %Unassign the last value because it is repeated
           % (it is equal to the start point).
Xb(N_b)=[]; %Unassign the last value because it is repeated
           % (it is equal to the start point).
Yb(N_b)=[]; %Unassign the last value because it is repeated
           % (it is equal to the start point).
Zb(N_b)=[]; %Unassign the last value because it is repeated
           % (it is equal to the start point).
N_a=size(Xa,1); %Re-compute the size of Xa.
N_b=size(Xb,1); %Re-compute the size of Xb.
X=zeros(N_a+N_b,1);
Y=zeros(N_a+N_b,1);
Z=zeros(N_a+N_b,1);
%Construction of the X,Y and Z vectors:
X(1:N_a)=Xa;
X(N_a+1:N_a+N_b)=Xb;
Y(1:N_a)=Ya;
Y(N_a+1:N_a+N_b)=Yb;
Z(1:N_a)=Za;
Z(N_a+1:N_a+N_b)=Zb;
k=1;
for i=1:1:N_a
    Dmin1=1e6;
    Dmin2=1e6;
    Cond1=0;
    if i<N_a
        for j=1:1:N_b
            D1=sqrt((Xa(i)-Xb(j))^2+(Ya(i)-Yb(j))^2);
            D2=sqrt((Xa(i+1)-Xb(j))^2+(Ya(i+1)-Yb(j))^2);
            if D1<Dmin1
                Dmin1=D1;
                Pos1=j;
            end
            if D2<Dmin2
                Dmin2=D2;
                Pos2=j;
            end
            if D1<Dmin1
                Dmin1=D1;
                Pos1=j;
            end
            if D2<Dmin2
                Dmin2=D2;
                Pos2=j;
            end
            if Cond1==0
                Cond1=1;
                Dmin1=D1;
                Dmin2=D2;
                Pos1=j;
                Pos2=j;
            end
        end
    end
end
\end{verbatim}
end
if D2<Dmin2
Dmin2=D2;
Pos2=j;
end
end
else
for j=1:1:N_b
D1=sqrt((Xa(i)-Xb(j))^2+(Ya(i)-Yb(j))^2);
if D1<Dmin1
Dmin1=D1;
Pos1=j;
end
end
Pos2=Pos1;
end
if Pos1==Pos2
Cond1=1;
end
n_up=N_a+Pos1;
if i==1
n_up_ant=n_up;
end
if n_up<N_a+N_b && i<N_a
if Cond1==0 && n_up==n_up_ant
tri(k,:)=i n_up i+1;
tri(k+1,:)=n_up n_up+1 i+1;
n_up_ant=n_up+1;
k=k+2;
elseif n_up==n_up_ant+1
tri(k,:)=i n_up_ant i+1;
tri(k+1,:)=n_up_ant n_up_ant+1 i+1;
n_up_ant=n_up_ant+1;
k=k+2;
elseif n_up>n_up_ant+2
while n_up_ant<n_up
tri(k,:)=i n_up_ant n_up_ant+1;
n_up_ant=n_up_ant+1;
k=k+1;
end
tri(k,:)=i n_up i+1;
k=k+1;
n_up_ant=n_up;
else
tri(k,:)=i n_up i+1;
n_up_ant=n_up;
k=k+1;
end
elseif n_up==N_a+N_b && i<N_a
if Cond1==0 && n_up==n_up_ant
tri(k,:)=i n_up i+1;
tri(k+1,:)=n_up N_a+1 i+1;
n_up_ant=n_up+1;
k=k+2;
else if n_up==n_up_ant+1
tri(k,:)=[i n_up_ant i+1];
tri(k+1,:)=[n_up_ant n_up_ant+1 i+1];
n_up_ant=n_up_ant+1;
k=k+2;
else if n_up>=n_up_ant+2
    while n_up_ant<n_up
        tri(k,:)=[i n_up_ant n_up_ant+1];
        n_up_ant=n_up_ant+1;
        k=k+1;
    end
    tri(k,:)=[i n_up i+1];
k=k+1;
n_up_ant=n_up;
else
    tri(k,:)=[i n_up i+1];
n_up_ant=n_up;
k=k+1;
end
else if i==N_a
    if n_up==N_a+N_b
        if n_up==n_up_ant+1
            tri(k,:)=[i n_up_ant n_up];
k=k+1;
        elseif n_up>=n_up_ant+2
            while n_up_ant<n_up
                tri(k,:)=[i n_up_ant n_up_ant+1];
                n_up_ant=n_up_ant+1;
                k=k+1;
            end
        end
        tri(k,:)=[i n_up i+1];
        if n_up==N_a+1
            tri(k+1,:)=[n_up N_a+1 1];
        end
    elseif n_up<N_a+N_b
        if Cond1==0 && n_up==n_up_ant
            while n_up_ant<N_a+N_b
                tri(k,:)=[i n_up_ant n_up_ant+1];
                n_up_ant=n_up_ant+1;
                k=k+1;
            end
        end
        tri(k,:)=[i N_a+N_b 1];
        tri(k+1,:)=[N_a+N_b N_a+1 1];
    else
        tri(k,:)=[i n_up i+1];
    end
    end
end
end
end
A.2.2.17. wetted_surface.m

```matlab
function [Sw] = wetted_surface(XA,XB)
%UNTITLED2 Summary of this function goes here
% Detailed explanation goes here

%First, the triangulation between the two cross-sections is computed:
[tri,Xt,Yt,Zt] =
tri_generation(XA(:,1),XA(:,2),XA(:,3),XB(:,1),XB(:,2),XB(:,3));

Sw=0;
for i=1:size(tri,1)
a=tri(i,1);
b=tri(i,2);
c=tri(i,3);
Xia=[Xt(a) Yt(a) Zt(a)];
Xib=[Xt(b) Yt(b) Zt(b)];
Xic=[Xt(c) Yt(c) Zt(c)];
Sw=(1/2)*norm(cross(Xib-Xia,Xic-Xia))+Sw;
end
end
```

A.2.2.18. wetted_surface2.m

```matlab
function [Sw] = wetted_surface2(XA,XB,R_case)
%UNTITLED2 Summary of this function goes here
% Detailed explanation goes here

Ca=[XA(:,1) XA(:,2)];
Cb=[XB(:,1) XB(:,2)];
Pa=perimeter_cross_section(Ca,R_case); %Perimeter of the cross-section A
Pb=perimeter_cross_section(Cb,R_case); %Perimeter of the cross-section B
Pm=(Pa+Pb)/2; %Mean perimeter
Delta_z=abs(XB(1,3)-XA(1,3));
Sw=Pm*Delta_z;
end
```
A.3. **CODE OF QUASI-3D GRAIN BURNBACK AND 1D QUASI-STeady FLOW SIMULATION**

This section is aimed to explain deeply the section 6.4.3 of the project report. This way, in the following Table 3 the names of the different Matlab® files used in the implementation of the Quasi-3D grain burnback and 1D quasi-steady flow simulation are identified. It should be pointed out that while there is a unique file corresponding to the main code, there are a total of 28 files used to implement 28 independent functions. Furthermore, in Table 3 the main code has also been identified with number 0 and the functions have been numbered from 1 to 28. Note, that not all of the functions are directly called from the main code; nevertheless, they are also necessary to perform the simulation. That is why they are included in the Table 3 too. It should be pointed out that in order to make videos, the function nº17 grain_video_evolution_3D.m should be called from the main code. Finally, note that all of the file’s names end with “.m” because they are Matlab® files.

