Modeling Unsteady and Steady 1-D Hydrodynamics under Different Hydraulic Conceptualizations: Model/Software Development, and Case Studies

Marcus N. Gomes Júnior†,*, Luis Miguel Castillo Rápalo‡, Paulo Tarso S. Oliveira£, Marcio H. Giacomoni††, and Eduardo M. Mendiondo§

I. Supplementary Material

This supplemental material presents the following:

- HydroHP - 1D Input Data in Sec. I-A
- Data Derived from ANA in Sec. I-B
- Mathematical Treatment at Domain Boundaries in Sec. I-C
- Appendix 2 - Algorithm 2 for HP Estimation on Python language in Sec. I-D
- Matlab codes of:
  - (i) HP Estimator Sec. I-E1
  - (ii) Read Input Data for SVE Model Sec. I-E2
  - (iii) SVE Model in Sec. I-E3
  - (iv) post-processing in Sec. I-E4
  - (v) Cross-Section, Top Width, and Water Surface Elevation Profiles in Sec. I-E5
  - (vi) Top Width, and Water Surface Elevation Profiles in regular sections in Sec. I-E6
  - (vii) Detailed output algorithm in Sec. I-E7

A. HydroHP - 1D Input Data

In order to improve the HydroHP - 1D use and aided with Excel sheets, it was developed an interface to set up the model parameters, boundary conditions and cross-sections data. All input data entered in Excel has comments to aid users. Excel version 2013 or higher is required.

1) General Data

In this sub-topic, we enter the basic gemetrical data of the channel such as the length, the number of nodes and the elevation of the first node. Fig. 1 shows an example.

- L is the channel lengths (m)

†School of Civil Environmental Engineering, and Construction Management, College of Engineering and Integrated Design, University of Texas at San Antonio, One UTSA Circle, San Antonio, Texas, 78249, BSE 1.310 (marcusnobrega.engcivil@gmail.com).
‡Department of Hydraulic Engineering and Sanitation, University of Sao Paulo, Sao Carlos School of Engineering, 13566-590 (luis.castillo@unah.hn).
£Faculty of Engineering, Architecture and Urbanism and Geography, Federal University of Mato Grosso do Sul, MS, 79070–900, Brazil. (paulo.t.oliveira@ufms.br)
††School of Civil Environmental Engineering, and Construction Management, College of Engineering and Integrated Design, University of Texas at San Antonio, One UTSA Circle, San Antonio, Texas, 78249, BSE 1.346 (marcio.giacomoni@utsa.edu).
§Department of Hydraulic Engineering and Sanitation, University of Sao Paulo, Sao Carlos School of Engineering, 13566-590 (emm@sc.usp.br).
* Corresponding author.
This work was financially supported by CAPES.

General Data

L	1097.88	m
N_x	100	
el	0.27432	m
g	9.81	m/s^2
n_m	0.025	
I_0	0.00025	m/m
t_f	360	min
dt	1	sec
animation_time	10	min
$s_{f,outlet}$	0.00025	m/m
$N_{\Delta y}$	1000	
C	0.5	
Δt_{min}	0.5	sec
Δt_{max}	0.5	sec
Date Begin	1/1/2022 12:00:00	
Date End	1/1/2022 12:32:00	

Fig. 1: Example of general data for the model set up. The parameters presented here controls the spatial domain, some of the outlet boundary conditions, adaptive time-step scheme, and output recording time.

- N_x is the number of sections that the channel lengths will be divided into
- e_l is the elevation of the first reach of the channel
- g is the gravity acceleration magnitude (9.81 m/s^2)
- n_m is the manning roughness coefficient for cases with a constant roughness coefficient
- I_0 is the bottom slope along the channel lengths (this not includes the outlet).
- t_f is the simulation period of time (min).
- Δt is the time-step if a constant time-step is used.
- animation_time is the interval of time considered for the results post-processing.
- $s_{\mathrm{f,outlet}}$ is the outlet slope if normal condition are established (see sec. I-A2)
- $N_{\Delta \mathrm{v}}$ is the number of discretization for the cross-section depths. C is the desired Courant number in order to ensure numerical stability
- $\Delta t_{\min}$ is the minimum time-step
- and $\Delta t_{\max}$ is the maximum time-step for the adaptive scheme employed in this paper.
- Date Begin is the date that starts the simulation. It is only activated if flag_elapsed_time is 1.
- Date End is the date that the simulation ends. It is only activated if flag_elapsed_time is 1

2) Boundary conditions

The boundary conditions and other modeling conditions are actiavated by flags. A flag equals 1 represent that a condition is imposed in the model. Fig. 2a) shows the flags that are required and Fig. 2b) summarize the HydroHP - 1D set up, where a) represent the general data, b) shows the model boundary conditions and simulating cases according to the flags entered, c) controls the Nash hydrograph, d) controls the tidal outlet boundary condition, e) enters the trapezoid cross-section data, f) controls either the circular or parabolic cross-section data, g) enters the tabular inflow hydrograph, h) inputs the stage hydrograph data, and i) controls the varying slope or elevation data.

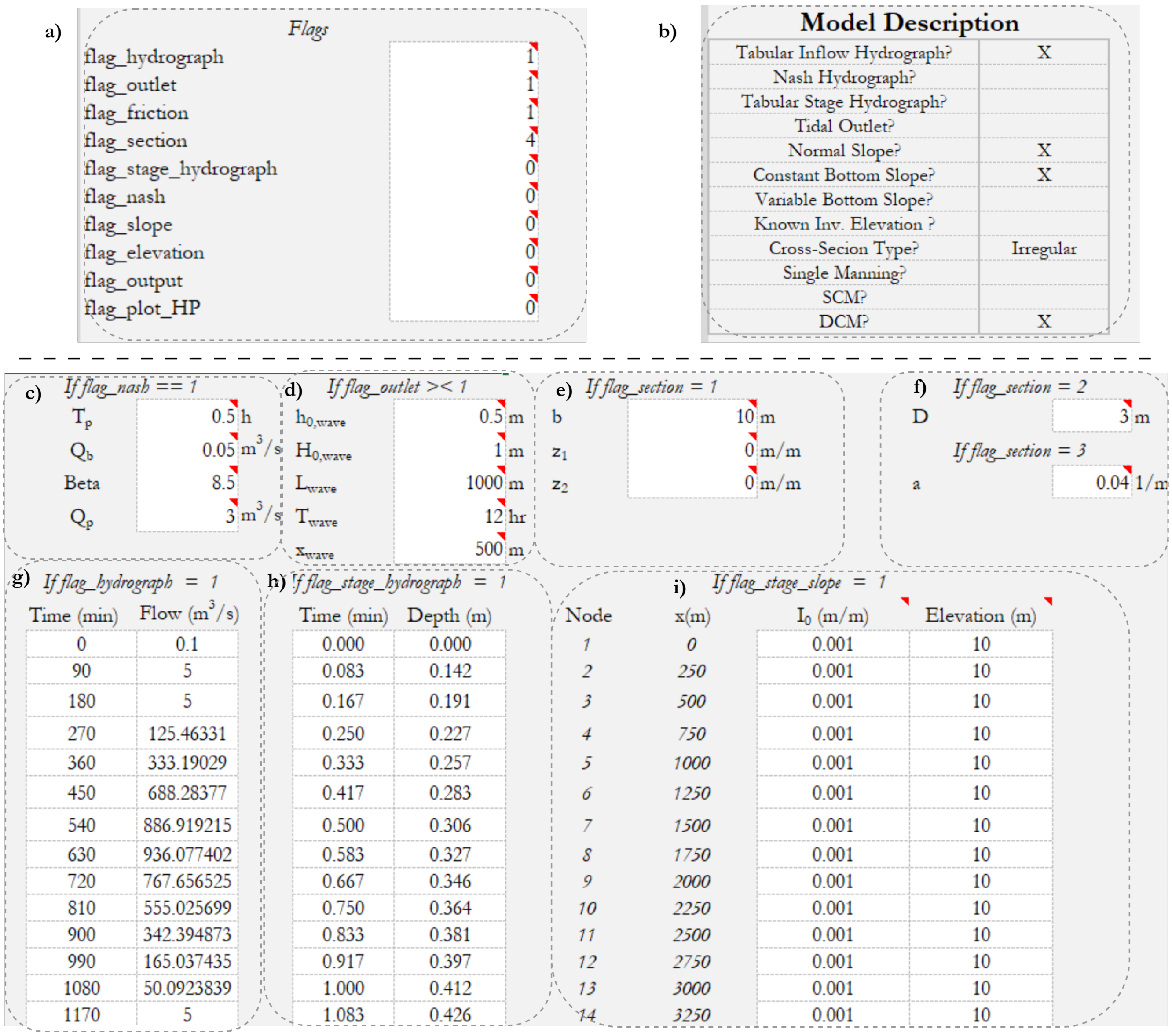

Fig. 2: Boundary conditions for the HydroHP - 1D model.

3) Hydrograph Conditions

The flag_hydrograph indicates the hydrograph shape, if it is defined by the user as showed in Fig. 2g), otherwise, it is assumed to employ a hydrograph with Nash shape (flag_nash==1). For the latter, Fig. 2c) shows the parameters for this condition. T_p indicates when is reached the peak time of the hydrograph (h). Q_b is the base flow along the hydrograph (m^3/s). Beta is a the shape factor of the Nash hydroprah. Q_p is the magnitude of the peak flow (m^3/s).

4) Outlet Conditions

Regarding the flag_outlet, this indicates if it is assumed normal conditions for the flow, otherwise, a wave function is employed (flag_outlet = 0) which is defined by the user as showed in Fig. 2h). Herein, in Fig. 2d) are shown the wave properties: $h_{0,wave}$ is the mean wave depth (m); $H_{0,wave}$ is the wave amplitude (m); L_{wave} is the wave length (m); T_{wave} is the wave period (hr); x_{wave} is the relative position from the reference (m). It is worth mentioning that if normal conditions are assumed, the flag_friction should be equals to 1.

5) Channel Conditions

If the conditions will not considered as constant along the channel, the flag_slope and flag_elevation (equal to 1) allows to specify the slope and elevation for each node within the channel, as showed in Fig. 2i).

6) Cross-section Conditions

The HydroHP - 1D includes four types of cross-section along the channel: Trapezoidal (1); Circular (2); Parabolic (3); Irregular (4). Once the kind of cross-section is defined in flag_section, it is necessary to set the parameters for the desired section as shown in Fig. 2e) and Fig. 2f). b is the bottom channel width (m). z_1 and z_2 are the left and right slopes (m/m), respectively. D is the channel diameter (m) and a is the parabola coefficient (1/m).

For the irregular cross-section case, as shown in Fig. 3, it is necessary to indicate the kind of model (flag_method) to calculate the hydraulic properties of the section: 1 indicates that the single cross-section method (SCM) will employed considering only the depth-varying manning coefficient modeled via Einstein's equation and entered in the table; and 2 indicates that the discrete cross-section method (DCM) will be used and considers the difference between the in-bank and the over-banks roughness. n_m is the in-bank roughness coefficient and n_f is the over-bank roughness coefficient.

Readme		
flag_length	0	
flag_method	2	
S_0	0.0001	m/m
n_m	0.035	
n_f	0.035	

Section Discretization				
Station (m)	Elevation (m)	Lengths (m)	n(i,i+1)	break_point divider
0	195.05		0.035	
5.0002	194.81		0.035	
10.0004	194.57		0.035	
15.0007	194.32		0.035	
20.0014	194.05		0.035	
25.0018	193.23		0.035	
30.0028	192.38		0.035	
35.003	191.55		0.035	
40.0036	190.73		0.035	
45.0041	190.41		0.035	
50.0044	190.1		0.035	

Fig. 3: Irregular cross-section input data.

```
%% Algorithm - Section Coordinates
% Developer: Marcus Nobrega
% Date 5/16/2022
% Goal - Determine cross-section coordinates for different types of
% cross-sections
%%%%%%%%%%%%%%% All Rights Reserved - contact: marcusnobrega.engcivil@gmail.com

clear all
% Single Sections
n_test = 0.02; % Roughness assumed
%% Triangular Section
hmax = 2; % maximum depth in m
b1 = 1; % left length in m
b2 = 2; % right length in m
x_1 = 0; % inicial x_coordinate for first value
y_1 = hmax; % inicial y_coordinate for first value
x = [x_1 (x_1 + b1) (x_1 + b1 + b2)]';
y = [y_1 (y_1 - hmax) (y_1)]';
x_triangular = x;
y_triangular = y;
n_channel_triangular = repmat(n_test,length(x_triangular)-1,1);
%% Parabolic Section
a = 1; % 1/m such that y = a*x^2 or x = sqrt(y/a)
hmax = 2; % maximum depth in m
step = 0.01; % height step in m
n_steps = floor(hmax/step);
y = linspace(0,hmax,n_steps);
x_right = sqrt(y/a);
x_left = flip(-x_right,2);
y_left = flip(y,2);
```

```
x = [x_left x_right]';
y = [y_left y]';
x_parabolic = x;
xmin = min(x_parabolic);
x_parabolic = x_parabolic + abs(xmin);
y_parabolic = y;
n_channel_parabolic = repmat(n_test,length(x_parabolic)-1,1);
%% Semi-Hyperbolic and Semi-Parabolic
% Hyperbole Equation -> y^2/a^2 - x^2/b^2 = 1
% a = 0.1;
% b = 0.01;
% xc = 0;
% yc = 0;
% hmax = 1; % maximum depth in m
% step = 0.01; % height step in m
% n_steps = floor(hmax/step);
% y = linspace(0,hmax,n_steps);
% x_left = xc + sqrt(a^2*(-1 + (y - yc).^2/(b^2)));
% x_left = flip(-x_left,2);
% % Parabolic Equation
% a = 0.01; % 1/m such that y = a*x^2 or x = sqrt(y/a)
% x_right = sqrt(y/a);
% % Final
% x = [x_left x_right]';
% y = [flip(y,2) y]';
% Composite Sections
%% Semi-Elliptical and Semi-Parabolic
% Ellipse Equation -> (x-xc)^2/a^2 + (y-yc)^2/b^2 = 1\
hmax = 2; % maximum depth in m
a = 2*hmax;
b = hmax;
xc = -a;
yc = 0;
step = 0.01; % height step in m
n_steps = floor(hmax/step);
y = linspace(0,hmax,n_steps);
x_left = xc + sqrt(a^2*(1 - (y - yc).^2/(b^2)));
x_left = flip(x_left,2);
% Parabolic Equation
a = 0.1; % 1/m such that y = a*x^2 or x = sqrt(y/a)
x_right = sqrt(y/a);
% Final
x = [x_left x_right]';
y = [flip(y,2) y]';
x_semi = x;
xmin = min(x_semi);
x_semi = x_semi + abs(xmin);
y_semi = y;
n_channel_semi = repmat(n_test,length(x_semi)-1,1);
%%%%%%%%%%%%%%%% Composite Sections%%%%%%%%%%%%%%%%
%% Road Gutter Cross-Section
hmax = 2; % maximum depth in m
b_1 = 0; % gutter width in m, typycally 0 if vertical
b_2 = 0.4; % gutter width in m
b_3 = 1.2; % wetted road width in (m)
h_1 = 0.15; % curb height (m)
h_2 = 0.10; % gutter height (m)
h_3 = 0.12; % water depth (m) ≤ h_1
x_1 = 0; % inicial x_coordinate for first value
y_1 = max([h_1 h_2 h_3]); % inicial y_coordinate for first value
x = [x_1 (x_1 + b_1) (x_1 + b_1 + b_2) (x_1 + b_1 + b_2 + b_3)]';
y = [y_1 (y_1 - h_1) (y_1 - h_1 + h_2) (y_1 - h_1 +  h_3)]';
x_gutter = x;
xmin = min(x_gutter);
x_gutter = x_gutter + abs(xmin);
y_gutter = y;
n_channel_road = repmat(n_test,length(x_gutter)-1,1);
%% Sucessive Trapezoid Gabion Channel
b0 = 0; % width within vertical points (m)
b = 2; % width of horizontal gabion (m)
h = 0.5; % height of the gabion (m)
n_vertical = 4; % number of vertical gabions
x_1 = 0; % inicial x_coordinate for first value
y_1 = h*n_vertical; % inicial y_coordinate for first value
x = 0;
y = 0;
for i = 1:(n_vertical*2)
```

```
    if i == 1
        x(i,1) = x_1;
        y(i,1) = y_1;
    else
        if mod(i,2) == 1 % Odd number
            x(i,1) = x(i-1,1) + b;
            y(i,1) = y(i-1,1);
        else
            x(i,1) = x(i-1,1) + b0;
            y(i,1) = y(i-1,1) - h;
        end
    end
end
x_left = x;
y_left = y;
x_right = 0; y_right = 0;
for i = 1:(n_vertical*2)
    if i == 1
        x_right(i,1) = x_left(end,1) + b;
        y_right(i,1) = y_left(end,1);
    else
        if mod(i,2) == 1 % Odd number
            x_right(i,1) = x_right(i-1,1) + b;
            y_right(i,1) = y_right(i-1,1);
        else
            x_right(i,1) = x_right(i-1,1) + b0;
            y_right(i,1) = y_right(i-1,1) + h;
        end
    end
end
x = [x_left;x_right]';
y = [y_left;y_right]';
x_gabion = x;
xmin = min(x_gabion);
x_gabion = x_gabion + abs(xmin);
y_gabion = y;
n_channel_triangular = repmat(n_test,length(x_gabion)-1,1);
%% Composite V-Notch and Francis Weir
b_rec = 0.75; % width of rectangular weir besides the v-notch (m)
hrec = 1; % rectangular height
h_vnot = 1; % v-notch height
alfa = pi/4; % 45 degree
x_1 = 0;
y_1 = hrec + h_vnot;
x = [x_1 (x_1) (x_1 + b_rec) (x_1 + b_rec + h_vnot/tan(atan(alfa))) (x_1 + b_rec + ...
    2*h_vnot/tan(atan(alfa))) (x_1 + b_rec + 2*h_vnot/tan(atan(alfa)) + b_rec) (x_1 + ...
    2*h_vnot/tan(atan(alfa)) + 2*b_rec)]';
y = [y_1 (y_1 - hrec) (y_1 - hrec) (y_1 - hrec - h_vnot) (y_1 - hrec) (y_1 - hrec) (y_1)]';
x_vnot = x;
y_vnot = y;
n_channel_trapezoid = repmat(n_test,length(x_vnot)-1,1);
%% Irregular Channel
y_irr = [343.6  342.6   341.7   341.5   341.5   342.1   342 342.3   343 343 340.2   341.6   341.3   ...
    339.3   338.6   339.3   340.5   342.7   342.7   342.3   342 341.9   341.7   341.5   342.3   ...
    342.7   343.2]';
l_irr = [20.1   50.5    90.9    17.1    30.2    9.4 6.7 4.9 2.1 13.8    3.9 2.5 3   3.7 3.3 3.4 0.6 ...
    5.8 5.8 15.8    17.7    7   18.9    38.1    27.4    62.7]';
x_irr(i,1) = 0;
for i = 1:length(l_irr)
    x_irr(i+1,1) = x_irr(i,1) + l_irr(i,1);
end
n_channel_triangular = repmat(n_test,length(x_irr)-1,1);
% x_final = [x_triangular x_parabolic x_semi x_gutter x_gabion x_vnot x_irr]';
% y_final = [y_triangular y_parabolic y_semi y_gutter y_gabion y_vnot x_irr]';
%% Plot Cross-Sections
subplot(4,2,1)
line_w = 2;
c = [64 64 64]/255;
font = 12;
set(gcf,'units','inches','position',[4,4,6.5,4])
set(gca,'FontSize',font)
plot(x_triangular,y_triangular,'LineWidth',line_w,'color',c)
xlabel('x(m)','Interpreter','latex','FontSize',font)
ylabel('y(m)','Interpreter','latex','FontSize',font)
grid on
set(gca,'FontSize',font)
subplot(4,2,2)
```

```
plot(x_parabolic,y_parabolic,'LineWidth',line_w,'color',c)
xlabel('x(m)','Interpreter','latex','FontSize',font)
ylabel('y(m)','Interpreter','latex','FontSize',font)
grid on
set(gca,'FontSize',font)
subplot(4,2,3)
plot(x_semi,y_semi,'LineWidth',line_w,'color',c)
xlabel('x(m)','Interpreter','latex','FontSize',font)
ylabel('y(m)','Interpreter','latex','FontSize',font)
grid on
set(gca,'FontSize',font)
subplot(4,2,4)
plot(x_gutter,y_gutter,'LineWidth',line_w,'color',c)
xlabel('x(m)','Interpreter','latex','FontSize',font)
ylabel('y(m)','Interpreter','latex','FontSize',font)
grid on
set(gca,'FontSize',font)
subplot(4,2,5)
plot(x_gabion,y_gabion,'LineWidth',line_w,'color',c)
xlabel('x(m)','Interpreter','latex','FontSize',font)
ylabel('y(m)','Interpreter','latex','FontSize',font)
grid on
set(gca,'FontSize',font)
subplot(4,2,6)
plot(x_vnot,y_vnot,'LineWidth',line_w,'color',c)
xlabel('x(m)','Interpreter','latex','FontSize',font)
ylabel('y(m)','Interpreter','latex','FontSize',font)
grid on
set(gca,'FontSize',font)
% Irr
subplot(4,2,[7:8])
y_irr = y_irr - min(y_irr);
plot(x_irr,y_irr,'LineWidth',line_w,'color',c)
xlabel('x(m)','Interpreter','latex','FontSize',font)
ylabel('y(m)','Interpreter','latex','FontSize',font)
grid on
set(gca,'FontSize',font)
exportgraphics(gcf,'Cross_Sections.pdf','ContentType','vector')
```

B. Data derived from ANA

Data can be obtained from hydroweb website, available at (https://www.snirh.gov.br/hidroweb/). The data format is given in .csv and requires a treatment to convert it into cross-sections, flows, and stages. The data treatment is performed in (https://www.labhidro.ufsc.br/hidroapp/), using the research conducted in [1].

C. Mathematical Treatment at Domain Boundaries

The Lax-Friedrichs method uses a central difference in space, which requires known state values on the neighborhoods. Therefore, in the borders of domain (i.e., $i = 1$ or $i = N_x$), one has to assume some sort of extrapolation. Herein, we use a zero-order extrapolation that varies according to the chosen boundary condition simulated.

1) Inflow Hydrograph Boundary Condition

For an inflow hydrograph boundary condition, we can write:

$$Q_1(t + \Delta t) - Q_h(t + \Delta t) = 0 \tag{1a}$$
$$A_1(t + \Delta t) - A_2(t) = 0 \tag{1b}$$

where Q_h is the known inflow hydrograph.

We can see from previous equation that if a very long time-step is used, problems might arise making the boundary sharped

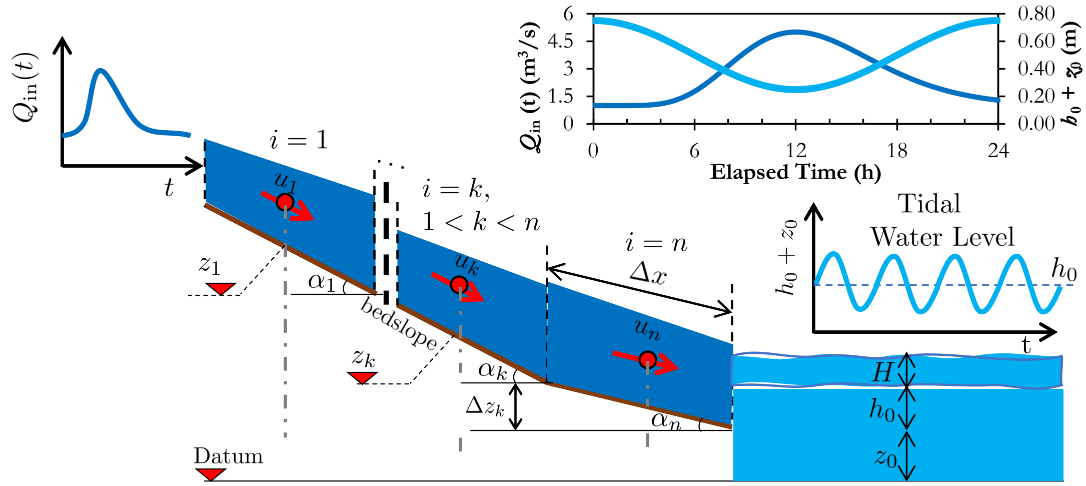

Fig. 4: Example of problem schematics that HydroHP - 1D can solve. The model allow simulating inflow hydrographs, stage hydrographs, normal slope, rating curves and other types of boundary conditions

or curved by the zero-order extrapolation. Moreover, since $A_1(t\Delta t)$ can be estimated, all other hydraulic properties can be derived from the table containing the hydraulic properties.

2) Stage-Hydrograph Boundary Condition

When the depths are known over time in the inlet of the channel, we can write:

$$h_1(t + \Delta t) - h_s(t + \Delta t) = 0 \tag{2a}$$

$$Q_1(t + \Delta t) = Q_2(t) \tag{2b}$$

With known values of $h_1(t + \Delta t)$, we seek values of every other state, such as $A_1(t + \Delta t)$, in the hydraulic properties table.

