

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

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

## 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 geometrical 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)

<sup>†</sup>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).

<sup>‡</sup>Department of Hydraulic Engineering and Sanitation, University of Sao Paulo, Sao Carlos School of Engineering, 13566-590 (luis.castillo@unah.hn).

<sup>£</sup>Faculty of Engineering, Architecture and Urbanism and Geography, Federal University of Mato Grosso do Sul, MS, 79070-900, Brazil. (paulo.t.oliveira@ufms.br)

<sup>††</sup>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).

<sup>§</sup>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	
$e_l$	0.27432	m
$g$	9.81	m/s <sup>2</sup>
$n_m$	0.025	
$I_0$	0.00025	m/m
$t_f$	360	min
$\Delta t$	1	sec
animation_time	10	min
$s_{f,outlet}$	0.00025	m/m
$N_{\Delta v}$	1000	
C	0.5	
$\Delta t_{min}$	0.5	sec
$\Delta t_{max}$	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<sup>2</sup>)
- $n_m$  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).
- $\Delta t$  is the time-step if a constant time-step is used.
- animation\_time is the interval of time considered for the results post-processing.
- $s_{f,outlet}$  is the outlet slope if normal condition are established (see sec. I-A2)
- $N_{\Delta v}$  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 activated 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.

a) **Flags**

flag_hydrograph	1
flag_outlet	1
flag_friction	1
flag_section	4
flag_stage_hydrograph	0
flag_nash	0
flag_slope	0
flag_elevation	0
flag_output	0
flag_plot_HP	0

b) **Model Description**

Tabular Inflow Hydrograph?	X
Nash Hydrograph?	
Tabular Stage Hydrograph?	
Tidal Outlet?	
Normal Slope?	X
Constant Bottom Slope?	X
Variable Bottom Slope?	
Known Inv. Elevation ?	
Cross-Section Type?	Irregular
Single Manning?	
SCM?	
DCM?	X

---

c) If flag\_nash == 1

$T_p$	0.5 h
$Q_b$	$0.05 \text{ m}^3/\text{s}$
Beta	8.5
$Q_p$	$3 \text{ m}^3/\text{s}$

d) If flag\_outlet >> 1

$h_{0,wave}$	0.5 m
$H_{0,wave}$	1 m
$L_{wave}$	1000 m
$T_{wave}$	12 hr
$x_{wave}$	500 m

e) If flag\_section = 1

b	10 m
$z_1$	0 m/m
$z_2$	0 m/m

f) If flag\_section = 2

D	3 m
---	-----

If flag\_section = 3

a	0.04 1/m
---	----------

g) If flag\_hydrograph = 1

Time (min)	Flow ( $\text{m}^3/\text{s}$ )
0	0.1
90	5
180	5
270	125.46331
360	333.19029
450	688.28377
540	886.919215
630	936.077402
720	767.656525
810	555.025699
900	342.394873
990	165.037435
1080	50.0923839
1170	5

h) If flag\_stage\_hydrograph = 1

Time (min)	Depth (m)
0.000	0.000
0.083	0.142
0.167	0.191
0.250	0.227
0.333	0.257
0.417	0.283
0.500	0.306
0.583	0.327
0.667	0.346
0.750	0.364
0.833	0.381
0.917	0.397
1.000	0.412
1.083	0.426

i) If flag\_stage\_slope = 1

Node	x(m)	$I_0$ (m/m)	Elevation (m)
1	0	0.001	10
2	250	0.001	10
3	500	0.001	10
4	750	0.001	10
5	1000	0.001	10
6	1250	0.001	10
7	1500	0.001	10
8	1750	0.001	10
9	2000	0.001	10
10	2250	0.001	10
11	2500	0.001	10
12	2750	0.001	10
13	3000	0.001	10
14	3250	0.001	10

**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 ( $\text{m}^3/\text{s}$ ). Beta is a the shape factor of the Nash hydrograph.  $Q_p$  is the magnitude of the peak flow ( $\text{m}^3/\text{s}$ ).

### 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		Section Discretization				
flag_length	0	Station (m)	Elevation (m)	Lengths (m)	$n(i,i+1)$	break_point divider
flag_method	2	0	195.05		0.035	
$S_0$	0.0001 m/m	5.0002	194.81		0.035	
$n_m$	0.035	10.0004	194.57		0.035	
$n_f$	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.

```

1 %% Algorithm - Section Coordinates
2 % Developer: Marcus Nobrega
3 % Date 5/16/2022
4 % Goal - Determine cross-section coordinates for different types of
5 % cross-sections
6 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% All Rights Reserved - contact: marcusnobrega.engcivil@gmail.com
7
8 clear all
9 % Single Sections
10 n_test = 0.02; % Roughness assumed
11 %% Triangular Section
12 hmax = 2; % maximum depth in m
13 b1 = 1; % left length in m
14 b2 = 2; % right length in m
15 x_1 = 0; % inicial x_coordinate for first value
16 y_1 = hmax; % inicial y_coordinate for first value
17 x = [x_1 (x_1 + b1) (x_1 + b1 + b2)]';
18 y = [y_1 (y_1 - hmax) (y_1)]';
19 x_triangular = x;
20 y_triangular = y;
21 n_channel_triangular = repmat(n_test,length(x_triangular)-1,1);
22 %% Parabolic Section
23 a = 1; % 1/m such that y = a*x^2 or x = sqrt(y/a)
24 hmax = 2; % maximum depth in m
25 step = 0.01; % height step in m
26 n_steps = floor(hmax/step);
27 y = linspace(0,hmax,n_steps);
28 x_right = sqrt(y/a);
29 x_left = flip(-x_right,2);
30 y_left = flip(y,2);

```

```

31 x = [x_left x_right]';
32 y = [y_left y]';
33 x_parabolic = x;
34 xmin = min(x_parabolic);
35 x_parabolic = x_parabolic + abs(xmin);
36 y_parabolic = y;
37 n_channel_parabolic = repmat(n_test,length(x_parabolic)-1,1);
38 %% Semi-Hyperbolic and Semi-Parabolic
39 % Hyperbole Equation ->  $y^2/a^2 - x^2/b^2 = 1$ 
40 % a = 0.1;
41 % b = 0.01;
42 % xc = 0;
43 % yc = 0;
44 % hmax = 1; % maximum depth in m
45 % step = 0.01; % height step in m
46 % n_steps = floor(hmax/step);
47 % y = linspace(0,hmax,n_steps);
48 % x_left = xc + sqrt(a^2*(-1 + (y - yc).^2/(b^2)));
49 % x_left = flip(-x_left,2);
50 % % Parabolic Equation
51 % a = 0.01; % 1/m such that  $y = a*x^2$  or  $x = \sqrt{y/a}$ 
52 % x_right = sqrt(y/a);
53 % % Final
54 % x = [x_left x_right]';
55 % y = [flip(y,2) y]';
56 % Composite Sections
57 %% Semi-Elliptical and Semi-Parabolic
58 % Ellipse Equation ->  $(x-xc)^2/a^2 + (y-yc)^2/b^2 = 1$ 
59 hmax = 2; % maximum depth in m
60 a = 2*hmax;
61 b = hmax;
62 xc = -a;
63 yc = 0;
64 step = 0.01; % height step in m
65 n_steps = floor(hmax/step);
66 y = linspace(0,hmax,n_steps);
67 x_left = xc + sqrt(a^2*(1 - (y - yc).^2/(b^2)));
68 x_left = flip(x_left,2);
69 % Parabolic Equation
70 a = 0.1; % 1/m such that  $y = a*x^2$  or  $x = \sqrt{y/a}$ 
71 x_right = sqrt(y/a);
72 % Final
73 x = [x_left x_right]';
74 y = [flip(y,2) y]';
75 x_semi = x;
76 xmin = min(x_semi);
77 x_semi = x_semi + abs(xmin);
78 y_semi = y;
79 n_channel_semi = repmat(n_test,length(x_semi)-1,1);
80 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Composite Sections%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
81 %% Road Gutter Cross-Section
82 hmax = 2; % maximum depth in m
83 b_1 = 0; % gutter width in m, typically 0 if vertical
84 b_2 = 0.4; % gutter width in m
85 b_3 = 1.2; % wetted road width in (m)
86 h_1 = 0.15; % curb height (m)
87 h_2 = 0.10; % gutter height (m)
88 h_3 = 0.12; % water depth (m) ≤ h_1
89 x_1 = 0; % inicial x_coordinate for first value
90 y_1 = max([h_1 h_2 h_3]); % inicial y_coordinate for first value
91 x = [x_1 (x_1 + b_1) (x_1 + b_1 + b_2) (x_1 + b_1 + b_2 + b_3)]';
92 y = [y_1 (y_1 - h_1) (y_1 - h_1 + h_2) (y_1 - h_1 + h_2 + h_3)]';
93 x_gutter = x;
94 xmin = min(x_gutter);
95 x_gutter = x_gutter + abs(xmin);
96 y_gutter = y;
97 n_channel_road = repmat(n_test,length(x_gutter)-1,1);
98 %% Sucessive Trapezoid Gabion Channel
99 b0 = 0; % width within vertical points (m)
100 b = 2; % width of horizontal gabion (m)
101 h = 0.5; % height of the gabion (m)
102 n_vertical = 4; % number of vertical gabions
103 x_1 = 0; % inicial x_coordinate for first value
104 y_1 = h*n_vertical; % inicial y_coordinate for first value
105 x = 0;
106 y = 0;
107 for i = 1:(n_vertical*2)

```

```

108     if i == 1
109         x(i,1) = x_1;
110         y(i,1) = y_1;
111     else
112         if mod(i,2) == 1 % Odd number
113             x(i,1) = x(i-1,1) + b;
114             y(i,1) = y(i-1,1);
115         else
116             x(i,1) = x(i-1,1) + b0;
117             y(i,1) = y(i-1,1) - h;
118         end
119     end
120 end
121 x_left = x;
122 y_left = y;
123 x_right = 0; y_right = 0;
124 for i = 1:(n_vertical*2)
125     if i == 1
126         x_right(i,1) = x_left(end,1) + b;
127         y_right(i,1) = y_left(end,1);
128     else
129         if mod(i,2) == 1 % Odd number
130             x_right(i,1) = x_right(i-1,1) + b;
131             y_right(i,1) = y_right(i-1,1);
132         else
133             x_right(i,1) = x_right(i-1,1) + b0;
134             y_right(i,1) = y_right(i-1,1) + h;
135         end
136     end
137 end
138 x = [x_left;x_right]';
139 y = [y_left;y_right]';
140 x_gabion = x;
141 xmin = min(x_gabion);
142 x_gabion = x_gabion + abs(xmin);
143 y_gabion = y;
144 n_channel_triangular = repmat(n_test,length(x_gabion)-1,1);
145 %% Composite V-Notch and Francis Weir
146 b_rec = 0.75; % width of rectangular weir besides the v-notch (m)
147 hrec = 1; % rectangular height
148 h_vnot = 1; % v-notch height
149 alfa = pi/4; % 45 degree
150 x_1 = 0;
151 y_1 = hrec + h_vnot;
152 x = [x_1 (x_1) (x_1 + b_rec) (x_1 + b_rec + h_vnot/tan(atan(alfa))) (x_1 + b_rec + ...
153     2*h_vnot/tan(atan(alfa))) (x_1 + b_rec + 2*h_vnot/tan(atan(alfa)) + b_rec) (x_1 + ...
154     2*h_vnot/tan(atan(alfa)) + 2*b_rec)]';
155 y = [y_1 (y_1 - hrec) (y_1 - hrec) (y_1 - hrec - h_vnot) (y_1 - hrec) (y_1 - hrec) (y_1)]';
156 x_vnot = x;
157 y_vnot = y;
158 n_channel_trapezoid = repmat(n_test,length(x_vnot)-1,1);
159 %% Irregular Channel
160 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 ...
161     339.3 338.6 339.3 340.5 342.7 342.7 342.3 342 341.9 341.7 341.5 342.3 ...
162     342.7 343.2]';
163 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 ...
164     5.8 5.8 15.8 17.7 7 18.9 38.1 27.4 62.7]';
165 x_irr(i,1) = 0;
166 for i = 1:length(l_irr)
167     x_irr(i+1,1) = x_irr(i,1) + l_irr(i,1);
168 end
169 n_channel_triangular = repmat(n_test,length(x_irr)-1,1);
170 % x_final = [x_triangular x_parabolic x_semi x_gutter x_gabion x_vnot x_irr]';
171 % y_final = [y_triangular y_parabolic y_semi y_gutter y_gabion y_vnot x_irr]';
172 %% Plot Cross-Sections
173 subplot(4,2,1)
174 line_w = 2;
175 c = [64 64 64]/255;
176 font = 12;
177 set(gcf,'units','inches','position',[4,4,6.5,4])
178 set(gca,'FontSize',font)
179 plot(x_triangular,y_triangular,'LineWidth',line_w,'color',c)
180 xlabel('x(m)','Interpreter','latex','FontSize',font)
181 ylabel('y(m)','Interpreter','latex','FontSize',font)
182 grid on
183 set(gca,'FontSize',font)
184 subplot(4,2,2)

```

```

180 plot(x_parabolic,y_parabolic,'LineWidth',line_w,'color',c)
181 xlabel('x(m)','Interpreter','latex','FontSize',font)
182 ylabel('y(m)','Interpreter','latex','FontSize',font)
183 grid on
184 set(gca,'FontSize',font)
185 subplot(4,2,3)
186 plot(x_semi,y_semi,'LineWidth',line_w,'color',c)
187 xlabel('x(m)','Interpreter','latex','FontSize',font)
188 ylabel('y(m)','Interpreter','latex','FontSize',font)
189 grid on
190 set(gca,'FontSize',font)
191 subplot(4,2,4)
192 plot(x_gutter,y_gutter,'LineWidth',line_w,'color',c)
193 xlabel('x(m)','Interpreter','latex','FontSize',font)
194 ylabel('y(m)','Interpreter','latex','FontSize',font)
195 grid on
196 set(gca,'FontSize',font)
197 subplot(4,2,5)
198 plot(x_gabion,y_gabion,'LineWidth',line_w,'color',c)
199 xlabel('x(m)','Interpreter','latex','FontSize',font)
200 ylabel('y(m)','Interpreter','latex','FontSize',font)
201 grid on
202 set(gca,'FontSize',font)
203 subplot(4,2,6)
204 plot(x_vnot,y_vnot,'LineWidth',line_w,'color',c)
205 xlabel('x(m)','Interpreter','latex','FontSize',font)
206 ylabel('y(m)','Interpreter','latex','FontSize',font)
207 grid on
208 set(gca,'FontSize',font)
209 % Irr
210 subplot(4,2,[7:8])
211 y_irr = y_irr - min(y_irr);
212 plot(x_irr,y_irr,'LineWidth',line_w,'color',c)
213 xlabel('x(m)','Interpreter','latex','FontSize',font)
214 ylabel('y(m)','Interpreter','latex','FontSize',font)
215 grid on
216 set(gca,'FontSize',font)
217 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

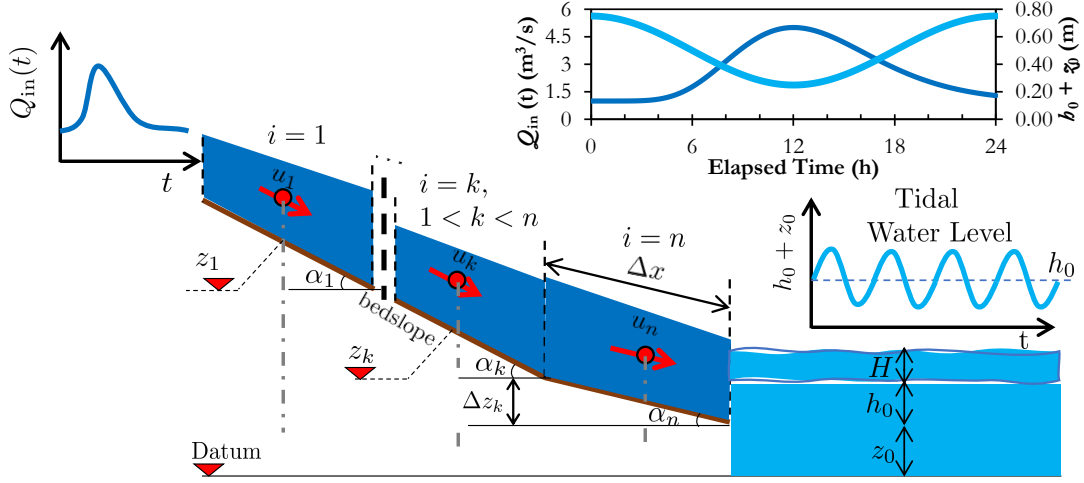
For an inflow hydrograph boundary condition, we can write:

$$Q_1(t + \Delta t) - Q_h(t + \Delta t) = 0 \quad (1a)$$

$$A_1(t + \Delta t) - A_2(t) = 0 \quad (1b)$$

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 sharpened



**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:

$$h_1(t + \Delta t) - h_s(t + \Delta t) = 0 \quad (2a)$$

$$Q_1(t + \Delta t) = Q_2(t) \quad (2b)$$

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:

$$h_1(t + \Delta t) - h_s(t + \Delta t) = 0 \quad (3a)$$

$$Q_1(t + \Delta t) - Q_h(t + \Delta t) = 0 \quad (3b)$$

## 4) Known Friction Slope at Outlet

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

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

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

where  $s_{out}$  is given.

## 5) Tidal Outlet Boundary Condition

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





### 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.

$$f_1(i) = y_{i+1} - \left( \frac{y_i - y_{i+1}}{x_i - x_{i+1}} \right) (x_i - k) \quad (6)$$

$$f_2 = x_{i+1} - \left( \frac{y_{i+1} - y}{y_i - y_{i+1}} \right) (x_{i+1} - x_i) \quad (7)$$

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)).

$$f_3(i) = \sqrt{(y_i - y_{i+1})^2 + (x_i - x_{i+1})^2} \quad (8)$$

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,  $\phi$  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