<table>
<thead>
<tr>
<th>Nº</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Main Code</td>
<td>General_Quasi_3D_flow_1D_discrete_cart_coord.m</td>
</tr>
<tr>
<td>1</td>
<td>Function</td>
<td>dif_A_p.m</td>
</tr>
<tr>
<td>2</td>
<td>Function</td>
<td>grain_cross_sections_pmt_1D.m</td>
</tr>
<tr>
<td>3</td>
<td>Function</td>
<td>interpolate_bt_cs.m</td>
</tr>
<tr>
<td>4</td>
<td>Function</td>
<td>nozzle_inlet_mach.m</td>
</tr>
<tr>
<td>5</td>
<td>Function</td>
<td>p_u_rho_calc.m</td>
</tr>
<tr>
<td>6</td>
<td>Function</td>
<td>perimeter_cross_section.m</td>
</tr>
<tr>
<td>7</td>
<td>Function</td>
<td>rocket_performance_1D_steady_Q3D.m</td>
</tr>
<tr>
<td>8*</td>
<td>Function</td>
<td>add_tri_matrix.m</td>
</tr>
<tr>
<td>9*</td>
<td>Function</td>
<td>cartesian_grid.m</td>
</tr>
<tr>
<td>10*</td>
<td>Function</td>
<td>case_radius_int.m</td>
</tr>
<tr>
<td>11*</td>
<td>Function</td>
<td>chamber_volume.m</td>
</tr>
<tr>
<td>12*</td>
<td>Function</td>
<td>curve_tangent_normal.m</td>
</tr>
<tr>
<td>13*</td>
<td>Function</td>
<td>curve_tangent_normal_3D.m</td>
</tr>
<tr>
<td>14*</td>
<td>Function</td>
<td>geometry_loading_3D.m</td>
</tr>
<tr>
<td>15*</td>
<td>Function</td>
<td>grain_3D_plot_matrix.m</td>
</tr>
<tr>
<td>16*</td>
<td>Function</td>
<td>grain_video_evolution_3D.m</td>
</tr>
</tbody>
</table>
Some of the functions of the Table 3 are repeated in the Table 1 as well as in the Table 2; precisely, all of the functions that have a star (***) in the number are repeated (functions located just at the end of the Table 3). In order to verify it, please consult the introduction of the sections A.1. and A.2. This way, the function nº8 is attached in section A.2.2.9. add_tri_matrix.m; the function nº9 in the section A.1.2.4. cartesian_grid.m; the function nº10 in the section A.2.2.10. case_radius_int.m; the function nº11 in the section A.2.2.2. chamber_volume.m; the function nº12 in the section A.1.2.3. curve_tangent_normal.m; the function nº13 in the section A.2.2.3. curve_tangent_normal_3D.m; the function nº14 in the section A.2.2.1. geometry_loading_3D.m; the function nº15 in the section A.2.2.7. grain_3D_plot_matrix.m; the function nº16 in the section A.2.2.12. grain_video_evolution_3D.m; the function nº17 in the section A.2.2.13. initial_geometry_organization_3D.m; the function nº18 in the section A.2.2.14. interface_tracker_in_P.m; the function nº19 in the section A.2.2.15. interpolate_cross_section.m; the function nº20 in the section A.1.2.13. mach_calculation.m; the function nº21 in the section A.2.2.4. minimum_distance_function_3D.m; the function nº22 in the section A.2.2.8. motor_case_generation.m; the function nº23 in the section A.1.2.14. order_vector_points.m; the function nº24 in the section A.1.2.9. phi_one_time_step_evolution.m; the function nº25 in the section A.1.2.7. port_area.m; the function nº26 in the section A.2.2.16. tri_generation.m, the function nº27 in the section A.1.2.16. units_conversion.m and finally, the nº28 in

<table>
<thead>
<tr>
<th></th>
<th>Function</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>17*</td>
<td>Function</td>
<td>initial_geometry_organization_3D.m</td>
</tr>
<tr>
<td>18*</td>
<td>Function</td>
<td>interface_tracker_in_P.m</td>
</tr>
<tr>
<td>19*</td>
<td>Function</td>
<td>interpolate_cross_section.m</td>
</tr>
<tr>
<td>20*</td>
<td>Function</td>
<td>mach_calculation.m</td>
</tr>
<tr>
<td>21*</td>
<td>Function</td>
<td>minimum_distance_function_3D.m</td>
</tr>
<tr>
<td>22*</td>
<td>Function</td>
<td>motor_case_generation.m</td>
</tr>
<tr>
<td>23*</td>
<td>Function</td>
<td>order_vector_points.m</td>
</tr>
<tr>
<td>24*</td>
<td>Function</td>
<td>phi_one_time_step_evolution.m</td>
</tr>
<tr>
<td>25*</td>
<td>Function</td>
<td>port_area.m</td>
</tr>
<tr>
<td>26*</td>
<td>Function</td>
<td>tri_generation.m</td>
</tr>
<tr>
<td>27*</td>
<td>Function</td>
<td>units_conversion.m</td>
</tr>
<tr>
<td>28*</td>
<td>Function</td>
<td>rocket_performance_0D_transient_quasi_3D.m</td>
</tr>
</tbody>
</table>

Table 3. List of names used to identify the main code and the functions.
the section A.2.6. rocket_performance_0D_transient_quasi_3D.m. So, here a brief description of only not repeated functions is presented as follows:

1) **dif_A_p.m.** This function calculates the differential of the area as a function of the longitudinal position (as a function of z). The mathematical background concerning this function is explained in the section 5.3. of the project report. The only input required by this function is the port area as a function of z. This port area is called as “A_p”. The output given by this function is the differential aforementioned. The Matlab® code is enclosed in the section A.3.2.1. dif_A_p.m.

2) **grain_cross_sections_pmt_1D.m.** This function extracts the interface curve from the ϕ function for each grain cross-sectional obtaining a ϕ function for each grain cross-section. Thereby, this function extracts the interface curve for each time step and calculates the grain geometric parameters; i.e. the burning area, the propellant volume and the port area and perimeter of each cross-section. The input parameters required by this function are the “X” and “Y” matrices of x and y coordinates of the Cartesian grid, the matrix of the ϕ function called “Phi”, the number of cross-sections called “N_cs” and the radius of the motor case, inter-alia.

The outputs given by this function are six: the matrix of the points of cross-sections called “Xg_cs”, the beeper to localize the position of cross-sections called “Gr_loc”, the burning area known as “A_b”, the propellant volume knows as “V_prop”, the vector “P_b” which contains the perimeter of each cross-section and the vector “A_p” which contains the port area of each cross-section. The mathematical background of this function can be also found in the section 4.4.2.3 of the project report. The Matlab® code is attached in the section A.3.2.2. grain_cross_sections_pmt_1D.m.

3) **interpolate_bt_cs.m.** This function performs the linear interpolation of a given property between two cross-sections. The mathematical background of this function is explained in the section 4.4.2.2. the project report. The inputs required by this function are two: “P_data” and “z”. The former is a matrix array with the property data as a function of z coordinate. While the first row of this matrix contains the z coordinate points, the second row has the property values at each z point. The latter is the z coordinate value where the output shall be evaluated. “P_z” is the output given by the function and it refers to the property evaluated between two cross-sections. The code of this function is found in A.3.2.3. interpolate_bt_cs.m.

4) **nozzle_inlet_mach.m.** This function calculates the number of Mach at the inlet of the nozzle. Consequently, the output of this function the Mach number. In order to calculate the Mach high numeric precision as well as
a relaxation factor for the calculation can be assigned inside the function. Additionally, three inputs are required by the function: the area of the nozzle inlet known as “A_inlet”, the area of the throat called as “A_t” and the $\gamma$ of the gases called “Gamma”. The Matlab® code of this function is found in the section A.3.2.4. nozzle_inlet_mach.m.

5) p_u_rho_calc.m This function calculates the density (“rho”), the static pressure (“p”) and the velocity (“u”) given the port area (“Ap”), the $\gamma$ of the gases and the flux-variables “f1”, “f2” and “f3” of the governing equations. “Ap”, $\gamma$, “f1”, “f2” and “f3” are the inputs of the function and “rho”, “p” and “u” the outputs. The mathematical background of the function can be found in the section 5.3. of the project report. The code is attached in the section A.3.2.5. p_u_rho_calc.m.

6) perimeter_cross_section.m. This function estimates the perimeter “P” given a closed curve “C” and the radius of the motor case “R_case”. So, whereas “P” is the output of the function “C” and “R_case” are the inputs. Note that “C” is a matrix of two columns, containing the x coordinates of the curve in the first one and the y coordinates in the second one. This function is mainly used to estimate the perimeter of the cross-sections (input data). The Matlab® code of the function is found in the section A.3.2.6. perimeter_cross_section.m.

7) rocket_performance_1D_steady_Q3D.m. This function is analogous to the rocket_performance_0D_transient.m for the 0D unsteady flow. Thereby, this function calculates the internal ballistics of the solid rocket motor with a 1D quasi-steady flow model. Consequently, by calculating the internal ballistics the performance of the SRM is obtained. The input parameters of this function are the initial rocket parameters, such as propellant density or parameters of the burning rate law, the vector of port area, the vector of cross-section perimeters, the motor case profile, etc. Then, using these inputs, the function computes the value of the rocket performance variables along the longitudinal axis of the SRM. Some rocket performance variables obtained are for instance, the static pressure, the density, the velocity and the number of Mach along the longitudinal axis of the SRM, the Mach at the nozzle inlet, etc. The mathematical background as well as the solver of this function is explained in the section 5.3. of the project report. The code of the function can be found in the section A.3.2.7. rocket_performance_1D_steady_Q3D.m.