3) Stage-Hydrograph with Inflow Hydrograph Boundary Condition

When both information is known, we can write:

$$h_1(t + \Delta t) - h_s(t + \Delta t) = 0 \tag{3a}$$

$$Q_1(t + \Delta t) - Q_h(t + \Delta t) = 0 \tag{3b}$$

4) Known Friction Slope at Outlet

In this case, the outlet friction slope is a constant value given by:

$$I_{f,N_x}(t + \Delta t) - s_{out} = 0 \tag{4a}$$

$$A_{N_x}(t + \Delta t) - A_{N_x - 1}(t) = 0 \tag{4b}$$

where s_{out} is given.

5) Tidal Outlet Boundary Condition

For tidal water level, we use a wave equation boundary condition such as:

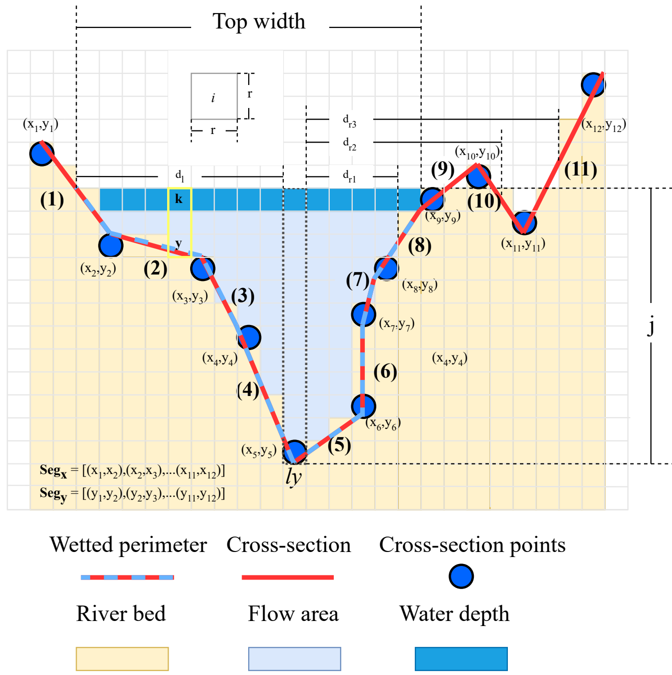

Fig. 5: Example of cross-section discretization with finite element and riverbed boundaries identification according to a water depth j.

$$h_{N_x}(t+\Delta t) = h_0 + \frac{H}{2}\cos\left(k_w x_w - \sigma t\right) \tag{5a}$$

$$I_{f,N_x}(t+\Delta t) = \frac{(z_{N_x-1} + h_{N_x-1}(t)) - (z_{N_x} + h_{N_x}(t))}{\Delta x} \tag{5b}$$

$$u_{N_x}(t+\Delta t) = \frac{1}{n_{Nx}} R_{hN_x}(t)^{2/3}\sqrt{I_{f,N_x}} \tag{5c}$$

$$Q_{N_x}(t+\Delta t) = A_{N_x}(t) u_{N_x}(t) \tag{5d}$$

where h_{N_x} is the water depth at node n, x_w is the relative position of the wave from a given reference, L_w is the wave length, $k_w = 2\pi/L_w$ is the wave number, T_w is the wave period, and σ is the wave angular frequency such that $\sigma = 2\pi/T_w$.

D. Algorithm 2: Finite Element discretization procedure with Nested For Loops

To assess depth-varying HP for the second algorithm, it was employed as a basis the Finite Element Method (FEM) to discretize the hole cross-section area into n regular elements, this results in a 2-D mesh of squares (a matrix), where the number of elements in the mesh are established by a resolution r as commonly done in many engineering applications [2]. The grid size is determined by the r which splits vertical and horizontal distances between coordinates, for instance, a r equals to 0.1m will divide into 10 elements a horizontal distance of 1 meter between two coordinates, and similarly for a vertical distance.

The algorithm begins by finding the lowest bottom elevation of the riverbed ly, then, two vectors are defined ($\boldsymbol{seg_x}$ and $\boldsymbol{seg_y}$) with consecutive pairs or coordinates for both axis, this aims to determine the flow area between the water depth j and the boundaries of the riverbed (see Fig. 5). The main loop is used to represent the water depth increasing, then, inside of this, three individual loops are used to $1)$ define the riverbed boundaries; $2)$ calculate the flow area, and $3)$ calculate the wet perimeter. The left HP are determined in terms of the aforementioned variables. Considering that the water depth is monotonically increasing from ly for every pixel in the mesh on the vertical axis, boundaries from the riverbed topography are identified for every j iteration, hence defining new boundaries to be reached before the water can overflow to the next height of the cross-section for each side. To this end, first, in the vector $\boldsymbol{seg_y}$ is identified between of which pair of coordinates or segments j belongs to. It is worth mentioning that through this method many segments could be considered, as shown in Fig. 5 where j intersect segments 1, 8, 10, and 11, to solve this, the pairs of horizontal coordinates from those segments in $\boldsymbol{seg_x}$ are filtered by considering the closer distance of the average of those pair of coordinates related to the station of ly for left and right sides, for instance, on the right side the distance d_{r1} is lower than d_{r2} and d_{r3}, for the left side there is just a segment to be considered.

1) *Flow area and centroid*

To calculate any HP is necessary to define which elements in the mesh belong to the flow area, for this, and considering the previous method to find boundaries in the riverbed, the value of 1 is assigned to elements in the flow area, otherwise, 0 is assigned to the left elements in the matrix. To this end, it was defined the function f_1, which returns the riverbed elevation for a specific station k within the cross-section, in this case, for every column in the matrix. According to Eq. (6) as shown in Fig. 5, derives from a linear interpolation between the two coordinates of the segment 2. It is worth to mention that there is also a second function f_2 (Eq. (7)) with similar logic of f_1 with the difference that f_2 returns the value of the k station in a segment according to an elevation y of the riverbed. Once every element in the matrix has a value, calculate the area as just the sum of all elements within the matrix.

$$f_1(i) = y_{\mathrm{i+1}} - \left(\frac{y_{\mathrm{i}} - y_{\mathrm{i+1}}}{x_{\mathrm{i}} - x_{\mathrm{i+1}}}\right)(x_{\mathrm{i}} - k) \tag{6}$$

$$f_2 = x_{\mathrm{i+1}} - \left(\frac{y_{\mathrm{i+1}} - y}{y_{\mathrm{i}} - y_{\mathrm{i+1}}}\right)(x_{\mathrm{i+1}} - x_{\mathrm{i}}) \tag{7}$$

where: y is the riverbed elevation; y_i and y_{i+1} are the two riverbed elevations in the segment in analysis; x_i and x_{i+1} are the two riverbed horizontal coordinates of the segment in analysis, and k is the horizontal coordinate of the station.

On the other hand, the vertical centroid for every column is calculated through the sum of all the values on the column and divided by two, plus the riverbed elevation obtained with the f_1 shown in Eq. (6).

2) *Wetted Perimeter*

This procedure is divided into two steps: first, with the f_3 (Eq. (8)) are calculated and accumulated the hypotenuses for all segments within the flow area (2, 3, 4, 5, 6, and 7), excluding those which are intersected by the j water depth (1 and 8). Second, for the hypotenuses' calculation of the first and last segments, is necessary to determine the intersection points on them due to the water depth j using (Eq. (6)) and (Eq. (7)), thus, knowing the coordinates, the distances are calculated using (Eq. (8)).

$$f_3(i) = \sqrt{(y_{\mathrm{i}} - y_{\mathrm{i+1}})^2 + (x_{\mathrm{i}} - x_{\mathrm{i+1}})^2} \tag{8}$$

where y_i, y_{i+1}, x_i and x_{i+1} represent the segment's coordinates.

3) *Hydraulic properties calculation*

As mentioned before, for each water depth in the cross-section and after the cumulative process of area, perimeter, and relative centroid values as shown in algorithm 1, HP as hydraulic radius Eq. (2), conveyance Eq. (5a), velocity, ϕ Eq. (3), flow, and top width are calculated. A pseudocode of the main algorithm is shown in Algorithm 1 to briefly introduce the algorithm structure.

4) *Main Python Code*

```
# %%% Cross Section Hydraulic Properties Estimator %%% #
# Developer: Luis Castillo
# Date 5/20/2022
# Goal: Determine hydrualic properties for regular or irregular cross-section

import numpy as np
import pandas as pd
import math
```

Algorithm 1 Finite Element Procedure with nested loops

Input: cross-section points δ, elements resolution r, Manning roughness coefficient man, and slope s. From δ, vectors $\boldsymbol{seg_x}$ and $\boldsymbol{seg_y}$ are created which contains the pairs of consecutive coordinates in the horizontal and vertical axis, respectively. In addition, values of maximum and minimum are extracted for each label (x_{max}, y_{min}, y_{max}, y_{min}), the lowest riverbed height ly, and mid the horizontal station of ly.

mg= matrix of zeros((($x_{max} - x_{min}) * r$, (y_{max}-$y_{min}) * r$)
for j=$y_{min} * r + 1$; $y_{max} * r$ **do**
  for $i : seg_y$ **do**
    if $seg_y[i][0] >= j/r > seg_y[i][1]$ or $seg_y[i][0] <= j/r < seg_y[i][1]$ **then**
      $\text{seg}_{y2} = append(i)$
      $seg_{x2} = append((\text{seg}_x[i][0] + seg_x[i][1])/2 - mid)$
    end if
  end for
  $\text{seg}_{x3} = array(\text{seg}_{x2})$
  lw = max argument((where($seg_{x3} < 0, seg_{x3}$, -inf))
  rw = min argument((where($seg_{x3} > 0, seg_{x3}$, inf))
  if $lf == rw$ **then**
    break the loop
  end if
  for $i = f_2(lw, j/r) * r - x_{min} * r : f_2(rw, j/r) * r - x_{min} * r$ **do**
    for $k : seg_x$ **do**
      compute: calculate flow area from the matrix.
      compute: calculate relative centroid for every column.
    end for
  end for
  for $i = lw + 1 : rw$ **do**
    per=append($f_3(i)$)
  end for
  compute: calculate distance for the first segment intersected by j.
  compute: calculate distance for the last segment intersected by j.
  compute: sum the cumulated area, perimeter, top width and vertical centroid for the j water depth and then reset values.
  compute: hydraulic radius, centroid, convenyance, streamflow, flow velocity for the j water depth.
end for

```python
import matplotlib.pyplot as plt
from matplotlib import pyplot
from numpy import exp

noise = 0.01
res = 10  # To be defined by the user, this resolution means the quantity of elements between ...
    point, i.e., between
          # two coordinates (1 and 2) on the vertical axis, and for a res = 10, 10 elements will be ...
              discretized between
          # 1 and 2 coordinates. the bigger the quantity of elements, the better representation, ...
              however, it takes more
          # time of processing.
man = 0.012  # To be defined by the user, Manning roughness coefficient
s = 0.00398  # To be defined by the user, slope of the cross-section

file = open("D:/Google_drive/Meu Drive/Papers/Paper - Nota_tecnica/j1.csv")
coors = pd.read_csv(file, delimiter=';', header=None).values
plt.plot(coors[:, 0], coors[:, 1])

Ymax, Ymin, Xmax, Xmin = max(coors[:, 1]), min(coors[:, 1]), max(coors[:, 0]), min(coors[:,0 ])   # ...
    Maximum and minimum values of the list of coordinates
for m in range(len(coors)):
    if coors[m][1] ≤ Ymin:  # Looking for the middle part of the cross-section
        middle = coors[m][0]
# --- Preallocate HP --- #
area, top, = np.zeros((int(Ymax*res - Ymin*res), 1)), np.zeros((int(Ymax*res - Ymin*res), 1))
perimeter_2, y = np.zeros((int(Ymax*res - Ymin*res), 1)), np.zeros((int(Ymax*res - Ymin*res), 1))
RH, centroid = np.zeros((int(Ymax*res - Ymin*res), 1)), np.zeros((int(Ymax*res - Ymin*res), 1))
con, phi = np.zeros((int(Ymax*res - Ymin*res), 1)), np.zeros((int(Ymax*res - Ymin*res), 1))
Q, center = np.zeros((int(Ymax*res - Ymin*res), 1)), np.zeros((int(Ymax*res - Ymin*res), 1))
```

```
seg_x, seg_y = np.zeros((len(coors[:, 0]) - 1, 2)), np.zeros((len(coors[:, 0]) - 1, 2))

for i in range(len(coors) - 1):
    seg_x[i, 0], seg_x[i, 1] = coors[i, 0], coors[i+1, 0]
    seg_y[i, 0], seg_y[i, 1] = coors[i, 1], coors[i+1, 1]


def per(i):
    return math.sqrt(pow(seg_y[i, 0]- seg_y[i, 1], 2) + pow(seg_x[i, 0]-seg_x[i, 1], 2))


def image_x(i, j):  # Function that according to the horizontal position of K, returns the vertical ...
    image of the segment
    if seg_y[i, 0] == seg_y[i, 1]:  # if there is a vertical wall
        return (seg_x[i, 0]) - (((seg_y[i, 0] - j)*(seg_x[i, 0]-seg_x[i, 1])) / ((seg_y[i, ...
            0]-seg_y[i, 0]*noise) - (seg_y[i, 1]+seg_y[i, 1]*noise)))
    return (seg_x[i, 0]) - (((seg_y[i, 0] - j)*(seg_x[i, 0]-seg_x[i, 1])) / (seg_y[i, 0] - seg_y[i, ...
        1]))


def image_y(i, j):  # Function that according to the horizontal position of K, returns the vertical ...
    image of the segment
    if seg_x[i, 0] == seg_x[i, 1]:  # if there is a horizontal wall
        return (seg_y[i, 0]) - ((seg_y[i, 0] - seg_y[i, 1])/((seg_x[i, 0]-seg_x[i, ...
            0]*noise)-(seg_x[i, 1]+seg_x[i, 1]*noise)))*(seg_x[i, 0] - j)
    return (seg_y[i, 0]) - ((seg_y[i, 0] - seg_y[i, 1])/(seg_x[i, 0]-seg_x[i, 1]))*(seg_x[i, 0] - j)


mg = np.zeros((int(round((Ymax-Ymin)*res)), int(round((Xmax-Xmin)*res))), dtype=int)  # Main Grid

for j in range(int(round(Ymin*res))+1, int(round(Ymax*res))):  # Looping thought the vertical axis
    seg_x_2, seg_y_2 = np.zeros((len(seg_y), 1)), np.zeros((len(seg_y), 1))
    for i in range(len(seg_y)):  # finding the upper boundary of the water deep
        if (seg_y[i, 0] ≥ j/res > seg_y[i, 1]) or (seg_y[i, 0] ≤ j/res ≤ seg_y[i, 1]):
            seg_y_2[i, 0] = i
            seg_x_2[i, 0] = (seg_x[i, 0]+seg_x[i, 1])/2 - middle
    left_wall = np.where(seg_x_2 < 0, seg_x_2, -np.inf).argmax()  # Finding the walls that contains ...
        the current
    right_wall = np.where(seg_x_2 > 0, seg_x_2, np.inf).argmin()   # water level

    if left_wall == right_wall:  # this condition is meet when water level is higher the profile
        break

    for i in np.arange(round(image_x(left_wall, j/res)*res) - Xmin*res,  # Looping thought the ...
        horizontal axis
                       round(image_x(right_wall, j/res)*res) - Xmin*res):   # Modifying the main grid
        for k in range(len(seg_x)):
            if (seg_x[k, 0] ≤ (i / res + Xmin) < seg_x[k, 1]):  # Looking for what segment "i" ...
                belongs to.
                break
        mg[round(Ymax*res-j): int(round(Ymax*res)) - int(round(image_y(k, (i/res + Xmin))*res)), ...
            int(i)] = 1
        center[int(j - Ymin*res), 0] = ((np.count_nonzero(mg[:, int(i)] == 1)/2)/res + (image_y(k, ...
            (i/res)))) * (np.count_nonzero(mg[:, int(i)] == 1)/pow(res, 2))

    perimeter = []
    for i in range(left_wall,
                   right_wall):  # all segments between the walls but not including they selfs
        perimeter.append(per(i))
    perimeter.append(math.sqrt(pow(j/res - seg_y[left_wall, 1], 2) +
                               pow(image_x(left_wall, j/res) - seg_x[left_wall, 1],
                                   2)))  # perimeter for the left boundary
    perimeter.append(math.sqrt(pow(j/res - seg_y[right_wall, 0], 2) +
                               pow(image_x(right_wall, j/res) - seg_x[right_wall, 0],
                                   2)))  # perimeter for the right boundary

    area[int(j - Ymin*res), 0] = np.sum(mg) / pow(res, 2)
    y[int(j - Ymin*res), 0] = j / res - Ymin
    perimeter_2[int(j - Ymin*res), 0] = np.sum(perimeter)
    RH[int(j - Ymin*res), 0] = (np.sum(mg) / pow(res, 2))/np.sum(perimeter)
    top[int(j - Ymin*res), 0] = image_x(right_wall, j/res)-image_x(left_wall, j/res)
    centroid[int(j - Ymin*res), 0] = np.sum(center)/(np.sum(mg))
    con[int(j - Ymin*res), 0] = (1/man)*(np.sum(mg) / pow(res, 2))*pow((np.sum(mg) / ...
        pow(res,2))/(np.sum(perimeter)), 2/3)
    phi[int(j - Ymin*res), 0] = (np.sum(mg) / pow(res, 2))*pow((np.sum(mg) / ...
        pow(res,2))/(np.sum(perimeter)), 2/3)
```

```
    Q[int(j - Ymin*res), 0] = (1/man)*(np.sum(mg) / pow(res, 2))*pow((np.sum(mg) / ...
        pow(res,2))/(np.sum(perimeter)), 2/3)*pow(s, 1/2)

# --- Filling with Nan all extra elements in the arrays --- #
area[int(j - Ymin*res): , 0], y[int(j - Ymin*res): , 0], perimeter_2[int(j - Ymin*res): , 0] = ...
    math.nan, math.nan, math.nan
RH[int(j - Ymin*res): , 0], top[int(j - Ymin*res): , 0], centroid[int(j - Ymin*res): , 0] = ...
    math.nan, math.nan, math.nan
con[int(j - Ymin*res): , 0], phi[int(j - Ymin*res): , 0], Q[int(j - Ymin*res): , 0] = math.nan, ...
    math.nan, math.nan
plt.imshow(mg)

# ---  Plotting the HP curves --- #
fig, (ax1, ax2, ax3, ax4, ax5, ax6, ax7, ax8) = plt.subplots(1, 8)
fig.suptitle('2b')
ax1.plot(area, y)
ax1.set_xlabel('Area $(m^2)$')
ax1.set_ylabel('water depth $(m)$')
ax2.plot(perimeter_2, y)
ax2.set_xlabel('Perimeter $(m)$')
ax3.plot(top,y)
ax3.set_xlabel('Top lenght $(m)$')
ax4.plot(RH, y)
ax4.set_xlabel('Hydraulic radius $(m)$')
ax5.plot(centroid,y)
ax5.set_xlabel('Centroid $(m)$')
ax6.plot(con, y)
ax6.set_xlabel('Conveyance $(m^3/s)$')
ax7.plot(phi, y)
ax7.set_xlabel('Phi $(m^3/s)$')
ax8.plot(Q, y)
ax8.set_xlabel('Flow $(m^3/s)$')
```

E. Matlab Codes

1) HP Estimator

A read-me file gives all details of how to fill the data in the spreadsheet. In summary, the user can select the method used to enter the coordinates (e.g., flag length) and the method used to calculate flows. Moreover, the user can enter the bottom slope and roughness coefficients of the inbank and outbank areas if the DCM is used.

A table with cells painted white allows the entry of x and y coordinates, as well as roughness coefficients, lengths, and the breakpoint dividers of the channel.

Overall, this function reads the input data and return plots of

- Cross-section geometry and stage-roughness plot
- Normalized Hydraulic Properties such as: a)

```
%%% Determining Irregular Cross-section Functions %%%
% Developer: Marcus Nobrega Gomes Junior
% Date: 2022/05/03
% Goal - Calculate Hydraulic Properties of Irregular and Regular Sections
% for a given cross-sections and Manning's roughness coefficients

function [y_table, A, P, Rh, y_bar, n_med, Beta, v, B, Q, x_absolute, y,s0] = ...
    HP_estimator(flag_plot_HP,dh)
input_table = xlsread('HyProSWE_Input_Data.xlsx','Irregular_Cross_Section');
input_data = input_table(1:5,1);
input_data_coordinates = input_table(2:end,3:end);
flag_length = input_data(1,1); % If == 1, use lengths as main input data, otherwise use absolute ...
    values of x (m)
flag_method = input_data(2,1); % If == 1, SCM, else DCM
s0 = input_data(3,1); % Slope in m/m
nm = input_data(4,1); % Main channel roughness
```