```

1  # %% Cross Section Hydraulic Properties Estimator %% #
2  # Developer: Luis Castillo
3  # Date 5/20/2022
4  # Goal: Determine hydraulic properties for regular or irregular cross-section
5
6  import numpy as np
7  import pandas as pd
8  import math

```

**Algorithm 1** Finite Element Procedure with nested loops

**Input:** cross-section points  $\delta$ , elements resolution  $r$ , Manning roughness coefficient  $man$ , and slope  $s$ . From  $\delta$ , vectors  $seg_x$  and  $seg_y$  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  $seg_y[i][0] \geq j/r > seg_y[i][1]$  or  $seg_y[i][0] \leq j/r < seg_y[i][1]$  then
       $seg_{y2} = \text{append}(i)$ 
       $seg_{x2} = \text{append}((seg_x[i][0] + seg_x[i][1])/2 - mid)$ 
    end if
  end for
   $seg_{x3} = \text{array}(seg_{x2})$ 
   $lw = \text{max argument}((\text{where}(seg_{x3} < 0, seg_{x3}, -\text{inf}))$ 
   $rw = \text{min argument}((\text{where}(seg_{x3} > 0, seg_{x3}, \text{inf}))$ 
  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
    |  $per = \text{append}(f_3(i))$ 
  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, conveyance, streamflow, flow velocity for the  $j$  water depth.
end for

```

```

9  import matplotlib.pyplot as plt
10 from matplotlib import pyplot
11 from numpy import exp
12
13 noise = 0.01
14 res = 10 # To be defined by the user, this resolution means the quantity of elements between ...
15         point, i.e., between
16         # two coordinates (1 and 2) on the vertical axis, and for a res = 10, 10 elements will be ...
17         discretized between
18         # 1 and 2 coordinates. the bigger the quantity of elements, the better representation, ...
19         however, it takes more
20         # time of processing.
21 man = 0.012 # To be defined by the user, Manning roughness coefficient
22 s = 0.00398 # To be defined by the user, slope of the cross-section
23
24
25 file = open("D:/Google_drive/Meu Drive/Papers/Paper - Nota_tecnica/j1.csv")
26 coors = pd.read_csv(file, delimiter=';', header=None).values
27 plt.plot(coors[:, 0], coors[:, 1])
28
29 Ymax, Ymin, Xmax, Xmin = max(coors[:, 1]), min(coors[:, 1]), max(coors[:, 0]), min(coors[:, 0]) # ...
30 Maximum and minimum values of the list of coordinates
31 for m in range(len(coors)):
32   if coors[m][1] ≤ Ymin: # Looking for the middle part of the cross-section
33     middle = coors[m][0]
34
35 # --- Preallocate HP --- #
36 area, top, = np.zeros((int(Ymax*res - Ymin*res), 1)), np.zeros((int(Ymax*res - Ymin*res), 1))
37 perimeter_2, y = np.zeros((int(Ymax*res - Ymin*res), 1)), np.zeros((int(Ymax*res - Ymin*res), 1))
38 RH, centroid = np.zeros((int(Ymax*res - Ymin*res), 1)), np.zeros((int(Ymax*res - Ymin*res), 1))
39 con, phi = np.zeros((int(Ymax*res - Ymin*res), 1)), np.zeros((int(Ymax*res - Ymin*res), 1))
40 Q, center = np.zeros((int(Ymax*res - Ymin*res), 1)), np.zeros((int(Ymax*res - Ymin*res), 1))

```

```

35 seg_x, seg_y = np.zeros((len(coors[:, 0]) - 1, 2)), np.zeros((len(coors[:, 0]) - 1, 2))
36
37 for i in range(len(coors) - 1):
38     seg_x[i, 0], seg_x[i, 1] = coors[i, 0], coors[i+1, 0]
39     seg_y[i, 0], seg_y[i, 1] = coors[i, 1], coors[i+1, 1]
40
41
42 def per(i):
43     return math.sqrt(pow(seg_y[i, 0]-seg_y[i, 1], 2) + pow(seg_x[i, 0]-seg_x[i, 1], 2))
44
45
46 def image_x(i, j): # Function that according to the horizontal position of K, returns the vertical ...
47     image of the segment
48     if seg_y[i, 0] == seg_y[i, 1]: # if there is a vertical wall
49         return (seg_x[i, 0]) - (((seg_y[i, 0] - j)*(seg_x[i, 0]-seg_x[i, 1])) / ((seg_y[i, ...
50             0]-seg_y[i, 0]*noise) - (seg_y[i, 1]+seg_y[i, 1]*noise)))
51     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, ...
52         1]))
53
54 def image_y(i, j): # Function that according to the horizontal position of K, returns the vertical ...
55     image of the segment
56     if seg_x[i, 0] == seg_x[i, 1]: # if there is a horizontal wall
57         return (seg_y[i, 0]) - ((seg_y[i, 0] - seg_y[i, 1])/((seg_x[i, 0]-seg_x[i, ...
58             0]*noise)-(seg_x[i, 1]+seg_x[i, 1]*noise)))*(seg_x[i, 0] - j)
59     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)
60
61 mg = np.zeros((int(round((Ymax-Ymin)*res)), int(round((Xmax-Xmin)*res))), dtype=int) # Main Grid
62
63 for j in range(int(round(Ymin*res))+1, int(round(Ymax*res))): # Looping thought the vertical axis
64     seg_x_2, seg_y_2 = np.zeros((len(seg_y), 1)), np.zeros((len(seg_y), 1))
65     for i in range(len(seg_y)): # finding the upper boundary of the water deep
66         if (seg_y[i, 0] >= j/res > seg_y[i, 1]) or (seg_y[i, 0] <= j/res <= seg_y[i, 1]):
67             seg_y_2[i, 0] = i
68             seg_x_2[i, 0] = (seg_x[i, 0]+seg_x[i, 1])/2 - middle
69     left_wall = np.where(seg_x_2 < 0, seg_x_2, -np.inf).argmax() # Finding the walls that contains ...
70     the current
71     right_wall = np.where(seg_x_2 > 0, seg_x_2, np.inf).argmin() # water level
72
73 if left_wall == right_wall: # this condition is meet when water level is higher the profile
74     break
75
76 for i in np.arange(round(image_x(left_wall, j/res)*res) - Xmin*res, # Looping thought the ...
77     horizontal axis
78     round(image_x(right_wall, j/res)*res) - Xmin*res): # Modifying the main grid
79     for k in range(len(seg_x)):
80         if (seg_x[k, 0] <= (i / res + Xmin) < seg_x[k, 1]): # Looking for what segment "i" ...
81             belongs to.
82             break
83     mg[round(Ymax*res-j): int(round(Ymax*res)) - int(round(image_y(k, (i/res + Xmin))*res)), ...
84         int(i)] = 1
85     center[int(j - Ymin*res), 0] = ((np.count_nonzero(mg[:, int(i)] == 1)/2)/res + (image_y(k, ...
86         (i/res)))) * (np.count_nonzero(mg[:, int(i)] == 1)/pow(res, 2))
87
88 perimeter = []
89 for i in range(left_wall,
90     right_wall): # all segments between the walls but not including they selfs
91     perimeter.append(per(i))
92     perimeter.append(math.sqrt(pow(j/res - seg_y[left_wall, 1], 2) +
93         pow(image_x(left_wall, j/res) - seg_x[left_wall, 1],
94             2))) # perimeter for the left boundary
95     perimeter.append(math.sqrt(pow(j/res - seg_y[right_wall, 0], 2) +
96         pow(image_x(right_wall, j/res) - seg_x[right_wall, 0],
97             2))) # perimeter for the right boundary
98
99 area[int(j - Ymin*res), 0] = np.sum(mg) / pow(res, 2)
100 y[int(j - Ymin*res), 0] = j / res - Ymin
101 perimeter_2[int(j - Ymin*res), 0] = np.sum(perimeter)
102 RH[int(j - Ymin*res), 0] = (np.sum(mg) / pow(res, 2))/np.sum(perimeter)
103 top[int(j - Ymin*res), 0] = image_x(right_wall, j/res)-image_x(left_wall, j/res)
104 centroid[int(j - Ymin*res), 0] = np.sum(center)/(np.sum(mg))
105 con[int(j - Ymin*res), 0] = (1/man)*(np.sum(mg) / pow(res, 2))*pow((np.sum(mg) / ...
106     pow(res,2))/(np.sum(perimeter)), 2/3)
107 phi[int(j - Ymin*res), 0] = (np.sum(mg) / pow(res, 2))*pow((np.sum(mg) / ...
108     pow(res,2))/(np.sum(perimeter)), 2/3)

```

```

99     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)
100
101 # --- Filling with Nan all extra elements in the arrays --- #
102 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
103 RH[int(j - Ymin*res): , 0], top[int(j - Ymin*res): , 0], centroid[int(j - Ymin*res): , 0] = ...
    math.nan, math.nan, math.nan
104 con[int(j - Ymin*res): , 0], phi[int(j - Ymin*res): , 0], Q[int(j - Ymin*res): , 0] = math.nan, ...
    math.nan, math.nan
105 plt.imshow(mg)
106
107 # --- Plotting the HP curves --- #
108 fig, (ax1, ax2, ax3, ax4, ax5, ax6, ax7, ax8) = plt.subplots(1, 8)
109 fig.suptitle('2b')
110 ax1.plot(area, y)
111 ax1.set_xlabel('Area $(m^2)$')
112 ax1.set_ylabel('water depth $(m)$')
113 ax2.plot(perimeter_2, y)
114 ax2.set_xlabel('Perimeter $(m)$')
115 ax3.plot(top, y)
116 ax3.set_xlabel('Top lenght $(m)$')
117 ax4.plot(RH, y)
118 ax4.set_xlabel('Hydraulic radius $(m)$')
119 ax5.plot(centroid, y)
120 ax5.set_xlabel('Centroid $(m)$')
121 ax6.plot(con, y)
122 ax6.set_xlabel('Conveyance $(m^3/s)$')
123 ax7.plot(phi, y)
124 ax7.set_xlabel('Phi $(m^3/s)$')
125 ax8.plot(Q, y)
126 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.

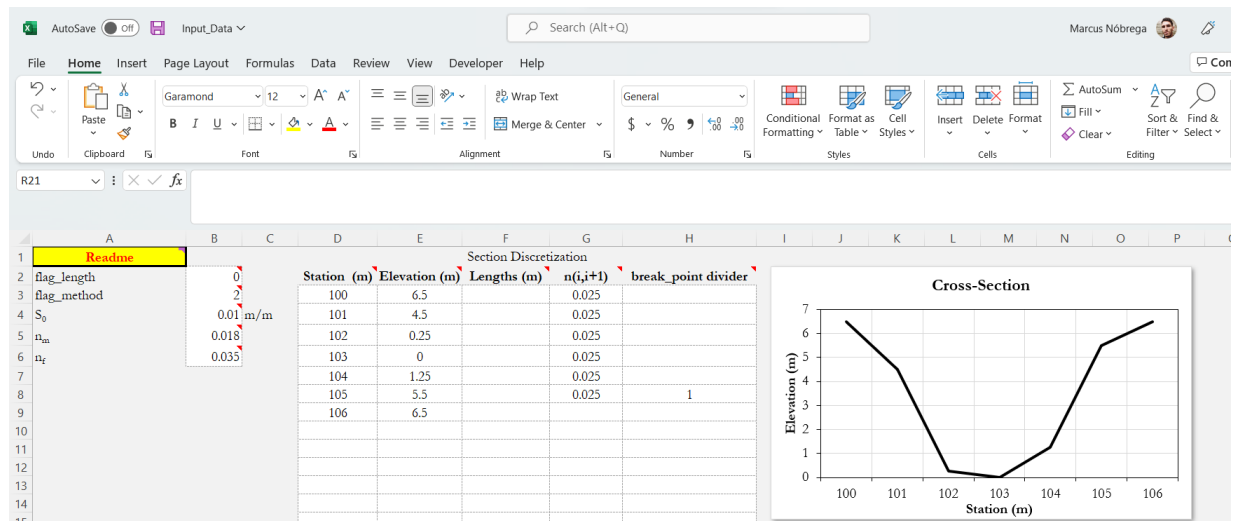
Overall, this function reads the input data and return plots of

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

```

1  %%% Determining Irregular Cross-section Functions %%%
2  % Developer: Marcus Nobrega Gomes Junior
3  % Date: 2022/05/03
4  % Goal - Calculate Hydraulic Properties of Irregular and Regular Sections
5  % for a given cross-sections and Manning's roughness coefficients
6
7  function [y_table, A, P, Rh, y_bar, n_med, Beta, v, B, Q, x_absolute, y,s0] = ...
    HP_estimator(flag_plot_HP,dh)
8  input_table = xlsread('HyProSWE_Input_Data.xlsx','Irregular_Cross_Section');
9  input_data = input_table(1:5,1);
10 input_data_coordinates = input_table(2:end,3:end);
11 flag_length = input_data(1,1); % If == 1, use lengths as main input data, otherwise use absolute ...
    values of x (m)
12 flag_method = input_data(2,1); % If == 1, SCM, else DCM
13 s0 = input_data(3,1); % Slope in m/m
14 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.

```

15 nf = input_data(5,1); % Overbanks channel roughness
16
17 if flag_method == 1
18     n_channel = input_data_coordinates(1:(end-1),4);
19 end
20
21 % Retrieving Data
22 x_absolute = input_data_coordinates(:,1);
23 elevations = input_data_coordinates(:,2);
24 lengths = input_data_coordinates(1:(end-1),3);
25 break_point_divider = input_data_coordinates(1:(end),5);
26
27 Δ = zeros(length(elevations),1);
28 for i = 1:(length(elevations)-1)
29     Δ(i) = abs(elevations(i+1,1) - elevations(i,1));
30 end
31 Δ_h = min(Δ(Δ > 0));
32 tic
33
34 % Checking input data consistency
35 if length(elevations) ≤ 3
36     error('Please, enter at least 4 points for elevation and 3 points for manning and lengths. If ...
37         you have a triangular shape, please enter the invert elevation twice and add a 0 length and ...
38         0 manning, such that you have 4 points for elevation and 3 points for manning and lengths')
39 end
40
41 points = (1:1:length(elevations))'; % stations from 1 to n
42
43 % Let's assume a maximum 1 cm difference in the depths
44 % Noise
45 noise_max = 0.01; % m
46 % Let's also assume a minimum 0.1 cm difference in the depths, that is, the
47 % noise
48 noise_min = 0.001; % m
49 noise = Δ_h/dh; % Noise in m from user input data
50 if noise > noise_max
51     noise = noise_max; % m
52 elseif noise < noise_min
53     noise = noise_min; % m
54 end
55
56 factor = 1; %precision = 1/factor * noise
57
58 [au,ia] = unique(elevations,'stable');
59 Same = ones(size(elevations));
60 Same(ia) = 0; % repetitive values
61 noise_i = rand(1,1)*noise;
62 small_number = noise/100;