As it has been aforementioned, in the section A.3.2. the Matlab® the codes of the functions here described are included.
Study of Grain Burnback and Performance of Solid Rocket Motors

A.3.1. MAIN CODE: General_Quasi_3D_flow_1D_discrete_cart_coord.m

%STUDY OF GRAIN BURNBACK AND PERFORMANCE OF SOLID ROCKET MOTORS--
%%
%AUTHOR: ARNAU PONS LORENTE
%TUTOR: Dr. RAMON CARRERAS PLANELLS
%CO-TUTOR: Dr. MANEL SORIA GUERRERO
%UNIVERSITY: ETSEIAT-UNIVERSITAT POLITÈCNICA DE CATALUNYA-
%BARCELONA TECH.
%DEFENSE: JANUARY 2013
%%
%QUASI 3D SOLID ROCKET MOTOR GRAIN BURNBACK ANALYSIS AND 1D QUASI
STEADY FLOW MODEL PERFORMANCE.
clc
clear all
close all

%---------------------------DATA INPUT---------------------------

%INPUT PARAMETERS
N=250; %Number of nodes per side in the cartesian
grid.
N_Delta_web=60; %Number of contour lines shown in the Phi
plot.
N_bs_plot=16; %Number of burnt steps shown in grain
evolution plot.
N_cs_i=1; %Number of grain cross-sections
interpolated between each reference cross-section.
N_z1=500; %Number of slices for the flow calculation
from the head end to the nozzle inlet.
Time=6;  %Simulation time [s]
n_steps=75; %Number of simulation time steps.
n_studied=2;  %Time step to study the flow variables
inside the grain.
t_0=0.5;  %Time duration of the first phase of the
simulation (heavy transient).
Delta_t0=0.005; %Time step during t_0 phase of the
simulation.
Delta_t1=(Time-floor(t_0/Delta_t0)*Delta_t0)/(n_steps-
floor(t_0/Delta_t0));  %Time step for the rest of simulation.
K_size=8;  %Parameter for the size of Xgrain_cs.
%Distance units selection:
units='mm';
%Units conversion:
[K_units] = units_conversion(units);
atm_units=101325;  %Conversion factor from Pa to atm.

%Rocket input parameters:
P_c0=1.1E6;  %Initial chamber pressure [Pa]
a=2.34015e-5;  %a parameter of the propellant burning
rate law: r=a*Pc^n
n_exp=0.36;  %n exponent of the propellant burning rate
law: r=a*Pc^n
gamma\_e=1.19577; \hspace{1em} \%Combustion products gas gamma parameter
\gamma=C_p/C_v
R\_air=287; \hspace{1em} \%R of air [J/KgK]
R\_e=322.21; \hspace{1em} \%R of the combustion products [J/KgK]
rho\_p=1800; \hspace{1em} \%Propellant density [Kg/m^3]
M\_mol=0.03875; \hspace{1em} \%Combustion products molar mass [kg/mol]
theta\_min\_case=90; \hspace{1em} \%Initial angle for the plot of the motor
case [º]
theta\_max\_case=270; \hspace{1em} \%Final angle for the plot of the motor
case [º]
P\_a=101325; \hspace{1em} \%Atmospheric pressure [Pa]
T\_c\_v=[2970 2980 2986.15]; \hspace{1em} \%Vector of chamber temperature
data relation with chamber pressure [K].
P\_c\_v=[1 3 6.9]; \hspace{1em} \%Vector of chamber pressure data
[MPa].
p\_T=polyfit(P\_c\_v,T\_c\_v,2); \hspace{1em} \%Polynomial coefficients of the
function adjust of \( T\_c=f(P\_c) \)
h\_f=5e6; \hspace{1em} \%Enthalpy of the propellant combustion
reaction.

\%GEOMETRY LOADING:
name\_file\_grain='NAWC\_motor\_n6\_grain.dat'; \hspace{1em} \%Name of the
propellant grain geometry file to be loaded.
name\_file\_case='NAWC\_6\_motor\_case\_profile.dat'; \hspace{1em} \%Name of the
motor case geometry file to be loaded.
[Xgr,Ygr,Zgr,G\_loc,R\_case,A\_e,A\_t,Zcp,Rcp] =
geometry\_loading\_3D(name\_file\_grain,name\_file\_case);
idx = strfind(name\_file\_grain,'.');
name\_file\_v=name\_file\_grain(1:idx-1); \hspace{1em} \%File name of
the grain burnback video evolution.

[M\_e] = mach\_calculation(A\_e,A\_t,gamma\_e); \hspace{1em} \%Nozzle exit
Mach number
[\text{Vc\_0}] = chamber\_volume(Zcp,Rcp)*K\_units^-3; \hspace{1em} \%Volume of the
combustion chamber in empty configuration (no propellant in) [m^3]

\%Assignment of the rocket motor input parameters to a data vector:
rocket\_parameters=zeros(1,13); \hspace{1em} \%rocket\_parameters: Vector
array of the rocket inputs parameters.
rocket\_parameters(1)=P\_c0; \hspace{1em} \%Initial chamber pressure
[Pa]
rocket\_parameters(2)=a; \hspace{1em} \%a parameter of the
propellant burning rate law: \( r=a*Pc^n \)
rocket\_parameters(3)=n\_exp; \hspace{1em} \%n exponent of the
propellant burning rate law: \( r=a*Pc^n \)
rocket\_parameters(4)=gamma\_e; \hspace{1em} \%Combustion products gas
\gamma=C_p/C_v
rocket\_parameters(5)=R\_air; \hspace{1em} \%R of air [J/KgK]
rocket\_parameters(6)=R\_e; \hspace{1em} \%R of the propellant
[J/KgK]
rocket\_parameters(7)=rho\_p; \hspace{1em} \%Propellant density
[Kg/m^3]
rocket\_parameters(8)=M\_mol; \hspace{1em} \%Combustion products molar
mass [kg/mol]
rocket\_parameters(9)=A\_e*K\_units^-2; \hspace{1em} \%Nozzle exit area [m^2]
rocket\_parameters(10)=A\_t*K\_units^-2; \hspace{1em} \%Nozzle true area [m^2]
rocket\_parameters(11)=M\_e; \hspace{1em} \%Nozzle exit Mach number
rocket_parameters(12)=P_a;  %Atmospheric pressure [Pa]
rocket_parameters(13)=Vc_0;  %Volume of the combustion chamber in empty configuration (no propellant in) [m^3]

%----------------------------GEOMETRICAL CALCULATIONS----------------------------%
%
%Organization of the initial geometry data given by the user:
[X_grain,Z_cs] =
initial_geometry_organization_3D(Xgr,Ygr,Zgr,G_loc);
N_cs=size(Z_cs,2)+(size(Z_cs,2)-1)*N_cs_i;  %Total number of cross-sections.
%Calculation of tangent and normal vectors of the interface curve:
[Tc,Nc] = curve_tangent_normal_3D(X_grain,G_loc);
%Cartesian square grid generation:
L=1.75*2*R_case;  %Width and height of the cartesian grid square.
Delta_xy=L/N;  %Step size Delta_x=Delta_y=Delta_xy, due to the grid is cartesian.
[X,Y] = cartesian_grid(N,L,Delta_xy);
Delta_web=R_case/N_Delta_web;  %Step size of the plotting contours of Phi function.

%----------------------------CALCULATIONS----------------------------%
%
%Signed Minimum Distance calculation for each grain cross-section:
[Dmin] = minimum_distance_function_3D(X,Y,X_grain,G_loc,Nc);

%Initialization of the Phi function as a signed minimum distance function:
Phi=zeros(N+1,N+1,size(G_loc,2),n_steps+1);
%Calculation of the initial grain cross-sections, burning area and propellant volume:
[Xgrain_0,Gr_loc_0,A_b0,V_prop_0,P_b0,A_p0] =
grain_cross_sections_pmt_1D(X,Y,Phi(:,:,1),Z_cs,N_cs_i,N_cs,R_case,K_units);
%Initialization of the Xgrain_cs matrix, which has the points of all the grain cross-sections for each time step:
Xgrain_cs=zeros(K_size*size(X,1)*N_cs,3,n_steps+1);
Gr_loc=zeros(N_cs,n_steps+1);
X_grain_cs(:,:,1)=Xgrain_0;
Gr_loc(:,1)=Gr_loc_0;

Delta_t=zeros(1,n_steps);  %Time step vector.
V_time=zeros(1,n_steps+1);  %Time vector [s].