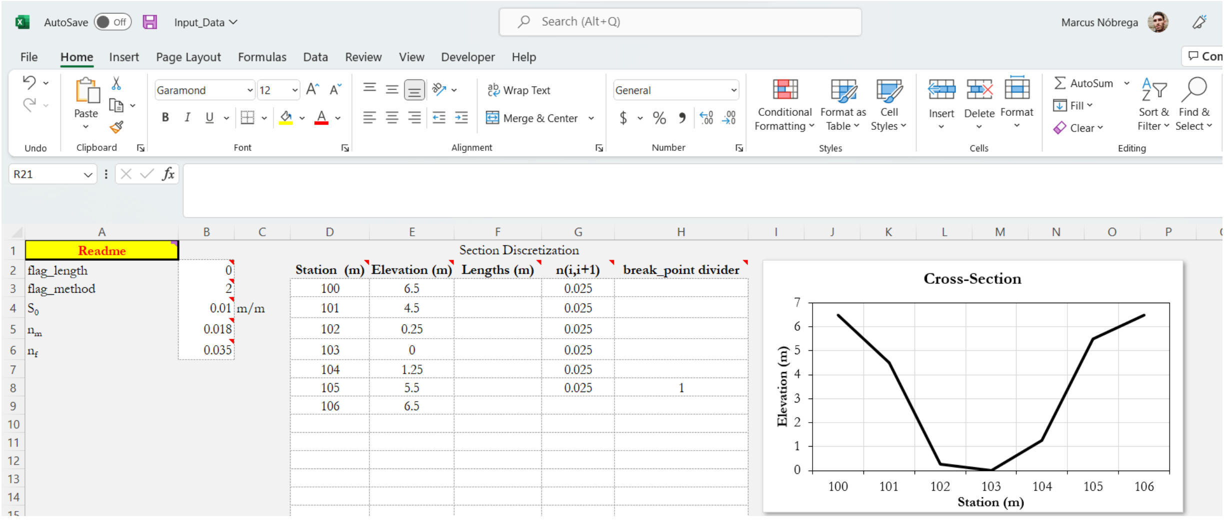

Fig. 6: Excel Spreadsheet input data file. Column B allows selecting the data entry method and the hydraulic assumption of the DCM or SCM model. Moreover, it allows entering the roughness coefficient for inbank and overbank areas. Columns D to H are relative to the cross-section. An automatic plot of the cross-section is displayed in the right of the data entry.

```
nf = input_data(5,1); % Overbanks channel roughness

if flag_method == 1
    n_channel = input_data_coordinates(1:(end-1),4);
end

% Retrieving Data
x_absolute = input_data_coordinates(:,1);
elevations = input_data_coordinates(:,2);
lengths = input_data_coordinates(1:(end-1),3);
break_point_divider = input_data_coordinates(1:(end),5);

Δ = zeros(length(elevations),1);
for i = 1:(length(elevations)-1)
    Δ(i) = abs(elevations(i+1,1) - elevations(i,1));
end
Δ_h = min(Δ(Δ > 0));
tic

% Checking input data consistency
if length(elevations) ≤ 3
    error('Please, enter at least 4 points for elevation and 3 points for manning and lengths. If ...
        you have a triangular shape, please enter the invert elevation twice and add a 0 length and ...
        0 manning, such that you have 4 points for elevation and 3 points for manning and lengths')
end

points = (1:1:length(elevations))'; % stations from 1 to n

% Let's assume a maximum 1 cm difference in the depths
% Noise
noise_max = 0.01; % m
% Let's also assume a minimum 0.1 cm difference in the depths, that is, the
% noise
noise_min = 0.001; % m
noise = Δ_h/dh; % Noise in m from user input data
if noise > noise_max
    noise = noise_max; % m
elseif noise < noise_min
    noise = noise_min; % m
end

factor = 1; %precision = 1/factor * noise

[au,ia] = unique(elevations,'stable');
Same = ones(size(elevations));
Same(ia) = 0; % repetitive values
noise_i = rand(1,1)*noise;
small_number = noise/100;
```

```
% New Elevation and X_values
ii = 0;
for i = 1:(length(elevations) - 1)
    el1 = elevations(i); el2 = elevations(i+1);
    x1 = x_absolute(i); x2 = x_absolute(i+1);
    if el1 == el2 || abs(el1 - el2) == noise
        elevations(i+1) = elevations(i+1) + noise;
        if elevations(i+1) == elevations(i)
            elevations(i+1) = elevations(i+1) + noise;
        end
    end
    if x1 == x2 || abs(x2 - x1) == noise
        x_absolute(i+1) = x_absolute(i+1) + noise;
        if x_absolute(i+1) == x_absolute(i)
            x_absolute(i+1) = x_absolute(i+1) + noise;
        end
    end
end

% if max(isnan(n_channel)) > 0
%     error('Please, enter (n-1) data for Manning coefficient, where n is the number of break-points')
% end

% Roughness Boundary Condition
if flag_method == 1
    n_channel(end+1,1) = 0; % adding last boundary condition
end

% Minimum elevation
min_el = min(elevations); % m
% y (bottom to up)
y = elevations - min_el;
pos_inv = find(y == 0); % position of invert elevation
% If we have more than 1 invert
pos_inv = pos_inv(1);

% x (left to right)
if flag_length == 1
    for i = 1:length(y) % coordinates of each measured point
        if i == 1
            x_absolute(i,1) = 0 + noise;
        else
            x_absolute(i,1) = x_absolute(i-1) + lengths(i-1) + noise;
        end
    end
else % Lengths are already assumed from the input data table
    for i = 1:length(y)
        if i ≠ length(y)
            lengths(i) = x_absolute(i+1) - x_absolute(i);
        end
    end
end

%  Alfa min
alfa_min_bound = noise/max(lengths(lengths>1e-8));
big_n = 100000*atan(asin(1)); % big number making sure it is a multiple of 1 rad, so that ...
    sin(atan(big_n)) = 1
min_length = min(lengths(lengths>0));

% Invert coordinates
x_invert = x_absolute(pos_inv,1);
y_invert = 0;

% Slopes (taking from x (left-right) y (down-up)
% For point 1 and for the last point
alfa_l = (y(1,1) - y(2,1))/lengths(1,1);

% Unsorted Values
x_left_unsorted = x_absolute(1:(pos_inv-1),1);
y_left_unsorted = y(1:(pos_inv-1),1);
x_right_unsorted = x_absolute(pos_inv + 1:end,1);
y_right_unsorted = y(pos_inv + 1:end,1);
if flag_method == 1
    n_left_unsorted = n_channel(1:(pos_inv-1),1);
    n_right_unsorted = n_channel(pos_inv:(end-1),1);
end
```

```
% Maximum depth (left and right)
max_left = max(y_left_unsorted); max_right = max(y_right_unsorted);
max_y = min(max_left, max_right);

% Refreshing values of ymax
pos_r = length(y_right_unsorted);
if max_left ≠ max_right
    if max_left > max_y % the maximum is located at left
        z = sort(y_left_unsorted,1,'descend');
        if length(z) == 1 % Case where we have a vertical wall
            z(2,1) = y_invert;
        end
        x_left_first = round(x_absolute(2) - (max_y - z(2))/alfa_l,2);
        % New values of x and y
        x_absolute(1) = x_left_first;
        y(1) = max_y;
        pos_r = length(y_right_unsorted);
    else
        pos_r = find(y_right_unsorted > max_y ,1,'first');
        alfa_r = (y_right_unsorted(pos_r) - y_right_unsorted(pos_r - ...
            1))/lengths(length(y_left_unsorted) + 1 + pos_r-1);
        z = sort(y_right_unsorted,1,'descend');
        x_rigth_last = round(x_absolute(end-1) + (max_y - z(2))/alfa_r,2);
        % New values of x and y
        x_absolute(end) = x_rigth_last;
        y(length(y_left_unsorted) + 1 + pos_r) = max_y;
    end
end
dim = 1:(length(y_left_unsorted) + 1 + pos_r);
y = y(dim,1);
x_absolute = x_absolute(dim,1);
% n_channel = n_channel(dim,1);
points = points(dim);

% New Unsorted Values with New max
x_left_unsorted = x_absolute(1:(pos_inv-1),1);
y_left_unsorted = y(1:(pos_inv-1),1);
x_right_unsorted = x_absolute(pos_inv + 1:end,1);
y_right_unsorted = y(pos_inv + 1:end,1);
if flag_method == 1
    n_left_unsorted = n_channel(1:(pos_inv-1),1);
    n_right_unsorted = n_channel(pos_inv:(end-1),1);
end

% Main Matrix
% table = [points,x_absolute,y,n_channel];

% % Vlookup Function
% Vlookup_eq = @(data,col1,val1,col2) data((find(data(:,col1)==val1,1)),col2); %Vlookup function as ...
    Excel
% Vlookup_leq = @(data,col1,val1,col2) data((find(data(:,col1)≤val1,1)),col2); %Vlookup function as ...
    Excel

% Sections left
numb_left = length(find(y_left_unsorted ≥ y_left_unsorted(end)));
% Sections right
numb_right = length(find(y_right_unsorted ≥ y_right_unsorted(1)));
% Tot sections
tot_sections = numb_left + numb_right - 1; % take one out because both sides are equal

y_l_prev = y_left_unsorted(2:length(y_left_unsorted));
y_l_next = y_left_unsorted(1:(length(y_left_unsorted)-1));

%%%% Precision
precision = 1/factor*noise; % m

%%%% small number ≥ 1 < 1e-8 + 1
sm = (1e-8 + 1);

%%%% Total_Noise
tot_noise = noise*sum(Same);
% Main loop
i = 0; int_n_p = 0; % integral of n*perimeter

%% Define Main Channel and Overbanks
pos_break = find(break_point_divider == 1); % Position where the divider occurs
% Main Channel Height
```

```
ym = y(pos_break); % Main channel height (m)
if pos_break > pos_inv % Left intersection
    % Left intersection
    posm_left = find(y_left_unsorted ≥ ym,1,'last');
    ym_left_up = y_left_unsorted(posm_left);
    xm_left_up = x_left_unsorted(posm_left);
    ym_left_down = y_left_unsorted(min(posm_left+1,length(y_left_unsorted)));
    xm_left_down = x_left_unsorted(min(posm_left+1,length(y_left_unsorted)));
    % Angles
    if (ym_left_up - ym_left_down ≤ length(y_left_unsorted)*noise)
        alfa_m_l = big_n;
    else
        alfa_m_l = (ym_left_up - ym_left_down )/(xm_left_down - xm_left_up); % Slope
    end
    xm_left = xm_left_down - (ym - ym_left_down )/alfa_m_l;
    ym_left = ym;
    % Polygons (left - inv - right)
    x_pol = [xm_left; x_left_unsorted((posm_left + 1:end),1); x_invert; ...
        x_right_unsorted(1:(pos_break-pos_inv),1)];
    y_pol = [ym_left; y_left_unsorted((posm_left + 1:end),1); y_invert; ...
        y_right_unsorted(1:(pos_break-pos_inv),1)];
    % Top-Width
    bm = abs(x_pol(1) - x_pol(end));
    % Area
    am = polyarea(x_pol,y_pol);
    % Perimeter
    polyin = polyshape(x_pol,y_pol);
    pm = perimeter(polyin) - bm; % Taking away the top width
else
    % Right Intersection
    posm_right = find(y_right_unsorted ≥ ym,1,'first');
    ym_right_up = y_right_unsorted(posm_right);
    xm_right_up = x_right_unsorted(posm_right);
    ym_right_down = y_right_unsorted(max(posm_right-1,1));
    xm_right_down = x_right_unsorted(max(posm_right-1,1));
    % Angles
    if (ym_right_up - ym_right_down < noise*length(y_right_unsorted)) % No depth
        alfa_m_r = big_n;
    else
        alfa_m_r = (ym_right_up - ym_right_down )/(xm_right_up - xm_right_down); % Slope
    end
    xm_right = xm_right_down + (ym - ym_right_down )/alfa_m_r;
    ym_right = ym;
    % Polygons (left - inv - right)
    x_pol = [x_left_unsorted(pos_break:end,1); x_invert; x_right_unsorted(1:(posm_right - 1),1); ...
        xm_right];
    y_pol = [y_left_unsorted(pos_break:end,1); y_invert; y_right_unsorted(1:(posm_right - 1),1); ...
        ym_right];
    % Top-Width
    bm = abs(x_pol(1) - x_pol(end));
    % Area
    am = polyarea(x_pol,y_pol);
    % Perimeter
    polyin = polyshape(x_pol,y_pol);
    pm = perimeter(polyin) - bm; % Taking away the top width
end
if flag_method ≠ 1
    % Number of floodplains
    if pos_break == 1 || pos_break == length(y)
        n_fp = 1;
    else
        n_fp = 2;
    end
end
while i < big_n
    %% Case where i == 1
    i = i + 1;
    n_P_left = 0;
    n_P_right = 0;
    n_P_left_extra = 0;
    n_P_right_extra = 0;
    B_extra = 0;
    P_extra = 0;
    P_extra_left = 0;
    P_extra_right = 0;
    if i == 1 % We are talking about the first point
```

```
%%% Initializing variables
y_table = 0; h = 0; B = 0; A = 0; Rh = 0; P = 0; Phi = 0; K_c = 0;
% Look to both sides from pos_inv (invert point)

% Left Direction
pos_left = find(y_left_unsorted>sm*y_invert,1,'last');
y_left_point = y_left_unsorted(pos_left,1);
x_left_point = x_left_unsorted(pos_left,1);
if flag_method == 1
    n_left_segment = n_left_unsorted(pos_left,1);
else
    n_left_segment = nm; % Main channel
end

% Right Direction
pos_right = find(y_right_unsorted>sm*y_invert,1,'first');
y_right_point = y_right_unsorted(pos_right,1);
x_right_point = x_right_unsorted(pos_right,1);
if flag_method == 1
    n_right_segment = n_right_unsorted(pos_right,1);
else
    n_right_segment = nm; % Main channel
end

%%%%%%%%%%%% Angles Calculations %%%%%%%%%%%%
%%%% Alfa Left %%%%
% Case 01 - Vertical Point
if (x_invert - x_left_point ≤ tot_noise) && (y_left_point - y_invert > tot_noise)
    alfa_l = big_n;
    alfa_l_tang = big_n;
end
% Case 02 - Horizontal Point
if (x_invert - x_left_point > tot_noise) && (y_left_point - y_invert ≤ tot_noise)
    alfa_l = big_n;
    alfa_l_tang = big_n;
end
% Case 03 - Horizontal and Vertical Point
if (x_invert - x_left_point ≤ tot_noise) && (y_left_point - y_invert ≤ tot_noise)
    alfa_l = big_n;
    alfa_l_tang = big_n;
end
% Case 04 - Poit with normal slopes
if (x_invert - x_left_point > tot_noise) && (y_left_point - y_invert > tot_noise)
    alfa_l = (y_left_point - y_invert)/(x_invert - x_left_point);
    alfa_l_tang = alfa_l;
end

%%%% Alfa Right %%%%
% Case 01 - Vertical Point
if (x_right_point - x_invert ≤ tot_noise) && (y_right_point- y_invert > tot_noise)
    alfa_r = big_n;
    alfa_r_tang = big_n;
end
% Case 02 - Horizontal Point
if (x_right_point - x_invert > tot_noise) && (y_right_point - y_invert ≤ tot_noise)
    alfa_r = big_n;
    alfa_r_tang = big_n;
end
% Case 03 - Horizontal and Vertical Point
if (x_right_point - x_invert ≤ tot_noise) && (y_right_point - y_invert ≤ tot_noise)
    alfa_r = big_n;
    alfa_r_tang = big_n;
end
% Case 04 - Poit with normal slopes
if (x_right_point - x_invert > tot_noise) && (y_right_point - y_invert > tot_noise)
    alfa_r = (y_right_point - y_invert)/(x_right_point - x_invert);
    alfa_r_tang = alfa_r;
end

% Min Angle
if alfa_l ≤ alfa_min_bound
    alfa_l_tang = big_n;
end
if alfa_r ≤ alfa_min_bound
    alfa_r_tang = big_n;
end
```

```
        if y_left_point ≤ y_right_point
            y_moving = y_left_point;
            xleft_point = x_absolute(pos_inv - 1,1);
            precision_section = min(y_left_point - y_invert,precision);
            n_points = floor((y_left_point - y_invert)/(precision_section)); % number of ...
                interpolated points
            if n_points == 1 % only one point means no slope
                if x_invert - x_left_point ≥ sm*noise && alfa_l == big_n
                    P_extra_left = sqrt((x_invert - x_left_point)^2 + (y_invert - y_left_point)^2);
                    n_P_left_extra = P_extra_left*n_left_segment^(3/2);
                    B_extra = (x_invert - x_left_point);
                else
                    B_extra = 0;
                    n_P_left_extra = 0;
                    P_extra_left;
                end
            end
            if n_points == 1 % only one point means no slope
                if x_right_point - x_invert > 1.0001*noise && alfa_r == big_n
                    P_extra_right = sqrt((x_invert - x_right_point)^2 + (y_invert - ...
                        y_right_point)^2) + B_extra;
                    B_extra = B_extra + (x_right_point - x_invert);
                    n_P_right_extra = (P_extra_right)*n_right_segment^(3/2);
                else
                    n_P_left_extra = 0;
                    P_extra_right = 0;
                end
            end
            P_extra = P_extra_right + P_extra_left;

            %%%%%%%%%%%% Main loop for i == 1 %%%%%%%%%%%%
            for j = 1:(n_points)
                h = precision_section;
                y_table(j+1,1) = y_table(j,1) + h;
                B(j+1,1) = h/alfa_l_tang + h/alfa_r_tang + B(j,1);
                A(j+1,1) = (B(j+1,1) + B(j,1))*h/2 + A(j,1); % Trapezoid
                P(j+1,1) = h/sin(atan(alfa_l_tang)) + h/sin(atan(alfa_r_tang)) + P(j,1);
                Rh(j+1,1) = A(j+1,1)/P(j+1,1);
                Phi(j+1,1) = A(j+1,1)*Rh(j+1,1)^(2/3);
                int_n_p = n_P_left_extra + n_P_right_extra + ...
                    n_left_segment^(3/2)*h/sin(atan(alfa_l_tang)) + ...
                    n_right_segment^(3/2)*h/sin(atan(alfa_r_tang)) + int_n_p;
                % Representative Roughness Coefficient
                if flag_method == 1
                    n_med(j+1,1) = (int_n_p/P(j+1,1))^(2/3);
                else
                    if y_table(j+1,1) > ym
                        yf = max(y_table(j+1,1) - ym,0); % Overbank depth
                        af = max(A(j+1,1) - (am + bm*yf),0); % Overbank flow area
                        pf = max(P(j+1,1) - pm,0); % Floodplain perimeter (m)
                        pm_star = max(pm + n_fp*yf,0);
                        am_star = max(am + bm*yf,0);
                        n_med(j+1,1) = (Phi(j+1,1))/(1/nf*af*(af/pf)^(2/3) + ...
                            1/nm*am_star*(am_star/pm_star)^(2/3));
                    else
                        yf = 0; % Overbank depth
                        af = 0; % Overbank flow area
                        pf = 0; % Floodplain perimeter (m)
                        pm_star = 0;
                        am_star = 0;
                        n_med(j+1,1) = nm;
                    end
                end
                K_c(j+1,1) = 1/n_med(j+1,1)*Phi(j+1,1);

                if j == (n_points) % final point
                    % Final point - make sure you have the exact surveyed point at the end
                    h_ = y_right_point - y_table(j,1);
                    y_table(j+1,1) = y_table(j,1) + h_;
                    B(j+1,1) = h_/alfa_l_tang + h_/alfa_r_tang + B(j,1) + B_extra;
                    A(j+1,1) = (B(j+1,1) + B(j,1))*h/2 + A(j,1); % Trapezoid
                    P(j+1,1) = h_/sin(atan(alfa_l_tang)) + h_/sin(atan(alfa_r_tang)) + P(j,1) + ...
                        P_extra;
                    Rh(j+1,1) = A(j+1,1)/P(j+1,1);
                    Phi(j+1,1) = A(j+1,1)*Rh(j+1,1)^(2/3);
                    if n_points == 1
                        int_n_p =  n_left_segment^(3/2)*h/sin(atan(alfa_l_tang)) + ...
```

```
                            n_right_segment^(3/2)*h/sin(atan(alfa_r_tang)) + n_P_right_extra + ...
                            n_P_left_extra;
                    else
                        int_n_p =  n_left_segment^(3/2)*h/sin(atan(alfa_l_tang)) + ...
                            n_right_segment^(3/2)*h/sin(atan(alfa_r_tang)) + int_n_p;
                    end
                    % Representative Roughness Coefficient
                    if flag_method == 1
                        n_med(j+1,1) = round((int_n_p/P(j+1,1))^(2/3),3);
                    else
                        if y_table(j+1,1) > ym
                            yf = max(y_table(j+1,1) - ym,0); % Overbank depth
                            af = max(A(j+1,1) - (am + bm*yf),0); % Overbank flow area
                            pf = max(P(j+1,1) - pm,0); % Floodplain perimeter (m)
                            pm_star = max(pm + n_fp*yf,0);
                            am_star = max(am + bm*yf,0);
                            n_med(j+1,1) = round((Phi(j+1,1))/(1/nf*af*(af/pf)^(2/3) + ...
                                1/nm*am_star*(am_star/pm_star)^(2/3)),3);
                        else
                            yf = 0; % Overbank depth
                            af = 0; % Overbank flow area
                            pf = 0; % Floodplain perimeter (m)
                            pm_star = 0;
                            am_star = 0;
                            n_med(j+1,1) = nm;
                        end
                    end
                    K_c(j+1,1) = 1/n_med(j+1,1)*Phi(j+1,1);
                end
            end
        else
            x_right_point = x_absolute(pos_inv + 1,1);
            precision_section = min(y_right_point - y_invert,precision);
            n_points = floor((y_right_point - y_invert)/(precision_section)); % number of ...
                interpolated points
            if n_points == 1 % only one point means no slope
                if x_right_point - x_invert ≥ sm*noise && alfa_r == big_n % Additional B_extra
                    P_extra = sqrt((x_right_point - x_invert)^2 + (y_right_point - y_invert)^2);
                    B_extra = x_right_point - x_invert;
                    n_P_right_extra = P_extra*n_right_segment^(3/2);
                else
                    B_extra = 0;
                    n_P_right_extra = 0;
                    P_extra = 0;
                end
            end
            y_moving = y_right_point;
            % For loop to calculate functions
            for j = 1:(n_points)
                h = precision_section;
                B(j+1,1) = h/alfa_l_tang + h/alfa_r_tang + B(j,1);
                y_table(j+1,1) = y_table(j,1) + h;
                A(j+1,1) = (B(j+1,1) + B(j,1))*h/2 + A(j,1); % Trapezoid
                P(j+1,1) = h/sin(atan(alfa_l_tang)) + h/sin(atan(alfa_r_tang)) + P(j,1);
                Rh(j+1,1) = A(j+1,1)/P(j+1,1);
                Phi(j+1,1) = A(j+1,1)*Rh(j+1,1)^(2/3);
                int_n_p = n_P_left_extra + n_P_right_extra + ...
                    n_left_segment^(3/2)*h/sin(atan(alfa_l_tang)) + ...
                    n_right_segment^(3/2)*h/sin(atan(alfa_r_tang)) + int_n_p;
                % Representative Roughness Coefficient
                if flag_method == 1
                    n_med(j+1,1) = round((int_n_p/P(j+1,1))^(2/3),3);
                else
                    if y_table(j+1,1) > ym
                        yf = max(y_table(j+1,1) - ym,0); % Overbank depth
                        af = max(A(j+1,1) - (am + bm*yf),0); % Overbank flow area
                        pf = max(P(j+1,1) - pm,0); % Floodplain perimeter (m)
                        pm_star = max(pm + n_fp*yf,0);
                        am_star = max(am + bm*yf,0);
                        n_med(j+1,1) = (Phi(j+1,1))/(1/nf*af*(af/pf)^(2/3) + ...
                            1/nm*am_star*(am_star/pm_star)^(2/3));
                    else
                        yf = 0; % Overbank depth
                        af = 0; % Overbank flow area
                        pf = 0; % Floodplain perimeter (m)
                        pm_star = 0;
                        am_star = 0;
```

```
                        n_med(j+1,1) = nm;
                    end
                end
                K_c(j+1,1) = 1/n_med(j+1,1)*Phi(j+1,1);
                if j == (n_points) % final point
                    % Final point - make sure you have the exact surveyed point at the end
                    h_ = y_right_point - y_table(j,1);
                    y_table(j+1,1) = y_table(j,1) + h_;
                    B(j+1,1) = h_/alfa_l_tang + h_/alfa_r_tang + B(j,1) + B_extra;
                    A(j+1,1) = (B(j+1,1) + B(j,1))*h/2 + A(j,1); % Trapezoid
                    P(j+1,1) = h_/sin(atan(alfa_l_tang)) + h_/sin(atan(alfa_r_tang)) + P(j,1) + ...
                        P_extra;
                    Rh(j+1,1) = A(j+1,1)/P(j+1,1);
                    Phi(j+1,1) = A(j+1,1)*Rh(j+1,1)^(2/3);
                    if n_points == 1
                        int_n_p =  n_left_segment^(3/2)*h/sin(atan(alfa_l_tang)) + ...
                            n_right_segment^(3/2)*h/sin(atan(alfa_r_tang)) + n_P_right_extra + ...
                            n_P_left_extra;
                    else
                        int_n_p =  n_left_segment^(3/2)*h/sin(atan(alfa_l_tang)) + ...
                            n_right_segment^(3/2)*h/sin(atan(alfa_r_tang)) + int_n_p;
                    end
                    % Representative Roughness Coefficient
                    if flag_method == 1
                        n_med(j+1,1) = (int_n_p/P(j+1,1))^(2/3);
                    else
                        if y_table(j+1,1) > ym
                            yf = max(y_table(j+1,1) - ym,0); % Overbank depth
                            af = max(A(j+1,1) - (am + bm*yf),0); % Overbank flow area
                            pf = max(P(j+1,1) - pm,0); % Floodplain perimeter (m)
                            pm_star = max(pm + n_fp*yf,0);
                            am_star = max(am + bm*yf,0);
                            n_med(j+1,1) = (Phi(j+1,1))/(1/nf*af*(af/pf)^(2/3) + ...
                                1/nm*am_star*(am_star/pm_star)^(2/3));
                        else
                            yf = 0; % Overbank depth
                            af = 0; % Overbank flow area
                            pf = 0; % Floodplain perimeter (m)
                            pm_star = 0;
                            am_star = 0;
                            n_med(j+1,1) = nm;
                        end
                    end
                    K_c(j+1,1) = 1/n_med(j+1,1)*Phi(j+1,1);
                end
            end
        end
        % Previous Positions
        pos_left_previous = pos_left;
        pos_right_previous = pos_right;
    else
        %% Case where i ≠ 1

        % Look to left sides from x_point_left and from right side of
        % x_point_right
        y_moving = y_table(end,1); % actual water depth

        % Left Direction
        pos_left = find(y_left_unsorted>sm*y_moving,1,'last');
        y_left_point = y_left_unsorted(pos_left,1);
        x_left_point = x_left_unsorted(pos_left,1);

        % Right Direction
        pos_right = find(y_right_unsorted>sm*y_moving,1,'first');
        y_right_point = y_right_unsorted(pos_right,1);
        x_right_point = x_right_unsorted(pos_right,1);

        % Roughness
        if y_moving ≤ ym % Inside of the channel
            if flag_method == 1
                n_left_segment = n_left_unsorted(pos_left,1);
                n_right_segment = n_right_unsorted(pos_right,1);
            else
                if (abs(y_left_unsorted(pos_left) - ym) ≤ noise*length(y_left_unsorted))
                    n_left_segment = nf; % Attention here
                else
                    n_left_segment = nm; % Attention here
```

```
                end
                if (abs(y_right_unsorted(pos_right) - ym) ≤ noise*length(y_right_unsorted))
                    n_right_segment = nf;  % Attention here
                else