```

```

61 % New Elevation and X_values
62 ii = 0;
63 for i = 1:(length(elevations) - 1)
64     el1 = elevations(i); el2 = elevations(i+1);
65     x1 = x_absolute(i); x2 = x_absolute(i+1);
66     if el1 == el2 || abs(el1 - el2) == noise
67         elevations(i+1) = elevations(i+1) + noise;
68         if elevations(i+1) == elevations(i)
69             elevations(i+1) = elevations(i+1) + noise;
70         end
71     end
72     if x1 == x2 || abs(x2 - x1) == noise
73         x_absolute(i+1) = x_absolute(i+1) + noise;
74         if x_absolute(i+1) == x_absolute(i)
75             x_absolute(i+1) = x_absolute(i+1) + noise;
76         end
77     end
78 end
79
80 % if max(isnan(n_channel)) > 0
81 %     error('Please, enter (n-1) data for Manning coefficient, where n is the number of break-points')
82 % end
83
84 % Roughness Boundary Condition
85 if flag_method == 1
86     n_channel(end+1,1) = 0; % adding last boundary condition
87 end
88
89 % Minimum elevation
90 min_el = min(elevations); % m
91 % y (bottom to up)
92 y = elevations - min_el;
93 pos_inv = find(y == 0); % position of invert elevation
94 % If we have more than 1 invert
95 pos_inv = pos_inv(1);
96
97 % x (left to right)
98 if flag_length == 1
99     for i = 1:length(y) % coordinates of each measured point
100         if i == 1
101             x_absolute(i,1) = 0 + noise;
102         else
103             x_absolute(i,1) = x_absolute(i-1) + lengths(i-1) + noise;
104         end
105     end
106 else % Lengths are already assumed from the input data table
107     for i = 1:length(y)
108         if i ≠ length(y)
109             lengths(i) = x_absolute(i+1) - x_absolute(i);
110         end
111     end
112 end
113
114 % Alfa min
115 alfa_min_bound = noise/max(lengths(lengths>1e-8));
116 big_n = 100000*atan(asin(1)); % big number making sure it is a multiple of 1 rad, so that ...
117     sin(atan(big_n)) = 1
118 min_length = min(lengths(lengths>0));
119
120 % Invert coordinates
121 x_invert = x_absolute(pos_inv,1);
122 y_invert = 0;
123
124 % Slopes (taking from x (left-right) y (down-up)
125 % For point 1 and for the last point
126 alfa_1 = (y(1,1) - y(2,1))/lengths(1,1);
127
128 % Unsorted Values
129 x_left_unsorted = x_absolute(1:(pos_inv-1),1);
130 y_left_unsorted = y(1:(pos_inv-1),1);
131 x_right_unsorted = x_absolute(pos_inv + 1:end,1);
132 y_right_unsorted = y(pos_inv + 1:end,1);
133 if flag_method == 1
134     n_left_unsorted = n_channel(1:(pos_inv-1),1);
135     n_right_unsorted = n_channel(pos_inv:(end-1),1);
136 end

```

```

137 % Maximum depth (left and right)
138 max_left = max(y_left_unsorted); max_right = max(y_right_unsorted);
139 max_y = min(max_left, max_right);
140
141 % Refreshing values of ymax
142 pos_r = length(y_right_unsorted);
143 if max_left ≠ max_right
144     if max_left > max_y % the maximum is located at left
145         z = sort(y_left_unsorted,1,'descend');
146         if length(z) == 1 % Case where we have a vertical wall
147             z(2,1) = y_invert;
148         end
149         x_left_first = round(x_absolute(2) - (max_y - z(2))/alfa_l,2);
150         % New values of x and y
151         x_absolute(1) = x_left_first;
152         y(1) = max_y;
153         pos_r = length(y_right_unsorted);
154     else
155         pos_r = find(y_right_unsorted > max_y ,1,'first');
156         alfa_r = (y_right_unsorted(pos_r) - y_right_unsorted(pos_r - ...
157             1))/lengths(length(y_left_unsorted) + 1 + pos_r-1);
158         z = sort(y_right_unsorted,1,'descend');
159         x_rigth_last = round(x_absolute(end-1) + (max_y - z(2))/alfa_r,2);
160         % New values of x and y
161         x_absolute(end) = x_rigth_last;
162         y(length(y_left_unsorted) + 1 + pos_r) = max_y;
163     end
164     dim = 1:(length(y_left_unsorted) + 1 + pos_r);
165     y = y(dim,1);
166     x_absolute = x_absolute(dim,1);
167     % n_channel = n_channel(dim,1);
168     points = points(dim);
169
170 % New Unsorted Values with New max
171 x_left_unsorted = x_absolute(1:(pos_inv-1),1);
172 y_left_unsorted = y(1:(pos_inv-1),1);
173 x_right_unsorted = x_absolute(pos_inv + 1:end,1);
174 y_right_unsorted = y(pos_inv + 1:end,1);
175 if flag_method == 1
176     n_left_unsorted = n_channel(1:(pos_inv-1),1);
177     n_right_unsorted = n_channel(pos_inv:(end-1),1);
178 end
179
180 % Main Matrix
181 % table = [points,x_absolute,y,n_channel];
182
183 % % Vlookup Function
184 % Vlookup_eq = @(data,col1,vall,col2) data((find(data(:,col1)==vall,1)),col2); %Vlookup function as ...
185 % Vlookup_leq = @(data,col1,vall,col2) data((find(data(:,col1)≤vall,1)),col2); %Vlookup function as ...
186 % Excel
187
188 % Sections left
189 numb_left = length(find(y_left_unsorted ≥ y_left_unsorted(end)));
190 % Sections right
191 numb_right = length(find(y_right_unsorted ≥ y_right_unsorted(1)));
192 % Tot sections
193 tot_sections = numb_left + numb_right - 1; % take one out because both sides are equal
194
195 y_l_prev = y_left_unsorted(2:length(y_left_unsorted));
196 y_l_next = y_left_unsorted(1:(length(y_left_unsorted)-1));
197
198 %%% Precision
199 precision = 1/factor*noise; % m
200
201 %%% small number ≥ 1 < 1e-8 + 1
202 sm = (1e-8 + 1);
203
204 %%% Total_Noise
205 tot_noise = noise*sum(Same);
206 % Main loop
207 i = 0; int_n_p = 0; % integral of n*perimeter
208
209 %% Define Main Channel and Overbanks
210 pos_break = find(break_point_divider == 1); % Position where the divider occurs
211 % Main Channel Height

```



```

211 ym = y(pos_break); % Main channel height (m)
212 if pos_break > pos_inv % Left intersection
213     % Left intersection
214     posm_left = find(y_left_unsorted ≥ ym,1,'last');
215     ym_left_up = y_left_unsorted(posm_left);
216     xm_left_up = x_left_unsorted(posm_left);
217     ym_left_down = y_left_unsorted(min(posm_left+1,length(y_left_unsorted)));
218     xm_left_down = x_left_unsorted(min(posm_left+1,length(y_left_unsorted)));
219     % Angles
220     if (ym_left_up - ym_left_down ≤ length(y_left_unsorted)*noise)
221         alfa_m_l = big_n;
222     else
223         alfa_m_l = (ym_left_up - ym_left_down)/(xm_left_down - xm_left_up); % Slope
224     end
225     xm_left = xm_left_down - (ym - ym_left_down)/alfa_m_l;
226     ym_left = ym;
227     % Polygons (left - inv - right)
228     x_pol = [xm_left; x_left_unsorted((posm_left + 1:end),1); x_invert; ...
                x_right_unsorted(1:(pos_break-pos_inv),1)];
229     y_pol = [ym_left; y_left_unsorted((posm_left + 1:end),1); y_invert; ...
                y_right_unsorted(1:(pos_break-pos_inv),1)];
230     % Top-Width
231     bm = abs(x_pol(1) - x_pol(end));
232     % Area
233     am = polyarea(x_pol,y_pol);
234     % Perimeter
235     polyin = polyshape(x_pol,y_pol);
236     pm = perimeter(polyin) - bm; % Taking away the top width
237 else
238     % Right Intersection
239     posm_right = find(y_right_unsorted ≥ ym,1,'first');
240     ym_right_up = y_right_unsorted(posm_right);
241     xm_right_up = x_right_unsorted(posm_right);
242     ym_right_down = y_right_unsorted(max(posm_right-1,1));
243     xm_right_down = x_right_unsorted(max(posm_right-1,1));
244     % Angles
245     if (ym_right_up - ym_right_down < noise*length(y_right_unsorted)) % No depth
246         alfa_m_r = big_n;
247     else
248         alfa_m_r = (ym_right_up - ym_right_down)/(xm_right_up - xm_right_down); % Slope
249     end
250     xm_right = xm_right_down + (ym - ym_right_down)/alfa_m_r;
251     ym_right = ym;
252     % Polygons (left - inv - right)
253     x_pol = [x_left_unsorted(pos_break:end,1); x_invert; x_right_unsorted(1:(posm_right - 1),1); ...
                xm_right];
254     y_pol = [y_left_unsorted(pos_break:end,1); y_invert; y_right_unsorted(1:(posm_right - 1),1); ...
                ym_right];
255     % Top-Width
256     bm = abs(x_pol(1) - x_pol(end));
257     % Area
258     am = polyarea(x_pol,y_pol);
259     % Perimeter
260     polyin = polyshape(x_pol,y_pol);
261     pm = perimeter(polyin) - bm; % Taking away the top width
262 end
263 if flag_method ≠ 1
264     % Number of floodplains
265     if pos_break == 1 || pos_break == length(y)
266         n_fp = 1;
267     else
268         n_fp = 2;
269     end
270 end
271 while i < big_n
272     %% Case where i == 1
273     i = i + 1;
274     n_P_left = 0;
275     n_P_right = 0;
276     n_P_left_extra = 0;
277     n_P_right_extra = 0;
278     B_extra = 0;
279     P_extra = 0;
280     P_extra_left = 0;
281     P_extra_right = 0;
282     if i == 1 % We are talking about the first point
283

```

```

284     %% Initializing variables
285     y_table = 0; h = 0; B = 0; A = 0; Rh = 0; P = 0; Phi = 0; K_c = 0;
286     % Look to both sides from pos_inv (invert point)
287
288     % Left Direction
289     pos_left = find(y_left_unsorted>sm*y_invert,1,'last');
290     y_left_point = y_left_unsorted(pos_left,1);
291     x_left_point = x_left_unsorted(pos_left,1);
292     if flag_method == 1
293         n_left_segment = n_left_unsorted(pos_left,1);
294     else
295         n_left_segment = nm; % Main channel
296     end
297
298     % Right Direction
299     pos_right = find(y_right_unsorted>sm*y_invert,1,'first');
300     y_right_point = y_right_unsorted(pos_right,1);
301     x_right_point = x_right_unsorted(pos_right,1);
302     if flag_method == 1
303         n_right_segment = n_right_unsorted(pos_right,1);
304     else
305         n_right_segment = nm; % Main channel
306     end
307
308     % Angles Calculations
309     %%% Alfa Left %%%
310     % Case 01 - Vertical Point
311     if (x_invert - x_left_point ≤ tot_noise) && (y_left_point - y_invert > tot_noise)
312         alfa_l = big_n;
313         alfa_l_tang = big_n;
314     end
315     % Case 02 - Horizontal Point
316     if (x_invert - x_left_point > tot_noise) && (y_left_point - y_invert ≤ tot_noise)
317         alfa_l = big_n;
318         alfa_l_tang = big_n;
319     end
320     % Case 03 - Horizontal and Vertical Point
321     if (x_invert - x_left_point ≤ tot_noise) && (y_left_point - y_invert ≤ tot_noise)
322         alfa_l = big_n;
323         alfa_l_tang = big_n;
324     end
325     % Case 04 - Point with normal slopes
326     if (x_invert - x_left_point > tot_noise) && (y_left_point - y_invert > tot_noise)
327         alfa_l = (y_left_point - y_invert)/(x_invert - x_left_point);
328         alfa_l_tang = alfa_l;
329     end
330
331     %%% Alfa Right %%%
332     % Case 01 - Vertical Point
333     if (x_right_point - x_invert ≤ tot_noise) && (y_right_point - y_invert > tot_noise)
334         alfa_r = big_n;
335         alfa_r_tang = big_n;
336     end
337     % Case 02 - Horizontal Point
338     if (x_right_point - x_invert > tot_noise) && (y_right_point - y_invert ≤ tot_noise)
339         alfa_r = big_n;
340         alfa_r_tang = big_n;
341     end
342     % Case 03 - Horizontal and Vertical Point
343     if (x_right_point - x_invert ≤ tot_noise) && (y_right_point - y_invert ≤ tot_noise)
344         alfa_r = big_n;
345         alfa_r_tang = big_n;
346     end
347     % Case 04 - Point with normal slopes
348     if (x_right_point - x_invert > tot_noise) && (y_right_point - y_invert > tot_noise)
349         alfa_r = (y_right_point - y_invert)/(x_right_point - x_invert);
350         alfa_r_tang = alfa_r;
351     end
352
353     % Min Angle
354     if alfa_l ≤ alfa_min_bound
355         alfa_l_tang = big_n;
356     end
357     if alfa_r ≤ alfa_min_bound
358         alfa_r_tang = big_n;
359     end
360