%Combustion chamber pressure considering a 0D steady flow model:
P_cm=zeros(n_steps+1,1);
P_cm(1)=P_c0;
%Initialization of rocket performance variables:
M_rocket=zeros(n_steps+1,16);  %M_rocket: Matrix array of the rocket performance variables for each time step.
Burning area [m^2]
\[ %M_{\text{rocket}}(n) = \text{variable} \]

- Propellant volume \([\text{m}^3]\) 
- Burning rate \([\text{m/s}]\) 
- Burning depth \([\text{m}]\) 
- Production of gas due to combustion \([\text{kg/s}]\) 
- Flow rate exiting the nozzle \([\text{kg/s}]\) 
- Combustion chamber pressure \([\text{Pa}]\) 
- Combustion chamber temperature \([\text{K}]\) 
- Combustion chamber volume \([\text{m}^3]\) 
- Combustion chamber density \([\text{kg/m}^3]\) 
- Exit pressure \([\text{Pa}]\) 
- Exit velocity \([\text{m/s}]\) 
- Propellant mass \([\text{kg}]\) 
- Characteristic velocity \([\text{m/s}]\) 
- Thrust coefficient \([\text{N}]\) 

Assignment of initial values of the rocket performance variables:

\[
\begin{align*}
M_{\text{rocket}}(1,1) &= A_{b0}; \\
M_{\text{rocket}}(1,2) &= V_{prop_0}; \\
M_{\text{rocket}}(1,3) &= a*P_{c0}^n_{\text{exp}}; \\
M_{\text{rocket}}(1,4) &= 0; \\
M_{\text{rocket}}(1,5) &= M_{\text{rocket}}(1,1)*\rho_{p}*M_{\text{rocket}}(1,3); \\
M_{\text{rocket}}(1,6) &= M_{\text{rocket}}(1,5); \\
M_{\text{rocket}}(1,7) &= P_{c0}; \\
M_{\text{rocket}}(1,8) &= \text{polyval}(p_T,P_{c0}*1e^{-6}); \\
M_{\text{rocket}}(1,9) &= \text{rocket\_parameters(13)}-V_{prop_0}; \\
M_{\text{rocket}}(1,10) &= 0; \\
M_{\text{rocket}}(1,11) &= P_{c0}/(1+(\gamma_e-1)/2*{\mathbf{M}_e}^2)^{(\gamma_e/(\gamma_e-1))}; \\
M_{\text{rocket}}(1,12) &= (2*\gamma_e*R_e*{\mathbf{M}_e}^2*V_{prop}(1,8))/(\gamma_e-1)*((M_{\text{rocket}}(1,11)/P_{c0})^((\gamma_e-1)/\gamma_e))^{(1/2)}; \\
M_{\text{rocket}}(1,13) &= \text{rocket\_parameters(16)}; \\
\end{align*}
\]
\[ M_{\text{rocket}}(1,13) = M_{\text{rocket}}(1,6) \times M_{\text{rocket}}(1,12) + (M_{\text{rocket}}(1,11) - P_a) \times \text{rocket parameters}(9); \]

\[ M_{\text{rocket}}(1,14) = \rho_p \times V_{\text{prop.0}}; \]

% Initial propellant mass [kg]
% Initial characteristic velocity [m/s]:
\[ M_{\text{rocket}}(1,15) = \sqrt{\text{rocket parameters}(6) \times M_{\text{rocket}}(1,8) / \sqrt{\text{rocket parameters}(4)}} \times (2 / (\text{rocket parameters}(4) + 1))^{(\text{rocket parameters}(4) + 1)/(2 \times (\text{rocket parameters}(4) - 1))}; \]

% Initial thrust coefficient []
\[ M_{\text{rocket}}(1,16) = \sqrt{\left(2 \times \text{rocket parameters}(4)^2 / (\text{rocket parameters}(4) - 1) \right) \times \left(2 / (\text{rocket parameters}(4) + 1) \right)^{(\text{rocket parameters}(4) + 1)/(\text{rocket parameters}(4) - 1)}} \times (1 - (M_{\text{rocket}}(1,11) / M_{\text{rocket}}(1,7))^{(\text{rocket parameters}(4) - 1)/(\text{rocket parameters}(4))}) \times ((M_{\text{rocket}}(1,11) - \text{rocket parameters}(12) / M_{\text{rocket}}(1,7)) / \text{rocket parameters}(9) / \text{rocket parameters}(10)); \]

% Initialization of the ballistics variables:
\[ p = \text{zeros}(N_z1+1,n_steps+1); \]
\[ u = \text{zeros}(N_z1+1,n_steps+1); \]
\[ \rho = \text{zeros}(N_z1+1,n_steps+1); \]
\[ M = \text{zeros}(N_z1+1,n_steps+1); \]
\[ z = \text{zeros}(N_z1+1,1); \]
\[ M_{\text{inlet}} = \text{zeros}(1,n_steps+1); \]
\[ [p(:,1),u(:,1),\rho(:,1),z(:,1),M(:,1),M_{\text{inlet}}] = \text{rocket performance}_1D_{\text{steady Q3D}}(M_{\text{rocket}}(:,1),\text{rocket parameters},A_p0,P_b0,Zcp,Rcp,Z_cs,K_units,N_z1,h_f); \]

% Calculation of the coupled evolution of the Phi function and the rocket performance:
\[ \text{for } n=1:n_steps \quad \text{ %Time loop} \]
\[ \text{if } V_{\text{time}}(n) < t_0 \]
\[ \Delta t(n) = \Delta t_0; \]
\[ \text{else} \]
\[ \Delta t(n) = \Delta t_1; \]
\[ \text{end} \]
\[ V_{\text{time}}(n+1) = V_{\text{time}}(n) + \Delta t(n); \]
\[ \text{rocket performance}_0D_{\text{transient quasi 3D}}(M_{\text{rocket}}(n,:),\text{rocket parameters},p_T,K_units,\Delta t(n)); \]

% Assignment of V burning rate for the Phi evolution:
\[ V = M_{\text{rocket}}(n+1,3) \times K_{\text{units}}; \]
\[ \text{Phi evolution for the next time step:} \]
\[ \text{for } k=1:1:\text{size}(G_{\text{loc}},2) \quad \text{ %Reference cross-sections loop} \]
\[ \text{phi one time step evolution}(\Phi(:,k,n),N,\Delta x,y,\Delta t(n),V); \]
\[ \text{end} \]
\[ \text{Calculation of the grain cross-sections and updating the burning area and propellant volume:} \]
\[ [X_{\text{grain cs}}(:,n+1),G_{\text{loc}}(:,n+1),M_{\text{rocket}}(n+1,1),M_{\text{rocket}}(n+1,2),P_b,A_p] = \text{grain cross sections pmt 1D}(X,Y,\Phi(:,,:,n+1),Z_{cs},N_{cs_i},N_{cs},R_{case},K_{units}); \]
\[ \text{Updating the chamber volume, nozzle mass flow:} \]
M_rocket(n+1,9)=rocket_parameters(13)-M_rocket(n+1,2);
M_rocket(n+1,6)=rocket_parameters(7)*M_rocket(n+1,1)*M_rocket(n+1,3);
%Calculation of p,u,rho with 1D model flow inside the combustion chamber:
[p(:,n+1),u(:,n+1),rho(:,n+1),z(:,1),M(:,n+1),M_inlet_ref,M_inlet(n+1)]=
rocket_performance_1D_steady_Q3D(M_rocket(n+1,:),rocket_parameters,
A_p,P_b,Zcp,Rcp,Z_cs,K_units,N_z1,h_f);
%Combustion chamber pressure considering a 0D steady flow model:
P_cm(n+1)=((M_rocket(n+1,1)/rocket_parameters(10))*rocket_parameters(2)*rocket_parameters(7)*M_rocket(n+1,15))^(1/(1-
rocket_parameters(3)));
end
%
-------------------------PLOTS-------------------------------
figure(1);
Az=50;
El=20;
[TRI_g,Xg,Yg,Zg] =
grain_3D_plot_matrix(Xgrain_cs(:,:,n_steps+1),Gr_loc(:,n_steps+1),N_cs);
trisurf(TRI_g,Yg,Zg,Xg,
    'FaceColor','interp',...%
    'EdgeColor','none',...%
    'FaceLighting','phong')
hold on;
[TRI_g0,Xg0,Yg0,Zg0] =
grain_3D_plot_matrix(Xgrain_0,Gr_loc_0,size(Gr_loc_0,1));
trisurf(TRI_g0,Yg0,Zg0,Xg0,
    'FaceColor','green',...%
    'EdgeColor','none',...%
    'FaceLighting','phong')
hold on;
[X_c,Y_c,Z_c] =
motor_case_generation(theta_min_case,theta_max_case,Zcp,Rcp);
surf(X_c,Z_c,Y_c,
    'FaceColor','blue',...%
    'EdgeColor','black',...%
    'FaceLighting','phong')
hold on;
camlight left
colormap(hot)
text1=['y [' units ']'];
text2=['z [' units ']'];
text3=['x [' units ']'];
xlabel(text1)
ylabel(text2)
zlabel(text3)
view([Az El])
figure(2);
set(2,'Name','Time evolution of A_b, m_dot_g, P_c, m_dot_d');
subplot(2,2,1);
plot(V_time,M_rocket(:,1),'color',[0 0 1])
hold on
xlim([V_time(1) V_time(n_steps+1)]);
xlabel('t [s]')
ylabel('A_b [m^2]')
title('Burning area vs. time')