                    n_right_segment = nm;  % Attention here
                end
            end
        else % Overbanks
            if flag_method == 1
                n_left_segment = n_left_unsorted(pos_left,1);
                n_right_segment = n_right_unsorted(pos_right,1);
            elseif y_left_unsorted(pos_left) - ym < noise*length(y_left_unsorted)% Check Noises
                n_left_segment = nm; % Attention here
                n_right_segment = nm;  % Attention here
            else
                n_left_segment = nf; % Attention here
                n_right_segment = nf;  % Attention here
            end
        end


        % Checking Discontinuities
        %%% Initializing Varaibles
        Delta_Area_left = 0; Delta_Area_right = 0;
        Delta_B_left = 0; Delta_B_right = 0;
        Delta_P_left = 0; Delta_P_right = 0;

        %%%%%%%%%%%% Angles Calculation %%%%%%%%%%%%
        if pos_left + 1 > length(y_left_unsorted)
            x_prev_left = x_invert;
            y_prev_left = y_invert;
        else
            x_prev_left = (x_left_unsorted(pos_left + 1,1));
            y_prev_left = (y_left_unsorted(pos_left + 1,1));
        end

        %%%% Alfa Left %%%%
        % Case 01 - Vertical Point
        if (x_prev_left - x_left_point ≤ tot_noise) && (y_left_point - y_prev_left > tot_noise)
            alfa_l = big_n;
            alfa_l_tang = big_n;
        end
        % Case 02 - Horizontal Point
        if (x_prev_left - x_left_point > tot_noise) && (y_left_point - y_prev_left ≤ tot_noise)
            alfa_l = big_n;
            alfa_l_tang = big_n;
        end
        % Case 03 - Horizontal and Vertical Point
        if (x_prev_left - x_left_point ≤ tot_noise) && (y_left_point - y_prev_left ≤ tot_noise)
            alfa_l = big_n;
            alfa_l_tang = big_n;
        end
        % Case 04 - Poit with normal slopes
        if (x_prev_left - x_left_point > tot_noise) && (y_left_point - y_prev_left > tot_noise)
            alfa_l = (y_left_point - y_prev_left)/(x_prev_left - x_left_point);
            alfa_l_tang = alfa_l;
        end
        if pos_right == 1
            x_prev_right = x_invert;
            y_prev_right = y_invert;
        else
            x_prev_right = x_right_unsorted(pos_right - 1,1);
            y_prev_right = y_right_unsorted(pos_right - 1,1);
        end
        %%%% Alfa Right %%%%
        % Case 01 - Vertical Point
        if (x_right_point - x_prev_right ≤ tot_noise) && (y_right_point- y_prev_right > tot_noise)
            alfa_r = big_n;
            alfa_r_tang = big_n;
        end
        % Case 02 - Horizontal Point
        if (x_right_point - x_prev_right > tot_noise) && (y_right_point - y_prev_right ≤ tot_noise)
            alfa_r = big_n;
            alfa_r_tang = big_n;
        end
        % Case 03 - Horizontal and Vertical Point
        if (x_right_point - x_prev_right ≤ tot_noise) && (y_right_point - y_prev_right ≤ tot_noise)
```

```
            alfa_r = big_n;
            alfa_r_tang = big_n;
        end
        % Case 04 - Poit with normal slopes
        if (x_right_point - x_prev_right > tot_noise) && (y_right_point - y_prev_right > tot_noise)
            alfa_r = (y_right_point - y_prev_right)/(x_right_point - x_prev_right);
            alfa_r_tang = alfa_r;
        end

        % Min Angle
        if alfa_l ≤ alfa_min_bound
            alfa_l_tang = big_n;
        end
        if alfa_r ≤ alfa_min_bound
            alfa_r_tang = big_n;
        end


        if (pos_left_previous - pos_left) > 1 % More than one movement

            % intersect
            if alfa_l_tang == 0
                x_intersect = x_left_unsorted(pos_left + 1,1);
            else
                x_intersect = x_left_unsorted(pos_left + 1,1) - (y_moving - ...
                    y_left_unsorted(pos_left + 1,1))/alfa_l;
            end
            x_pol = []; y_pol = [];
            for nn = 1:(pos_left_previous - pos_left)
                x_pol = [x_pol; x_left_unsorted(pos_left_previous - nn + 1)];
                y_pol = [y_pol; y_left_unsorted(pos_left_previous - nn + 1)];
            end
            % Adding intersection
            x_pol = [x_pol;x_intersect];
            y_pol = [y_pol;y_moving];
            % Delta B
            Delta_B_left = abs(x_pol(1) - x_pol(end));
            % Delta A
            Delta_Area_left = polyarea(x_pol,y_pol);
            % Delta P
            polyin = polyshape(x_pol,y_pol);
            Delta_P_left = perimeter(polyin) - Delta_B_left; % Taking away top width
            n_P_left = Delta_P_left*n_left_segment^(3/2);
            % Delta Rh left
            % Phi left
            % Conductance Left
        end


        % Checking Discontinuities
        if (pos_right - pos_right_previous) > 1 % More than one movement
            % intersect
            if alfa_r_tang == 0
                x_intersect = x_right_unsorted(pos_right - 1,1);
            else
                x_intersect = x_right_unsorted(pos_right - 1,1) + (y_moving - ...
                    y_right_unsorted(pos_right - 1,1))/alfa_r;
            end
            x_pol = []; y_pol = [];
            for nn = 1:(pos_right - pos_right_previous)
                x_pol = [x_pol; x_right_unsorted(pos_right_previous + nn - 1)];
                y_pol = [y_pol; y_right_unsorted(pos_right_previous + nn - 1)];
            end
            % Adding intersection
            x_pol = [x_pol;x_intersect];
            y_pol = [y_pol;y_moving];
            % Delta B
            Delta_B_right = abs(x_pol(1) - x_pol(end));
            % Delta A
            Delta_Area_right = polyarea(x_pol,y_pol);
            % Delta P
            polyin = polyshape(x_pol,y_pol);
            Delta_P_right = perimeter(polyin) - Delta_B_right; % Taking away top width
            % Manning * Perimeter
            n_P_right = Delta_P_right*n_right_segment^(3/2);
        end
        y_moving_end = min(y_right_point,y_left_point);
```

```
        %         if (y_moving_end - y_moving)/(precision/100) < 1
        %             error('Please, increase precision. Instability!')
        %         end
        precision_section = min(y_moving_end - y_moving,precision); % meters
        if y_moving_end - y_moving < precision
            ttt = 1;
        end
        n_points = floor((y_moving_end - y_moving)/(precision_section)); % number of interpolated ...
            points
        % For loop to calculate functions
        if n_points == 1 % only one point means no slope
            if y_moving_end == y_right_point && y_moving_end == y_left_point && alfa_l == big_n && ...
                alfa_r == big_n
                B_extra = x_right_point - x_prev_right + x_prev_left - x_left_point;
                P_extra_left = sqrt((x_prev_left - x_left_point)^2 + (y_prev_left - y_left_point)^2);
                P_extra_right = sqrt((x_right_point - x_prev_right)^2 + (y_right_point - ...
                    y_prev_right)^2);
            elseif y_moving_end == y_right_point && alfa_r == big_n
                if pos_right == 1
                    P_extra_right = sqrt((x_right_point - x_invert)^2 + (y_right_point - y_invert)^2);
                    B_extra = x_right_point - x_invert;
                else
                    P_extra_right = sqrt((x_right_point - x_prev_right)^2 + (y_right_point -  ...
                        y_prev_right)^2);
                    B_extra = x_right_point - x_prev_right;
                end
            else % y_moving == y_left
                if pos_left + 1 > length(x_left_unsorted) && alfa_l == big_n
                    P_extra_left = sqrt((x_invert - x_left_point)^2 + (y_invert - y_left_point)^2);
                    B_extra = x_invert - x_left_point;
                elseif alfa_l == big_n
                    P_extra_left = sqrt((x_prev_left - x_left_point)^2 + (y_prev_left - ...
                        y_left_point)^2);
                    B_extra = x_prev_left - x_left_point;
                end
                % Right
                if pos_right == 1 && alfa_r == big_n
                    P_extra_right = sqrt((x_invert - x_right_point)^2 + (y_invert - y_right_point)^2);
                    B_extra = x_right_point - x_invert + B_extra;
                elseif alfa_r == big_n
                    P_extra_left = sqrt((x_prev_right - x_right_point)^2 + (y_right_point -  ...
                        y_prev_right^2));
                    B_extra = x_right_point - x_prev_right + B_extra;
                end

            end
            P_extra = P_extra_left + P_extra_right;
            n_P_right_extra = P_extra_right*n_right_segment^(3/2);
            n_P_left_extra = P_extra_left*n_left_segment^(3/2);
        else
            B_extra = 0;
            n_P_right_extra = 0;
            n_P_left_extra = 0;
            P_extra = 0;
            P_extra_left = 0;
            P_extra_right = 0;
        end

        dim_table = length(y_table);
        %%%%%%%%%%%% Main loop for i ≠ 1 %%%%%%%%%%%%

        for j = 1:(n_points)
            k = dim_table + j;
            if  j == 1 % We have to add values from discontinuity (Deltas)
                h = precision_section; % meters
                y_table(k,1) = y_table(k-1,1) + h;
                % Roughness
                if y_table(k,1) ≤ ym % Inside of the channel
                    if flag_method == 1
                        n_left_segment = n_left_unsorted(pos_left,1);
                        n_right_segment = n_right_unsorted(pos_right,1);
                    else
                        if (abs(y_left_unsorted(pos_left) - ym) ≤ noise*length(y_left_unsorted))
                            n_left_segment = nf; % Attention here
                        else
                            n_left_segment = nm; % Attention here
                        end
```

```
                        if (abs(y_right_unsorted(pos_right) - ym) ≤ noise*length(y_right_unsorted))
                            n_right_segment = nf;  % Attention here
                        else
                            n_right_segment = nm;  % Attention here
                        end
                    end
                else % Overbanks
                    if flag_method == 1
                        n_left_segment = n_left_unsorted(pos_left,1);
                        n_right_segment = n_right_unsorted(pos_right,1);
                    elseif y_left_unsorted(pos_left) - ym < noise*length(y_left_unsorted)% Check Noises
                        n_left_segment = nm; % Attention here
                        n_right_segment = nm;  % Attention here
                    else
                        n_left_segment = nf; % Attention here
                        n_right_segment = nf;  % Attention here
                    end
                end
                B(k,1) = B(k-1,1) + Delta_B_left + Delta_B_right +  h/alfa_l_tang + h/alfa_r_tang;
                A(k,1) = A(k-1,1) + (B(k,1) + B(k-1,1))*h/2 + Delta_Area_left + Delta_Area_right;
                P(k,1) = h/sin(atan(alfa_l_tang)) + h/sin(atan(alfa_r_tang)) + P(k-1,1) + ...
                    Delta_P_left + Delta_P_right;
                Rh(k,1) = A(k,1)/P(k,1);
                Phi(k,1) = A(k,1)*Rh(k,1)^(2/3);
                int_n_p =  n_P_left + n_P_right + n_P_right_extra + n_P_left_extra + ...
                    n_left_segment^(3/2)*h/sin(atan(alfa_l_tang)) + ...
                    n_right_segment^(3/2)*h/sin(atan(alfa_r_tang)) + int_n_p;
                % Representative Roughness Coefficient
                if flag_method == 1
                    n_med(k,1) = (int_n_p/P(k,1))^(2/3);
                else
                    if y_table(k,1) > ym
                        yf = max(y_table(k,1) - ym,0); % Overbank depth
                        af = max(A(k,1) - (am + bm*yf),0); % Overbank flow area
                        pf = max(P(k,1) - pm,0); % Floodplain perimeter (m)
                        pm_star = max(pm + n_fp*yf,0);
                        am_star = max(am + bm*yf,0);
                        n_med(k,1) = (Phi(k,1))/(1/nf*af*(af/pf)^(2/3) + ...
                            1/nm*am_star*(am_star/pm_star)^(2/3));
                    else
                        yf = 0; % Overbank depth
                        af = 0; % Overbank flow area
                        pf = 0; % Floodplain perimeter (m)
                        pm_star = 0;
                        am_star = 0;
                        n_med(k,1) = nm;
                    end
                end
                K_c(k,1) = 1/n_med(k,1)*Phi(k,1);
            else
                % Functions in terms of depth
                h = precision_section;
                y_table(k,1) = h + y_table(k-1,1);
                % Roughness
                if y_table(k,1) ≤ ym % Inside of the channel
                    if flag_method == 1
                        n_left_segment = n_left_unsorted(pos_left,1);
                        n_right_segment = n_right_unsorted(pos_right,1);
                    else
                        if (abs(y_left_unsorted(pos_left) - ym) ≤ noise*length(y_left_unsorted))
                            n_left_segment = nf; % Attention here
                        else
                            n_left_segment = nm; % Attention here
                        end
                        if (abs(y_right_unsorted(pos_right) - ym) ≤ noise*length(y_right_unsorted))
                            n_right_segment = nf;  % Attention here
                        else
                            n_right_segment = nm;  % Attention here
                        end
                    end
                else % Overbanks
                    if flag_method == 1
                        n_left_segment = n_left_unsorted(pos_left,1);
                        n_right_segment = n_right_unsorted(pos_right,1);
                    elseif y_left_unsorted(pos_left) - ym < noise*length(y_left_unsorted)% Check Noises
                        n_left_segment = nm; % Attention here
                        n_right_segment = nm;  % Attention here
```

```
                else
                    n_left_segment = nf; % Attention here
                    n_right_segment = nf;  % Attention here
                end
            end
            B(k,1) = h/alfa_l_tang + h/alfa_r_tang + B(k-1,1);
            A(k,1) = (B(k,1) + B(k-1,1))*h/2 + A(k-1,1); % Trapezoid
            P(k,1) = h/sin(atan(alfa_l_tang)) + h/sin(atan(alfa_r_tang)) + P(k-1,1);
            Rh(k,1) = A(k,1)/P(k,1);
            Phi(k,1) = A(k,1)*Rh(k,1)^(2/3);
            int_n_p = n_left_segment^(3/2)*h/sin(atan(alfa_l_tang)) + ...
                n_right_segment^(3/2)*h/sin(atan(alfa_r_tang)) + int_n_p;
            % Representative Roughness Coefficient
            if flag_method == 1
                n_med(k,1) = (int_n_p/P(k,1))^(2/3);
            else
                if y_table(k,1) > ym
                    yf = max(y_table(k,1) - ym,0); % Overbank depth
                    af = max(A(k,1) - (am + bm*yf),0); % Overbank flow area
                    pf = max(P(k,1) - pm,0); % Floodplain perimeter (m)
                    pm_star = max(pm + n_fp*yf,0);
                    am_star = max(am + bm*yf,0);
                    n_med(k,1) = (Phi(k,1))/(1/nf*af*(af/pf)^(2/3) + ...
                        1/nm*am_star*(am_star/pm_star)^(2/3));
                else
                    yf = 0; % Overbank depth
                    af = 0; % Overbank flow area
                    pf = 0; % Floodplain perimeter (m)
                    pm_star = 0;
                    am_star = 0;
                    n_med(k,1) = nm;
                end
            end
            K_c(k,1) = 1/n_med(k,1)*Phi(k,1);
        end

        if j == (n_points) % final point
            % Final point - make sure you have the exact surveyed point at the end
            h_ = y_moving_end - y_table(k-1,1);
            y_table(k,1) = y_table(k-1,1) + h_;
            % Roughness
            if y_table(k,1) ≤ ym % Inside of the channel
                if flag_method == 1
                    n_left_segment = n_left_unsorted(pos_left,1);
                    n_right_segment = n_right_unsorted(pos_right,1);
                else
                    if (abs(y_left_unsorted(pos_left) - ym) ≤ noise*length(y_left_unsorted))
                        n_left_segment = nf; % Attention here
                    else
                        n_left_segment = nm; % Attention here
                    end
                    if (abs(y_right_unsorted(pos_right) - ym) ≤ noise*length(y_right_unsorted))
                        n_right_segment = nf;  % Attention here
                    else
                        n_right_segment = nm;  % Attention here
                    end
                end
            else % Overbanks
                if flag_method == 1
                    n_left_segment = n_left_unsorted(pos_left,1);
                    n_right_segment = n_right_unsorted(pos_right,1);
                elseif y_left_unsorted(pos_left) - ym < noise*length(y_left_unsorted)% Check Noises
                    n_left_segment = nm; % Attention here
                    n_right_segment = nm;  % Attention here
                else
                    n_left_segment = nf; % Attention here
                    n_right_segment = nf;  % Attention here
                end
            end
            B(k,1) = h_/alfa_l_tang + h_/alfa_r_tang + B(k-1,1) + B_extra;
            A(k,1) = (B(k,1) + B(k-1,1))*h_/2 + A(k-1,1); % Trapezoid
            P(k,1) = h_/sin(atan(alfa_l_tang)) + h_/sin(atan(alfa_r_tang)) + P(k-1,1) + P_extra;
            Rh(k,1) = A(k,1)/P(k,1);
            Phi(k,1) = A(k,1)*Rh(k,1)^(2/3);
            int_n_p = n_left_segment^(3/2)*h/sin(atan(alfa_l_tang)) + ...
                n_right_segment^(3/2)*h/sin(atan(alfa_r_tang)) + int_n_p;
            % Representative Roughness Coefficient
```

```
                if flag_method == 1
                    n_med(k,1) = (int_n_p/P(k,1))^(2/3);
                else
                    if y_table(k,1) ≥ ym
                        yf = max(y_table(k,1) - ym,0); % Overbank depth
                        af = max(A(k,1) - (am + bm*yf),0); % Overbank flow area
                        pf = max(P(k,1) - pm,0); % Floodplain perimeter (m)
                        pm_star = max(pm + n_fp*yf,0);
                        am_star = max(am + bm*yf,0);
                        n_med(k,1) = (Phi(k,1))/(1/nf*af*(af/pf)^(2/3) + ...
                            1/nm*am_star*(am_star/pm_star)^(2/3));
                    else
                        yf = 0; % Overbank depth
                        af = 0; % Overbank flow area
                        pf = 0; % Floodplain perimeter (m)
                        pm_star = 0;
                        am_star = 0;
                        n_med(k,1) = nm;
                    end
                end
                K_c(k,1) = 1/n_med(k,1)*Phi(k,1);
            end
        end
        % Previous Positions
        pos_left_previous = pos_left;
        pos_right_previous = pos_right;
    end
    % Checking i
    if round(y_table(end),3) == round(max_y,3) % Stop de algorithm
        i = big_n;
    end
end

% Centroid Coordinates
int_a_y = 0; % Integral of A(y)dy
for i = 1:(length(A))
    if i == 1
        y_bar(i,1) = 0;
        int_a_y(i,1) = 0;
    else
        int_a_y(i,1) = (A(i) - A(i-1))*(y_table(i) + y_table(i-1))/2 + int_a_y(i-1);
        y_bar(i,1) = int_a_y(i,1)/A(i,1);
    end
end

% Flow Discharge Calculations
Q = K_c*sqrt(s0);

% Velocity
v = Q./A; % m/s

% Beta - Boussinesq factor
kappa = 0.41;
g = 9.81; % m/s2
Beta = (1 + (g*n_med.^2)./(Rh.^(1/3)*kappa^2));

%% Plotting Results
% Plotting Channel
if flag_plot_HP == 1
close all
subplot(1,2,1)
set(gcf,'units','inches','position',[4,4,6.5,4])
mark_size = 5;
plot(x_absolute,y,'linewidth',2,'color','black')
xlabel('x ($m$)','Interpreter','latex');
ylabel('y ($m$)','Interpreter','latex');
xlim([min(x_absolute) max(x_absolute)])
grid on
hold on
scatter(x_absolute,y,'black')
subplot(1,2,2)
n_med(1,1) = inf;
plot(n_med(2:end,1),y_table(2:end,1),'linewidth',2,'color','black')
xlabel('Manning`s coefficient (SI)','Interpreter','latex');
ylabel('y ($m$)','Interpreter','latex');
xlim([0.9*min(n_med) 1.1*max(n_med(¬isinf(n_med)))])
grid on
```

```
exportgraphics(gcf,'Cross_Section.pdf','ContentType','vector')


subplot(2,4,1)
set(gcf,'units','inches','position',[4,2,7.5,5])
sz = 5;
c = linspace(1,sz,length(y_table));
scatter(A,y_table,sz,c,'filled')
grid on
grid on
xlabel('Area ($m^2$)','Interpreter','latex');
ylabel('y ($m$)','Interpreter','latex');
% xlim([0 4])
subplot(2,4,2)
grid on
scatter(P,y_table,sz,c,'filled')
grid on
xlabel('Perimeter ($m$)','Interpreter','latex');
ylabel('y ($m$)','Interpreter','latex');
% xlim([0 4])
subplot(2,4,3)
grid on
scatter(Rh,y_table,sz,c,'filled')
grid on
xlabel('Hydraulic Radius ($m$)','Interpreter','latex');
ylabel('y ($m$)','Interpreter','latex');
% xlim([0 4])
subplot(2,4,4)
grid on
scatter(B,y_table,sz,c,'filled')
grid on
xlabel('Top width ($m$)','Interpreter','latex');
ylabel('y ($m$)','Interpreter','latex');
subplot(2,4,5)
grid on
scatter(K_c,y_table,sz,c,'filled')
grid on
xlabel('Conveyance ($m^3/s$)','Interpreter','latex');
ylabel('y ($m$)','Interpreter','latex');
subplot(2,4,6)
sz = 5;
c = linspace(1,sz,length(y_table));
scatter(Phi,y_table,sz,c,'filled')
grid on
xlabel('$\Phi$ ($m^{5/3}$)','Interpreter','latex');
ylabel('y ($m$)','Interpreter','latex');
subplot(2,4,7)
scatter(y_bar,y_table,sz,c,'filled')
grid on
xlabel('$\bar{y}$ (m)','Interpreter','latex');
ylabel('y ($m$)','Interpreter','latex');
subplot(2,4,8)
scatter(Q,y_table,sz,c,'filled')
grid on
xlabel('Flow discharge ($m^3/s$)','Interpreter','latex');
ylabel('y ($m$)','Interpreter','latex');
exportgraphics(gcf,'Hydraulic_Properties.pdf','ContentType','vector')
toc

% Rating Curve
close all
subplot(3,1,1)
set(gcf,'units','inches','position',[4,4,6.5,4])
mark_size = 5;
plot(x_absolute,y,'linewidth',2,'color','black')
xlabel('x ($m$)','Interpreter','latex');
ylabel('y ($m$)','Interpreter','latex');
xlim([min(x_absolute) max(x_absolute)])
grid on
subplot(3,1,2)
scatter(Q,y_table,sz,c,'filled')
xlabel('Flow discharge ($m^3/s$)','Interpreter','latex');
ylabel('y ($m$)','Interpreter','latex');
grid on
box on
% Velocity
subplot(3,1,3)
```

```
scatter(Q./A,y_table,sz,c,'filled')
xlabel('Velocity ($m/s$)','Interpreter','latex');
ylabel('y ($m$)','Interpreter','latex');
grid on
box on
exportgraphics(gcf,'Rating Curve.pdf','ContentType','vector')

% Plotting Normalized Values
set(gcf,'units','inches','position',[4,2,8,4])
subplot(1,5,1)
scatter(Q/max(Q),y_table/max(y_table),sz,c,'filled')
xlabel('$Q/Q_p$','Interpreter','latex');
ylabel('$y/y_{max} $','Interpreter','latex');
title(['$Q_p (m^3/s) = $ ',num2str(round(max(Q),2))],'interpreter','latex')
axis equal
grid on
xlim([0 1]); ylim([0 1]);
subplot(1,5,2)
scatter(A/max(A),y_table/max(y_table),sz,c,'filled')
xlabel('$A/A_{max}$','Interpreter','latex');
ylabel('$y/y_{max}$','Interpreter','latex');
title(['$A_{max} (m^2) = $ ',num2str(round(max(A),2))],'interpreter','latex')
axis equal
grid on
xlim([0 1]); ylim([0 1]);
subplot(1,5,3)
scatter(Phi/max(Phi),y_table/max(y_table),sz,c,'filled')
xlabel('$\Phi/\Phi_{max}$','Interpreter','latex');
ylabel('$y/y_{max} $','Interpreter','latex');
title(['$\Phi_{max} (m^2) = $ ',num2str(round(max(Phi),2))],'interpreter','latex')
axis equal
grid on
xlim([0 1]); ylim([0 1]);
subplot(1,5,4)
scatter(K_c/max(K_c),y_table/max(y_table),sz,c,'filled')
xlabel('$K_c/K_{c,max}$','Interpreter','latex');
ylabel('$y/y_{max} $','Interpreter','latex');
title(['$K_{c,max} (m^3/s) = $ ',num2str(round(max(K_c),2))],'interpreter','latex')
axis equal
grid on
xlim([0 1]); ylim([0 1]);
subplot(1,5,5)
scatter((Q./A)/(max(Q./A)),y_table/max(y_table),sz,c,'filled')
xlabel('$v/v_{c,max}$','Interpreter','latex');
ylabel('$y/y_{max} $','Interpreter','latex');
title(['$v_{max} (m/s) = $ ',num2str(round(max(Q./A),2))],'interpreter','latex')
axis equal
grid on
xlim([0 1]); ylim([0 1]);
exportgraphics(gcf,'Normalized_Values.pdf','ContentType','vector')
close all

end
end
```