```

```

361     if y_left_point ≤ y_right_point
362         y_moving = y_left_point;
363         x_left_point = x_absolute(pos_inv - 1,1);
364         precision_section = min(y_left_point - y_invert,precision);
365         n_points = floor((y_left_point - y_invert)/(precision_section)); % number of ...
366         interpolated points
367         if n_points == 1 % only one point means no slope
368             if x_invert - x_left_point ≥ sm*noise && alfa_l == big_n
369                 P_extra_left = sqrt((x_invert - x_left_point)^2 + (y_invert - y_left_point)^2);
370                 n_P_left_extra = P_extra_left*n_left_segment^(3/2);
371                 B_extra = (x_invert - x_left_point);
372             else
373                 B_extra = 0;
374                 n_P_left_extra = 0;
375                 P_extra_left;
376             end
377         end
378         if n_points == 1 % only one point means no slope
379             if x_right_point - x_invert > 1.0001*noise && alfa_r == big_n
380                 P_extra_right = sqrt((x_invert - x_right_point)^2 + (y_invert - ...
381                     y_right_point)^2) + B_extra;
382                 B_extra = B_extra + (x_right_point - x_invert);
383                 n_P_right_extra = (P_extra_right)*n_right_segment^(3/2);
384             else
385                 n_P_left_extra = 0;
386                 P_extra_right = 0;
387             end
388         end
389         P_extra = P_extra_right + P_extra_left;
390
391         % Main loop for i == 1
392         for j = 1:(n_points)
393             h = precision_section;
394             y_table(j+1,1) = y_table(j,1) + h;
395             B(j+1,1) = h/alfa_l_tang + h/alfa_r_tang + B(j,1);
396             A(j+1,1) = (B(j+1,1) + B(j,1))*h/2 + A(j,1); % Trapezoid
397             P(j+1,1) = h/sin(atan(alfa_l_tang)) + h/sin(atan(alfa_r_tang)) + P(j,1);
398             Rh(j+1,1) = A(j+1,1)/P(j+1,1);
399             Phi(j+1,1) = A(j+1,1)*Rh(j+1,1)^(2/3);
400             int_n_p = n_P_left_extra + n_P_right_extra + ...
401                 n_left_segment^(3/2)*h/sin(atan(alfa_l_tang)) + ...
402                 n_right_segment^(3/2)*h/sin(atan(alfa_r_tang)) + int_n_p;
403             % Representative Roughness Coefficient
404             if flag_method == 1
405                 n_med(j+1,1) = (int_n_p/P(j+1,1))^(2/3);
406             else
407                 if y_table(j+1,1) > ym
408                     yf = max(y_table(j+1,1) - ym,0); % Overbank depth
409                     af = max(A(j+1,1) - (am + bm*yf),0); % Overbank flow area
410                     pf = max(P(j+1,1) - pm,0); % Floodplain perimeter (m)
411                     pm_star = max(pm + n_fp*yf,0);
412                     am_star = max(am + bm*yf,0);
413                     n_med(j+1,1) = (Phi(j+1,1))/(1/nf*af*(af/pf)^(2/3) + ...
414                         1/nm*am_star*(am_star/pm_star)^(2/3));
415                 else
416                     yf = 0; % Overbank depth
417                     af = 0; % Overbank flow area
418                     pf = 0; % Floodplain perimeter (m)
419                     pm_star = 0;
420                     am_star = 0;
421                     n_med(j+1,1) = nm;
422                 end
423             end
424             K_c(j+1,1) = 1/n_med(j+1,1)*Phi(j+1,1);
425
426             if j == (n_points) % final point
427                 % Final point - make sure you have the exact surveyed point at the end
428                 h_ = y_right_point - y_table(j,1);
429                 y_table(j+1,1) = y_table(j,1) + h_;
430                 B(j+1,1) = h_/alfa_l_tang + h_/alfa_r_tang + B(j,1) + B_extra;
431                 A(j+1,1) = (B(j+1,1) + B(j,1))*h_/2 + A(j,1); % Trapezoid
432                 P(j+1,1) = h_/sin(atan(alfa_l_tang)) + h_/sin(atan(alfa_r_tang)) + P(j,1) + ...
433                     P_extra;
434                 Rh(j+1,1) = A(j+1,1)/P(j+1,1);
435                 Phi(j+1,1) = A(j+1,1)*Rh(j+1,1)^(2/3);
436                 if n_points == 1
437                     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;
432 else
433     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;
434 end
435 % Representative Roughness Coefficient
436 if flag_method == 1
437     n_med(j+1,1) = round((int_n_p/P(j+1,1))^(2/3),3);
438 else
439     if y_table(j+1,1) > ym
440         yf = max(y_table(j+1,1) - ym,0); % Overbank depth
441         af = max(A(j+1,1) - (am + bm*yf),0); % Overbank flow area
442         pf = max(P(j+1,1) - pm,0); % Floodplain perimeter (m)
443         pm_star = max(pm + n_fp*yf,0);
444         am_star = max(am + bm*yf,0);
445         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);
446     else
447         yf = 0; % Overbank depth
448         af = 0; % Overbank flow area
449         pf = 0; % Floodplain perimeter (m)
450         pm_star = 0;
451         am_star = 0;
452         n_med(j+1,1) = nm;
453     end
454 end
455 K_c(j+1,1) = 1/n_med(j+1,1)*Phi(j+1,1);
456 end
457 end
458 else
459     x_right_point = x_absolute(pos_inv + 1,1);
460     precision_section = min(y_right_point - y_invert,precision);
461     n_points = floor((y_right_point - y_invert)/(precision_section)); % number of ...
        interpolated points
462 if n_points == 1 % only one point means no slope
463     if x_right_point - x_invert >= sm*noise && alfa_r == big_n % Additional B_extra
464         P_extra = sqrt((x_right_point - x_invert)^2 + (y_right_point - y_invert)^2);
465         B_extra = x_right_point - x_invert;
466         n_P_right_extra = P_extra*n_right_segment^(3/2);
467     else
468         B_extra = 0;
469         n_P_right_extra = 0;
470         P_extra = 0;
471     end
472 end
473 y_moving = y_right_point;
474 % For loop to calculate functions
475 for j = 1:(n_points)
476     h = precision_section;
477     B(j+1,1) = h/alfa_l_tang + h/alfa_r_tang + B(j,1);
478     y_table(j+1,1) = y_table(j,1) + h;
479     A(j+1,1) = (B(j+1,1) + B(j,1))*h/2 + A(j,1); % Trapezoid
480     P(j+1,1) = h/sin(atan(alfa_l_tang)) + h/sin(atan(alfa_r_tang)) + P(j,1);
481     Rh(j+1,1) = A(j+1,1)/P(j+1,1);
482     Phi(j+1,1) = A(j+1,1)*Rh(j+1,1)^(2/3);
483     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;
484 % Representative Roughness Coefficient
485 if flag_method == 1
486     n_med(j+1,1) = round((int_n_p/P(j+1,1))^(2/3),3);
487 else
488     if y_table(j+1,1) > ym
489         yf = max(y_table(j+1,1) - ym,0); % Overbank depth
490         af = max(A(j+1,1) - (am + bm*yf),0); % Overbank flow area
491         pf = max(P(j+1,1) - pm,0); % Floodplain perimeter (m)
492         pm_star = max(pm + n_fp*yf,0);
493         am_star = max(am + bm*yf,0);
494         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));
495     else
496         yf = 0; % Overbank depth
497         af = 0; % Overbank flow area
498         pf = 0; % Floodplain perimeter (m)
499         pm_star = 0;
500         am_star = 0;

```

```

501         n_med(j+1,1) = nm;
502     end
503 end
504 K_c(j+1,1) = 1/n_med(j+1,1)*Phi(j+1,1);
505 if j == (n_points) % final point
506     % Final point - make sure you have the exact surveyed point at the end
507     h_ = y_right_point - y_table(j,1);
508     y_table(j+1,1) = y_table(j,1) + h_;
509     B(j+1,1) = h_/alfa_l_tang + h_/alfa_r_tang + B(j,1) + B_extra;
510     A(j+1,1) = (B(j+1,1) + B(j,1))*h/2 + A(j,1); % Trapezoid
511     P(j+1,1) = h_/sin(atan(alfa_l_tang)) + h_/sin(atan(alfa_r_tang)) + P(j,1) + ...
        P_extra;
512     Rh(j+1,1) = A(j+1,1)/P(j+1,1);
513     Phi(j+1,1) = A(j+1,1)*Rh(j+1,1)^(2/3);
514     if n_points == 1
515         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;
516     else
517         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;
518     end
519     % Representative Roughness Coefficient
520     if flag_method == 1
521         n_med(j+1,1) = (int_n_p/P(j+1,1))^(2/3);
522     else
523         if y_table(j+1,1) > ym
524             yf = max(y_table(j+1,1) - ym,0); % Overbank depth
525             af = max(A(j+1,1) - (am + bm*yf),0); % Overbank flow area
526             pf = max(P(j+1,1) - pm,0); % Floodplain perimeter (m)
527             pm_star = max(pm + n_fp*yf,0);
528             am_star = max(am + bm*yf,0);
529             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));
530         else
531             yf = 0; % Overbank depth
532             af = 0; % Overbank flow area
533             pf = 0; % Floodplain perimeter (m)
534             pm_star = 0;
535             am_star = 0;
536             n_med(j+1,1) = nm;
537         end
538     end
539     K_c(j+1,1) = 1/n_med(j+1,1)*Phi(j+1,1);
540 end
541 end
542 end
543 % Previous Positions
544 pos_left_previous = pos_left;
545 pos_right_previous = pos_right;
546 else
547     %% Case where i ≠ 1
548
549     % Look to left sides from x_point_left and from right side of
550     % x_point_right
551     y_moving = y_table(end,1); % actual water depth
552
553     % Left Direction
554     pos_left = find(y_left_unsorted>sm*y_moving,1,'last');
555     y_left_point = y_left_unsorted(pos_left,1);
556     x_left_point = x_left_unsorted(pos_left,1);
557
558     % Right Direction
559     pos_right = find(y_right_unsorted>sm*y_moving,1,'first');
560     y_right_point = y_right_unsorted(pos_right,1);
561     x_right_point = x_right_unsorted(pos_right,1);
562
563     % Roughness
564     if y_moving ≤ ym % Inside of the channel
565         if flag_method == 1
566             n_left_segment = n_left_unsorted(pos_left,1);
567             n_right_segment = n_right_unsorted(pos_right,1);
568         else
569             if (abs(y_left_unsorted(pos_left) - ym) ≤ noise*length(y_left_unsorted))
570                 n_left_segment = nf; % Attention here
571             else
572                 n_left_segment = nm; % Attention here

```

```

573         end
574         if (abs(y_right_unsorted(pos_right) - ym) ≤ noise*length(y_right_unsorted))
575             n_right_segment = nf; % Attention here
576         else
577             n_right_segment = nm; % Attention here
578         end
579     end
580 else % Overbanks
581     if flag_method == 1
582         n_left_segment = n_left_unsorted(pos_left,1);
583         n_right_segment = n_right_unsorted(pos_right,1);
584     elseif y_left_unsorted(pos_left) - ym < noise*length(y_left_unsorted)% Check Noises
585         n_left_segment = nm; % Attention here
586         n_right_segment = nm; % Attention here
587     else
588         n_left_segment = nf; % Attention here
589         n_right_segment = nf; % Attention here
590     end
591 end
592
593
594 % Checking Discontinuities
595 %% Initializing Variables
596 Delta_Area_left = 0; Delta_Area_right = 0;
597 Delta_B_left = 0; Delta_B_right = 0;
598 Delta_P_left = 0; Delta_P_right = 0;
599
600 % Angles Calculation
601 if pos_left + 1 > length(y_left_unsorted)
602     x_prev_left = x_invert;
603     y_prev_left = y_invert;
604 else
605     x_prev_left = (x_left_unsorted(pos_left + 1,1));
606     y_prev_left = (y_left_unsorted(pos_left + 1,1));
607 end
608
609 % Alfa Left
610 % Case 01 - Vertical Point
611 if (x_prev_left - x_left_point ≤ tot_noise) && (y_left_point - y_prev_left > tot_noise)
612     alfa_l = big_n;
613     alfa_l_tang = big_n;
614 end
615 % Case 02 - Horizontal Point
616 if (x_prev_left - x_left_point > tot_noise) && (y_left_point - y_prev_left ≤ tot_noise)
617     alfa_l = big_n;
618     alfa_l_tang = big_n;
619 end
620 % Case 03 - Horizontal and Vertical Point
621 if (x_prev_left - x_left_point ≤ tot_noise) && (y_left_point - y_prev_left ≤ tot_noise)
622     alfa_l = big_n;
623     alfa_l_tang = big_n;
624 end
625 % Case 04 - Point with normal slopes
626 if (x_prev_left - x_left_point > tot_noise) && (y_left_point - y_prev_left > tot_noise)
627     alfa_l = (y_left_point - y_prev_left)/(x_prev_left - x_left_point);
628     alfa_l_tang = alfa_l;
629 end
630 if pos_right == 1
631     x_prev_right = x_invert;
632     y_prev_right = y_invert;
633 else
634     x_prev_right = x_right_unsorted(pos_right - 1,1);
635     y_prev_right = y_right_unsorted(pos_right - 1,1);
636 end
637 % Alfa Right
638 % Case 01 - Vertical Point
639 if (x_right_point - x_prev_right ≤ tot_noise) && (y_right_point - y_prev_right > tot_noise)
640     alfa_r = big_n;
641     alfa_r_tang = big_n;
642 end
643 % Case 02 - Horizontal Point
644 if (x_right_point - x_prev_right > tot_noise) && (y_right_point - y_prev_right ≤ tot_noise)
645     alfa_r = big_n;
646     alfa_r_tang = big_n;
647 end
648 % Case 03 - Horizontal and Vertical Point
649 if (x_right_point - x_prev_right ≤ tot_noise) && (y_right_point - y_prev_right ≤ tot_noise)

```

```

650     alfa_r = big_n;
651     alfa_r_tang = big_n;
652 end
653 % Case 04 - Poit with normal slopes
654 if (x_right_point - x_prev_right > tot_noise) && (y_right_point - y_prev_right > tot_noise)
655     alfa_r = (y_right_point - y_prev_right)/(x_right_point - x_prev_right);
656     alfa_r_tang = alfa_r;
657 end
658
659 % Min Angle
660 if alfa_l ≤ alfa_min_bound
661     alfa_l_tang = big_n;
662 end
663 if alfa_r ≤ alfa_min_bound
664     alfa_r_tang = big_n;
665 end
666
667
668 if (pos_left_previous - pos_left) > 1 % More than one movement
669
670     % intersect
671     if alfa_l_tang == 0
672         x_intersect = x_left_unsorted(pos_left + 1,1);
673     else
674         x_intersect = x_left_unsorted(pos_left + 1,1) - (y_moving - ...
675             y_left_unsorted(pos_left + 1,1))/alfa_l;
676     end
677     x_pol = []; y_pol = [];
678     for nn = 1:(pos_left_previous - pos_left)
679         x_pol = [x_pol; x_left_unsorted(pos_left_previous - nn + 1)];
680         y_pol = [y_pol; y_left_unsorted(pos_left_previous - nn + 1)];
681     end
682     % Adding intersection
683     x_pol = [x_pol;x_intersect];
684     y_pol = [y_pol;y_moving];
685     % Delta B
686     Delta_B_left = abs(x_pol(1) - x_pol(end));
687     % Delta A
688     Delta_Area_left = polyarea(x_pol,y_pol);
689     % Delta P
690     polyin = polyshape(x_pol,y_pol);
691     Delta_P_left = perimeter(polyin) - Delta_B_left; % Taking away top width
692     n_P_left = Delta_P_left*n_left_segment^(3/2);
693     % Delta Rh left
694     % Phi left
695     % Conductance Left
696 end
697
698 % Checking Discontinuities
699 if (pos_right - pos_right_previous) > 1 % More than one movement
700     % intersect
701     if alfa_r_tang == 0
702         x_intersect = x_right_unsorted(pos_right - 1,1);
703     else
704         x_intersect = x_right_unsorted(pos_right - 1,1) + (y_moving - ...
705             y_right_unsorted(pos_right - 1,1))/alfa_r;
706     end
707     x_pol = []; y_pol = [];
708     for nn = 1:(pos_right - pos_right_previous)
709         x_pol = [x_pol; x_right_unsorted(pos_right_previous + nn - 1)];
710         y_pol = [y_pol; y_right_unsorted(pos_right_previous + nn - 1)];
711     end
712     % Adding intersection
713     x_pol = [x_pol;x_intersect];
714     y_pol = [y_pol;y_moving];
715     % Delta B
716     Delta_B_right = abs(x_pol(1) - x_pol(end));
717     % Delta A
718     Delta_Area_right = polyarea(x_pol,y_pol);
719     % Delta P
720     polyin = polyshape(x_pol,y_pol);
721     Delta_P_right = perimeter(polyin) - Delta_B_right; % Taking away top width
722     % Manning * Perimeter
723     n_P_right = Delta_P_right*n_right_segment^(3/2);
724 end
725 y_moving_end = min(y_right_point,y_left_point);