subplot(2,2,2);
plot(V_time,M_rocket(:,5),'color',[0 0 1])
hold on
xlim([V_time(1) V_time(n_steps+1)]);
xlabel('t [s]')
ylabel('m_d_o_t_g [kg/s]')
title('Mass flow rate due to propellant combustion vs. time')

subplot(2,2,3);
plot(V_time,M_rocket(:,7)/atm_units,'color',[0 0 1])
hold on
xlim([V_time(1) V_time(n_steps+1)]);
xlabel('t [s]')
ylabel('P_c [atm]')
title('Combustion chamber pressure vs. time')

subplot(2,2,4);
plot(V_time,M_rocket(:,6),'color',[0 0 1])
hold on
xlim([V_time(1) V_time(n_steps+1)]);
xlabel('t [s]')
ylabel('m_d_o_t_d [kg/s]')
title('Mass flow rate exiting the nozzle vs. time')

figure(3);
set(3,'Name','Time evolution of r_dot, V_prop, P_cm, T_c')

subplot(2,2,1);
plot(V_time,1000*M_rocket(:,3),'color',[0 0 1])
hold on
xlim([V_time(1) V_time(n_steps+1)]);
xlabel('t [s]')
ylabel('r_d_o_t [mm/s]')
title('Burning rate vs. time')

subplot(2,2,2);
plot(V_time,M_rocket(:,2)*1000,'color',[0 0 1])
hold on
xlim([V_time(1) V_time(n_steps+1)]);
xlabel('t [s]')
ylabel('V_p_r_o_p_e_l_l_a_n_t [L]')
title('Propellant volume versus time')

subplot(2,2,3);
plot(V_time,P_cm/atm_units,'color',[0 0 1])
hold on
xlim([V_time(1) V_time(n_steps+1)]);
xlabel('t [s]')
ylabel('P_c_m [atm]')
title('Combustion chamber pressure (0D steady flow) versus time')
subplot(2,2,4); plot(V_time,M_rocket(:,8),'color',[0 0 1]) hold on xlim([V_time(1) V_time(n_steps+1)]); xlabel('t [s]') ylabel('T_c [K]') title('Combustion chamber temperature vs. time')

figure(4); set(4,'Name','Time evolution of T, V_e, rho_c, M_prop');

subplot(2,2,1); plot(V_time,M_rocket(:,13),'color',[0 0 1]) hold on xlim([V_time(1) V_time(n_steps+1)]); xlabel('t [s]') ylabel('F [N]') title('Thrust vs. time')

subplot(2,2,2); plot(V_time,M_rocket(:,12),'color',[0 0 1]) hold on xlim([V_time(1) V_time(n_steps+1)]); xlabel('t [s]') ylabel('V_e_xt [m/s]') title('Nozzle exit velocity vs. time')

subplot(2,2,3); plot(V_time,M_rocket(:,10),'color',[0 0 1]) hold on xlim([V_time(1) V_time(n_steps+1)]); xlabel('t [s]') ylabel('
\rho_c [kg/m^3]') title('Combustion chamber density vs. time')

subplot(2,2,4); plot(V_time,rho_p*M_rocket(:,2),color,[0 0 1]) hold on xlim([V_time(1) V_time(n_steps+1)]); xlabel('t [s]') ylabel('M Propel [kg]') title('Propellant mass vs. time')

figure(5); set(5,'Name','Time evolution of c*, C_f, A_b, P_e');

subplot(2,2,1); plot(V_time,M_rocket(:,15),color,[0 0 1]) hold on xlim([V_time(1) V_time(n_steps+1)]); xlabel('t [s]') ylabel('c* [m/s]') c_max=max(M_rocket(:,15)); ylim([0.995*c_max 1.005*c_max]) title('Characteristic velocity vs. time')
Study of Grain Burnback and Performance of Solid Rocket Motors

```matlab
subplot(2,2,2);
plot(V_time,M_rocket(:,16),'color',[0 0 1])
hold on
xlim([V_time(1) V_time(n_steps+1)]);
ylabel('C_f')
title('Thrust coefficient vs. time')

subplot(2,2,3);
plot(M_rocket(:,4),M_rocket(:,1),'color',[0 0 1])
hold on
xlim([M_rocket(1,4) M_rocket(n_steps+1,4)]);
text2=['Web [' units ']' ];
xlabel(text2)
ylabel('A_b [m^2]')
title('Burning area vs. burnt depth')

subplot(2,2,4);
plot(V_time,M_rocket(:,11)/atm_units,'color',[0 0 1])
hold on
xlim([V_time(1) V_time(n_steps+1)]);
xlabel('t [s]')
ylabel('P_e_x_i_t [atm]')
title('Nozzle exit pressure vs. time')

figure(6);
subplot(2,2,1);
plot(z,u(:,n_studied))
xlabel('z [m]')
ylabel('u [m/s]')

subplot(2,2,2);
plot(z,M(:,n_studied))
xlabel('z [m]')
ylabel('Mach')

subplot(2,2,3);
plot(z,rho(:,n_studied))
xlabel('z [m]')
ylabel('$\rho$ [kg/m^3]')

subplot(2,2,4);
plot(z,p(:,n_studied)/atm_units)
xlabel('z [m]')
ylabel('P [atm]')
```

141
A.3.2. FUNCTIONS

The functions presented in this section are listed considering the order given in the Table 3. Additionally, it should be pointed out that the repeated functions of the Table 3 are not included again in this section.

A.3.2.1. dif_A_p.m

```matlab
function [D_Ap] = dif_A_p(A_p)
%UNTITLED2 Summary of this function goes here
% Detailed explanation goes here
D_Ap=zeros(2,size(A_p,2));
for i=1:size(A_p,2)
    if i==size(A_p,2)
        D_Ap(2,i)=(A_p(2,i)-A_p(2,i-1))/(A_p(1,i)-A_p(1,i-1));
    else
        D_Ap(2,i)=(A_p(2,i+1)-A_p(2,i))/(A_p(1,i+1)-A_p(1,i));
    end
end
end
```

A.3.2.2. grain_cross_sections_pmt_1D.m

```matlab
function [Xg_cs,Gr_loc,A_b,V_prop,P_b,A_p] = grain_cross_sections_pmt_1D(X,Y,Phi,Z_cs,N_cs_i,N_cs,R_case,K_units)
%UNTITLED Summary of this function goes here
% Detailed explanation goes here
Gr_loc=zeros(N_cs,1);
N_Xg_cs=8*size(X,1)*N_cs;
Xg_cs=zeros(N_Xg_cs,3); %Matrix with [x y z] coordinates for the points of all the interpolated cross-sections.
A_p=zeros(2,size(Z_cs,2));
P_b=zeros(2,N_cs);
%Filling of the Xg_cs matrix with the reference cross-sections of the grain given by the Phi function, and the interpolated cross-sections between each reference cross-section. Also, calculation of the burning area A_b and the propellant volume V_p_in:
A_b=0;
V_p_in=0;
N_in_t=0;
k=1;
l=1;
for i=1:size(Z_cs,2)-1
```
%i-th reference cross-section:
[Cin1,P_1] = interface_tracker_in_P(X,Y,Phi(:,:,i),R_case);
[A_p1] = port_area(X,Y,Phi(:,:,i),R_case);
P_b(1,l)=Z_cs(i)*K_units^-1;
P_b(2,l)=P_1*K_units^-1;
l=l+1;
A_p(2,i)=A_p1*K_units^-2;
N_in1=size(Cin1,1);
Vz_1=Z_cs(i)*ones(N_in1,1);
%i+1-th reference cross-section:
[Cin2,P_2] = interface_tracker_in_P(X,Y,Phi(:,:,i+1),R_case);
[A_p2] = port_area(X,Y,Phi(:,:,i+1),R_case);
N_in2=size(Cin2,1);
Vz_2=Z_cs(i+1)*ones(N_in2,1);