2) Read Input Data - SVE

This script reads the excel input data and converts them into Matlab arrays.

```
%%% --------- HyProSWE Model ----------- %%%
% Script to read input data
% Developer: Marcus Nobrega Gomes Junior
% 5/1/2023
% Goal: Solution of 1-D SVE for given cross-section functions of Area, Perimeter, and
% top Width
% If you have any issues, please contact me at
% marcusnobrega.engcivil@gmail.com

% ---------- Please, don't change anything below ------------ %

%% Read Input Data %%
data = readtable('HyProSWE_Input_Data.xlsx','Sheet','Input_Data');
```

```
b = 0; Z1 = 0; Z2 = 0; a = 0; D = 0;


% General Data
general_data = table2array(data(1:16,2));
L = general_data(1,1);
Nx = general_data(2,1);
el = general_data(3,1);
g = general_data(4,1);
nm = general_data(5,1);
I0 = general_data(6,1);
tf = general_data(7,1);
dt = general_data(8,1);
animation_time = general_data(9,1);
s_outlet = general_data(10,1);
dh = general_data(11,1);
alpha = general_data(12,1);
dtmin = general_data(13,1);
dtmax = general_data(14,1);


% Flags
flags = table2array(data(19:29,2));
flag_hydrograph = flags(1,1);
flag_outlet = flags(2,1);
flag_friction = flags(3,1);
flag_section = flags(4,1);
flag_stage_hydrograph = flags(5,1);
flag_nash = flags(6,1);
flag_slope = flags(7,1);
flag_elevation = flags(8,1);
flag_output = flags(9,1);
flag_plot_HP = flags(10,1);
flag_elapsed_time = flags(11,1);
if flag_elapsed_time ≠ 1
    Date_Begin = general_data(15,1);
    Date_Begin = datetime(datestr(Date_Begin+datenum('30-Dec-1899')));
    Date_End = general_data(16,1);
    Date_End = datetime(datestr(Date_End+datenum('30-Dec-1899')));
end

if flag_nash == 1
    nash_data = table2array(data(1:4,5));
    % Hydrograph
    Tp = nash_data(1,1);
    Qb = nash_data(2,1);
    Beta = nash_data(3,1);
    Qp = nash_data(4,1);
else
    % Input Hydrograph
    input_hydrograph_data = table2array(data(8:end,4:5));
    time_ = input_hydrograph_data(1:end,1);
    Qe1_ = input_hydrograph_data(1:end,2);
    Qe1 = zeros(size(Qe1_,1) - sum(isnan(Qe1_)),1);
    time = zeros(size(time_,1) - sum(isnan(time_)),1);
    % Taking away nans
    for i = 1:length(Qe1)
        if isnan(Qe1_(i)) || isnan(time_(i))
            break
        else
            Qe1(i,1) = Qe1_(i,1);
            time(i,1) = time_(i,1);
        end
    end
    clear Qe1_ time_
end

if flag_stage_hydrograph ≠ 0
    % Stage Hydrograph
    input_stage_data = table2array(data(8:end,7:8));
    time_stage_ = input_stage_data(1:end,1);
    he1_ = data(1:end,2);
    he1 = zeros(size(he1_,1) - sum(isnan(he1_)),1);
    time_stage = zeros(size(time_stage_,1) - sum(isnan(time_stage_)),1);
    % Taking away nans
    for i = 1:length(he1)
        if isnan(he1_(i)) || isnan(time_stage_(i))
```

```
            break
        else
            he1(i,1) = he1_(i,1);
            time_stage(i,1) = time_stage_(i,1);
        end
    end
    clear Qe1_ time_stage_
end

if flag_slope ≠ 0
    % Slope
    input_slope_data = table2array(data(8:end,7:8));
    station = input_slope_data(1:end,1);
    bottom_slope = input_slope_data(2:end,2);
    slopes_not_nan = zeros(size(bottom_slope,1) - sum(isnan(bottom_slope)),1);
    station_index = zeros(size(station,1) - sum(isnan(station)),1);
    % Taking away nans
    for i = 1:length(station_index)
        if isnan(bottom_slope(i)) || isnan(station(i))
            break
        else
            slopes_not_nan(i,1) = bottom_slope(i,1);
            station_index(i,1) = station(i,1);
        end
    end
    clear station_index station bottom_slope
    bottom_slope = slopes_not_nan;
end

if flag_elevation ≠ 0
    % Slope
    input_slope_data = table2array(data(8:end,7:8));
    station = input_slope_data(1:end,1);
    elevation_cell = table2array(data(8:end,7:8));
    inv_el_ = zeros(size(elevation_cell,1) - sum(isnan(elevation_cell)),1);
    station_index = zeros(size(station,1) - sum(isnan(station)),1);
    % Taking away nans
    for i = 1:length(station_index)
        if isnan(elevation_cell(i)) || isnan(station(i))
            break
        else
            inv_el_(i,1) = elevation_cell(i,1);
            station_index(i,1) = station(i,1);
        end
    end
    inv_el = inv_el_; % Invert Elevation
    clear station_index station elevation_cell inv_el_
end

% Outlet
if flag_outlet ≠1
    input_slope_wave = table2array(data(8:5,8));
    h_0_wave = input_slope_wave(1,1);
    H_0_wave = input_slope_wave(2,1);
    L_wave = input_slope_wave(3,1);
    T_wave = input_slope_wave(4,1);
    x_wave = input_slope_wave(5,1);
end

% Section
if flag_section == 1
    input_slope_trapezoid = table2array(data(1:3,11));
    b = input_slope_trapezoid(1,1);
    Z1 = input_slope_trapezoid(2,2);
    Z2 = input_slope_trapezoid(3,3);
elseif flag_section == 2
    input_slope_circular = table2array(data(1,14));
    D = input_slope_circular(1,1);
elseif flag_section == 3
    input_slope_parabolic = table2array(data(3,14));
    a = data(1,1);
else
    % Read HP estimator data
    [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr, x_cross, ...
        y_cross,s0] = HP_estimator(flag_plot_HP,dh);
    irr_table = [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];

```

```
    % Some Boundary Conditions
    % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
    % [   1,     2,      3,       4,      5,              6,           7,   8,       9,   10]
    irr_table(1,6) = irr_table(2,6); irr_table(1,7) = 0; irr_table(1,8) = 0;
    % Second Line
    irr_table(2,2) = 0; irr_table(2,3) = 0; irr_table(2,4) = 0; irr_table(2,5) = 0; irr_table(2,7) ...
        = 0; irr_table(2,8) = 0; irr_table(2,9) = 0;
%     z = irr_table;
%     second = 0*z(2,:);
%     second(1,1) = 0.5*10^-3; second(1,6) = z(1,6); second(1,10) = z(1,10)/2;
%     z = [z(1,:) ; second; z(2:end,:)];
%     irr_table = z;
end


% Contraint at observed flow
if flag_hydrograph == 1
    if max(time) ≠ tf
        z = round(tf - max(time),0);
        for i = 1:z
            Qe1(end + 1,1) = 0;
            time(end+1,1) = time(end,1) + 1;
        end
    end
end


% Contraint at stage hydrograph
if flag_stage_hydrograph == 1
    if max(time_stage) ≠ tf
        z = round(tf - max(time_stage),0);
        for i = 1:z
            he1(end + 1,1) = 0;
            time_stage(end+1,1) = time_stage(end,1) + 1;
        end
    end
end
```

3) SVE Model

The following algorithm solves the 1-D SVE using the Lax-Friedrichs method. To run the SVE Model, 3 functions are required: The SVE Model V1, the Read Input Data, and the HP Estimator, explained in the previous section.

```
%%% --------- HyProSWE Model ----------- %%%
% Developer: Marcus Nobrega Gomes Junior
% 5/1/2023
% Goal: Solution of 1-D SVE for given cross-section functions of Area, Perimeter, and
% top Width
% If you have any issues, please contact me at
% marcusnobrega.engcivil@gmail.com


% -------------------- All Rights Reserved ---------------------- %

%% 1.0 -  Pre-Processing

clear all
clc
warning('off') % Deactivate Warnings

% Reading the Input Data
Read_Input_Data % Here we read the .xlsx input data file. Please don't change the name of this file.

% Checking if at least one boundary condition is considered
if flag_hydrograph ≠ 1 && flag_nash ≠ 1 && flag_stage_hydrograph ≠ 1 && flag_outlet ≠ 0
    error('Please enter at least 1 internal boundary condition.')
end

% Checking if there is conflicting boundary conditions
if flag_hydrograph == 1 && flag_nash == 1
    error('Please choose either an observed inflow hydrograph entered in a tabular format or a ...
        nash-type hydrograph.')
```

```
end

% Checking if there is conflicting cross section
if flag_section > 4
    error('Please, enter a the index indicating which type of cross-section is being simulated. ...
        Read the instruction in the .xlsx file')
end

% Checking if there is conflicting cross section
if flag_stage_hydrograph == 1 && flag_hydrograph == 1
    error('Please, the inlet can only have either a stage hydrograph or a flow hydrograph')
end
%% 2.0 - Initial Boundary Conditions
% ------------- Inflow Hydrograph ------------- %
if flag_hydrograph == 1
    % We already read the hydrograph in Read_Input_Data file
elseif flag_nash == 1
    % 2nd option - Model the hydrograph using a nash function
    %%% Q(t) = Qb(t) + (Qp(t) - Qb(t))*(t/TP*EXP(1 - t/TP))^Beta
    Inflow_Hydrograph_fun = @(t)(Qb + (Qp - Qb).*(t/(Tp*3600).*exp(1 - (t)/(Tp*3600))).^Beta);
    time = [0 tf]'; % begin and end in min
else
    time = [0 tf]'; % begin and end in min
end

if flag_stage_hydrograph == 1
    Stage_Hydrograph = he1;
end

% -------------  Outlet Boundary Condition ------------- %
% flag_outlet = 1; % 1 = normal depth, flag_outlet >< 1, stage hydrograph
% at the outlet following a wave function

if flag_outlet ≠ 1
    %%% Wave Properties for Outlet Stage Hydrograph
    %     x_wave = L_wave/1; % point position in wave x direction;
    k_wave = 2*pi/L_wave;
    sigma_wave = 2*pi./(T_wave*3600);
    h_wave_function = @(t)(h_0_wave + H_0_wave/2.*cos(k_wave.*x_wave - sigma_wave*t));
end

% Time Calculations
time = time*60; % time in seconds
[a1,¬] = size(time); % Length of time
tt_h = time(a1,1); % End of hydrograph in seconds
tt = min(tf*60,tt_h); % End of simulation in seconds
Nt = tt/dt; % Number of time-steps in the simulations

% Recording Times
time_records_min = animation_time; % Minutes
time_store = [0:time_records_min*60:tt]; % number of steps necessary to reach the recording vector
Nat = time_records_min*60/dt; % Number of time-steps within an animation time
tint = linspace(0,tt,Nt); % Generate Nt points within 0 and tt(sec)

time_save = zeros(length(time_store),1); % Time
Flow_Area = zeros(length(time_store),Nx); % Flow area
Discharge = zeros(length(time_store),Nx); % Flow discharge
Depth = zeros(length(time_store),Nx); % Depth
Velocity = zeros(length(time_store),Nx); % Velocity
Froude = zeros(length(time_store),Nx); % Froude
Courant = zeros(length(time_store),Nx); % Courant number

if flag_hydrograph == 1
    Qe1int = max(interp1(time,Qe1(:,1),tint,'pchip'),0); % Interpolated flow
    % Assuming no negative flows
    Qe1int = Qe1int';
elseif flag_nash == 1
    Qe1int = Inflow_Hydrograph_fun(tint)';
else
    tiny_flow = 1e-8;
    Qe1int = tiny_flow*ones(1,length(tint)); % No inflow hydrograph
end

if flag_stage_hydrograph == 1
    he1int = max(interp1(time_stage*60,he1(:,1),tint,'pchip'),0); % Interpolated depth
    he1int = he1int';
end
```

```
%% 3.0 - Pre-Allocation of Arrays

% Channel Discretization
dx = L/(Nx-1); % Channel discretization length in meters

% Friction Data
flag_friction = 1; % If 1, Manning, otherwise DW

% Manning
nm = repmat(nm,Nx,1); % Bottom slope in m/m for all reaches

% Pre-allocating arrays
% Matrices
x = (0:dx:L)'; % x discretization in meters
y = zeros(Nx,2);
q1 = zeros(Nx,2);
q2 = zeros(Nx,2);
f1 = zeros(Nx,2);
f2 = zeros(Nx,2);
J2 = zeros(Nx,2);
q1_back = q1(1:(Nx-2),2);
q1_forward = zeros(Nx-2,2);
q2_back = zeros(Nx-2,2);
q2_forward = zeros(Nx-2,2);
f1_back = zeros(Nx-2,2);
f1_forward = zeros(Nx-2,2);
f2_back = zeros(Nx-2,2);
f2_forward = zeros(Nx-2,2);
J2_back = zeros(Nx-2,2);
J2_forward = zeros(Nx-2,2);
ybar = zeros(Nx,2);
Fr = zeros(Nx,2);
Cn = zeros(Nx,2);

%% 4.0 Channel Data (Cross Section)
% Slope
if flag_slope ≠ 1 && flag_elevation ≠ 1
    I0 = repmat(I0,(Nx-1),1); % Bottom slope in m/m for all reaches. This is only valid for ...
        closed-form sections
elseif flag_slope == 1
    I0 = bottom_slope; % From read input data script
end

if flag_elevation == 1 % We are entering the elevations of each node
    for i = 1:(Nx-1)
        if i+1 > length(inv_el)
            error('Please make sure to add enough invert elevation data.')
        end
        I0(i,1) = (inv_el(i+1) - inv_el(i))/dx;
    end
end

% Outlet Slope
if flag_outlet == 1
    I0(end+1) = s_outlet;
else
    I0(end+1) = s_outlet; % Let's assume a boundary condition
end

% Intializing channel data
sm = 1e-12; % Small number
b = sm + b; Z1 = sm + Z1; Z2 = sm + Z2; D = sm + D; a = sm + a;
% flag_section - If 1, trapezoid, if 2, circular, if 3, paraboloid, if 4 - Irregular

% Invert Elevations
if flag_elevation ≠1
    inv_el = zeros(Nx,1);
    for i = 1:Nx
        if i == 1
            inv_el(i) = el;
        else
            inv_el(i) = inv_el(i-1) - (I0(i-1)*dx);
        end
    end
end