```

```

725 %         if (y_moving_end - y_moving)/(precision/100) < 1
726 %             error('Please, increase precision. Instability!')
727 %         end
728 precision_section = min(y_moving_end - y_moving, precision); % meters
729 if y_moving_end - y_moving < precision
730     ttt = 1;
731 end
732 n_points = floor((y_moving_end - y_moving)/(precision_section)); % number of interpolated ...
733 points
734 % For loop to calculate functions
735 if n_points == 1 % only one point means no slope
736     if y_moving_end == y_right_point && y_moving_end == y_left_point && alfa_l == big_n && ...
737         alfa_r == big_n
738         B_extra = x_right_point - x_prev_right + x_prev_left - x_left_point;
739         P_extra_left = sqrt((x_prev_left - x_left_point)^2 + (y_prev_left - y_left_point)^2);
740         P_extra_right = sqrt((x_right_point - x_prev_right)^2 + (y_right_point - ...
741             y_prev_right)^2);
742     elseif y_moving_end == y_right_point && alfa_r == big_n
743         if pos_right == 1
744             P_extra_right = sqrt((x_right_point - x_invert)^2 + (y_right_point - y_invert)^2);
745             B_extra = x_right_point - x_invert;
746         else
747             P_extra_right = sqrt((x_right_point - x_prev_right)^2 + (y_right_point - ...
748                 y_prev_right)^2);
749             B_extra = x_right_point - x_prev_right;
750         end
751     else % y_moving == y_left
752         if pos_left + 1 > length(x_left_unsorted) && alfa_l == big_n
753             P_extra_left = sqrt((x_invert - x_left_point)^2 + (y_invert - y_left_point)^2);
754             B_extra = x_invert - x_left_point;
755         elseif alfa_l == big_n
756             P_extra_left = sqrt((x_prev_left - x_left_point)^2 + (y_prev_left - ...
757                 y_left_point)^2);
758             B_extra = x_prev_left - x_left_point;
759         end
760     % Right
761     if pos_right == 1 && alfa_r == big_n
762         P_extra_right = sqrt((x_invert - x_right_point)^2 + (y_invert - y_right_point)^2);
763         B_extra = x_right_point - x_invert + B_extra;
764     elseif alfa_r == big_n
765         P_extra_left = sqrt((x_prev_right - x_right_point)^2 + (y_right_point - ...
766             y_prev_right)^2);
767         B_extra = x_right_point - x_prev_right + B_extra;
768     end
769 end
770 P_extra = P_extra_left + P_extra_right;
771 n_P_right_extra = P_extra_right*n_right_segment^(3/2);
772 n_P_left_extra = P_extra_left*n_left_segment^(3/2);
773 else
774     B_extra = 0;
775     n_P_right_extra = 0;
776     n_P_left_extra = 0;
777     P_extra = 0;
778     P_extra_left = 0;
779     P_extra_right = 0;
780 end
781
782 dim_table = length(y_table);
783 %%%%%%%%%%% Main loop for i ≠ 1 %%%%%%%%%%%
784
785 for j = 1:(n_points)
786     k = dim_table + j;
787     if j == 1 % We have to add values from discontinuity (Deltas)
788         h = precision_section; % meters
789         y_table(k,1) = y_table(k-1,1) + h;
790         % Roughness
791         if y_table(k,1) ≤ ym % Inside of the channel
792             if flag_method == 1
793                 n_left_segment = n_left_unsorted(pos_left,1);
794                 n_right_segment = n_right_unsorted(pos_right,1);
795             else
796                 if (abs(y_left_unsorted(pos_left) - ym) ≤ noise*length(y_left_unsorted))
797                     n_left_segment = nf; % Attention here
798                 else
799                     n_left_segment = nm; % Attention here
800                 end
801             end
802         end
803     end
804 end

```



```

796         if (abs(y_right_unsorted(pos_right) - ym) ≤ noise*length(y_right_unsorted))
797             n_right_segment = nf; % Attention here
798         else
799             n_right_segment = nm; % Attention here
800         end
801     end
802 else % Overbanks
803     if flag_method == 1
804         n_left_segment = n_left_unsorted(pos_left,1);
805         n_right_segment = n_right_unsorted(pos_right,1);
806     elseif y_left_unsorted(pos_left) - ym < noise*length(y_left_unsorted) % Check Noises
807         n_left_segment = nm; % Attention here
808         n_right_segment = nm; % Attention here
809     else
810         n_left_segment = nf; % Attention here
811         n_right_segment = nf; % Attention here
812     end
813 end
814 B(k,1) = B(k-1,1) + Delta_B_left + Delta_B_right + h/alfa_l_tang + h/alfa_r_tang;
815 A(k,1) = A(k-1,1) + (B(k,1) + B(k-1,1))*h/2 + Delta_Area_left + Delta_Area_right;
816 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;
817 Rh(k,1) = A(k,1)/P(k,1);
818 Phi(k,1) = A(k,1)*Rh(k,1)^(2/3);
819 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;
820 % Representative Roughness Coefficient
821 if flag_method == 1
822     n_med(k,1) = (int_n_p/P(k,1))^(2/3);
823 else
824     if y_table(k,1) > ym
825         yf = max(y_table(k,1) - ym,0); % Overbank depth
826         af = max(A(k,1) - (am + bm*yf),0); % Overbank flow area
827         pf = max(P(k,1) - pm,0); % Floodplain perimeter (m)
828         pm_star = max(pm + n_fp*yf,0);
829         am_star = max(am + bm*yf,0);
830         n_med(k,1) = (Phi(k,1))/(1/nf*af*(af/pf)^(2/3) + ...
            1/nm*am_star*(am_star/pm_star)^(2/3));
831     else
832         yf = 0; % Overbank depth
833         af = 0; % Overbank flow area
834         pf = 0; % Floodplain perimeter (m)
835         pm_star = 0;
836         am_star = 0;
837         n_med(k,1) = nm;
838     end
839 end
840 K_c(k,1) = 1/n_med(k,1)*Phi(k,1);
841 else
842     % Functions in terms of depth
843     h = precision_section;
844     y_table(k,1) = h + y_table(k-1,1);
845     % Roughness
846     if y_table(k,1) ≤ ym % Inside of the channel
847         if flag_method == 1
848             n_left_segment = n_left_unsorted(pos_left,1);
849             n_right_segment = n_right_unsorted(pos_right,1);
850         else
851             if (abs(y_left_unsorted(pos_left) - ym) ≤ noise*length(y_left_unsorted))
852                 n_left_segment = nf; % Attention here
853             else
854                 n_left_segment = nm; % Attention here
855             end
856             if (abs(y_right_unsorted(pos_right) - ym) ≤ noise*length(y_right_unsorted))
857                 n_right_segment = nf; % Attention here
858             else
859                 n_right_segment = nm; % Attention here
860             end
861         end
862     else % Overbanks
863         if flag_method == 1
864             n_left_segment = n_left_unsorted(pos_left,1);
865             n_right_segment = n_right_unsorted(pos_right,1);
866         elseif y_left_unsorted(pos_left) - ym < noise*length(y_left_unsorted) % Check Noises
867             n_left_segment = nm; % Attention here
868             n_right_segment = nm; % Attention here

```

```

869         else
870             n_left_segment = nf; % Attention here
871             n_right_segment = nf; % Attention here
872         end
873     end
874     B(k,1) = h/alfa_l_tang + h/alfa_r_tang + B(k-1,1);
875     A(k,1) = (B(k,1) + B(k-1,1))*h/2 + A(k-1,1); % Trapezoid
876     P(k,1) = h/sin(atan(alfa_l_tang)) + h/sin(atan(alfa_r_tang)) + P(k-1,1);
877     Rh(k,1) = A(k,1)/P(k,1);
878     Phi(k,1) = A(k,1)*Rh(k,1)^(2/3);
879     int_n_p = n_left_segment^(3/2)*h/sin(atan(alfa_l_tang)) + ...
880             n_right_segment^(3/2)*h/sin(atan(alfa_r_tang)) + int_n_p;
881     % Representative Roughness Coefficient
882     if flag_method == 1
883         n_med(k,1) = (int_n_p/P(k,1))^(2/3);
884     else
885         if y_table(k,1) > ym
886             yf = max(y_table(k,1) - ym,0); % Overbank depth
887             af = max(A(k,1) - (am + bm*yf),0); % Overbank flow area
888             pf = max(P(k,1) - pm,0); % Floodplain perimeter (m)
889             pm_star = max(pm + n_fp*yf,0);
890             am_star = max(am + bm*yf,0);
891             n_med(k,1) = (Phi(k,1))/(1/nf*af*(af/pf)^(2/3) + ...
892                             1/nm*am_star*(am_star/pm_star)^(2/3));
893         else
894             yf = 0; % Overbank depth
895             af = 0; % Overbank flow area
896             pf = 0; % Floodplain perimeter (m)
897             pm_star = 0;
898             am_star = 0;
899             n_med(k,1) = nm;
900         end
901     end
902     K_c(k,1) = 1/n_med(k,1)*Phi(k,1);
903 end
904
905 if j == (n_points) % final point
906     % Final point - make sure you have the exact surveyed point at the end
907     h_ = y_moving_end - y_table(k-1,1);
908     y_table(k,1) = y_table(k-1,1) + h_;
909     % Roughness
910     if y_table(k,1) ≤ ym % Inside of the channel
911         if flag_method == 1
912             n_left_segment = n_left_unsorted(pos_left,1);
913             n_right_segment = n_right_unsorted(pos_right,1);
914         else
915             if (abs(y_left_unsorted(pos_left) - ym) ≤ noise*length(y_left_unsorted))
916                 n_left_segment = nf; % Attention here
917             else
918                 n_left_segment = nm; % Attention here
919             end
920             if (abs(y_right_unsorted(pos_right) - ym) ≤ noise*length(y_right_unsorted))
921                 n_right_segment = nf; % Attention here
922             else
923                 n_right_segment = nm; % Attention here
924             end
925         end
926     else % Overbanks
927         if flag_method == 1
928             n_left_segment = n_left_unsorted(pos_left,1);
929             n_right_segment = n_right_unsorted(pos_right,1);
930         elseif y_left_unsorted(pos_left) - ym < noise*length(y_left_unsorted) % Check Noises
931             n_left_segment = nm; % Attention here
932             n_right_segment = nm; % Attention here
933         else
934             n_left_segment = nf; % Attention here
935             n_right_segment = nf; % Attention here
936         end
937     end
938     B(k,1) = h_/alfa_l_tang + h_/alfa_r_tang + B(k-1,1) + B_extra;
939     A(k,1) = (B(k,1) + B(k-1,1))*h_/2 + A(k-1,1); % Trapezoid
940     P(k,1) = h_/sin(atan(alfa_l_tang)) + h_/sin(atan(alfa_r_tang)) + P(k-1,1) + P_extra;
941     Rh(k,1) = A(k,1)/P(k,1);
942     Phi(k,1) = A(k,1)*Rh(k,1)^(2/3);
943     int_n_p = n_left_segment^(3/2)*h/sin(atan(alfa_l_tang)) + ...
944             n_right_segment^(3/2)*h/sin(atan(alfa_r_tang)) + int_n_p;
945     % Representative Roughness Coefficient

```

```

943         if flag_method == 1
944             n_med(k,1) = (int_n_p/P(k,1))^(2/3);
945         else
946             if y_table(k,1) ≥ ym
947                 yf = max(y_table(k,1) - ym,0); % Overbank depth
948                 af = max(A(k,1) - (am + bm*yf),0); % Overbank flow area
949                 pf = max(P(k,1) - pm,0); % Floodplain perimeter (m)
950                 pm_star = max(pm + n_fp*yf,0);
951                 am_star = max(am + bm*yf,0);
952                 n_med(k,1) = (Phi(k,1))/(1/nf*af*(af/pf)^(2/3) + ...
953                     1/nm*am_star*(am_star/pm_star)^(2/3));
954             else
955                 yf = 0; % Overbank depth
956                 af = 0; % Overbank flow area
957                 pf = 0; % Floodplain perimeter (m)
958                 pm_star = 0;
959                 am_star = 0;
960                 n_med(k,1) = nm;
961             end
962         end
963         K_c(k,1) = 1/n_med(k,1)*Phi(k,1);
964     end
965     % Previous Positions
966     pos_left_previous = pos_left;
967     pos_right_previous = pos_right;
968 end
969 % Checking i
970 if round(y_table(end),3) == round(max_y,3) % Stop de algorithm
971     i = big_n;
972 end
973 end
974
975 % Centroid Coordinates
976 int_a_y = 0; % Integral of A(y)dy
977 for i = 1:(length(A))
978     if i == 1
979         y_bar(i,1) = 0;
980         int_a_y(i,1) = 0;
981     else
982         int_a_y(i,1) = (A(i) - A(i-1))*(y_table(i) + y_table(i-1))/2 + int_a_y(i-1);
983         y_bar(i,1) = int_a_y(i,1)/A(i,1);
984     end
985 end
986
987 % Flow Discharge Calculations
988 Q = K_c*sqrt(s0);
989
990 % Velocity
991 v = Q./A; % m/s
992
993 % Beta - Boussinesq factor
994 kappa = 0.41;
995 g = 9.81; % m/s2
996 Beta = (1 + (g*n_med.^2)./(Rh.^(1/3)*kappa^2));
997
998 %% Plotting Results
999 % Plotting Channel
1000 if flag_plot_HP == 1
1001     close all
1002     subplot(1,2,1)
1003     set(gcf,'units','inches','position',[4,4,6.5,4])
1004     mark_size = 5;
1005     plot(x_absolute,y,'linewidth',2,'color','black')
1006     xlabel('x ($m$)','Interpreter','latex');
1007     ylabel('y ($m$)','Interpreter','latex');
1008     xlim([min(x_absolute) max(x_absolute)])
1009     grid on
1010     hold on
1011     scatter(x_absolute,y,'black')
1012     subplot(1,2,2)
1013     n_med(1,1) = inf;
1014     plot(n_med(2:end,1),y_table(2:end,1),'linewidth',2,'color','black')
1015     xlabel('Manning`s coefficient (SI)','Interpreter','latex');
1016     ylabel('y ($m$)','Interpreter','latex');
1017     xlim([0.9*min(n_med) 1.1*max(n_med(~isinf(n_med)))])
1018     grid on

```

```

1019 exportgraphics(gcf, 'Cross_Section.pdf', 'ContentType', 'vector')
1020
1021
1022 subplot(2,4,1)
1023 set(gcf, 'units', 'inches', 'position', [4,2,7.5,5])
1024 sz = 5;
1025 c = linspace(1,sz,length(y_table));
1026 scatter(A,y_table,sz,c,'filled')
1027 grid on
1028 grid on
1029 xlabel('Area ($m^2$)', 'Interpreter', 'latex');
1030 ylabel('y ($m$)', 'Interpreter', 'latex');
1031 % xlim([0 4])
1032 subplot(2,4,2)
1033 grid on
1034 scatter(P,y_table,sz,c,'filled')
1035 grid on
1036 xlabel('Perimeter ($m$)', 'Interpreter', 'latex');
1037 ylabel('y ($m$)', 'Interpreter', 'latex');
1038 % xlim([0 4])
1039 subplot(2,4,3)
1040 grid on
1041 scatter(Rh,y_table,sz,c,'filled')
1042 grid on
1043 xlabel('Hydraulic Radius ($m$)', 'Interpreter', 'latex');
1044 ylabel('y ($m$)', 'Interpreter', 'latex');
1045 % xlim([0 4])
1046 subplot(2,4,4)
1047 grid on
1048 scatter(B,y_table,sz,c,'filled')
1049 grid on
1050 xlabel('Top width ($m$)', 'Interpreter', 'latex');
1051 ylabel('y ($m$)', 'Interpreter', 'latex');
1052 subplot(2,4,5)
1053 grid on
1054 scatter(K_c,y_table,sz,c,'filled')
1055 grid on
1056 xlabel('Conveyance ($m^3/s$)', 'Interpreter', 'latex');
1057 ylabel('y ($m$)', 'Interpreter', 'latex');
1058 subplot(2,4,6)
1059 sz = 5;
1060 c = linspace(1,sz,length(y_table));
1061 scatter(Phi,y_table,sz,c,'filled')
1062 grid on
1063 xlabel('$\Phi$ ($m^{5/3}$)', 'Interpreter', 'latex');
1064 ylabel('y ($m$)', 'Interpreter', 'latex');
1065 subplot(2,4,7)
1066 scatter(y_bar,y_table,sz,c,'filled')
1067 grid on
1068 xlabel('$\bar{y}$ (m)', 'Interpreter', 'latex');
1069 ylabel('y ($m$)', 'Interpreter', 'latex');
1070 subplot(2,4,8)
1071 scatter(Q,y_table,sz,c,'filled')
1072 grid on
1073 xlabel('Flow discharge ($m^3/s$)', 'Interpreter', 'latex');
1074 ylabel('y ($m$)', 'Interpreter', 'latex');
1075 exportgraphics(gcf, 'Hydraulic_Properties.pdf', 'ContentType', 'vector')
1076 toc
1077
1078 % Rating Curve
1079 close all
1080 subplot(3,1,1)
1081 set(gcf, 'units', 'inches', 'position', [4,4,6.5,4])
1082 mark_size = 5;
1083 plot(x_absolute,y, 'linewidth', 2, 'color', 'black')
1084 xlabel('x ($m$)', 'Interpreter', 'latex');
1085 ylabel('y ($m$)', 'Interpreter', 'latex');
1086 xlim([min(x_absolute) max(x_absolute)])
1087 grid on
1088 subplot(3,1,2)
1089 scatter(Q,y_table,sz,c,'filled')
1090 xlabel('Flow discharge ($m^3/s$)', 'Interpreter', 'latex');
1091 ylabel('y ($m$)', 'Interpreter', 'latex');
1092 grid on
1093 box on
1094 % Velocity
1095 subplot(3,1,3)