%Filling the X_gS matrix with the points of the i-th reference cross-section:
i_0=N_in_t+1;
N_in_t=N_in1+N_in_t;
Xg_cs(i_0:N_in_t,:)=[Cin1 Vz_1];
Gr_loc(k)=N_in_t;
k=k+1;
Lz=Z_cs(i+1)-z_cs_ant;
Delta_z=Lz/(N_cs_i+1);
%Calculation of the wet surface between the previous interpolated cross-section and the cross-section A:
if i>1
    if P_1==0 && P_csi~=0
        delta_z2=Z_cs(i)-z_cs_ant;
        A_b=P_csi*delta_z2+A_b;  % Burning area.
    elseif P_csi==0 && P_1~=0
        delta_z2=Z_cs(i)-z_cs_ant;
        A_b=P_1*delta_z2+A_b;  % Burning area.
    else
        P_m=(P_csi+P_1)/2;  % Mean perimeter
        delta_z2=Z_cs(i)-z_cs_ant;
        A_b=P_m*delta_z2+A_b;  % Burning area.
    end
end
%Calculation of the port volume between the cross-sections A and B:
A_pm=(A_p1+A_p2)/2;  %Mean port area, between the cross-sections A and B
V_p_in=A_pm*Lz+V_p_in;
z_cs_ant=Z_cs(i);
for j=1:N_cs_i
    %Calculation of the z-coordinate of the interpolated section:
z_cs=Z_cs(i)+j*Delta_z;
    %Interpolation of a cross-section between the cross-sections A and B:
    [X_csi] =
        interpolate_cross_section(Cin1(:,:,1),Cin1(:,:,2),Vz_1,Cin2(:,:,1),Cin2(:,:,2),Vz_2,z_cs);
    P_csi=perimeter_cross_section(X_csi(:,:,1:2),R_case);
P_b(1,l)=z_cs*K_units^-1;
P_b(2,l)=P_csi*K_units^-1;
l=l+1;
%Calculation of the wet surface between the two considered cross-sections:
if \( P_1 = 0 \) && \( P_{\text{csi}} = 0 \)
\[
\delta z_2 = z_{\text{cs}} - z_{\text{cs ant}}; \quad A_b = P_{\text{csi}} \delta z_2 + A_b; \quad \%\text{Burning area.}
\]
elseif \( P_{\text{csi}} = 0 \) && \( P_1 = 0 \)
\[
\delta z_2 = z_{\text{cs}} - z_{\text{cs ant}}; \quad A_b = P_1 \delta z_2 + A_b; \quad \%\text{Burning area.}
\]
else
\[
P_m = (P_{\text{csi}} + P_1) / 2; \quad \%\text{Mean perimeter}
\]
\[
\delta z_2 = z_{\text{cs}} - z_{\text{cs ant}}; \quad A_b = P_m \delta z_2 + A_b; \quad \%\text{Burning area.}
\]
end

\( z_{\text{cs ant}} = z_{\text{cs}}; \)
\( P_1 = P_{\text{csi}}; \)
\( N_{\text{csi}} = \text{size}(X_{\text{csi}},1); \)
\( i_0 = N_{\text{in t}} + 1; \)
\( N_{\text{in t}} = N_{\text{csi}} + N_{\text{in t}}; \)
\%Filling the X_gs matrix with the points of the j-th interpolated
\%cross-section between the cross-sections A and B:
\[
Xg_{\text{cs}}(i_0:N_{\text{in t}},:) = X_{\text{csi}};
\]
\( \text{Gr loc}(k) = N_{\text{in t}}; \)
\( k = k + 1; \)
end

\%Assignment of the last cross-section:
in \( i = \text{size}(Z_{\text{cs}},2) - 1 \)
\( i_0 = N_{\text{in t}} + 1; \)
\( N_{\text{in t}} = N_{\text{in t}} + N_{\text{in t}}; \)
\( Xg_{\text{cs}}(i_0:N_{\text{in t}},:) = [C_{\text{in 2}} \ V_{z_2}]; \)
\( \text{Gr loc}(k) = N_{\text{in t}}; \)
\( P_b(1,1) = Z_{\text{cs}}(i+1) * K_{\text{units}}^{-1}; \)
\( P_b(2,1) = P_2 * K_{\text{units}}^{-1}; \)
\( A_p(2,i+1) = A_p2 * K_{\text{units}}^{-2}; \)
\[ \text{if } P_1 = 0 \&\& \ P_2 = 0 \]
\[ \delta z_2 = Z_{\text{cs}}(\text{size}(Z_{\text{cs}},2)) - z_{\text{cs ant}}; \]
\[ A_b = P_2 \delta z_2 + A_b; \]
elseif \( P_2 = 0 \) && \( P_1 = 0 \)
\[ \delta z_2 = Z_{\text{cs}}(\text{size}(Z_{\text{cs}},2)) - z_{\text{cs ant}}; \]
\[ A_b = P_1 \delta z_2 + A_b; \]
else
\[ P_m = (P_1 + P_2) / 2; \quad \%\text{Mean perimeter} \]
\[ \delta z_2 = Z_{\text{cs}}(\text{size}(Z_{\text{cs}},2)) - z_{\text{cs ant}}; \]
\[ A_b = P_m \delta z_2 + A_b; \]
end
end

\%Convert the units of \( A_b, A_p \) and \( V_p\_in \) so that \( A_b \) and \( A_p \) are in \([\text{m}^2]\) \%and \( V_p\_in \) is \([\text{m}^3]\):
\[ A_b = A_b * K_{\text{units}}^{-2}; \]
\[ V_{\text{prop}} = (p_1 * R_{\text{case}}^2 * (Z_{\text{cs}}(\text{size}(Z_{\text{cs}},2)) - Z_{\text{cs}}(1)) - V_{p\_in}) * K_{\text{units}}^{-3}; \]
\[ A_p(1,:) = Z_{\text{cs}} * K_{\text{units}}^{-1}; \]
end
A.3.2.3. interpolate_bt_cs.m

```matlab
function [P_z] = interpolate_bt_cs(P_data,z)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
%Lineal interpolation of a given property between two cross_sections.
%P_z: the property evaluated between two cross-sections.
P_data: matrix array with the property data as a function of z-coordinate.
   %the first row has the z-coordinate points and the second row has
%the property values at each z-coordinate.
%z: z-coordinate where the P_z shall be evaluated.

%First find the two cross-sections two interpolate with:
for i=1:size(P_data,2)
    if z<=P_data(1,i)
        i_0=i;
        break;
    end
end
if i_0==1
    i_0=2;
end
P_z=P_data(2,i_0-1)+((P_data(2,i_0)-P_data(2,i_0-1))/(P_data(1,i_0)-P_data(1,i_0-1)))*(z-P_data(1,i_0-1));
```

A.3.2.4. nozzle_inlet_mach.m

```matlab
function [M] = nozzle_inlet_mach(A_inlet,A_t,Gamma)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

delta=1e-6; %Numeric precision
fr=0.5; %Relaxation factor for the calculation

M=0;
dif=2*delta;
while dif>delta
    x=M;
    M=(A_t/A_inlet)*((1+((Gamma-1)/2)*M^2)/(Gamma+1))^{((Gamma+1)/(2*(Gamma-1)))};
    dif=abs(M-x);
    M=x+fr*(M-x);
end
```

A.3.2.5. p_u_rho_calc.m

```matlab
function [p,u,rho] = p_u_rho_calc(f1,f2,f3,Ap,Gamma)
%UNTITLED2 Summary of this function goes here
```
% Detailed explanation goes here
p=(f2+sqrt(f2^2*Gamma^2+2*f1*f3*(1-Gamma^2)))/(Ap*(1+Gamma));
u=(f2/f1)-(Ap/f1)*p;
rho=(f1/(Ap*u));
end

A.3.2.6. perimeter_cross_section.m

function [P] = perimeter_cross_section(C,R_case)
%UNTITLED Summary of this function goes here
% Detailed explanation goes here