```

```
% -------------  Geometrical Functions for all Cros-Sections ------------ %
syms b_ y_ Z1_ Z2_ Q_ I0_ D_ a_
dim_all = 1e-6*(y_ + Z1_ + Z2_ + a_ + D_ + b_);
if flag_section == 1
    B = b_ + y_*(Z1_ + Z2_) + + dim_all; % user defined function (top width)
    B_function = matlabFunction(B);
    P = b_ + y_*(sqrt(1 + Z1_^2) + sqrt(1 + Z2_^2)) + dim_all; % Perimeter Function % user defined ...
        function
    P_function = matlabFunction(P);
    A = (2*b_ + y_*(Z1_ + Z2_))*y_/2 + dim_all; % Area function % user defined function
    A_function = matlabFunction(A); % Function describing the area in terms of y
    centroid = y_ - int(A,y_)./A + dim_all; % 1st order momentum
    ybar_function = matlabFunction(centroid); % Function describing ybar in terms of y
end
if flag_section == 2
    % Circular Section
    theta = 2*acos(1 - 2.*y_./D_) + dim_all;
    B = D_.*sin(theta/2) ; % top width
    B_function = matlabFunction(B);
    P = theta.*D_/2 ; % perimeter
    P_function = matlabFunction(P);
    A = D_.^2/8.*(theta - sin(theta)) ; % area
    A_function = matlabFunction(A); % Function describing the area in terms of y
    Ybar = y_ - (D_.*(- cos(theta/2)/2 + 2.*sin(theta/2).^3./(3*(theta - sin(theta))))); % Very ...
        much attention here
    ybar_function = matlabFunction(Ybar);
end

if flag_section == 3
    % Parabolic Section
    % Area Function
    A = 4.*(y_.^3/2)./(3*sqrt(a_)) + dim_all; % m2
    A_function = matlabFunction(A); % Function describing the area in terms of y
    % Top Width
    B = 3/2.*A./y_ + dim_all; % m
    B_function = matlabFunction(B);
    % Hydraulic Perimeter
    P = dim_all + sqrt(y_)./sqrt(a_).*(sqrt(1 + 4*a_.*y_) + 1./(2*a_).*(log(2*sqrt(a_).*sqrt(y_) + ...
        sqrt(1 + 4*a_.*y_))));
    P_function = matlabFunction(P);
    Y_bar = y_ - 2/5*y_ + dim_all;
    ybar_function = matlabFunction(Y_bar);
end

if flag_section ≠ 4
    %%%%%%% Hydraulic Radius %%%%%%%
    Rh = A/P; % Hydraulic Radius Function
    Rh_function = matlabFunction(Rh); % Function describing the hydraulic radius in terms of y
end

% Vlookup Function
Vlookup_eq = @(data,col1,val1,col2) data((find(data(:,col1)==val1,1,'first')),col2); %Vlookup ...
    function as Excel
Vlookup_l = @(data,col1,val1,col2) data((find(data(:,col1)<val1,1,'last')),col2); %Vlookup function ...
    as Excel]
Vlookup_g = @(data,col1,val1,col2) data((find(data(:,col1)>val1,1,'first')),col2); %Vlookup ...
    function as Excel
fv = 1 + 1e-4; % Factor to avoid fails in vlookup function

% Minimum Value
min_depth = 0.02; % m
min_area = Vlookup_l(irr_table,1,min_depth*fv,2);

% Initial Guess
if flag_section == 1
    y0_guess = 1;
elseif flag_section == 2
    y0_guess = D/2;
elseif flag_section == 3
    y0_guess = 1;
end

%% 5.0- Initial Values for Simulation
Q0 = Qe1int(1,1); % Flow at inlet section at time 0
if flag_stage_hydrograph == 1
    h0 = he1int(1,1); % Water depth at x = 0 at time = 0
end
```

```
if flag_friction == 1
    if flag_section ≠ 4
        if Q0 == 0
            Q0 = sm; % Numerical Constraint
        end
        y0 = uniformeM(nm,Q0,b,Z1,Z2,a,D,I0,P,A,y0_guess)    ; % normal depth using manning equation
        % Stage_Hydrograph Boundary Condition
        if flag_stage_hydrograph == 1
            y0(1,1) = h0;
        end
        % More Initial Boundary Conditions for Area, Velocity, Perimeter and Rh
        A0 = A_function(D,Z1,Z2,a,b,y0); % Cross section area in m2
        u0 = (Q0./A0)'; % Initial velocity in m/s
        P0 = P_function(D,Z1,Z2,a,b,y0); % Hydraulic perimeter in m
        Rh0 = A0./P0; % Hydraulic radius at time 0
        % Boundary Conditions
        y(:,1) = y0; % all sub-reaches with y0 at the beginning
        q1(:,1) = A0; % all sub-reaches with same area A0 at the beginning
        q2(:,1) = Q0; % Assuming permanent conditions at the beginning
        f1(:,1) = q2(:,1);
        % f2 depends on ybar
    else % Irregular Cross-Section
        % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
        % [   1,     2,      3,       4,     5,             6,           7,      8,       9,    10]

        if max(irr_table(:,10)) == 0 % No outflow and S = 0
            % Here we are modeling a channel with no slope
            % We search Everything Using the Depth instead of the Flow
            col1 = 1; % Searching with the Col of Y
            % Stage_Hydrograph Boundary Condition
            if flag_stage_hydrograph == 1
                y0(1,1) = max(h0,irr_table(2,1));
                s_v = y0; % Searching Variable
            else
                error('Please, add a minimum slope value or enter a stage-hydrograph boundary ...
                    condition.')
            end
            y0 = Vlookup_g(irr_table,col1,s_v*fv,1);
            A0 = Vlookup_g(irr_table,col1,s_v*fv,2);
            P0 = Vlookup_g(irr_table,col1,s_v*fv,3);
            Rh0 = Vlookup_g(irr_table,col1,s_v*fv,4);

        else % Now we are modeling a channel with slope
            if (flag_hydrograph == 1 || flag_nash == 1) && flag_stage_hydrograph ≠ 1
                col1 = 10; % Col with Q
                Q_min_table = irr_table(3,10); % ATTENTION HERE
                s_v = max(Q0,Q_min_table); % Searching Variable
            elseif flag_stage_hydrograph == 1
                col1 = 1; % Col with y or h
                h_min_table = irr_table(3,1);
                s_v = max(h_min_table,h0); % Searching Variable
            elseif flag_outlet == 0
                col1 = 1; % Col with y or h
                h_min_table = irr_table(3,1);
                s_v = max(h_min_table,0); % Searching Variable
                Q_min_table = irr_table(3,10);
                Q0 = Q_min_table;
            end
            Q0 = max(irr_table(2,end),Q0); % Allowing minimum value of Q0 larger than 0
            y0 = Vlookup_l(irr_table,col1,s_v*fv,1);
            A0 = Vlookup_l(irr_table,col1,s_v*fv,2);
            P0 = Vlookup_l(irr_table,col1,s_v*fv,3);
            Rh0 = Vlookup_l(irr_table,col1,s_v*fv,4);
        end
        % Boundary Conditions
        y(:,1) = y0; % all sub-reaches with y0 at the beginning
        q1(:,1) = A0; % all sub-reaches with same area A0 at the beginning
        q2(:,1) = Q0; % Assuming permanent conditions at the beginning
        f1(:,1) = q2(:,1);
    end
    if flag_outlet ≠1 % Bay or Ocean Boundary Condition
        % Stage Hydrograph Boundary Condition
        time_wave = 0; % time in seconds
        y(Nx,1) = h_wave_function(time_wave);
        if flag_section ≠ 4
            q1(Nx,1) = A_function(D,Z1,Z2,a,b,y(Nx,2));
        else
```

```
            % We search Everything Using the Depth instead of the Flow
            col1 = 1; % Searching with the Col of Flow
            q1(Nx,1) = Vlookup_g(irr_table,col1,y(Nx,2)*fv,2);
        end
    end

    % Hydraulic Radius
    if flag_section ≠ 4
        Rh_outlet = Rh_function(D,Z1,Z2,a,b,y(Nx,2));
    else
        for mm = 1:(length(irr_table(1,:))-1)
            interp_base = q1(Nx,1); % Value that will be used for interpolation (area)
            area_smaller = Vlookup_l(irr_table,2,interp_base,2); % Smaller values
            if isempty(area_smaller)
                area_smaller = 0;
            end
            area_larger = Vlookup_g(irr_table,2,interp_base,2); % Larger values
            col1 = 2; % Interpolating from area values
            if interp_base ≤ min_area
                var_outlet(mm,1,1) = irr_table(2,mm); % Smaller values
            else
                var_outlet(mm,1,1) = Vlookup_l(irr_table,col1,interp_base,mm); % Smaller values
            end
            var_outlet(mm,1,2) = Vlookup_g(irr_table,col1,interp_base,mm); % Larger values
            alfa_var_outlet(mm,1) = sqrt((interp_base - area_smaller)/(area_larger - area_smaller));
        end


        % [y_table, A, P, Rh, y_bar, n_med, Beta, v, B, Q]
        % [   1,     2, 3, 4,     5,     6,      7,  8,  9, 10]
        col_var = 4; % Calculating Hydraulic Radius
        % Var* = Var(-) + alfa*(Var(+) - Var(-))
        Rh_outlet = var_outlet(col_var,1,1) + alfa_var_outlet(col_var,1)*(var_outlet(col_var,1,2) - ...
            var_outlet(col_var,1,1)); % Interpolated Hydraulic Radius
        % Var* = Var(-) + alfa*(Var(+) - Var(-))
        col_var = 6;
        nm(end,1) = var_outlet(col_var,1,1) + alfa_var_outlet(col_var,1)*(var_outlet(col_var,1,2) - ...
            var_outlet(col_var,1,1));
    end

        if flag_outlet == 1
            u = (1./nm(Nx)).*Rh_outlet^(2/3)*I0(Nx)^0.5; % Normal depth at the outlet
            flow_dir = 1;
        else
            wse_dif = y(Nx-1,2-1) + inv_el(Nx-1) - y(Nx,2) - inv_el(Nx); % Difference in wse
            out_slope = abs(wse_dif)/dx; % Friction slope at the outlet as a diffusive model
            if wse_dif < 0
                ttt = 1;
            end

            if flag_stage_hydrograph ≠ 1 && flag_nash ≠ 1 && flag_hydrograph ≠ 1
                % Only Outlet Tidal B.C.
                if wse_dif > 0 && y(Nx-1,2-1) ≤ fv*1e-3
                    out_slope = 0;
                end
            end
            u = (1./nm(Nx)).*Rh_outlet^(2/3)*out_slope^0.5; % Normal velocity at the outlet
            if wse_dif > 0
                flow_dir = 1; % Flowing towards the outlet
            else
                flow_dir = -1; % Flowing to inside of the channel
            end
        end
else
    error('HyProSWE not coded for Darcy-Weisbach. Wait for the new version or change the method for ...
        Manning.')
end
q2(Nx,1) = q1(Nx,1)*u*flow_dir; % Area x Velocity

%%% State Space Format %%%
% dq/dt + dF/dx = S, we solve for A(x,t) and Q(x,t)
% q = [A Q]' = [q1 q2]'
% F [Q (Qv + gAybar]' = [q2 (q2^2)/q1 + g.q1.ybar]' = [f1 f2]'
% where ybar is the centroid depth from the top
% S = [0 gA(I0 - If)]'

% ybar = y - int(A(y)) / A(y) from y = 0 to y = y0
```

```
if flag_section ≠ 4
    ybar = ybar_function(D,Z1,Z2,a,b,y0);
else
    % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
    % [   1,     2,      3,       4,     5,             6,           7,      8,       9,
    % ybar = y - ybar*
    %      ybar(:,1) =  Vlookup_leq(irr_table,col1,Q0*fv,1) - Vlookup_leq(irr_table,col1,Q0*fv,5);
    ybar(:,1) = Vlookup_g(irr_table,col1,s_v*fv,5);
end
f2(:,1) = q2(:,1).*abs(q2(:,1))./q1(:,1) + g*q1(:,1).*ybar(:,1);
f2(isnan(f2)) = 0; % Attention Here

% Friction S = [J1 J2]' with J1 = 0 and J2 calculated as follows:
if flag_friction == 1
    J2(:,1) = g*q1(:,1).*(I0(:) - q2(:,1).*abs(q2(:,1)).*nm(:)./(q1(:,1).^2.*Rh0.^(4/3))); % Manning
else
    J2(:,1) = g*q1(:,1).*(I0(:) - f*q2(:,1).*abs(q2(:,1))./((q1(:,1).^2).*8*g.*Rh0));
end

J2(isnan(J2)) = 0; % Attention Here

% Froude Number
if flag_section ≠ 4
    Fr(:,1)=abs(q2(:,1)./q1(:,1))./((g*A_function(D,Z1,Z2,a,b,y0)./B_function(D,Z1,Z2,a,b,y0)).^0.5);% ...
        Froude Number
else
    % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
    % [   1,     2,      3,       4,     5,             6,           7,      8,       9,
    A_f_irr = Vlookup_g(irr_table,col1,s_v*fv,2)*ones(length(q1(:,1)),1);
    B_f_irr = Vlookup_g(irr_table,col1,s_v*fv,9)*ones(length(q1(:,1)),1);
    Fr(:,1)=abs(q2(:,1)./q1(:,1))./((g*A_f_irr./B_f_irr).^0.5);% Froude Number
end
% Courant Number
% Cn = c / (dx / dt), where c = v + sqrt(g.Hm), where Hm = A / B
if flag_section ≠ 4
    Hm = A_function(D,Z1,Z2,a,b,y0)./B_function(D,Z1,Z2,a,b,y0);
    Cn(:,1)=(abs(q2(:,1)./q1(:,1))+(g*Hm).^0.5)/(dx/dt);% Courant Number
else
    Hm = A_f_irr./B_f_irr;
    Cn(:,1) = (abs(q2(:,1)./q1(:,1))+(g*Hm).^0.5)/(dx/dt);
end

% Depth in terms of Area function
% let c be the area in terms of Z1,Z2,b, and y, such that A(y) = c
% we want to solve y for A(y) = c

syms c_
if flag_section ≠ 4
    fun_solve = (A - c_); % with c = area, we solve for y.
    options = optimoptions ('fsolve', 'Display', ...
        'none','FunctionTolerance',1e-2,'MaxFunctionEvaluations',Nx*10);
end
if flag_section == 1
    % We have an analytical solution for this case
    z = solve(fun_solve,y_); % solving for y_ = y and c = A(y)
    h_function = matlabFunction(z); % h(A) = z;
else
    % Non-linear set of equations for circular pipe, we need to use fsolve
end
if flag_section ≠ 4
    fun_solve = matlabFunction(fun_solve); % Transforming into an equation
end
%% 6.0 -  Main Loop %%
n = 1; % initializing counter
x_i = 2:(Nx-1); % vector for interior sections varying from 2 to (Nx - 1)
tic % starts measuring time
% Interpolation Variables
if flag_section == 4
    var_inlet = zeros((length(irr_table(1,:))-1),1,2); var_outlet = var_inlet;
    alfa_var_inlet = zeros((length(irr_table(1,:))-1),1,1); alfa_var_outlet = alfa_var_inlet;
    var_middle = zeros((length(irr_table(1,:))-1),length(x_i),2);
    alfa_var_middle = zeros((length(irr_table(1,:))-1),length(x_i),1);
end

% Initialization of some variables
time_end_min = (Nt)*dt;
time = 0;
```

```
time_previous = 0;
time_step = dt; % sec
t_store_prev = 0;

while time ≤ (time_end_min) % Main loop
    try
    n = n + 1; % Time-step index
    time = time + time_step; % Seconds
    time_save_model(n) = time; % Seconds

    % Model Status
    percentage_timestepsec_maxCourant_maxh = [time/(tt)*100, time_step, max(max(Cn)), max(max(y))]

    % Agregating Inflows to the New Time-step
    if flag_hydrograph == 1 || flag_nash == 1
        z1 = find(tint > time_previous,1,'first'); % begin of the time-step
        z2 = find(tint ≤ time,1,'last'); % end of the time-step
        if isempty(z1)
            z1 = 1;
        end
        if isempty(z2) || z2 < z1
            z2 = z1;
        end
        if time_step ≥ dt
            Q0 = mean(Qe1int(z1:z2));
        else
            Q0 = Qe1int(z1);
        end
    end
    if time > 4.08*10^3
        ttt = 1;
    end
    % Agregating Stages to the New Time-step
    if flag_stage_hydrograph == 1
        z1 = find(tint > time_previous,1,'first'); % begin of the time-step
        z2 = find(tint ≤ time,1,'last'); % end of the time-step
        if isempty(z1)
            z1 = 1;
        end
        if isempty(z2) || z2 < z1
            z2 = z1;
        end
        if time_step ≥ dt
            h0 = mean(he1int(z1:z2));
        else
            h0 = he1int(z1);
        end
    end

    % Stop Program if Complex Number Occurs
    if imag(max(Cn(:,2-1))) > 0 || imag(max(q2(:,2-1)))
        error('Complex number possibly due to changing the regime from free flow to pressurized flow.')
    end
    %%%%% -  Boundary Conditions -  %%%%%
    %% Channel's begin (INLET)
    if flag_stage_hydrograph == 1
        %         h0 = he1int(n,1); % Water depth at x = 0 at time = time
        if flag_section == 4
            if h0 > max(irr_table(:,1))
                error('The maximum water depth is larger than the channel height.')
            end
            q1(1,2) = Vlookup_g(irr_table,1,h0,2); % Smaller values
        else
            q1(1,2) = A_function(D,Z1,Z2,a,b,h0);
        end
    else
        q1(1,2) = q1(2,1); % Area at section 1 is equals area of section 2 from previous time-step
    end
    if flag_hydrograph == 1 || flag_nash == 1
        %         q2(1,2) = Qe1int(n,1); % Flow at section 1 is the inflow hydrograph
        q2(1,2) = Q0; % Flow at section 1 is the inflow hydrograph
    else
        q2(1,2) = q2(2,1); % Flow at section 1 equals flow at section 2 from previous time-step
    end

    if flag_hydrograph == 0 && flag_nash == 0 && flag_stage_hydrograph == 0 && flag_outlet == 0
        q2(1,2) = q2(2,1); % Flow at section 1 equals flow at section 2 from previous time-step
```

```
%          q2(1,1) = q2(2,1);
    end

    % Interpolating All Values from I_rr_table using q1 as basis
    % Explanation: area is given in m2. P, Rh, and other variables are
    % in m. So we have a quadratically similar triangle relationship
    if flag_section == 4
        for mm = 1:(length(irr_table(1,:))-1)
            interp_base = q1(1,2); % Value that will be used for interpolation (area)
            if interp_base ≤ min_area % Col with area = 0
                area_smaller = 0; % Smaller values
            else
                area_smaller = Vlookup_l(irr_table,2,interp_base,2); % Smaller values
            end
            area_larger = Vlookup_g(irr_table,2,interp_base,2); % Larger values
            col1 = 2; % Interpolating from area values
            if interp_base ≤ min_area % Col with area = 0
                var_inlet(mm,1,1) = irr_table(2,mm); % Smaller values
            else
                var_inlet(mm,1,1) = Vlookup_l(irr_table,col1,interp_base,mm); % Smaller values
            end
            var_inlet(mm,1,2) = Vlookup_g(irr_table,col1,interp_base,mm); % Larger values
            alfa_var_inlet(mm,1) = sqrt((interp_base - area_smaller)/(area_larger - area_smaller));
        end
    end

    if flag_section == 1 % Trapezoid or Rectangular
        if Z1 > 0 || Z2 > 0 % Trapezoidal channel
            y(1,2) = max(h_function(D,Z1,Z2,a,b,q1(1,2)')); % water depth in terms of area q1
            % In this previous function, we solve h = y in terms of A = q1 = c
        else
            y(1,2) = q1(1,2)/b; % water depth in terms of area q1 for rectangular channels
        end
    elseif flag_section > 1 % circular or paraboloid or irregular
        y0_guess = y(1,2-1);
        c = q1(1,2)*fv;  % WEIRDO. I HAVE TO CHECK IT OUT ... ISNT IT (2-1)?
        if flag_section ≠ 4
            fun = @(y_)fun_solve(D,Z1,Z2,a,b,c,y_);
            y(1,2) = fsolve(fun,y0_guess,options); % non-linear solver
        else % Irregular section
            % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
            % [   1,     2,      3,       4,     5,             6,           7,     8,       9,
            col1 = 2; % Col with A
            col_var = 1;
            % Var* = Var(-) + alfa*(Var(+) - Var(-))
            y(1,2) = var_inlet(col_var,1,1) + alfa_var_inlet(col_var,1)*(var_inlet(col_var,1,2) - ...
                var_inlet(col_var,1,1));
            %              y(1,2) = Vlookup_leq(irr_table,col1,c,1);
        end
    end
    % ybar
    if flag_section ≠ 4
        ybar(1,2) = ybar_function(D,Z1,Z2,a,b,y(1,2));
        % f1 and f2
        f1(1,2) = q2(1,2);
        f2(1,2) = q2(1,2).*abs(q2(1,2))./q1(1,2) + g*q1(1,2).*ybar(1,2);
        % Hydraulic Radius
        Rh_inlet = Rh_function(D,Z1,Z2,a,b,y(1,2));
        % Friction
        if flag_friction == 1
            J2(1,2) = g*q1(1,2).*(I0(1) - ...
                q2(1,2).*abs(q2(1,2)).*nm(1).^2./(q1(1,2).^2*Rh_inlet.^(4/3))); % Manning
        else
            J2(1,2) = (I0(1) - f*q2(1,2).*abs(q2(1,2))./((q1(1,2).^2)*8*g.*Rh_inlet));
        end
        % Froude
        Fr(1,2)=abs(q2(1,2)./q1(1,2))./((g*A_function(D,Z1,Z2,a,b,y(1,2))./B_function(D,Z1,Z2,a,b,y(1,2))).^0.5);%
            Froude Number
        % Courant
        Hm = A_function(D,Z1,Z2,a,b,y(1,2))./B_function(D,Z1,Z2,a,b,y(1,2));
        Cn(1,2)=(abs(q2(1,2)./q1(1,2))+(g*Hm).^0.5)/(dx/time_step);% Courant Number
        if isinf(Cn(1,2)) || isinan(Cn(1,2))
            Cn(1,2) = 0;
        end
    else
        % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
        % [   1,     2,      3,       4,     5,             6,           7,   8,       9,   10]
```

```
        col1 = 2; % Col with A
        col_var = 5;
        % Var* = Var(-) + alfa*(Var(+) - Var(-))
        ybar(1,2) = var_inlet(col_var,1,1) + alfa_var_inlet(col_var,1)*(var_inlet(col_var,1,2) - ...
            var_inlet(col_var,1,1));
        % f1 and f2
        f1(1,2) = q2(1,2);
        f2(1,2) = q2(1,2).*abs(q2(1,2))./q1(1,2) + g*q1(1,2).*ybar(1,2);
        % Hydraulic Radius
        col_var = 4;
        % Var* = Var(-) + alfa*(Var(+) - Var(-))
        Rh_inlet = var_inlet(col_var,1,1) + alfa_var_inlet(col_var,1)*(var_inlet(col_var,1,2) - ...
            var_inlet(col_var,1,1));
        % Friction
        if flag_friction == 1
            col_var = 6;
            % Var* = Var(-) + alfa*(Var(+) - Var(-))
            nm(1) = var_inlet(col_var,1,1) + alfa_var_inlet(col_var,1)*(var_inlet(col_var,1,2) - ...
                var_inlet(col_var,1,1));
            if isnan(nm(1,1))
                nm = irr_table(2,6)*ones(length(q1(:,1)),1);
            end
            J2(1,2) = g*c.*(I0(1) - q2(1,2).*abs(q2(1,2)).*nm(1).^2./(c.^2*Rh_inlet.^(4/3))); % Manning
        else
            J2(1,2) = (I0(1) - f*q2(1,2).*abs(q2(1,2))./((q1(1,2).^2)*8*g.*Rh_inlet));
        end
        % Froude
        % Var* = Var(-) + alfa*(Var(+) - Var(-))
        A_f_irr = q1(1,2);
        col_var = 9;
        B_f_irr = var_inlet(col_var,1,1) + alfa_var_inlet(col_var,1)*(var_inlet(col_var,1,2) - ...
            var_inlet(col_var,1,1));
        %         B_f_irr = Vlookup_leq(irr_table,col1,c,9);
        Fr(1,2) = abs(q2(1,2)./q1(1,2))./((g*A_f_irr./B_f_irr)^0.5);% Froude Number
        % Courant
        Hm = A_f_irr./B_f_irr;
        Cn(1,2) = (abs(q2(1,2)./q1(1,2))+(g*Hm).^0.5)/(dx/time_step);% Courant Number
    end

    %% Right side of the channel (outlet)
    if flag_outlet == 1 % Normal Depth
        q1(Nx,2) = q1(Nx-1,2-1); % Boundary Condition (same area)
        % Interpolating All Values from I_rr_table using q1 as basis
        % Explanation: area is given in m2. P, Rh, and other variables are
        % in m. So we have a quadratically similar triangle relationship
        if flag_section == 4
            for mm = 1:(length(irr_table(1,:))-1)
                interp_base = q1(Nx,2); % Value that will be used for interpolation (area)
                if interp_base ≤ min_area % Area
                    area_smaller = 0;
                else
                    area_smaller = Vlookup_l(irr_table,2,interp_base,2); % Smaller values
                end
                area_larger = Vlookup_g(irr_table,2,interp_base,2); % Larger values
                col1 = 2; % Interpolating from area values
                if interp_base ≤ min_area % Area
                    var_outlet(mm,1,1) = irr_table(2,mm); % Smaller values
                else
                    var_outlet(mm,1,1) = Vlookup_l(irr_table,col1,interp_base,mm); % Smaller values
                end
                var_outlet(mm,1,2) = Vlookup_g(irr_table,col1,interp_base,mm); % Larger values
                alfa_var_outlet(mm,1) = sqrt((interp_base - area_smaller)/(area_larger - ...
                    area_smaller));
            end
        end
        if flag_section == 1
            if Z1 > 0 || Z2 > 0
                y(Nx,2) = max(h_function(D,Z1,Z2,a,b,q1(Nx,2)')); % water depth in terms of area q1
            else
                y(Nx,2) = q1(Nx,2)/b; % water depth in terms of area q1 for rectangular channels
            end
        elseif flag_section ≥ 2 % circular or paraboloid or irregular
            % If we do not have a stage-hydrograph boundary condition
            y0_guess = y(Nx,2-1);
            if flag_section ≠ 4
                fun = @(y_)fun_solve(D,Z1,Z2,a,b,c,y_);
                y(Nx,2) = fsolve(fun,y0_guess,options); % non-linear solver
```

```
            else
                % [y_table, A, P, Rh, y_bar, n_med, Beta, v, B, Q]
                % [    1,     2, 3, 4,     5,     6,       7,   8,   9, 10]
                col_var = 1;
                % Var* = Var(-) + alfa*(Var(+) - Var(-))
                y(Nx,2) = var_outlet(col_var,1,1) + ...
                    alfa_var_outlet(col_var,1)*(var_outlet(col_var,1,2) - var_outlet(col_var,1,1));
            end
        end
    else
        % Stage Hydrograph Boundary Condition. We are modeling a tidal
        % outlet condition
        time_wave = time; % time in seconds
        y(Nx,2) = h_wave_function(time_wave);
        if flag_section ≠ 4
            q1(Nx,2) = A_function(D,Z1,Z2,a,b,y(Nx,2));
        else
            % We search Everything Using the Depth instead of the Flow
            col1 = 1; % Searching with the Col of Flow


            area_smaller = Vlookup_l(irr_table,col1,y(Nx,2)*fv,2);
            area_greater = Vlookup_g(irr_table,col1,y(Nx,2)*fv,2);
            y_smaller = Vlookup_l(irr_table,col1,y(Nx,2)*fv,1);
            y_greater = Vlookup_g(irr_table,col1,y(Nx,2)*fv,1);

            Δ_y = y(Nx,2) - Vlookup_l(irr_table,col1,y(Nx,2)*fv,1);
            q1(Nx,2) = area_smaller + (area_greater - area_smaller)*(Δ_y/(y_greater - y_smaller))^2;
        end
        %         q1(Nx,2) = q1(Nx-1,2-1)
    end
    % Hydraulic Radius
    if flag_section ≠ 4
        Rh_outlet = Rh_function(D,Z1,Z2,a,b,y(Nx,2));
    else
        for mm = 1:(length(irr_table(1,:))-1)
            interp_base = q1(Nx,2); % Value that will be used for interpolation (area)
            if interp_base ≤ min_area
                area_smaller = 0; % Smaller values
            else
                area_smaller = Vlookup_l(irr_table,2,interp_base,2); % Smaller values
            end
            area_larger = Vlookup_g(irr_table,2,interp_base,2); % Larger values
            col1 = 2; % Interpolating from area values
            if interp_base ≤ min_area
                var_outlet(mm,1,1) = irr_table(2,mm);
            else
                var_outlet(mm,1,1) = Vlookup_l(irr_table,col1,interp_base,mm); % Smaller values
            end
            var_outlet(mm,1,2) = Vlookup_g(irr_table,col1,interp_base,mm); % Larger values
            alfa_var_outlet(mm,1) = sqrt((interp_base - area_smaller)/(area_larger - area_smaller));
        end
        % [y_table, A, P, Rh, y_bar, n_med, Beta, v, B, Q]
        % [    1,     2, 3, 4,     5,     6,       7,   8,   9, 10]
        col_var = 4; % Calculating Hydraulic Radius
        % Var* = Var(-) + alfa*(Var(+) - Var(-))
        Rh_outlet = var_outlet(col_var,1,1) + alfa_var_outlet(col_var,1)*(var_outlet(col_var,1,2) - ...
            var_outlet(col_var,1,1)); % Interpolated Hydraulic Radius
        % Var* = Var(-) + alfa*(Var(+) - Var(-))
        col_var = 6;
        nm(end,1) = var_outlet(col_var,1,1) + alfa_var_outlet(col_var,1)*(var_outlet(col_var,1,2) - ...
            var_outlet(col_var,1,1));
    end
    if flag_friction == 1
        if flag_outlet == 1
            u = (1./nm(Nx)).*Rh_outlet^(2/3)*I0(Nx)^0.5; % Normal depth at the outlet
            flow_dir = 1;
        else
            wse_dif = y(Nx-1,2-1) + inv_el(Nx-1) - y(Nx,2) - inv_el(Nx); % Difference in wse
            out_slope = abs(wse_dif)/dx; % Friction slope at the outlet as a diffusive model
            if wse_dif < 0
                ttt = 1;
            end

            u = (1./nm(Nx)).*Rh_outlet^(2/3)*out_slope^0.5; % Normal velocity at the outlet
            if wse_dif > 0
                flow_dir = 1; % Flowing towards the outlet
```

```
            else
                flow_dir = -1; % Flowing to inside of the channel
            end
        end
    else
        u = sqrt(8*g*Rh_outlet*I0(Nx)/f); % outlet velocity
    end
    % Outlet Flow
    q2(Nx,2) = q1(Nx,2)*u*flow_dir; % Area x Velocity
    if isnan(q2(Nx,2))
        ttt = 1;
    end
    % Outlet Flow Under No Inflow Hydrograph & Not Enough WSE_dif
    if flag_stage_hydrograph ≠ 1 && flag_nash ≠ 1 && flag_hydrograph ≠ 1
        % Only Outlet Tidal B.C.
        if wse_dif > 0 && y(Nx-1,2-1) ≤ fv*1e-3
            q2(Nx,2) = q1(Nx,2)*dx/(time_step); % Making sure all available depth becomes outflow ...
                in the outlet
        end
    end

    % ybar
    if flag_section ≠ 4
        ybar(Nx,2) = ybar_function(D,Z1,Z2,a,b,y(Nx,2));
    else
        % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
        % [   1,    2,     3,      4,    5,            6,           7,  8,      9,  10]
        % ybar = y - ybar*
        col1 = 2; % A
        %         ybar(Nx,2) =  Vlookup_leq(irr_table,col1,c,1) - Vlookup_leq(irr_table,col1,c,5);
%         if q1(Nx,2) == 0
%             ybar(Nx,2) = 0;
%         else
%             ybar(Nx,2) = Vlookup_l(irr_table,col1,q1(Nx,2),5);
%         end
        col_var = 5;
        % Var* = Var(-) + alfa*(Var(+) - Var(-))
        ybar(Nx,2) = var_outlet(col_var,1,1) + alfa_var_outlet(col_var,1)*(var_outlet(col_var,1,2) ...
            - var_outlet(col_var,1,1));
    end
    % f1 and f2
    f1(Nx,2) = q2(Nx,2); % f1 - Flow
    zzz = q2(Nx,2).*abs(q2(Nx,2))./q1(Nx,2) + g*q1(Nx,2).*ybar(Nx,2); % f2 = (Qv + gAy_bar)
    zzz(isnan(zzz)) = 0;
    f2(Nx,2) = zzz; % f2 = (Qv + gAy_bar)

    % J2
    % Friction
    if flag_friction == 1
        J2(Nx,2) = g*q1(Nx,2).*(I0(Nx) - ...
            q2(Nx,2).*abs(q2(Nx,2)).*nm(Nx)^2./(q1(Nx,2).^2*Rh_outlet.^(4/3))); % Manning --> ...
            gA*(I0 - If), If = n^2*Q*abs*Q)/(Rh^(4/3)*A^2)
    else
        J2(Nx,2) = g*q1(Nx,2).*(I0(Nx) - f*q2(:,2).*abs(q2(Nx,2))./((q1(Nx,2).^2)*8*g*Rh_outlet));
    end
    J2(isnan(J2)) = 0; % Attention Here
    % Froude
    if flag_section ≠ 4
        Fr(Nx,2)=abs(q2(Nx,2)./q1(Nx,2))./((g*A_function(D,Z1,Z2,a,b,y(Nx,2))./B_function(D,Z1,Z2,a,b,y(Nx,2)))^0.
            Froude Number
        % Courant
        Hm = A_function(D,Z1,Z2,a,b,y(Nx,2))./B_function(D,Z1,Z2,a,b,y(Nx,2));
        Cn(Nx,2)=(abs(q2(Nx,2)./q1(Nx,2))+(g*Hm).^0.5)/(dx/time_step);% Courant Number
        if isnan(Cn(Nx,2)) || isinf(Cn(Nx,2))
            Cn(Nx,2) = 0;
        end
    else
        % Froude
        % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
        % [   1,    2,     3,      4,    5,            6,           7,  8,      9,  10]
        col1 = 2; % Col with A
        A_f_irr = c;
        col_var = 9;
        % Var* = Var(-) + alfa*(Var(+) - Var(-))
        B_f_irr = var_outlet(col_var,1,1) + alfa_var_outlet(col_var,1)*(var_outlet(col_var,1,2) - ...
            var_outlet(col_var,1,1));
        %         B_f_irr = Vlookup_l(irr_table,col1,c,9);
```

```
        Fr(Nx,2) = abs(q2(Nx,2)./q1(Nx,2))./((g*A_f_irr./B_f_irr)^0.5);% Froude Number
        % Courant
        Hm = A_f_irr./B_f_irr;
        if y(Nx,2) ≤ min_depth
            Cn(Nx,2) = 0;
        else
            Cn(Nx,2)=(abs(q2(Nx,2)./q1(Nx,2))+(g*Hm).^0.5)/(dx/time_step);% Courant Number
            if isnan(Cn(Nx,2)) || isinf(Cn(Nx,2))
                Cn(Nx,2) = 0;
            end
        end
    end

    %% Main Loop for Non-Boundary Cells from 2 to (Nx - 1)
    % vectorized calculations
    q1_back = q1(1:(Nx-2),(2-1));
    q1_forward = q1(3:(Nx),(2-1));
    q2_back = q2(1:(Nx-2),(2-1));
    q2_forward = q2(3:(Nx),(2-1));
    f1_back = f1(1:(Nx-2),(2-1));
    f1_forward = f1(3:(Nx),(2-1));
    f2_back = f2(1:(Nx-2),(2-1));
    f2_forward = f2(3:(Nx),(2-1));
    J2_back = J2(1:(Nx-2),(2-1));
    J2_forward = J2(3:(Nx),(2-1));

    % Lax-Friedrichs Method
    % Given a hyperbolic partial derivative system of equations described
    % by:
    % pq/pt + pF/px - S = 0, where p is the partial derivative, one can
    % solve this equation by performing a forward discretization for q and a
    % central discretization for F. Moreover, S = (Sback + Sforward)/2
    % Expliciting the system of equations for q, it follows that:

    q1(x_i,2) = 0.5.*(q1_forward + q1_back) - 0.5*time_step/dx*(f1_forward - f1_back); %% attention ...
        here in f1forward
    q2(x_i,2) = 0.5*(q2_forward + q2_back) - 0.5*time_step/dx*(f2_forward - f2_back) + ...
        0.5*time_step*(J2_back + J2_forward);

    if q1(Nx-1,2) > 0.0
        ttt = 1;
    end
    % There is no such thing as a negative water depth, so we apply a
    % constraint
%     if min(q1(x_i,2)) < 0
%         zzz = q1(x_i,2); zzz(zzz<0) = 0; q1(x_i,2) = zzz;
%         ttt = 1;
%     end
    % Interpolating All Values from I_rr_table using q1 as basis
    if flag_section == 4
        for mm = 1:(length(irr_table(1,:))-1)
            for hh = 1:length(x_i)
                interp_base = q1(hh+1,2); % Value that will be used for interpolation (area)
                if interp_base ≤ min_area
                    area_smaller = 0;
                else
                    area_smaller = Vlookup_l(irr_table,2,interp_base,2); % Smaller values
                end
                area_larger = Vlookup_g(irr_table,2,interp_base,2); % Larger values
                col1 = 2; % Interpolating from area values
                if interp_base ≤ min_area
                    var_middle(mm,hh,1) = irr_table(2,mm); % Smaller values
                else
                    var_middle(mm,hh,1) = Vlookup_l(irr_table,col1,interp_base,mm); % Smaller values
                end
                var_middle(mm,hh,2) = Vlookup_g(irr_table,col1,interp_base,mm); % Larger values
                alfa_var_middle(mm,hh,1) = sqrt((interp_base - area_smaller)/(area_larger - ...
                    area_smaller));
            end
        end
    end

    if flag_section == 1
        if Z1>0 || Z2>0
            y(x_i,2) = max(h_function(D,Z1,Z2,a,b,q1(x_i,2)')); % water depth in terms of area q1
        else
            y(x_i,2)=q1(x_i,2)/b;
```

```
        end
    elseif flag_section > 1
        y0_guess = y(x_i,2-1);
        c = q1(x_i,2)*fv; % It has to be a line vector (area)
        if flag_section ≠ 4
            fun = @(y_)fun_solve(D,Z1,Z2,a,b,c,y_);
            y(x_i,2) = fsolve(fun,y0_guess,options); % non-linear solver
        else
            % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
            % [   1,     2,      3,      4,     5,             6,          7,   8,      9,   10]
            col1 = 2; % Col with A
            for i = 1:length(x_i)
                cc = c(i); % be careful here
                col_var = 1;
                % Var* = Var(-) + alfa*(Var(+) - Var(-))
                y(i+1,2) = var_middle(col_var,i,1) + ...
                    alfa_var_middle(col_var,i)*(var_middle(col_var,i,2) - var_middle(col_var,i,1));
            end
        end
    end
    % Hydraulic Radius
    if flag_section ≠ 4
        Rh_middle = Rh_function(D,Z1,Z2,a,b,y(x_i,2));
        % ybar
        ybar(x_i,2) = ybar_function(D,Z1,Z2,a,b,y(x_i,2));
        % f1 and f2
        f1(x_i,2) = q2(x_i,2);
        f2(x_i,2) = q2(x_i,2).*abs(q2(x_i,2))./q1(x_i,2) + g*q1(x_i,2).*ybar(x_i,2);
        % Froude
        Hm = A_function(D,Z1,Z2,a,b,y(x_i,2))./B_function(D,Z1,Z2,a,b,y(x_i,2));
        Fr(x_i,2)=abs(q2(x_i,2)./q1(x_i,2))./((g*Hm).^0.5);% Froude Number
        % Courant
        Cn(x_i,2)=(abs(q2(x_i,2)./q1(x_i,2))+(g*Hm).^0.5)/(dx/time_step);% Courant Number
        % Friction
        if flag_friction == 1
            J2(x_i,2) = g*q1(x_i,2).*(I0(x_i) - ...
                q2(x_i,2).*abs(q2(x_i,2).*nm(x_i).^2./(q1(x_i,2).^2.*Rh_middle.^(4/3))));
        else
            J2(x_i,2) = g*q1(x_i,2).*(I0(x_i) - ...
                f*q2(x_i,2).*abs(q2(x_i,2))./((q1(x_i,2).^2)*8*g*Rh_midle));
        end
        % Stability Check
        if max(Cn(:,2)) > 1
            error('Please, decrease the time-step')
        end
    else
        for jj = 1:length(x_i)
            cc = c(jj); % Area
            % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
            % [   1,     2,      3,      4,     5,             6,          7,   8,      9,   10]
            col_var = 4;
            % Var* = Var(-) + alfa*(Var(+) - Var(-))
            Rh_middle(jj,1) = var_middle(col_var,jj,1) + ...
                alfa_var_middle(col_var,jj)*(var_middle(col_var,jj,2) - var_middle(col_var,jj,1));
            col_var = 5;
            ybar(jj+1,2) = var_middle(col_var,jj,1) + ...
                alfa_var_middle(col_var,jj)*(var_middle(col_var,jj,2) - var_middle(col_var,jj,1));
            col_var = 6;
            nm(jj+1,1) = var_middle(col_var,jj,1) + ...
                alfa_var_middle(col_var,jj)*(var_middle(col_var,jj,2) - var_middle(col_var,jj,1));
            % f1 and f2
            f1(jj+1,2) = q2(jj+1,2);
            f2(jj+1,2) = q2(jj+1,2).*abs(q2(jj+1,2))./q1(jj+1,2) + g*q1(jj+1,2).*ybar(jj+1,2);
            % Froude
            A_f_irr = q1(jj+1,2);
            col_var = 9;
            B_f_irr = var_middle(col_var,jj,1) + ...
                alfa_var_middle(col_var,jj)*(var_middle(col_var,jj,2) - var_middle(col_var,jj,1));
            Hm = A_f_irr./B_f_irr;
            Fr(jj+1,2) = abs(q2(jj+1,2)./q1(jj+1,2))./((g*Hm).^0.5);% Froude Number
            % Courant
            if y(jj+1,2) > 0.005 % 0.5 cm
                Cn(jj+1,2) = (abs(q2(jj+1,2)./q1(jj+1,2))+(g*Hm).^0.5)/(dx/time_step);% Courant Number
            else
                Cn(jj+1,2) = 0;
            end
            if isinf(Cn(jj+1,2))
```

```
                Cn(jj+1,2) = 0;
            end
            % Friction
            if flag_friction == 1
                J2(jj+1,2) = g*A_f_irr.*(I0(jj+1,1) - ...
                    q2(jj+1,2).*abs(q2(jj+1,2).*nm(jj+1,1).^2./(A_f_irr.^2.*Rh_middle(jj,1)^(4/3))));
            else
                J2(jj+1,2) = g*q1(jj+1,2).*(I0(jj+1,2) - ...
                    f*q2(jj+1,2).*abs(q2(jj+1,2))./((q1(jj+1,2).^2)*8*g*Rh_midle(jj,1)));
            end
            % Stability Check
            if Cn(jj+1,2) > 1 && y(jj+1,2) ≥ min_depth && q1(jj+1,2) ≥ min_area
                error('Please, decrease the time-step')
            end
        end
    end


    % Constraint at dry areas
    % -- the idea is that dry cells have no hydraulic properties
    if min(q1(:,2)) ≤  min_area || min(y(:,2)) ≤  min_depth
        idx1 = q1(:,2) ≤ min_area; idx2 = y(:,2) ≤ min_area; idx = idx1 + idx2; % Both
        idx = logical([zeros(size(idx,1),1), idx]);
        q1(idx) = 0; q2(idx) = 0; f1(idx) = 0; f2(idx) = 0; J2(idx) = 0; y(idx) = 0;
        Fr(idx) = 0; Cn(idx) = 0;
    end
        % Adaptive Time-Step - Outlet not considered
        idx_courant = Cn(1:end-1,2) ≤ 0;
        zzz = Cn(1:end-1,2); zzz(idx_courant) = nan;
        dt_courant_1 = zzz/time_step; % Cn/time_step = dx/ (v + sqrt(Hm*g)), this the the time-step ...
            for Courant = 1
        time_step = min(alpha./dt_courant_1); % Calculated
        time_step = min(time_step,dtmax);
        time_step = max(time_step,dtmin);
        time_previous = time;

        if time_step < 1
            ttt = 1;
        end

    % Saving hydrographs and depths with user defined recording time-step
    if n == 1
        % Do nothing, it is already solved, we just have to save the data
        % for the next time-step
        t_store = 1;
        time_save(1,1) = time;
    else
        t_store = find(time_store ≤ time,1,'last'); % Time that is being recorded in min
        if t_store > t_store_prev
            time_save(t_store,1) = time;
            Flow_Area(t_store,:) = q1(:,2); % m2
            Discharge(t_store,:) = q2(:,2); % m3/s
            Depth(t_store,:) = y(:,2); % m
            Velocity(t_store,:) = q2(:,2)./q1(:,1); % m/s
            Froude(t_store,:) = Fr(:,2);
            Courant(t_store,:) = Cn(:,2);
            t_store_prev = t_store;
        end
    end
    % Refreshing States
%     idx = y < 1e-3; q1(idx) = 0; q2(idx)

    q1(:,1) = q1(:,2);
    q2(:,1) = q2(:,2);
    f1(:,1) = f1(:,2);
    f2(:,1) = f2(:,2);
    J2(:,1) = J2(:,2);
    y(:,1) = y(:,2);
    Cn(:,1) = Cn(:,2);
    Fr(:,1) = Fr(:,2);

    if time > 2*1000
        ttt = 1;
    end

    catch ME
        % If this condition is reached, we are reducing the time-step to
```

```
        % 50% and doing the calculations again
        idx = q1(:,2) ≤ min_area;
        vel = abs(q2(:,2)./q1(:,2)) + sqrt(g*y(:,2)); vel(idx) = 0;
        dtnew = min(alpha*dx./(vel));
        time = time - time_step; % Seconds
        time_step = dtnew; % Halving the time-step
        n = n - 1;
    end
end
%% 7.0 - Post-Processing
water_depths = Depth;
%%% Post Processing Figures %%%
% Call function
warning('on');
post_processing
close all
toc
disp(['Thank you for using HyProSWE. If you have any questions, please contact me at ...
    marcusnobrega.engcivil@gmail.com).'])
disp(['Also, please check your current matlab folder. The outputs are there.'])
```