```

```

1096 scatter(Q./A,y_table,sz,c,'filled')
1097 xlabel('Velocity ($m/s$)','Interpreter','latex');
1098 ylabel('y ($m$)','Interpreter','latex');
1099 grid on
1100 box on
1101 exportgraphics(gcf,'Rating Curve.pdf','ContentType','vector')
1102
1103 % Plotting Normalized Values
1104 set(gcf,'units','inches','position',[4,2,8,4])
1105 subplot(1,5,1)
1106 scatter(Q/max(Q),y_table/max(y_table),sz,c,'filled')
1107 xlabel('$Q/Q_{p}$','Interpreter','latex');
1108 ylabel('$y/y_{\{max\}}$','$','Interpreter','latex');
1109 title(['$Q_p$ (m^3/s) = $ ',num2str(round(max(Q),2))'],'interpreter','latex')
1110 axis equal
1111 grid on
1112 xlim([0 1]); ylim([0 1]);
1113 subplot(1,5,2)
1114 scatter(A/max(A),y_table/max(y_table),sz,c,'filled')
1115 xlabel('$A/A_{\{max\}}$','$','Interpreter','latex');
1116 ylabel('$y/y_{\{max\}}$','$','Interpreter','latex');
1117 title(['$A_{\{max\}}$ (m^2) = $ ',num2str(round(max(A),2))'],'interpreter','latex')
1118 axis equal
1119 grid on
1120 xlim([0 1]); ylim([0 1]);
1121 subplot(1,5,3)
1122 scatter(Phi/max(Phi),y_table/max(y_table),sz,c,'filled')
1123 xlabel('$\Phi/\Phi_{\{max\}}$','$','Interpreter','latex');
1124 ylabel('$y/y_{\{max\}}$','$','Interpreter','latex');
1125 title(['$\Phi_{\{max\}}$ (m^2) = $ ',num2str(round(max(Phi),2))'],'interpreter','latex')
1126 axis equal
1127 grid on
1128 xlim([0 1]); ylim([0 1]);
1129 subplot(1,5,4)
1130 scatter(K_c/max(K_c),y_table/max(y_table),sz,c,'filled')
1131 xlabel('$K_c/K_{\{c,max\}}$','$','Interpreter','latex');
1132 ylabel('$y/y_{\{max\}}$','$','Interpreter','latex');
1133 title(['$K_{\{c,max\}}$ (m^3/s) = $ ',num2str(round(max(K_c),2))'],'interpreter','latex')
1134 axis equal
1135 grid on
1136 xlim([0 1]); ylim([0 1]);
1137 subplot(1,5,5)
1138 scatter((Q./A)/(max(Q./A)),y_table/max(y_table),sz,c,'filled')
1139 xlabel('$v/v_{\{c,max\}}$','$','Interpreter','latex');
1140 ylabel('$y/y_{\{max\}}$','$','Interpreter','latex');
1141 title(['$V_{\{max\}}$ (m/s) = $ ',num2str(round(max(Q./A),2))'],'interpreter','latex')
1142 axis equal
1143 grid on
1144 xlim([0 1]); ylim([0 1]);
1145 exportgraphics(gcf,'Normalized_Values.pdf','ContentType','vector')
1146 close all
1147
1148 end
1149 end

```

## 2) Read Input Data - SVE

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

```

1  %% ----- HyProSWE Model ----- %%
2  % Script to read input data
3  % Developer: Marcus Nobrega Gomes Junior
4  % 5/1/2023
5  % Goal: Solution of 1-D SVE for given cross-section functions of Area, Perimeter, and
6  % top Width
7  % If you have any issues, please contact me at
8  % marcusnobrega.engcivil@gmail.com
9
10 % ----- Please, don't change anything below ----- %
11
12 %% Read Input Data %%
13 data = readtable('HyProSWE_Input_Data.xlsx','Sheet','Input_Data');

```

```

14 b = 0; Z1 = 0; Z2 = 0; a = 0; D = 0;
15
16
17 % General Data
18 general_data = table2array(data(1:16,2));
19 L = general_data(1,1);
20 Nx = general_data(2,1);
21 e1 = general_data(3,1);
22 g = general_data(4,1);
23 nm = general_data(5,1);
24 I0 = general_data(6,1);
25 tf = general_data(7,1);
26 dt = general_data(8,1);
27 animation_time = general_data(9,1);
28 s_outlet = general_data(10,1);
29 dh = general_data(11,1);
30 alpha = general_data(12,1);
31 dtmin = general_data(13,1);
32 dtmax = general_data(14,1);
33
34
35 % Flags
36 flags = table2array(data(19:29,2));
37 flag_hydrograph = flags(1,1);
38 flag_outlet = flags(2,1);
39 flag_friction = flags(3,1);
40 flag_section = flags(4,1);
41 flag_stage_hydrograph = flags(5,1);
42 flag_nash = flags(6,1);
43 flag_slope = flags(7,1);
44 flag_elevation = flags(8,1);
45 flag_output = flags(9,1);
46 flag_plot_HP = flags(10,1);
47 flag_elapsed_time = flags(11,1);
48 if flag_elapsed_time ~= 1
49     Date_Begin = general_data(15,1);
50     Date_Begin = datetime(datestr(Date_Begin+datenum('30-Dec-1899')));
51     Date_End = general_data(16,1);
52     Date_End = datetime(datestr(Date_End+datenum('30-Dec-1899')));
53 end
54
55 if flag_nash == 1
56     nash_data = table2array(data(1:4,5));
57     % Hydrograph
58     Tp = nash_data(1,1);
59     Qb = nash_data(2,1);
60     Beta = nash_data(3,1);
61     Qp = nash_data(4,1);
62 else
63     % Input Hydrograph
64     input_hydrograph_data = table2array(data(8:end,4:5));
65     time_ = input_hydrograph_data(1:end,1);
66     Qe1_ = input_hydrograph_data(1:end,2);
67     Qe1 = zeros(size(Qe1_,1) - sum(isnan(Qe1_)),1);
68     time = zeros(size(time_,1) - sum(isnan(time_)),1);
69     % Taking away nans
70     for i = 1:length(Qe1)
71         if isnan(Qe1_(i)) || isnan(time_(i))
72             break
73         else
74             Qe1(i,1) = Qe1_(i,1);
75             time(i,1) = time_(i,1);
76         end
77     end
78     clear Qe1_ time_
79 end
80
81 if flag_stage_hydrograph ~= 0
82     % Stage Hydrograph
83     input_stage_data = table2array(data(8:end,7:8));
84     time_stage_ = input_stage_data(1:end,1);
85     he1_ = data(1:end,2);
86     he1 = zeros(size(he1_,1) - sum(isnan(he1_)),1);
87     time_stage = zeros(size(time_stage_,1) - sum(isnan(time_stage_)),1);
88     % Taking away nans
89     for i = 1:length(he1)
90         if isnan(he1_(i)) || isnan(time_stage_(i))

```

```

91         break
92     else
93         hel(i,1) = hel_(i,1);
94         time_stage(i,1) = time_stage_(i,1);
95     end
96 end
97 clear Qel_ time_stage_
98 end
99
100 if flag_slope ≠ 0
101     % Slope
102     input_slope_data = table2array(data(8:end,7:8));
103     station = input_slope_data(1:end,1);
104     bottom_slope = input_slope_data(2:end,2);
105     slopes_not_nan = zeros(size(bottom_slope,1) - sum(isnan(bottom_slope)),1);
106     station_index = zeros(size(station,1) - sum(isnan(station)),1);
107     % Taking away nans
108     for i = 1:length(station_index)
109         if isnan(bottom_slope(i)) || isnan(station(i))
110             break
111         else
112             slopes_not_nan(i,1) = bottom_slope(i,1);
113             station_index(i,1) = station(i,1);
114         end
115     end
116     clear station_index station bottom_slope
117     bottom_slope = slopes_not_nan;
118 end
119
120 if flag_elevation ≠ 0
121     % Slope
122     input_slope_data = table2array(data(8:end,7:8));
123     station = input_slope_data(1:end,1);
124     elevation_cell = table2array(data(8:end,7:8));
125     inv_el_ = zeros(size(elevation_cell,1) - sum(isnan(elevation_cell)),1);
126     station_index = zeros(size(station,1) - sum(isnan(station)),1);
127     % Taking away nans
128     for i = 1:length(station_index)
129         if isnan(elevation_cell(i)) || isnan(station(i))
130             break
131         else
132             inv_el_(i,1) = elevation_cell(i,1);
133             station_index(i,1) = station(i,1);
134         end
135     end
136     inv_el = inv_el_; % Invert Elevation
137     clear station_index station elevation_cell inv_el_
138 end
139
140 % Outlet
141 if flag_outlet ≠ 1
142     input_slope_wave = table2array(data(8:5,8));
143     h_0_wave = input_slope_wave(1,1);
144     H_0_wave = input_slope_wave(2,1);
145     L_wave = input_slope_wave(3,1);
146     T_wave = input_slope_wave(4,1);
147     x_wave = input_slope_wave(5,1);
148 end
149
150 % Section
151 if flag_section == 1
152     input_slope_trapezoid = table2array(data(1:3,11));
153     b = input_slope_trapezoid(1,1);
154     Z1 = input_slope_trapezoid(2,2);
155     Z2 = input_slope_trapezoid(3,3);
156 elseif flag_section == 2
157     input_slope_circular = table2array(data(1,14));
158     D = input_slope_circular(1,1);
159 elseif flag_section == 3
160     input_slope_parabolic = table2array(data(3,14));
161     a = data(1,1);
162 else
163     % Read HP estimator data
164     [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr, x_cross, ...
165      y_cross,s0] = HP_estimator(flag_plot_HP,dh);
166     irr_table = [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];

```

```

167 % Some Boundary Conditions
168 % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
169 % [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
170 irr_table(1,6) = irr_table(2,6); irr_table(1,7) = 0; irr_table(1,8) = 0;
171 % Second Line
172 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;
173 % z = irr_table;
174 % second = 0*z(2,:);
175 % second(1,1) = 0.5*10^-3; second(1,6) = z(1,6); second(1,10) = z(1,10)/2;
176 % z = [z(1,:) ; second; z(2:end,:)];
177 % irr_table = z;
178 end
179
180
181 % Contrainst at observed flow
182 if flag_hydrograph == 1
183     if max(time) ≠ tf
184         z = round(tf - max(time),0);
185         for i = 1:z
186             Qel(end + 1,1) = 0;
187             time(end+1,1) = time(end,1) + 1;
188         end
189     end
190 end
191
192
193 % Contrainst at stage hydrograph
194 if flag_stage_hydrograph == 1
195     if max(time_stage) ≠ tf
196         z = round(tf - max(time_stage),0);
197         for i = 1:z
198             hel(end + 1,1) = 0;
199             time_stage(end+1,1) = time_stage(end,1) + 1;
200         end
201     end
202 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.

```

1  %%% ----- HyProSWE Model ----- %%%
2  % Developer: Marcus Nobrega Gomes Junior
3  % 5/1/2023
4  % Goal: Solution of 1-D SVE for given cross-section functions of Area, Perimeter, and
5  % top Width
6  % If you have any issues, please contact me at
7  % marcusnobrega.engcivil@gmail.com
8
9
10 % ----- All Rights Reserved ----- %
11
12 % 1.0 - Pre-Processing
13
14 clear all
15 clc
16 warning('off') % Deactivate Warnings
17
18 % Reading the Input Data
19 Read_Input_Data % Here we read the .xlsx input data file. Please don't change the name of this file.
20
21 % Checking if at least one boundary condition is considered
22 if flag_hydrograph ≠ 1 && flag_nash ≠ 1 && flag_stage_hydrograph ≠ 1 && flag_outlet ≠ 0
23     error('Please enter at least 1 internal boundary condition.')
24 end
25
26 % Checking if there is conflicting boundary conditions
27 if flag_hydrograph == 1 && flag_nash == 1
28     error('Please choose either an observed inflow hydrograph entered in a tabular format or a ...
        nash-type hydrograph.')

```



```

29 end
30
31 % Checking if there is conflicting cross section
32 if flag_section > 4
33     error('Please, enter a the index indicating which type of cross-section is being simulated. ...
           Read the instruction in the .xlsx file')
34 end
35
36 % Checking if there is conflicting cross section
37 if flag_stage_hydrograph == 1 && flag_hydrograph == 1
38     error('Please, the inlet can only have either a stage hydrograph or a flow hydrograph')
39 end
40 %% 2.0 - Initial Boundary Conditions
41 % ----- Inflow Hydrograph ----- %
42 if flag_hydrograph == 1
43     % We already read the hydrograph in Read_Input_Data file
44 elseif flag_nash == 1
45     % 2nd option - Model the hydrograph using a nash function
46     %%  $Q(t) = Q_b(t) + (Q_p(t) - Q_b(t)) \cdot (t/TP \cdot \exp(1 - t/TP))^{\text{Beta}}$ 
47     Inflow_Hydrograph_fun = @(t) (Qb + (Qp - Qb) .* (t/(Tp*3600) .* exp(1 - (t)/(Tp*3600))) .^ Beta);
48     time = [0 tf]'; % begin and end in min
49 else
50     time = [0 tf]'; % begin and end in min
51 end
52
53 if flag_stage_hydrograph == 1
54     Stage_Hydrograph = hel;
55 end
56
57 % ----- Outlet Boundary Condition ----- %
58 % flag_outlet = 1; % 1 = normal depth, flag_outlet > 1, stage hydrograph
59 % at the outlet following a wave function
60
61 if flag_outlet ≠ 1
62     %% Wave Properties for Outlet Stage Hydrograph
63     % x_wave = L_wave/1; % point position in wave x direction;
64     k_wave = 2*pi/L_wave;
65     sigma_wave = 2*pi./(T_wave*3600);
66     h_wave_function = @(t) (h_0_wave + H_0_wave/2 .* cos(k_wave.*x_wave - sigma_wave*t));
67 end
68
69 % Time Calculations
70 time = time*60; % time in seconds
71 [al,-] = size(time); % Length of time
72 tt_h = time(al,1); % End of hydrograph in seconds
73 tt = min(tf*60,tt_h); % End of simulation in seconds
74 Nt = tt/dt; % Number of time-steps in the simulations
75
76 % Recording Times
77 time_records_min = animation_time; % Minutes
78 time_store = [0:time_records_min*60:tt]; % number of steps necessary to reach the recording vector
79 Nat = time_records_min*60/dt; % Number of time-steps within an animation time
80 tint = linspace(0,tt,Nt); % Generate Nt points within 0 and tt(sec)
81
82 time_save = zeros(length(time_store),1); % Time
83 Flow_Area = zeros(length(time_store),Nx); % Flow area
84 Discharge = zeros(length(time_store),Nx); % Flow discharge
85 Depth = zeros(length(time_store),Nx); % Depth
86 Velocity = zeros(length(time_store),Nx); % Velocity
87 Froude = zeros(length(time_store),Nx); % Froude
88 Courant = zeros(length(time_store),Nx); % Courant number
89
90 if flag_hydrograph == 1
91     Qelint = max(interp1(time,Qel(:,1),tint,'pchip'),0); % Interpolated flow
92     % Assuming no negative flows
93     Qelint = Qelint';
94 elseif flag_nash == 1
95     Qelint = Inflow_Hydrograph_fun(tint)';
96 else
97     tiny_flow = 1e-8;
98     Qelint = tiny_flow*ones(1,length(tint)); % No inflow hydrograph
99 end
100
101 if flag_stage_hydrograph == 1
102     helint = max(interp1(time_stage*60,hel(:,1),tint,'pchip'),0); % Interpolated depth
103     helint = helint';
104 end

```