P=0;
di=0;
for i=1:size(C,1)
    if i==size(C,1)
        Xa=C(i,1);
        Ya=C(i,2);
        Xb=C(1,1);
        Yb=C(1,2);
    else
        Xa=C(i,1);
        Ya=C(i,2);
        Xb=C(i+1,1);
        Yb=C(i+1,2);
    end
    Ra=sqrt(Xa^2+Ya^2);
    Rb=sqrt(Xb^2+Yb^2);
    if Ra>=R_case && Rb>=R_case
        di=0;
    elseif Ra<R_case && Rb<R_case
        di=sqrt((Xa-Xb)^2+(Ya-Yb)^2);
    elseif Ra<R_case && Rb>R_case || Ra>R_case && Rb<R_case
        a=(Xb-Xa)^2+(Yb-Ya)^2;
        b=2*Xa*(Xb-Xa)+2*Ya*(Yb-Ya);
        c=Xa^2+Ya^2-R_case^2;
        Disc=b^2-4*a*c;
        if Disc>=0
            s1=(-b+sqrt(Disc))/(2*a);
            s2=(-b-sqrt(Disc))/(2*a);
            if s1>=0 && s1<=1
                s=s1;
                x=Xa+(Xb-Xa)*s;
                y=Ya+(Yb-Ya)*s;
                if Ra<R_case && Rb>R_case
                    di=sqrt((x-Xa)^2+(y-Ya)^2);
                elseif Ra>R_case && Rb<R_case
                    di=sqrt((x-Xb)^2+(y-Yb)^2);
                end
            elseif s2>=0 && s2<=1
                s=s2;
                x=Xa+(Xb-Xa)*s;
                y=Ya+(Yb-Ya)*s;
                if Ra<R_case && Rb>R_case
                    di=sqrt((x-Xa)^2+(y-Ya)^2);
                elseif Ra>R_case && Rb>R_case
                    di=sqrt((x-Xb)^2+(y-Yb)^2);
                end
            end
        end
    end
end
elseif Ra>R_case && Rb<R_case
di=sqrt((x-Xb)^2+(y-Yb)^2);
else
di=0;
end
else
di=0;
end
end
P=di+P;
end

A.3.2.7. rocket_performance_1D_steady_Q3D.m

function [p,u,rho,rho_u,z,M,M_inlet_ref,M_inlet,M_inlet_v,P0_n] = rocket_performance_1D_steady_Q3D(M_rocket_n,rocket_parameters,A_p, P_b,A_b0,Zcp,Rcp,Z_cs,K_units,N_z1,fr,epsilon,P0_n0,Tc,h_f,n_iterations)
%UNTITLED Summary of this function goes here
% Detailed explanation goes here

%Finding the nozzle inlet z-position:
for i=1:1:size(Rcp,1)
    if Rcp(i)==Rcp(1)
        i_0=i;
    end
end
Zcp=Zcp*K_units^-1;
Z_cs=Z_cs*K_units^-1;
z_inlet=Zcp(i_0);
z_g0=Z_cs(1);
z_gf=Z_cs(size(Z_cs,2));
A_inlet=pi*(Rcp(1)*K_units^-1)^2;
v_Ap_add=[z_gf+0.5*(z_inlet-z_gf) z_inlet; A_inlet A_inlet];
A_p=[A_p v_Ap_add];
z=zeros(1,N_z1);
p=zeros(1,N_z1);
u=zeros(1,N_z1);
rho=zeros(1,N_z1);
M=zeros(1,N_z1);
rho_u=zeros(1,N_z1);
df_rho_u=zeros(1,N_z1);
df_p=zeros(1,N_z1);
Ap_i=zeros(1,N_z1);
d_Ap_i=zeros(1,N_z1);
Pb_i=zeros(1,N_z1);
Delta_P=1e5;
a=sqrt(rocket_parameters(4)*rocket_parameters(6)*Tc);
Delta_z1=(z_inlet-z_g0)/(N_z1-1);
[D_Ap] = dif_A_p(A_p);

%INSIDE GRAIN REGION FLOW CALCULATION:
\[ \text{dif} = 5 \ast \epsilon; \]
\[ \Gamma_C = \sqrt{\text{rocket}_\text{parameters}(4) \ast \left(2 / (\text{rocket}_\text{parameters}(4) + 1)\right)} \]
\[ c_{\text{star}} = \sqrt{\text{rocket}_\text{parameters}(6) \ast T_c / \Gamma_C}; \]
\[ P_0 = \left(A_b_0 / \text{rocket}_\text{parameters}(10)\right) \ast \text{rocket}_\text{parameters}(2) \ast \text{rocket}_\text{parameters}(7) \ast c_{\text{star}} \right)^{(1 / (1 - \text{rocket}_\text{parameters}(3))};} \]
\[ n = 1; \]
\[ M_{\text{inlet}} = 0; \]

\begin{verbatim}
while dif > epsilon && n <= n_iterations
  p(1) = P0;
  u(1) = 0;
  rho_u(1) = 0;
  rho(1) = p(1) / (\text{rocket}_\text{parameters}(6) \ast T_c);
  P0_ant = P0;
  P0_n(n) = P0;
  M_ant = M_inlet;
  for i = 1:1:N_z1
    z(i) = z_g0 + (i - 1) \ast \Delta_z1;
    if i == N_z1
      z(i) = z_{\text{inlet}};
    end
    M(i) = u(i) / (\sqrt{\text{rocket}_\text{parameters}(4) \ast p(i) / \rho(i)});
    Ap_i(i) = \text{interpolate}_\text{bt}_cs(A_p, z(i));
    d_Ap_i(i) = \text{interpolate}_\text{bt}_cs(D_Ap, z(i));
    if z(i) <= z_{gf}
      Pb_i(i) = \text{interpolate}_\text{bt}_cs(P_b, z(i));
      \[ df_{\rho_u}(i) = \left(\rho(i) \ast u(i) / Ap_i(i)\right) \ast d_{Ap_i(i)} + \left((\text{rocket}_\text{parameters}(7) \ast M_{\text{rocket_n}(3)} \ast Pb_i(i)) / \left(1 - (M(i)^2)\right)\right) \ast (h_f / a^2) + \left((\text{rocket}_\text{parameters}(4) - 1) / (\text{rocket}_\text{parameters}(4) + 1 / 2) \ast M(i)^2 + (\text{rocket}_\text{parameters}(4) \ast \text{rocket}_\text{parameters}(7))\right) \ast (u(i) / (1 - (M(i)^2) \ast Ap_i(i))) \ast d_{Ap_i(i)}; \]
      \[ df_u(i) = \left(-1 \ast (u(i) / (1 - M(i)^2) \ast Ap_i(i))\right) \ast d_{Ap_i(i)}; \]
      \[ df_p(i) = \left(M(i)^2 \ast \text{rocket}_\text{parameters}(4) \ast p(i) / (Ap_i(i) \ast (1 - M(i)^2))\right) \ast \text{interpolate}_\text{bt}_cs(P_b, z(i)); \]
    else
      \[ df_{\rho_u}(i) = \left(-1 \ast \rho(i) \ast u(i) / Ap_i(i)\right) \ast d_{Ap_i(i)}; \]
      \[ df_u(i) = \left(-1 \ast (u(i) / (1 - M(i)^2) \ast Ap_i(i))\right) \ast d_{Ap_i(i)}; \]
      \[ df_p(i) = \left(M(i)^2 \ast \text{rocket}_\text{parameters}(4) \ast p(i) / (Ap_i(i) \ast (1 - M(i)^2))\right) \ast \text{interpolate}_\text{bt}_cs(P_b, z(i)); \]
    end
  end
  rho_u(i+1) = rho_u(i) + df_{\rho_u}(i) \ast \Delta_z1;
  u(i+1) = u(i) + df_u(i) \ast \Delta_z1;
  p(i+1) = p(i) + df_p(i) \ast \Delta_z1;
  rho(i+1) = rho_u(i+1) / u(i+1);
  M(i+1) = u(i+1) / (\sqrt{\text{rocket}_\text{parameters}(4) \ast (p(i+1) / \rho(i+1))});
end
p_{\text{inlet}} = p(N_z1+1);
rho_{\text{inlet}} = rho(N_z1+1);
\end{verbatim}
\[ \text{M\_inlet\_ref} = \text{nozzle\_inlet\_mach(A\_inlet,rocket\_parameters(10),rocket\_parameters(4))}; \]
\[ \text{M\_inlet} = \frac{u(i+1)}{\sqrt{\text{rocket\_parameters(4)} \times (p\_inlet/rho\_inlet)} }; \]
\[ \text{M\_inlet\_v(n)} = \text{M\_inlet}; \]
\[ \text{dif} = \text{abs(M\_inlet-M\_inlet\_ref)}; \]
% if n>2
% \text{dM} = (\text{M\_inlet\_v(n-1)}-\text{M\_inlet\_v(n-2)})/(P0-P0\_n(n-2));
% \text{P0} = \text{P0\_ant-fr*(P0/dM)};
% else
% \text{P0} = \text{P0\_ant+Delta\_P};
% end
\[ \text{P0} = \text{P0\_ant-Delta\_P}; \]
\[ n = n+1; \]
end
ANNEX B. SOME RELEVANT RESULTS OF 2D GRAIN DESIGNS