4) SVE Post Processing

```
%%% --------- HyProSWE Model ----------- %%%
% Post-Processing Routine
% Developer: Marcus Nobrega Gomes Junior
% 5/1/2023
% Goal: Solution of 1-D SVE for given cross-section functions of Area, Perimeter, and
% top Width
% If you have any issues, please contact me at
% marcusnobrega.engcivil@gmail.com

%% Creating Modeling Results Folder
% Create the folder name
folderName = 'Modeling_Results';

% Check if the folder already exists
if ¬exist(folderName, 'dir')
    % If it doesn't exist, create the folder
    mkdir(folderName);
    disp('Folder "Modeling_Results" created successfully!');
else
    disp('Data sucessfully exported in Modeling_Results Folder');
end

%% Post Processing Graphs
clf
close all


color_plot = [21, 179, 196]/255; % You can change it if you want

% Surfplot
t_save = [0:Nat:tt/dt];
t_save(1,1) = 1;
set(gcf,'units','inches','position',[2,0,8,10])
subplot(3,1,1)
surf(x,tint(t_save)/3600,Froude);
view(0,90);
kk = colorbar ; colormap('jet')
shading interp
xlabel('x (m)','Interpreter','latex')
ylabel('t (h)','Interpreter','latex')
ylabel(kk,'Froude Number','Interpreter','latex')
zlabel ('Froude Number','Interpreter','Latex');
xlim([0 L]);
ylim([0 tt/60/60]);
set(gca,'FontName','Garamond','FontSize',12,'FontWeight','Bold','LineWidth', 1.5);
set(gca,'TickLength',[0.02 0.01])
set(gca,'TickDir','out')

subplot(3,1,2)
surf(x,tint(t_save)/60/60,Depth);
```

```
view(0,90);
kk = colorbar ; colormap('jet')
shading interp
xlabel('x (m)','Interpreter','latex')
ylabel('t (h)','Interpreter','latex')
ylabel(kk,'y (m)','Interpreter','latex')
zlabel ('y (m)','Interpreter','Latex');
xlim([0 L]);
ylim([0 tt/60/60]);
set(gca,'FontName','Garamond','FontSize',12,'FontWeight','Bold','LineWidth', 1.5);
set(gca,'TickLength',[0.02 0.01])
set(gca,'TickDir','out')

subplot(3,1,3)
wse = Depth + repmat(inv_el',[size(Depth,1),1]);
surf(x,tint(t_save)/60/60,wse);
view(0,90);
kk = colorbar ; colormap('jet')
shading interp
xlabel('x (m)','Interpreter','latex')
ylabel('t (h)','Interpreter','latex')
ylabel(kk,'WSE (m)','Interpreter','latex')
zlabel ('WSE (m)','Interpreter','Latex');
xlim([0 L]);
ylim([0 tt/60/60]);
set(gca,'FontName','Garamond','FontSize',12,'FontWeight','Bold','LineWidth', 1.5);
set(gca,'TickLength',[0.02 0.01])
set(gca,'TickDir','out')
exportgraphics(gcf,fullfile(folderName,'Surf_Plots.pdf'),'ContentType','image','Colorspace','rgb','Resolution',600
clf
close all

if flag_section == 2 % circular
% Video
obj = VideoWriter('Circular_Depth.avi','Motion JPEG AVI');
obj.Quality = 100;
obj.FrameRate = 20;
open(obj)
set(gcf,'units','inches','position',[2,2,10,3])
    for n=1:1:(Nt/Nat)
        if n == 1
            t = 1;
            pos = 1;
        else
            t=time_save(n);
            pos = n;
        end
        % Circle Function
        xcir = linspace(0,2*pi,100); % 100 points within 0 and 360 deg
        cir = @(r,ctr) [r*cos(xcir)+ctr(1); r*sin(xcir)+ctr(2)];
        c1 = cir(D/2, [D/2; D/2]);

        % Boundary Circle
        % (x - xc)^2 + (y - yc)^2 = D^2/4
        % where xc = D/2 and yc = D/2
        xc = D/2; yc = D/2;
        y01 = Depth(pos,1);
        y02 = Depth(pos,ceil(ceil(Nx/2)));
        y03 = Depth(pos,Nx);
        y0_c = [y01; y02; y03];
        % For a given known y, we have to find two xs, such that
        % x^2 + (-2xc)x + ( (y0 - yc)^2 - xc^2 - D^2/4 )
        % or ax^2 + bx + c, with
        % a = 1; b = -2xc; c = (y0 - yc)^2 - xc^2 - D^2/4
        % x = (- b +- sqrt(b^2 - 4ac)) / (2a)
        a = 1;
        b = -2*xc;
        c = xc^2 + (y0_c - yc).^2 - D^2/4;
        Delta = b^2 - 4*a.*c;
        x1 = (-b + sqrt(Delta))/(2*a);
        x2 = (-b - sqrt(Delta))/(2*a);
        % Now we found the intersection of the circle and a line with know
        % depth
        subplot(1,3,1)
        title(['t = ',num2str(round(round(t,2),0)),' [sec]'])
        ylim([0 D]);
        xlim([0 D]);
```

```
        viscircles([D/2 D/2],D/2,'Color','black');
%         plot(c1(1,:),c1(2,:),'Color','black');
        hold on
        x_water = linspace(x2(1),x1(1),100);
        y_water = repmat(y01,1,100);
        plot(x_water,y_water,'Color',color_plot,'linewidth',2);
%         fill([c1(1,:) fliplr(c1(1,:))], [y_water fliplr(c2(1,:))],color_plot)
        ylabel('y(m)','Interpreter','latex')
        xlabel('B(m)','Interpreter','latex')
        legend('Entrance','interpreter','latex')
        hold off
        grid on
        set(gca,'FontName','Garamond','FontSize',12);
        set(gca,'TickLength',[0.02 0.01])
        set(gca,'TickDir','out');
        box on
        % second section
        subplot(1,3,2)
        title(['t = ',num2str(round(round(t,2),0)),' [sec]'])
        ylim([0 D]);
        xlim([0 D]);
        viscircles([D/2 D/2],D/2,'Color','black');
        hold on
        x_water = linspace(x2(2),x1(2),100);
        y_water = repmat(y02,1,100);
        plot(x_water,y_water,'Color',color_plot,'linewidth',2);
        ylabel('y(m)','Interpreter','latex')
        xlabel('B(m)','Interpreter','latex')
        legend('x = L/2','interpreter','latex')
        hold off
        legend('L/2','interpreter','latex')
        % third section
        grid on
        set(gca,'FontName','Garamond','FontSize',12);
        set(gca,'TickLength',[0.02 0.01])
        set(gca,'TickDir','out');
        box on
        subplot(1,3,3)
        title(['t = ',num2str(round(round(t/60),0)),' [sec]'])
        ylim([0 D]);
        xlim([0 D]);
        viscircles([D/2 D/2],D/2,'Color','black');
        hold on
        x_water = linspace(x2(3),x1(3),100);
        y_water = repmat(y03,1,100);
        plot(x_water,y_water,'color',color_plot,'linewidth',2);
        hold off
        ylabel('y(m)','Interpreter','latex')
        xlabel('B(m)','Interpreter','latex')
        legend('Exit','interpreter','latex')
        grid on
        set(gca,'FontName','Garamond','FontSize',12);
        set(gca,'TickLength',[0.02 0.01])
        set(gca,'TickDir','out');
        box on
        % Save frame
        title(['t = ',num2str(round(round(t,2),0)),' [sec]'])
        f = getframe(gcf);
        writeVideo(obj,f);
        hold off
        clf
    end
obj.close();
end

if flag_section == 3 % paraboloid
% Video
obj = VideoWriter('Parabolic_Depth.avi','Motion JPEG AVI');
obj.Quality = 100;
obj.FrameRate = 20;
open(obj)
set(gcf,'units','inches','position',[2,2,10,3])
    for n=1:1:(Nt/Nat)
        if n == 1
            t = 1;
            pos = 1;
        else
```

```
            t=time_save(n);
            pos = n;
        end
        % Save frame
        Plot_Title = 'Time = %d (sec)';
        sgtitle(sprintf(Plot_Title, time_store(n)),'fontsize',18,'interpreter','latex')
        % Parabolic Function
        % y = a*x^2 => xmax = sqrt((ymax/a))
        ymax = max(max(Depth));
        xmax = sqrt(ymax/a); % x to left and right directions
        xpar = linspace(-xmax,xmax,100); % 100 points within -xmax and xmax deg
        ypar = a.*xpar.^2;
        % Now we found bottom of the channel
        % We still need to find xleft and xright for a given y
        y01 = Depth(pos,1);
        y02 = Depth(pos,ceil(Nx/2));
        y03 = Depth(pos,Nx);
        y0_c = [y01; y02; y03];
        xright = sqrt(y0_c/a);
        xleft = - xright;
        subplot(1,3,1)
        title(['t = ',num2str(round(round(t/60,2),0)),' [min]'])
        ylim([0 ymax]);
        xlim([0 ymax]);
        plot(xpar,ypar,'Color','black','LineWidth',2);
        hold on
        x_water = linspace(xleft(1),xright(1),100);
        y_water = linspace(y01,y01,100);
        plot(x_water,y_water,'color',color_plot,'linewidth',2);
        ylabel('y(m)','Interpreter','latex')
        xlabel('B(m)','Interpreter','latex')
        legend('Entrance','interpreter','latex')
        grid on
        set(gca,'FontName','Garamond','FontSize',12);
        set(gca,'TickLength',[0.02 0.01])
        set(gca,'TickDir','out');
        hold off
        % second section
        subplot(1,3,2)
        title(['t = ',num2str(round(round(t/60,2),0)),' [min]'])
        ylim([0 ymax]);
        xlim([0 ymax]);
        plot(xpar,ypar,'Color','black','LineWidth',2);
        hold on
        x_water = linspace(xleft(2),xright(2),100);
        y_water = linspace(y02,y02,100);
        plot(x_water,y_water,'color',color_plot,'linewidth',2);
        ylabel('y(m)','Interpreter','latex')
        xlabel('B(m)','Interpreter','latex')
        legend('x = L/2','interpreter','latex')
        grid on
        set(gca,'FontName','Garamond','FontSize',12);
        set(gca,'TickLength',[0.02 0.01])
        set(gca,'TickDir','out');
        hold off
        % third section
        subplot(1,3,3)
        title(['t = ',num2str(round(round(t/60,2),0)),' [min]'])
        ylim([0 ymax]);
        xlim([0 ymax]);
        plot(xpar,ypar,'Color','black','LineWidth',2);
        hold on
        x_water = linspace(xleft(3),xright(3),100);
        y_water = linspace(y03,y03,100);
        plot(x_water,y_water,'Color',color_plot,'linewidth',2);
        ylabel('y(m)','Interpreter','latex')
        xlabel('B(m)','Interpreter','latex')
        legend('Outlet','interpreter','latex')
        grid on
        set(gca,'FontName','Garamond','FontSize',12);
        set(gca,'TickLength',[0.02 0.01])
        set(gca,'TickDir','out');
        f = getframe(gcf);
        writeVideo(obj,f);
        hold off
        clf
    end
```

```
obj.close();
end


%% Plots
% Time Scale
if flag_elapsed_time == 1
    close all
    flag_date = 3; % 1 min, 2 hour, 3 day, 4 month
    date_string = {'Elased time (min)','Elapsed time (h)','Elapsed time (days)','Elapsed time ...
        (months)'};
    if flag_date == 1
        time_scale = 1;
    elseif flag_date == 2
        time_scale = 1/60;
    elseif flag_date == 3
        time_scale = 1/60/24;
    else
        time_scale = 1/60/24/30;
    end
    set(gcf,'units','inches','position',[2,0,8,10])
    subplot(3,2,1)
    % Flows
    plot(time_save/60,Discharge(:,1),'LineStyle','--','LineWidth',2,'Color','k')
    hold on
    plot(time_save/60,Discharge(:,ceil(Nx/2)),'LineStyle',':','LineWidth',2,'Color','k')
    hold on
    plot(time_save/60,Discharge(:,Nx),'LineStyle','-','LineWidth',2,'Color','k')
    hold on
    xlabel(date_string(flag_date),'interpreter','latex');
    ylabel('Flow Discharge (m\textsuperscript{3}/s)','Interpreter','latex');
    legend('Entrance','L/2','Outlet','Interpreter','Latex','location','best')
    % Velocity
    subplot(3,2,2)
    plot(time_save/60,Velocity(:,1),'LineStyle','--','LineWidth',2,'Color','k')
    hold on
    plot(time_save/60,Velocity(:,ceil(Nx/2)),'LineStyle',':','LineWidth',2,'Color','k')
    hold on
    plot(time_save/60,Velocity(:,Nx),'LineStyle','-','LineWidth',2,'Color','k')
    %%% Normal Depth Velocity %%%
    xlabel(date_string(flag_date),'interpreter','latex');
    ylabel('Velocity (m/s)','Interpreter','latex');
    legend('Entrance','L/2','Outlet','Interpreter','Latex','Location','best')
    % Water Depth
    subplot(3,2,3)
    plot(time_save/60,Depth(:,1),'LineStyle','--','LineWidth',2,'Color','k')
    hold on
    plot(time_save/60,Depth(:,ceil(Nx/2)),'LineStyle',':','LineWidth',2,'Color','k')
    hold on
    plot(time_save/60,Depth(:,Nx),'LineStyle','-','LineWidth',2,'Color','k')
    xlabel(date_string(flag_date),'interpreter','latex');
    ylabel('Water Depths (m)','Interpreter','latex');
    legend('Entrance','L/2','Outlet','Interpreter','Latex','Location','best')
    % Froude Number
    subplot(3,2,4)
    plot(time_save/60,Froude(:,1),'LineStyle','--','LineWidth',2,'Color','k')
    hold on
    plot(time_save/60,Froude(:,ceil(Nx/2)),'LineStyle',':','LineWidth',2,'Color','k')
    hold on
    plot(time_save/60,Froude(:,Nx),'LineStyle','-','LineWidth',2,'Color','k')
    xlabel(date_string(flag_date),'interpreter','latex');
    ylabel('Froude Number','Interpreter','latex');
    legend('Entrance','L/2','Outlet','Interpreter','Latex','Location','best')

    % Courant Number
    subplot(3,2,5)
    plot(time_save/60,Courant(:,1),'LineStyle','--','LineWidth',2,'Color','k')
    hold on
    plot(time_save/60,Courant(:,ceil(Nx/2)),'LineStyle',':','LineWidth',2,'Color','k')
    hold on
    plot(time_save/60,Courant(:,Nx),'LineStyle','-','LineWidth',2,'Color','k')
    xlabel(date_string(flag_date),'interpreter','latex');
    ylabel('Courant Number','Interpreter','latex');
    legend('Entrance','L/2','Outlet','Interpreter','Latex','Location','best')

    % Rating Curve
    % Solving for normal Depth
```

```
    ymin = min(min(Depth));
    ymax = max(max(Depth));
    hs = 1; % 1 node
    % hs = ceil(1);
    if flag_section ≠ 4
            y_m = [ymin:0.01:ymax]'; % meters
            Qn = ...
                1/nm(hs).*A_function(D,Z1,Z2,a,b,y_m).*Rh_function(D,Z1,Z2,a,b,y_m).^(2/3).*I0(hs)^0.5;
    else
        % [y_table, A, P, Rh, y_bar, n_med, Beta, v, B, Q]
        % [   1,     2, 3, 4,     5,     6,       7,  8,   9, 10]
        col1 = 2; % Col with A
        for jj = 1:length(Flow_Area(:,1))
            Qn(jj,1) = Vlookup_g(irr_table,col1,Flow_Area(jj,hs),10); % Attention here
            y_m(jj,1) = Vlookup_g(irr_table,col1,Flow_Area(jj,hs),1);
            rh_i = Vlookup_g(irr_table,col1,Flow_Area(jj,hs),4);
        end
    end
    subplot(3,2,6)
    tbegin = 30; % (steps), considering initial stabilization of the domain
    plot(Discharge(2:end,hs),Depth(2:end,hs),'LineStyle','--','LineWidth',2,'Color','k')
    hold on
    plot(Discharge(2:end,ceil(Nx/2)),Depth(2:end,ceil(Nx/2)),'LineStyle',':','LineWidth',2,'Color','k')
    hold on
    plot(Qn,y_m,'LineStyle','-','LineWidth',2,'Color','k')
    xlabel('Flow Discharge (m\textsuperscript{3}/s)','Interpreter','latex');
    ylabel('Water Depth (m)','Interpreter','latex');
    ylim([ymin 1.1*max([max(y_m),max(y(ceil(Nx)))])]);
    legend('Q(Inlet)','Q(Nx/2)','$Q_{n}$ (L)','Interpreter','Latex','Location','best')
    hold off
    exportgraphics(gcf,fullfile(folderName,'Summary_Charts.pdf'),'ContentType','vector')
    clf
    close all
else
    close all
    % Time Calculation
    time_duration = time_save/3600/24 + Date_Begin;
    set(gcf,'units','inches','position',[2,0,8,10])
    date_string = {''};
    flag_date = 1;
    subplot(3,2,1)
    % Flows
    plot(time_duration,Discharge(:,1),'LineStyle','--','LineWidth',2,'Color','k')
    hold on
    plot(time_duration,Discharge(:,ceil(Nx/2)),'LineStyle',':','LineWidth',2,'Color','k')
    hold on
    plot(time_duration,Discharge(:,Nx),'LineStyle','-','LineWidth',2,'Color','k')
    hold on
    xlabel(date_string(flag_date),'interpreter','latex');
    ylabel('Flow Discharge (m\textsuperscript{3}/s)','Interpreter','latex');
    legend('Entrance','L/2','Outlet','Interpreter','Latex','location','best')
    % Velocity
    subplot(3,2,2)
    plot(time_duration,Velocity(:,1),'LineStyle','--','LineWidth',2,'Color','k')
    hold on
    plot(time_duration,Velocity(:,ceil(Nx/2)),'LineStyle',':','LineWidth',2,'Color','k')
    hold on
    plot(time_duration,Velocity(:,Nx),'LineStyle','-','LineWidth',2,'Color','k')
    %%% Normal Depth Velocity %%%
    xlabel(date_string(flag_date),'interpreter','latex');
    ylabel('Velocity (m/s)','Interpreter','latex');
    legend('Entrance','L/2','Outlet','Interpreter','Latex','Location','best')
    % Water Depth
    subplot(3,2,3)
    plot(time_duration,Depth(:,1),'LineStyle','--','LineWidth',2,'Color','k')
    hold on
    plot(time_duration,Depth(:,ceil(Nx/2)),'LineStyle',':','LineWidth',2,'Color','k')
    hold on
    plot(time_duration,Depth(:,Nx),'LineStyle','-','LineWidth',2,'Color','k')
    xlabel(date_string(flag_date),'interpreter','latex');
    ylabel('Water Depths (m)','Interpreter','latex');
    legend('Entrance','L/2','Outlet','Interpreter','Latex','Location','best')
    % Froude Number
    subplot(3,2,4)
    plot(time_duration,Froude(:,1),'LineStyle','--','LineWidth',2,'Color','k')
    hold on
    plot(time_duration,Froude(:,ceil(Nx/2)),'LineStyle',':','LineWidth',2,'Color','k')
```

```
    hold on
    plot(time_duration,Froude(:,Nx),'LineStyle','-','LineWidth',2,'Color','k')
    xlabel(date_string(flag_date),'interpreter','latex');
    ylabel('Froude Number','Interpreter','latex');
    legend('Entrance','L/2','Outlet','Interpreter','Latex','Location','best')

    % Courant Number
    subplot(3,2,5)
    plot(time_duration,Courant(:,1),'LineStyle','--','LineWidth',2,'Color','k')
    hold on
    plot(time_duration,Courant(:,ceil(Nx/2)),'LineStyle',':','LineWidth',2,'Color','k')
    hold on
    plot(time_duration,Courant(:,Nx),'LineStyle','-','LineWidth',2,'Color','k')
    xlabel(date_string(flag_date),'interpreter','latex');
    ylabel('Courant Number','Interpreter','latex');
    legend('Entrance','L/2','Outlet','Interpreter','Latex','Location','best')

    % Rating Curve
    % Solving for normal Depth
    ymin = min(min(Depth));
    ymax = max(max(Depth));
    hs = 1; % 1 node
    % hs = ceil(1);
    if flag_section ≠ 4
            y_m = [ymin:0.01:ymax]'; % meters
            Qn = ...
                1/nm(hs).*A_function(D,Z1,Z2,a,b,y_m).*Rh_function(D,Z1,Z2,a,b,y_m).^(2/3).*I0(hs)^0.5;
    else
        % [y_table, A, P, Rh, y_bar, n_med, Beta, v, B, Q]
        % [   1,     2, 3, 4,     5,     6,      7,  8,   9, 10]
        col1 = 2; % Col with A
        for jj = 1:length(Flow_Area(:,1))
            Qn(jj,1) = Vlookup_g(irr_table,col1,Flow_Area(jj,hs),10); % Attention here
            y_m(jj,1) = Vlookup_g(irr_table,col1,Flow_Area(jj,hs),1);
            rh_i = Vlookup_g(irr_table,col1,Flow_Area(jj,hs),4);
        end
    end
    subplot(3,2,6)
    tbegin = 30; % (steps), considering initial stabilization of the domain
    plot(Discharge(2:end,hs),Depth(2:end,hs),'LineStyle','--','LineWidth',2,'Color','k')
    hold on
    plot(Discharge(2:end,ceil(Nx/2)),Depth(2:end,ceil(Nx/2)),'LineStyle',':','LineWidth',2,'Color','k')
    hold on
    plot(Qn,y_m,'LineStyle','-','LineWidth',2,'Color','k')
    xlabel('Flow Discharge (m\textsuperscript{3}/s)','Interpreter','latex');
    ylabel('Water Depth (m)','Interpreter','latex');
    ylim([ymin 1.1*max([max(y_m),max(y(ceil(Nx)))])]);
    legend('Q(Inlet)','Q(Nx/2)','$Q_{n}$ (L)','Interpreter','Latex','Location','best')
    hold off
    exportgraphics(gcf,fullfile(folderName,'Summary_Charts.pdf'),'ContentType','vector')
    clf
    close all
end

%% States Post-Processing
states_post_processing
%% Cross-Section Post-Processing
if flag_section == 4
    cross_section_post_processing
end

%% Lateral Profiles
if flag_section ≠ 4
    wse_top_width_regular
end
%% Detailed Output
Detailed_Output_Script
```