```

105
106 %% 3.0 - Pre-Allocation of Arrays
107
108 % Channel Discretization
109 dx = L/(Nx-1); % Channel discretization length in meters
110
111 % Friction Data
112 flag_friction = 1; % If 1, Manning, otherwise DW
113
114 % Manning
115 nm = repmat(nm,Nx,1); % Bottom slope in m/m for all reaches
116
117 % Pre-allocating arrays
118 % Matrices
119 x = (0:dx:L)'; % x discretization in meters
120 y = zeros(Nx,2);
121 q1 = zeros(Nx,2);
122 q2 = zeros(Nx,2);
123 f1 = zeros(Nx,2);
124 f2 = zeros(Nx,2);
125 J2 = zeros(Nx,2);
126 q1_back = q1(1:(Nx-2),2);
127 q1_forward = zeros(Nx-2,2);
128 q2_back = zeros(Nx-2,2);
129 q2_forward = zeros(Nx-2,2);
130 f1_back = zeros(Nx-2,2);
131 f1_forward = zeros(Nx-2,2);
132 f2_back = zeros(Nx-2,2);
133 f2_forward = zeros(Nx-2,2);
134 J2_back = zeros(Nx-2,2);
135 J2_forward = zeros(Nx-2,2);
136 ybar = zeros(Nx,2);
137 Fr = zeros(Nx,2);
138 Cn = zeros(Nx,2);
139
140 %% 4.0 Channel Data (Cross Section)
141 % Slope
142 if flag_slope ~= 1 && flag_elevation ~= 1
143     I0 = repmat(I0,(Nx-1),1); % Bottom slope in m/m for all reaches. This is only valid for ...
        closed-form sections
144 elseif flag_slope == 1
145     I0 = bottom_slope; % From read input data script
146 end
147
148 if flag_elevation == 1 % We are entering the elevations of each node
149     for i = 1:(Nx-1)
150         if i+1 > length(inv_el)
151             error('Please make sure to add enough invert elevation data.')
152         end
153         I0(i,1) = (inv_el(i+1) - inv_el(i))/dx;
154     end
155 end
156
157 % Outlet Slope
158 if flag_outlet == 1
159     I0(end+1) = s_outlet;
160 else
161     I0(end+1) = s_outlet; % Let's assume a boundary condition
162 end
163
164 % Intializing channel data
165 sm = 1e-12; % Small number
166 b = sm + b; Z1 = sm + Z1; Z2 = sm + Z2; D = sm + D; a = sm + a;
167 % flag_section - If 1, trapezoid, if 2, circular, if 3, paraboloid, if 4 - Irregular
168
169 % Invert Elevations
170 if flag_elevation ~= 1
171     inv_el = zeros(Nx,1);
172     for i = 1:Nx
173         if i == 1
174             inv_el(i) = el;
175         else
176             inv_el(i) = inv_el(i-1) - (I0(i-1)*dx);
177         end
178     end
179 end
180

```

```

181 % ----- Geometrical Functions for all Cros-Sections ----- %
182 syms b_ y_ Z1_ Z2_ Q_ I0_ D_ a_
183 dim_all = 1e-6*(y_ + Z1_ + Z2_ + a_ + D_ + b_);
184 if flag_section == 1
185     B = b_ + y_*(Z1_ + Z2_) + dim_all; % user defined function (top width)
186     B_function = matlabFunction(B);
187     P = b_ + y_*(sqrt(1 + Z1_^2) + sqrt(1 + Z2_^2)) + dim_all; % Perimeter Function % user defined ...
        function
188     P_function = matlabFunction(P);
189     A = (2*b_ + y_*(Z1_ + Z2_))*y_/2 + dim_all; % Area function % user defined function
190     A_function = matlabFunction(A); % Function describing the area in terms of y
191     centroid = y_ - int(A,y_)./A + dim_all; % 1st order momentum
192     ybar_function = matlabFunction(centroid); % Function describing ybar in terms of y
193 end
194 if flag_section == 2
195     % Circular Section
196     theta = 2*acos(1 - 2.*y_/D_) + dim_all;
197     B = D_.*sin(theta/2) ; % top width
198     B_function = matlabFunction(B);
199     P = theta.*D_/2 ; % perimeter
200     P_function = matlabFunction(P);
201     A = D_.^2/8.*(theta - sin(theta)) ; % area
202     A_function = matlabFunction(A); % Function describing the area in terms of y
203     Ybar = y_ - (D_.*(- cos(theta/2)/2 + 2.*sin(theta/2).^3./(3*(theta - sin(theta))))); % Very ...
        much attention here
204     ybar_function = matlabFunction(Ybar);
205 end
206
207 if flag_section == 3
208     % Parabolic Section
209     % Area Function
210     A = 4.*(y_.^3/2)./(3*sqrt(a_)) + dim_all; % m2
211     A_function = matlabFunction(A); % Function describing the area in terms of y
212     % Top Width
213     B = 3/2.*A./y_ + dim_all; % m
214     B_function = matlabFunction(B);
215     % Hydraulic Perimeter
216     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_))));
217     P_function = matlabFunction(P);
218     Y_bar = y_ - 2/5*y_ + dim_all;
219     ybar_function = matlabFunction(Y_bar);
220 end
221
222 if flag_section ~= 4
223     % Hydraulic Radius %%%%%%%%%
224     Rh = A/P; % Hydraulic Radius Function
225     Rh_function = matlabFunction(Rh); % Function describing the hydraulic radius in terms of y
226 end
227
228 % Vlookup Function
229 Vlookup_eq = @(data,col1,vall,col2) data((find(data(:,col1)==vall,1,'first')),col2); %Vlookup ...
        function as Excel
230 Vlookup_l = @(data,col1,vall,col2) data((find(data(:,col1)<vall,1,'last')),col2); %Vlookup function ...
        as Excel]
231 Vlookup_g = @(data,col1,vall,col2) data((find(data(:,col1)>vall,1,'first')),col2); %Vlookup ...
        function as Excel
232 fv = 1 + 1e-4; % Factor to avoid fails in vlookup function
233
234 % Minimum Value
235 min_depth = 0.02; % m
236 min_area = Vlookup_l(irr_table,1,min_depth*fv,2);
237
238 % Initial Guess
239 if flag_section == 1
240     y0_guess = 1;
241 elseif flag_section == 2
242     y0_guess = D/2;
243 elseif flag_section == 3
244     y0_guess = 1;
245 end
246
247 %% 5.0- Initial Values for Simulation
248 Q0 = Qelint(1,1); % Flow at inlet section at time 0
249 if flag_stage_hydrograph == 1
250     h0 = helint(1,1); % Water depth at x = 0 at time = 0
251 end

```

```

252 if flag_friction == 1
253     if flag_section ≠ 4
254         if Q0 == 0
255             Q0 = sm; % Numerical Constraint
256         end
257         y0 = uniformeM(nm,Q0,b,Z1,Z2,a,D,I0,P,A,y0_guess) ; % normal depth using manning equation
258         % Stage_Hydrograph Boundary Condition
259         if flag_stage_hydrograph == 1
260             y0(1,1) = h0;
261         end
262         % More Initial Boundary Conditions for Area, Velocity, Perimeter and Rh
263         A0 = A_function(D,Z1,Z2,a,b,y0); % Cross section area in m2
264         u0 = (Q0./A0)'; % Initial velocity in m/s
265         P0 = P_function(D,Z1,Z2,a,b,y0); % Hydraulic perimeter in m
266         Rh0 = A0./P0; % Hydraulic radius at time 0
267         % Boundary Conditions
268         y(:,1) = y0; % all sub-reaches with y0 at the beginning
269         q1(:,1) = A0; % all sub-reaches with same area A0 at the beginning
270         q2(:,1) = Q0; % Assuming permanent conditions at the beginning
271         f1(:,1) = q2(:,1);
272         % f2 depends on ybar
273     else % Irregular Cross-Section
274         % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
275         % [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
276
277         if max(irr_table(:,10)) == 0 % No outflow and S = 0
278             % Here we are modeling a channel with no slope
279             % We search Everything Using the Depth instead of the Flow
280             coll = 1; % Searching with the Col of Y
281             % Stage_Hydrograph Boundary Condition
282             if flag_stage_hydrograph == 1
283                 y0(1,1) = max(h0,irr_table(2,1));
284                 s_v = y0; % Searching Variable
285             else
286                 error('Please, add a minimum slope value or enter a stage-hydrograph boundary ...
287                     condition.')

```

```

328         % We search Everything Using the Depth instead of the Flow
329         coll = 1; % Searching with the Col of Flow
330         q1(Nx,1) = Vlookup_g(irr_table,coll,y(Nx,2)*fv,2);
331     end
332 end
333
334 % Hydraulic Radius
335 if flag_section ≠ 4
336     Rh_outlet = Rh_function(D,Z1,Z2,a,b,y(Nx,2));
337 else
338     for mm = 1:(length(irr_table(1,:))-1)
339         interp_base = q1(Nx,1); % Value that will be used for interpolation (area)
340         area_smaller = Vlookup_l(irr_table,2,interp_base,2); % Smaller values
341         if isempty(area_smaller)
342             area_smaller = 0;
343         end
344         area_larger = Vlookup_g(irr_table,2,interp_base,2); % Larger values
345         coll = 2; % Interpolating from area values
346         if interp_base ≤ min_area
347             var_outlet(mm,1,1) = irr_table(2,mm); % Smaller values
348         else
349             var_outlet(mm,1,1) = Vlookup_l(irr_table,coll,interp_base,mm); % Smaller values
350         end
351         var_outlet(mm,1,2) = Vlookup_g(irr_table,coll,interp_base,mm); % Larger values
352         alfa_var_outlet(mm,1) = sqrt((interp_base - area_smaller)/(area_larger - area_smaller));
353     end
354
355     % [y_table, A, P, Rh, y_bar, n_med, Beta, v, B, Q]
356     % [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
357     col_var = 4; % Calculating Hydraulic Radius
358     % Var* = Var(-) + alfa*(Var(+) - Var(-))
359     Rh_outlet = var_outlet(col_var,1,1) + alfa_var_outlet(col_var,1)*(var_outlet(col_var,1,2) - ...
360         var_outlet(col_var,1,1)); % Interpolated Hydraulic Radius
361     % Var* = Var(-) + alfa*(Var(+) - Var(-))
362     col_var = 6;
363     nm(end,1) = var_outlet(col_var,1,1) + alfa_var_outlet(col_var,1)*(var_outlet(col_var,1,2) - ...
364         var_outlet(col_var,1,1));
365 end
366
367 if flag_outlet == 1
368     u = (1./nm(Nx)).*Rh_outlet^(2/3)*I0(Nx)^0.5; % Normal depth at the outlet
369     flow_dir = 1;
370 else
371     wse_dif = y(Nx-1,2-1) + inv_el(Nx-1) - y(Nx,2) - inv_el(Nx); % Difference in wse
372     out_slope = abs(wse_dif)/dx; % Friction slope at the outlet as a diffusive model
373     if wse_dif < 0
374         ttt = 1;
375     end
376
377     if flag_stage_hydrograph ≠ 1 && flag_nash ≠ 1 && flag_hydrograph ≠ 1
378         % Only Outlet Tidal B.C.
379         if wse_dif > 0 && y(Nx-1,2-1) ≤ fv*1e-3
380             out_slope = 0;
381         end
382     end
383     u = (1./nm(Nx)).*Rh_outlet^(2/3)*out_slope^0.5; % Normal velocity at the outlet
384     if wse_dif > 0
385         flow_dir = 1; % Flowing towards the outlet
386     else
387         flow_dir = -1; % Flowing to inside of the channel
388     end
389 end
390 else
391     error('HyProSWE not coded for Darcy-Weisbach. Wait for the new version or change the method for ...
392         Manning.')
393 end
394
395 q2(Nx,1) = q1(Nx,1)*u*flow_dir; % Area x Velocity
396
397 %%% State Space Format %%%
398 % dq/dt + dF/dx = S, we solve for A(x,t) and Q(x,t)
399 % q = [A Q]' = [q1 q2]'
400 % F [Q (Qv + gAybar)] = [q2 (q2^2)/q1 + g.q1.ybar]' = [f1 f2]'
401 % where ybar is the centroid depth from the top
402 % S = [0 gA(I0 - If)]'
403
404 % ybar = y - int(A(y)) / A(y) from y = 0 to y = y0

```

```

402 if flag_section ≠ 4
403     ybar = ybar_function(D,Z1,Z2,a,b,y0);
404 else
405     % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
406     % [ 1, 2, 3, 4, 5, 6, 7, 8, 9,
407     % ybar = y - ybar*
408     % ybar(:,1) = Vlookup_leq(irr_table,coll,Q0*fv,1) - Vlookup_leq(irr_table,coll,Q0*fv,5);
409     ybar(:,1) = Vlookup_g(irr_table,coll,s_v*fv,5);
410 end
411 f2(:,1) = q2(:,1).*abs(q2(:,1))./q1(:,1) + g*q1(:,1).*ybar(:,1);
412 f2(isnan(f2)) = 0; % Attention Here
413
414 % Friction S = [J1 J2]' with J1 = 0 and J2 calculated as follows:
415 if flag_friction == 1
416     J2(:,1) = g*q1(:,1).*(I0(:) - q2(:,1).*abs(q2(:,1)).*nm(:)./(q1(:,1).^2.*Rh0.^(4/3))); % Manning
417 else
418     J2(:,1) = g*q1(:,1).*(I0(:) - f*q2(:,1).*abs(q2(:,1))./((q1(:,1).^2).*8*g.*Rh0));
419 end
420
421 J2(isnan(J2)) = 0; % Attention Here
422
423 % Froude Number
424 if flag_section ≠ 4
425     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);% ...
426     Froude Number
427 else
428     % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
429     % [ 1, 2, 3, 4, 5, 6, 7, 8, 9,
430     A_f_irr = Vlookup_g(irr_table,coll,s_v*fv,2)*ones(length(q1(:,1)),1);
431     B_f_irr = Vlookup_g(irr_table,coll,s_v*fv,9)*ones(length(q1(:,1)),1);
432     Fr(:,1)=abs(q2(:,1))./q1(:,1))./((g*A_f_irr./B_f_irr).^0.5);% Froude Number
433 end
434 % Courant Number
435 % Cn = c / (dx / dt), where c = v + sqrt(g.Hm), where Hm = A / B
436 if flag_section ≠ 4
437     Hm = A_function(D,Z1,Z2,a,b,y0)./B_function(D,Z1,Z2,a,b,y0);
438     Cn(:,1)=(abs(q2(:,1))./q1(:,1))+(g*Hm).^0.5)/(dx/dt);% Courant Number
439 else
440     Hm = A_f_irr./B_f_irr;
441     Cn(:,1) = (abs(q2(:,1))./q1(:,1))+(g*Hm).^0.5)/(dx/dt);
442 end
443
444 % Depth in terms of Area function
445 % let c be the area in terms of Z1,Z2,b, and y, such that A(y) = c
446 % we want to solve y for A(y) = c
447
448 syms c_
449 if flag_section ≠ 4
450     fun_solve = (A - c_); % with c = area, we solve for y.
451     options = optimoptions('fsolve','Display',...
452         'none','FunctionTolerance',1e-2,'MaxFunctionEvaluations',Nx*10);
453 end
454 if flag_section == 1
455     % We have an analytical solution for this case
456     z = solve(fun_solve,y_); % solving for y_ = y and c = A(y)
457     h_function = matlabFunction(z); % h(A) = z;
458 else
459     % Non-linear set of equations for circular pipe, we need to use fsolve
460 end
461 if flag_section ≠ 4
462     fun_solve = matlabFunction(fun_solve); % Transforming into an equation
463 end
464 %% 6.0 - Main Loop %%
465 n = 1; % initializing counter
466 x_i = 2:(Nx-1); % vector for interior sections varying from 2 to (Nx - 1)
467 tic % starts measuring time
468 % Interpolation Variables
469 if flag_section == 4
470     var_inlet = zeros((length(irr_table(1,:))-1),1,2); var_outlet = var_inlet;
471     alfa_var_inlet = zeros((length(irr_table(1,:))-1),1,1); alfa_var_outlet = alfa_var_inlet;
472     var_middle = zeros((length(irr_table(1,:))-1),length(x_i),2);
473     alfa_var_middle = zeros((length(irr_table(1,:))-1),length(x_i),1);
474 end
475
476 % Initialization of some variables
477 time_end_min = (Nt)*dt;
478 time = 0;

```