In this ANNEX B some relevant 2D grain designs simulated by the 2D grain burnback and 0D unsteady flow simulation tool are presented. This annex tends to be the supplementary document of the section 7.2 of the project report. Due to the volume of the project report, it has been decided not to include all of the tested 2D grain geometries there. In this way, in this ANNEX B the following 2D geometries have been included: the star of 10 slotted tips, the tri-slot, the dogbone and the anchor. Note that in order to be able to compare the performance results obtained for these four cases, the same input parameters shall be used except for uploaded the text file. This SRM related and numerical input parameters used for the study are shown in Figure 1. In addition, in order to prove that the results obtained with the simulation tool are coherent, it is recommended to consult the Figure 2 where the scheme of thrust vs. time curve for different 2D grain geometries is shown.

Figure 1. Inputs of the simulation tool for the study some relevant 2D grain designs.

Figure 2. Thrust curve for several SRM 2D grain geometries (source [1]).
B.1. STAR OF 10 SLOTTED TIPS

In Figure 3 the initial grain geometry of the star of 10 slotted tips is shown. This plot is outputted by the Matlab® in order to verify by the user that the geometry uploaded is correct. This way, the black line represents the motor case and the blue line the initial grain surface.

In Figure 4 the interface evolution at equally spaced time steps is shown. This is a good representation of how grain geometry evolves during the grain burnback. Note that despite of the background grid is Cartesian and with a moderated number of nodes, the front reconstruction is very precise and capable of tracking the sharp cusps formed by this grain geometry. Generally, a polar grid is capable of modeling better geometries with axisymmetric geometries. Nonetheless, the
Cartesian grid here used is enough to deal with the grain burnback of the star of 10 slotted tips. In order to consider the enclosed performance curves, note that one of the advantages of this grain geometry is its high volumetric fraction.

Figure 5. Burning area, mass flow rate due to propellant combustion, chamber pressure and mass flow rate exiting the nozzle time evolutions during the burning process of the star of 10 slotted tips grain.

Figure 6. Burning rate, burning perimeter, port area and chamber temperature time evolutions during the burning process of the star of 10 slotted tips grain.
Study of Grain Burnback and Performance of Solid Rocket Motors

Figure 7. Thrust, nozzle exit velocity, chamber gas density and propellant mass time evolutions during the burning process of the star of 10 slotted tips grain.

Figure 8. Characteristic velocity, thrust coefficient and nozzle exit pressure time evolutions and burning perimeter vs. burnt depth evolution during the burning process of the star of 10 slotted tips grain.

Figure 5, Figure 6, Figure 7 and Figure 8 show the typical performance curves obtained for the star of 10 slotted tips, which is coherent with the result offered by Figure 2. The most characteristic feature of this grain geometry is its regressive thrust curve which is very suitable to provide a moderated acceleration to the payload. Note that when the burning time goes by, the acceleration of the rocket increases if the force is kept constant.
B.2. TRI-SLOT

In Figure 9 the tri-slot initial grain geometry is shown. This plot is outputted by the Matlab® in order to verify by the user that the geometry uploaded is correct. This way, the black line represents the motor case and the blue line the initial grain surface.

In Figure 10 the interface evolution at equally spaced time steps is shown. This is a good representation of how grain geometry evolves during the grain burnback. Note that despite of the background grid is Cartesian and with a moderated number of nodes, the front reconstruction is very precise and capable of tracking
the three sharp cusps formed by this grain geometry. Generally, a polar grid is capable of modeling better geometries with axisymmetric geometries (this grain geometry can be considered as se-axissymmetric geometry). Nonetheless, the Cartesian grid here used is enough to deal with the grain burnback of this tri-slot 2D grain geometry.

![Figure 11](image1)

**Figure 11.** Burning area, mass flow rate due to propellant combustion, chamber pressure and mass flow rate exiting the nozzle time evolutions during the burning process of the tri-slot grain.

![Figure 12](image2)

**Figure 12.** Burning rate, burning perimeter, port area and chamber temperature time evolutions during the burning process of the tri-slot grain.
Figure 13. Thrust, nozzle exit velocity, chamber gas density and propellant mass time evolutions during the burning process of the tri-slot grain.

Figure 14. Characteristic velocity, thrust coefficient and nozzle exit pressure time evolutions and burning perimeter vs. burnt depth evolution during the burning process of the tri-slot grain.

Figure 11, Figure 12, Figure 13 and Figure 14 show the typical performance curves obtained by the tri-slot grain geometry. The most characteristic feature of this grain geometry is its progressive-regressive thrust curve. For this case, it is not found a clear reference in Figure 2.
In Figure 15 the initial grain geometry of the dogbone is shown. This plot is outputted by the Matlab® in order to verify by the user that the geometry uploaded is correct. This way, the black line represents the motor case and the blue line the initial grain surface.

In Figure 16 the interface evolution at equally spaced time steps is shown. This is a good representation of how grain geometry evolves during the grain burnback. Note that despite of the background grid is Cartesian and with a moderated number of nodes, the front reconstruction is very precise and capable of tracking
the forms this grain geometry. Generally, a polar grid is capable of modeling better geometries with axisymmetric geometries (this grain geometry can be considered as se-axisymmetric geometry). Nonetheless, the Cartesian grid here used is enough to deal with the grain burnback this 2D grain geometry.

Figure 17. Burning area, mass flow rate due to propellant combustion, chamber pressure and mass flow rate exiting the nozzle time evolutions during the burning process of the dogbone grain.

Figure 18. Burning rate, burning perimeter, port area and chamber temperature time evolutions during the burning process of the dogbone grain.
Figure 19. Thrust, nozzle exit velocity, chamber gas density and propellant mass time evolutions during the burning process of the dogbone grain.

Figure 20. Characteristic velocity, thrust coefficient and nozzle exit pressure time evolutions and burning perimeter vs. burnt depth evolution during the burning process of the dogbone grain.

Figure 17, Figure 18, Figure 19 and Figure 20 show the typical performance curves obtained by the dogbone grain geometry. The most characteristic feature of this grain geometry is its regressive thrust curve. For this case too, it is not found a clear reference in Figure 2.
B.4. ANCHOR

Figure 21. Anchor grain initial geometry.

In Figure 21 the initial grain geometry of the anchor is shown. This plot is outputted by the Matlab® in order to verify by the user that the geometry uploaded is correct. This way, the black line represents the motor case and the blue line the initial grain surface.

Figure 22. Anchor grain evolution during the rocket operation.

In Figure 22 the interface evolution at equally spaced time steps is shown. This is a good representation of how grain geometry evolves during the grain burnback. Note that despite of the background grid is Cartesian and with a moderated
number of nodes, the front reconstruction is very precise and capable of tracking the forms this grain geometry..

Figure 23. Burning area, mass flow rate due to propellant combustion, chamber pressure and mass flow rate exiting the nozzle time evolutions during the burning process of the anchor grain.

Figure 24. Burning rate, burning perimeter, port area and chamber temperature time evolutions during the burning process of the anchor grain.
Figure 25. Thrust, nozzle exit velocity, chamber gas density and propellant mass time evolutions during the burning process of the anchor grain.

Figure 26. Characteristic velocity, thrust coefficient and nozzle exit pressure time evolutions and burning perimeter vs. burnt depth evolution during the burning process of the anchor grain.

Figure 23, Figure 24, Figure 25 and Figure 26 show the typical performance curves obtained by the anchor grain geometry. The most characteristic feature of this grain geometry is its regressive thrust curve, which regresses moderately compared with the star of 10 slotted tips. This results verifies the thrust vs. time curve shown in Figure 2.
B.5. REFERENCES

[This page is intentionally left blank.]