5) Cross-Section Post Processing

The following matlab script shows the post processing of cross-section data.

```
    % Post-Processing Routine
% Model: HyPro-SWE
% Developer: Marcus Nobrega
% Last Update: 4/29/2023
% Goal: Create animations of water depth, top width, and water surface
% elevation

close all
close(video);

Video_Name = 'Depth_WSE_Top_Width.mp4';

% Set up video
video = VideoWriter(Video_Name,'MPEG-4');
open(video);

% Define water depths for each time
depths = Depth(:,1)';

% Preallocate Top Width
B2 = zeros(size(Flow_Area));

% Time
t = time_save; % Sec

% Define tick size
ticksize = [0.015 0.01];

% Define Tick Position
tickposition = 'in';


% Define polygon for the cross-section
polygon = polyshape(x_cross,y_cross);

% Water Surface Elevation
wse = Depth + repmat(inv_el',[size(Depth,1),1]);

% Color
color_plot = [21, 179, 196]/255;
set(gcf,'units','inches','position',[2,0,8,10])

if flag_elapsed_time ≠ 1
    % Time Calculation
    time_duration = time_save/3600/24 + Date_Begin;
end

% Iterate through all time steps
set(gca,'FontSize',14,'FontName','Garamond')
for i=1:(length(t))

    if flag_elapsed_time == 1
        Plot_Title = 'Time = %d (sec)';
        sgtitle(sprintf(Plot_Title, time_store(i)),'fontsize',18,'interpreter','latex')
    else
        sgtitle(string(time_duration(i)),'fontsize',18,'interpreter','latex');
    end
    for j = 1:3 % 3 Cross-sections
        if j == 1
            sec = 1;
        elseif j == 2
            sec = ceil(Nx/2);
        else
            sec = Nx;
        end
        depths = Depth(i,sec)';
        hold on
        subplot(3,3,(j))
        % Set title with time and water depth
        % Define the water depth for this time step
        depth_line = depths*ones(1,length(x_cross));
        plot(x_cross, y_cross, '-k', 'LineWidth', 2,Marker='*'); hold on
        % Find where depth line intersects cross-section polygon
        [x_intersect, y_intersect] = polyxpoly(x_cross,y_cross,x_cross,depth_line);
        if length(x_intersect)  > 1
            % Finding Inside Values
            idx1 = x_cross ≥ x_intersect(1);
```

```
            idx2 = x_cross ≤ x_intersect(end);
            idx = logical(idx1.*idx2); % Both cases
            x_pol = [x_intersect(1), x_cross(idx)', x_intersect(end)];
            y_pol = [y_intersect(1), y_cross(idx)', y_intersect(2)];
            hold on
            % If the depth line intersects the polygon, plot it
            if ¬isempty(x_intersect) && ¬isempty(y_intersect)
                depth_plot = depth_line(1)*ones(size(x_pol));
                fill([x_pol fliplr(x_pol)], [y_pol fliplr(depth_plot)],color_plot)
            else
                error('Call developer')
            end
        end
        box on
        if j == 1
            ylabel('Depth [m]','Interpreter','latex')
        end
        xlabel('Station [m]','Interpreter','latex')
        title(sprintf('x = %0.2f m, h = %0.2f m', round((sec-1)*dx,2), ...
            depths),'fontsize',16,'interpreter','latex');
        set(gca,'FontSize',12,'FontName','Garamond')
        % Set Tick Postion and Tick Size
        set(gca,'TickLength',ticksize)
        set(gca,'TickDir',tickposition)
    end

    % --------------- Plotting Channel Width ----------------- %
    subplot(3,3,[4 5 6]);
    if flag_section ≠ 4
        B2 = B_function(D,Z1,Z2,a,b,y);
    else
        for pos_b = 1:length(Flow_Area(1,:))
            % [y_table, A, P, Rh, y_bar, n_med, Beta, v, B, Q]
            % [   1,     2, 3, 4,     5,     6,       7,  8,   9, 10]
            B2(i,pos_b) = Vlookup_g(irr_table,col1,Flow_Area(i,pos_b),9);
        end
    end
    offset = max(x_cross)/2; % From station data
    right_margin = B2(i,:)/2 + offset; left_margin = -B2(i,:)/2 + offset;
    plot(x,right_margin,'k','LineWidth',2); set(gca,'YDir','reverse');
    hold on
    plot(x,left_margin,'k','LineWidth',2); set(gca,'YDir','reverse');
    hold on
    fill([x' fliplr(x')], [left_margin fliplr(right_margin)],color_plot)
    xlabel('$x$ [m]','Interpreter','latex');
    ylabel('Station [m]','Interpreter','latex');
    ylim([0, max(x_cross)]);
    grid on
    title(sprintf('$B_{{max}}(t)$ = %0.2f m', max(right_margin - ...
        left_margin)),'fontsize',16,'interpreter','latex');
    set(gca,'FontSize',12,'FontName','Garamond')
    % Set Tick Postion and Tick Size
    set(gca,'TickLength',ticksize)
    set(gca,'TickDir',tickposition)

    % ---------------------  Ploting Water Surface Elevation ------- %
    subplot(3,3,[7 8 9])
    plot(x,inv_el,'LineWidth',4,'LineStyle','-','Color','k');
    hold on
    plot(x,wse(i,:),'k','LineWidth',2,'LineStyle','-','Color',color_plot);
    fill([x' fliplr(x')], [inv_el' fliplr(wse(i,:))],color_plot)
    xlabel('$x$ [m]','Interpreter','latex');
    ylabel('Water Surface Elevation [m]','Interpreter','latex');
    ylim([0.98*min(min(wse - Depth)) max(max(1.01*wse))])
    grid on
    title(sprintf('$WSE_{{max}}(t)$ = %0.2f m', max(wse(i,:))),'fontsize',16,'interpreter','latex');
    set(gca,'FontSize',12,'FontName','Garamond')
    % Set Tick Postion and Tick Size
    set(gca,'TickLength',ticksize)
    set(gca,'TickDir',tickposition)

    % Save the frame for the video
    % Set background color and write to video
    frame = getframe(gcf);
    writeVideo(video,frame);
    hold off
    clf
```

```matlab
end
% Close video writer
close(video);
close all
```

6) Water Surface Elevation Profiles

The following matlab script shows the code to generate water surface elevation profiles in regular sections.

```matlab
    % Post-Processing Routine
% Model: HyPro-SWE
% Developer: Marcus Nobrega
% Last Update: 4/29/2023
% Goal: Create animations of WSE and Top Width for regular sections

close all

Video_Name = 'WSE_Top_Width.avi';

% Set up video
video = VideoWriter(Video_Name,'MPEG-4');
open(video);

% Define water depths for each time
depths = Depth(:,1)';

% Preallocate Top Width
B2 = zeros(size(Flow_Area));

% Time
t = time_save; % Sec

% Define tick size
ticksize = [0.02 0.01];



% Water Surface Elevation
wse = Depth + repmat(inv_el',[size(Depth,1),1]);

% Color
color_plot = [21, 179, 196]/255;
set(gcf,'units','inches','position',[2,0,8,10])

% Iterate through all time steps
set(gca,'FontSize',14,'FontName','Garamond')
for i=1:length(t)
    Plot_Title = 'Time = %d (sec)';
    sgtitle(sprintf(Plot_Title, time_store(i)),'fontsize',18,'interpreter','latex')
        % --------------- Plotting Channel Width ----------------- %
    subplot(2,3,[1 2 3]);
    if flag_section ≠ 4
        B2 = B_function(D,Z1,Z2,a,b,Depth);
    else
        for pos_b = 1:length(Flow_Area(1,:))
            % [y_table, A, P, Rh, y_bar, n_med, Beta, v, B, Q]
            % [   1,     2, 3, 4,     5,     6,       7,  8,   9, 10]
            B2(i,pos_b) = Vlookup_g(irr_table,col1,Flow_Area(i,pos_b),9);
        end
    end
    if flag_section == 1
         offset = b/2 + (Z1 + Z2)/2*max(max(depths));
         xmax_plot = (Z1 + Z2)*max(max(depths)) + b;
    elseif flag_section == 2
        offset = D/2;
        xmax_plot = D;
    elseif flag_section == 3
        offset = xmax/2;
        xmax_plot = xmax;
    else
        offset = max(x_cross)/2; % From station data
        xmax_plot = max(x_cross);
```

```
    end
    right_margin = B2(i,:)/2 + offset; left_margin = -B2(i,:)/2 + offset;
    plot(x,right_margin,'k','LineWidth',2); set(gca,'YDir','reverse');
    hold on
    plot(x,left_margin,'k','LineWidth',2); set(gca,'YDir','reverse');
    hold on
    fill([x' fliplr(x')], [left_margin fliplr(right_margin)],color_plot)
    xlabel('$x$ [m]','Interpreter','latex');
    ylabel('Station [m]','Interpreter','latex');
    ylim([0, xmax_plot]);
    xlim([0, max(x)]);
    grid on
    title(sprintf('$B_{{max}}(t)$ = %.2f m', max(right_margin - ...
        left_margin)),'fontsize',16,'interpreter','latex');
    set(gca,'FontSize',12,'FontName','Garamond')

    % ---------------------  Ploting Water Surface Elevation ------- %
    subplot(2,3,[4 5 6])
    plot(x,inv_el,'LineWidth',4,'LineStyle','-','Color','k');
    hold on
    plot(x,wse(i,:),'k','LineWidth',2,'LineStyle','-','Color',color_plot);
    fill([x' fliplr(x')], [inv_el' fliplr(wse(i,:))],color_plot)
    xlabel('$x$ [m]','Interpreter','latex');
    ylabel('Water Surface Elevation [m]','Interpreter','latex');
    ylim([0.98*min(min(wse - Depth)) max(max(1.01*wse))])
    grid on
    title(sprintf('$WSE_{{max}}(t)$ = %.2f m', max(wse(i,:))),'fontsize',16,'interpreter','latex');

    % Save the frame for the video
    set(gca,'FontSize',12,'FontName','Garamond')
    % Set background color and write to video
    frame = getframe(gcf);
    writeVideo(video,frame);
    hold off
end
% Close video writer
close(video);
close all
```

7) Detailed Output

The following script generates .csv outputs summarizing the collected data from the simulation.

```
    % HyProSWE Model
% Output .csv script
% Developer: Marcus Nobrega
% Goal: Create a detailed output from modeling results
% Last updated: 4/30/2023


%%% ----------------------- All rights reserved --------------------- %%

% Number of states
ns = 6;
% 0 - time, 1 - flow, 2 - depth, 3 - velocity, 4 - Courant, 5 - Froude, 6,
% 7 WSE

% Concatenate data
t = time_store; % time vector
h = Depth; % water level matrix
q = Discharge; % flow rate matrix
v = Velocity; % velocity matrix
f = Froude; % Froude number matrix
c = Courant; % Courant number matrix
z = x; % distance matrix

% Round Data
decimal_places = 3;

data = zeros(size(Depth,1),size(Depth,2),ns);

data(:,:,1) = Depth;
```

```
data(:,:,2) = Discharge;
data(:,:,3) = Velocity;
data(:,:,4) = Froude;
data(:,:,5) = Courant;
data(:,:,6) = wse;


if flag_output == 1
    for i = 1:(Nx*ns)
        j = floor((i-1)/ns);
        x_cell = j*dx;
        if mod(i-1,ns) == 0 || (i-1)/ns == 1
            states_title(1,i) = cellstr(sprintf('Depth (m), x(m) = %0.2f',x_cell));
        elseif mod(i-1,ns) == 1 || (i-1)/ns == 2
            states_title(1,i) = cellstr(sprintf('Discharge (m^3/s), x(m) = %0.2f',x_cell));
        elseif mod(i-1,ns) == 2 || (i-1)/ns == 3
            states_title(1,i) = cellstr(sprintf('Velocity (m/s), x(m) = %0.2f',x_cell));
        elseif mod(i-1,ns) == 3 || (i-1)/ns == 4
            states_title(1,i) = cellstr(sprintf('Froude (-), x(m) = %0.2f',x_cell));
        elseif mod(i-1,ns) == 4 || (i-1)/ns == 5
            states_title(1,i) = cellstr(sprintf('Courant Number (-), x(m) = %0.2f',x_cell));
        elseif mod(i-1,ns) == 5 || (i-1)/ns == 6
            states_title(1,i) = cellstr(sprintf('Water Surface Elevation (m), x(m) = %0.2f',x_cell));
        end
    end
else
    for i = 1:(Nx*ns)
        if mod(i,Nx) ≠ 0
            j = mod(i,Nx);
            x_cell = (j-1)*dx; % m
        else
            j = Nx;
            x_cell = (j-1)*dx; % m
        end
        if floor(i/Nx) == 0 || i/Nx == 1
            states_title(1,i) = cellstr(sprintf('Depth (m), x(m) = %0.2f',x_cell));
        elseif floor(i/Nx) == 1 || i/Nx == 2
            states_title(1,i) = cellstr(sprintf('Discharge (m^3/s), x(m) = %0.2f',x_cell));
        elseif floor(i/Nx) == 2 || i/Nx == 3
            states_title(1,i) = cellstr(sprintf('Velocity (m/s), x(m) = %0.2f',x_cell));
        elseif floor(i/Nx) == 3 || i/Nx == 4
            states_title(1,i) = cellstr(sprintf('Froude (-), x(m) = %0.2f',x_cell));
        elseif floor(i/Nx) == 4 || i/Nx == 5
            states_title(1,i) = cellstr(sprintf('Courant Number (-), x(m) = %0.2f',x_cell));
        elseif floor(i/Nx) == 5 || i/Nx == 6
            states_title(1,i) = cellstr(sprintf('Water Surface Elevation (m), x(m) = %0.2f',x_cell));
        end
    end
end
% states_title(1,end+1) = cellstr(sprintf('Water Surface Elevation (m), x(m) = %0.2f',dx*(Nx-1)));
time_string = {'Time (sec)'};
% Table Headers
table_headers = [time_string, states_title];
data_save = zeros(length(time_store),ns*Nx);


if flag_output == 1
    % Detailed Output for each section with all states together
    for i = 1:length(time_store)
        % For all time
        for j = 1:ns
            if j == 2
                ttt = 1;
            end
            % For all states
            for k = 1:Nx
                % For all nodes
                %               data_table = round(data(i,k,j),decimal_places);
                %               data_save(i,ns*(k-1) +  j) = data_table;
                data_table = round(data(i,k,j),decimal_places);
                data_save(i,ns*(k-1) +  j) = data_table;
            end
        end
    end
else
    % Detailed Output for each state for each section
    for i = 1:length(time_store)
```

```
        % For all time
        for j = 1:ns
            % For all states
            for k = 1:Nx
                % For all nodes
                data_table = round(data(i,k,j),decimal_places);
                data_save(i,k + (j-1)*Nx) = data_table;
            end
        end
    end
end



data_save = [time_save, data_save]; % Concatenating dataset to the time
T = array2table(data_save,'VariableNames',table_headers);
writetable(T,'Detailed_Output.csv','Delimiter',',');
disp('Attention: Data exported in .CSV');


%% Detailed Output per Cross-Section (Similarly as HEC-RAS)
i_prev = 1;
if flag_elapsed_time == 1
    time_str = 'Elapsed Time (sec)';
else
    time_str = 'Time';
end
Titles_Section = {'x(m)',time_str,' Depth (m)','Discharge (m3/s)','Velocity (m/s)','Froude ...
    (-)','Courant Number (-)','WSE (m)'};
for i = 1:Nx
    % Through each section
    for j = 1:length(time_store)
        % Through each time
        for k = 1:ns
            row = length(time_store)*(i-1) + j;
            data_save_XS(row,k) = data(j,i,k);
        end
    end
end

zzz = data_save_XS;
clear data_table data_save_XS
for i = 1:Nx
    x_cell = (i-1)*dx;
    if i == 1
        section(1,1) = x_cell;
    end
    row = length(time_store)*(i-1) + 1;
    row_i = length(time_store)*(i);
    data_save_XS((row + i-1):(row_i + i-1),:) = zzz(row:row_i,:);
    data_save_XS(row_i+1 + i - 1,:) = nan;

    section((row + i-1):(row_i + i-1),:) = x_cell;
    section(row_i+1 + i - 1,:) = NaN;
end
section(size(data_save_XS,1),1) = x_cell;
% section(end:size(data_save_XS,1)) = [];
% section(section == 0) = NaN;
if flag_elapsed_time == 1
    time_vector = time_save;
else
    time_vector = time_begin + time_save/86400; % Days minutes and seconds
end

Δ = 0;
for i = 1:Nx
    row = length(time_store)*(i-1) + 1;
    row_i = length(time_store)*(i);
    time_vector_total((row + i-1):(row_i + i-1),1) = time_vector;
    time_vector_total(row_i+1 + i - 1,:) = nan;
end

data_save = [section, time_vector_total, data_save_XS]; % Concatenating dataset to the time
T = array2table(data_save,'VariableNames',Titles_Section);

T.Properties.VariableNames(1:size(data_save,2)) = Titles_Section;
writetable(T,'Detailed_Output_XS.csv','Delimiter',',');
```

```
disp('Attention: XS Data exported in .CSV');
```

References

[1] K. I. S. d. Souza *et al.*, “Definição de áreas de preservação permanente com função de proteção aos recursos hídricos naturais,” 2021.
[2] L. Sabat and C. K. Kundu, “History of finite element method: A review,” p. 395–404, Jul 2020. [Online]. Available: http://dx.doi.org/10.1007/978-981-15-4577-1_32