```

477 time_previous = 0;
478 time_step = dt; % sec
479 t_store_prev = 0;
480
481 while time ≤ (time_end_min) % Main loop
482     try
483         n = n + 1; % Time-step index
484         time = time + time_step; % Seconds
485         time_save_model(n) = time; % Seconds
486
487         % Model Status
488         percentage_timestepsec_maxCourant_maxh = [time/(tt)*100, time_step, max(max(Cn)), max(max(y))]
489
490         % Agregating Inflows to the New Time-step
491         if flag_hydrograph == 1 || flag_nash == 1
492             z1 = find(tint > time_previous,1,'first'); % begin of the time-step
493             z2 = find(tint ≤ time,1,'last'); % end of the time-step
494             if isempty(z1)
495                 z1 = 1;
496             end
497             if isempty(z2) || z2 < z1
498                 z2 = z1;
499             end
500             if time_step ≥ dt
501                 Q0 = mean(Qelint(z1:z2));
502             else
503                 Q0 = Qelint(z1);
504             end
505         end
506         if time > 4.08*10^3
507             ttt = 1;
508         end
509         % Agregating Stages to the New Time-step
510         if flag_stage_hydrograph == 1
511             z1 = find(tint > time_previous,1,'first'); % begin of the time-step
512             z2 = find(tint ≤ time,1,'last'); % end of the time-step
513             if isempty(z1)
514                 z1 = 1;
515             end
516             if isempty(z2) || z2 < z1
517                 z2 = z1;
518             end
519             if time_step ≥ dt
520                 h0 = mean(helint(z1:z2));
521             else
522                 h0 = helint(z1);
523             end
524         end
525
526         % Stop Program if Complex Number Occurs
527         if imag(max(Cn(:,2-1))) > 0 || imag(max(q2(:,2-1)))
528             error('Complex number possibly due to changing the regime from free flow to pressurized flow.')
529         end
530         %%%%% - Boundary Conditions - %%%%%
531         % Channel's begin (INLET)
532         if flag_stage_hydrograph == 1
533             % h0 = helint(n,1); % Water depth at x = 0 at time = time
534             if flag_section == 4
535                 if h0 > max(irr_table(:,1))
536                     error('The maximum water depth is larger than the channel height.')
537                 end
538                 q1(1,2) = Vlookup_g(irr_table,1,h0,2); % Smaller values
539             else
540                 q1(1,2) = A_function(D,Z1,Z2,a,b,h0);
541             end
542         else
543             q1(1,2) = q1(2,1); % Area at section 1 is equals area of section 2 from previous time-step
544         end
545         if flag_hydrograph == 1 || flag_nash == 1
546             % q2(1,2) = Qelint(n,1); % Flow at section 1 is the inflow hydrograph
547             q2(1,2) = Q0; % Flow at section 1 is the inflow hydrograph
548         else
549             q2(1,2) = q2(2,1); % Flow at section 1 equals flow at section 2 from previous time-step
550         end
551
552         if flag_hydrograph == 0 && flag_nash == 0 && flag_stage_hydrograph == 0 && flag_outlet == 0
553             q2(1,2) = q2(2,1); % Flow at section 1 equals flow at section 2 from previous time-step

```

```

554 %      q2(1,1) = q2(2,1);
555 end
556
557 % Interpolating All Values from Irr_table using q1 as basis
558 % Explanation: area is given in m2. P, Rh, and other variables are
559 % in m. So we have a quadratically similar triangle relationship
560 if flag_section == 4
561     for mm = 1:(length(irr_table(1,:))-1)
562         interp_base = q1(1,2); % Value that will be used for interpolation (area)
563         if interp_base ≤ min_area % Col with area = 0
564             area_smaller = 0; % Smaller values
565         else
566             area_smaller = Vlookup_l(irr_table,2,interp_base,2); % Smaller values
567         end
568         area_larger = Vlookup_g(irr_table,2,interp_base,2); % Larger values
569         coll = 2; % Interpolating from area values
570         if interp_base ≤ min_area % Col with area = 0
571             var_inlet(mm,1,1) = irr_table(2,mm); % Smaller values
572         else
573             var_inlet(mm,1,1) = Vlookup_l(irr_table,coll,interp_base,mm); % Smaller values
574         end
575         var_inlet(mm,1,2) = Vlookup_g(irr_table,coll,interp_base,mm); % Larger values
576         alfa_var_inlet(mm,1) = sqrt((interp_base - area_smaller)/(area_larger - area_smaller));
577     end
578 end
579
580 if flag_section == 1 % Trapezoid or Rectangular
581     if Z1 > 0 || Z2 > 0 % Trapezoidal channel
582         y(1,2) = max(h_function(D,Z1,Z2,a,b,q1(1,2)')); % water depth in terms of area q1
583         % In this previous function, we solve h = y in terms of A = q1 = c
584     else
585         y(1,2) = q1(1,2)/b; % water depth in terms of area q1 for rectangular channels
586     end
587 elseif flag_section > 1 % circular or paraboloid or irregular
588     y0_guess = y(1,2-1);
589     c = q1(1,2)*fv; % WEIRDO. I HAVE TO CHECK IT OUT ... ISNT IT (2-1)?
590     if flag_section ≠ 4
591         fun = @(y_) fun_solve(D,Z1,Z2,a,b,c,y_);
592         y(1,2) = fsolve(fun,y0_guess,options); % non-linear solver
593     else % Irregular section
594         % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
595         % [ 1, 2, 3, 4, 5, 6, 7, 8, 9,
596         coll = 2; % Col with A
597         col_var = 1;
598         % Var* = Var(-) + alfa*(Var(+) - Var(-))
599         y(1,2) = var_inlet(col_var,1,1) + alfa_var_inlet(col_var,1)*(var_inlet(col_var,1,2) - ...
600             var_inlet(col_var,1,1));
601         %      y(1,2) = Vlookup_leq(irr_table,coll,c,1);
602     end
603 end
604 % ybar
605 if flag_section ≠ 4
606     ybar(1,2) = ybar_function(D,Z1,Z2,a,b,y(1,2));
607     % f1 and f2
608     f1(1,2) = q2(1,2);
609     f2(1,2) = q2(1,2).*abs(q2(1,2))./q1(1,2) + g*q1(1,2).*ybar(1,2);
610     % Hydraulic Radius
611     Rh_inlet = Rh_function(D,Z1,Z2,a,b,y(1,2));
612     % Friction
613     if flag_friction == 1
614         J2(1,2) = g*q1(1,2).*(I0(1) - ...
615             q2(1,2).*abs(q2(1,2)).*nm(1).^2./(q1(1,2).^2*Rh_inlet.^(4/3))); % Manning
616     else
617         J2(1,2) = (I0(1) - f*q2(1,2).*abs(q2(1,2))./((q1(1,2).^2)*g.*Rh_inlet));
618     end
619     % Froude
620     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); %
621     Froude Number
622     % Courant
623     Hm = A_function(D,Z1,Z2,a,b,y(1,2))./B_function(D,Z1,Z2,a,b,y(1,2));
624     Cn(1,2) = (abs(q2(1,2))./q1(1,2)) + (g*Hm).^0.5 / (dx/time_step); % Courant Number
625     if isinf(Cn(1,2)) || isnan(Cn(1,2))
626         Cn(1,2) = 0;
627     end
628 else
629     % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
630     % [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

```



```

628     coll = 2; % Col with A
629     col_var = 5;
630     % Var* = Var(-) + alfa*(Var(+) - Var(-))
631     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));
632     % f1 and f2
633     f1(1,2) = q2(1,2);
634     f2(1,2) = q2(1,2).*abs(q2(1,2))./q1(1,2) + g*q1(1,2).*ybar(1,2);
635     % Hydraulic Radius
636     col_var = 4;
637     % Var* = Var(-) + alfa*(Var(+) - Var(-))
638     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));
639     % Friction
640     if flag_friction == 1
641         col_var = 6;
642         % Var* = Var(-) + alfa*(Var(+) - Var(-))
643         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));
644         if isnan(nm(1,1))
645             nm = irr_table(2,6)*ones(length(q1(:,1)),1);
646         end
647         J2(1,2) = g*c.*(I0(1) - q2(1,2).*abs(q2(1,2)).*nm(1).^2./(c.^2*Rh_inlet.^(4/3))); % Manning
648     else
649         J2(1,2) = (I0(1) - f*q2(1,2).*abs(q2(1,2))./((q1(1,2).^2)*8*g.*Rh_inlet));
650     end
651     % Froude
652     % Var* = Var(-) + alfa*(Var(+) - Var(-))
653     A_f_irr = q1(1,2);
654     col_var = 9;
655     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));
656     % B_f_irr = Vlookup_leq(irr_table,coll,c,9);
657     Fr(1,2) = abs(q2(1,2))./q1(1,2))./((g*A_f_irr./B_f_irr)^0.5); % Froude Number
658     % Courant
659     Hm = A_f_irr./B_f_irr;
660     Cn(1,2) = (abs(q2(1,2))./q1(1,2)) + (g*Hm).^0.5)/(dx/time_step); % Courant Number
661 end
662
663 %% Right side of the channel (outlet)
664 if flag_outlet == 1 % Normal Depth
665     q1(Nx,2) = q1(Nx-1,2-1); % Boundary Condition (same area)
666     % Interpolating All Values from I_rr_table using q1 as basis
667     % Explanation: area is given in m2. P, Rh, and other variables are
668     % in m. So we have a quadratically similar triangle relationship
669     if flag_section == 4
670         for mm = 1:(length(irr_table(1,:))-1)
671             interp_base = q1(Nx,2); % Value that will be used for interpolation (area)
672             if interp_base ≤ min_area % Area
673                 area_smaller = 0;
674             else
675                 area_smaller = Vlookup_l(irr_table,2,interp_base,2); % Smaller values
676             end
677             area_larger = Vlookup_g(irr_table,2,interp_base,2); % Larger values
678             coll = 2; % Interpolating from area values
679             if interp_base ≤ min_area % Area
680                 var_outlet(mm,1,1) = irr_table(2,mm); % Smaller values
681             else
682                 var_outlet(mm,1,1) = Vlookup_l(irr_table,coll,interp_base,mm); % Smaller values
683             end
684             var_outlet(mm,1,2) = Vlookup_g(irr_table,coll,interp_base,mm); % Larger values
685             alfa_var_outlet(mm,1) = sqrt((interp_base - area_smaller)/(area_larger - ...
                area_smaller));
686         end
687     end
688     if flag_section == 1
689         if Z1 > 0 || Z2 > 0
690             y(Nx,2) = max(h_function(D,Z1,Z2,a,b,q1(Nx,2)')); % water depth in terms of area q1
691         else
692             y(Nx,2) = q1(Nx,2)/b; % water depth in terms of area q1 for rectangular channels
693         end
694     elseif flag_section ≥ 2 % circular or paraboloid or irregular
695         % If we do not have a stage-hydrograph boundary condition
696         y0_guess = y(Nx,2-1);
697         if flag_section ≠ 4
698             fun = @(y_) fun_solve(D,Z1,Z2,a,b,c,y_);
699             y(Nx,2) = fsolve(fun,y0_guess,options); % non-linear solver

```

```

700     else
701         % [y_table, A, P, Rh, y_bar, n_med, Beta, v, B, Q]
702         % [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
703         col_var = 1;
704         % Var* = Var(-) + alfa*(Var(+) - Var(-))
705         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));
706     end
707 end
708 else
709     % Stage Hydrograph Boundary Condition. We are modeling a tidal
710     % outlet condition
711     time_wave = time; % time in seconds
712     y(Nx,2) = h_wave_function(time_wave);
713     if flag_section ≠ 4
714         q1(Nx,2) = A_function(D,Z1,Z2,a,b,y(Nx,2));
715     else
716         % We search Everything Using the Depth instead of the Flow
717         coll = 1; % Searching with the Col of Flow
718
719
720         area_smaller = Vlookup_l(irr_table,coll,y(Nx,2)*fv,2);
721         area_greater = Vlookup_g(irr_table,coll,y(Nx,2)*fv,2);
722         y_smaller = Vlookup_l(irr_table,coll,y(Nx,2)*fv,1);
723         y_greater = Vlookup_g(irr_table,coll,y(Nx,2)*fv,1);
724
725         Δ_y = y(Nx,2) - Vlookup_l(irr_table,coll,y(Nx,2)*fv,1);
726         q1(Nx,2) = area_smaller + (area_greater - area_smaller)*(Δ_y/(y_greater - y_smaller))^2;
727     end
728     % q1(Nx,2) = q1(Nx-1,2-1)
729 end
730 % Hydraulic Radius
731 if flag_section ≠ 4
732     Rh_outlet = Rh_function(D,Z1,Z2,a,b,y(Nx,2));
733 else
734     for mm = 1:(length(irr_table(1,:))-1)
735         interp_base = q1(Nx,2); % Value that will be used for interpolation (area)
736         if interp_base ≤ min_area
737             area_smaller = 0; % Smaller values
738         else
739             area_smaller = Vlookup_l(irr_table,2,interp_base,2); % Smaller values
740         end
741         area_larger = Vlookup_g(irr_table,2,interp_base,2); % Larger values
742         coll = 2; % Interpolating from area values
743         if interp_base ≤ min_area
744             var_outlet(mm,1,1) = irr_table(2,mm);
745         else
746             var_outlet(mm,1,1) = Vlookup_l(irr_table,coll,interp_base,mm); % Smaller values
747         end
748         var_outlet(mm,1,2) = Vlookup_g(irr_table,coll,interp_base,mm); % Larger values
749         alfa_var_outlet(mm,1) = sqrt((interp_base - area_smaller)/(area_larger - area_smaller));
750     end
751     % [y_table, A, P, Rh, y_bar, n_med, Beta, v, B, Q]
752     % [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
753     col_var = 4; % Calculating Hydraulic Radius
754     % Var* = Var(-) + alfa*(Var(+) - Var(-))
755     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
756     % Var* = Var(-) + alfa*(Var(+) - Var(-))
757     col_var = 6;
758     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));
759 end
760 if flag_friction == 1
761     if flag_outlet == 1
762         u = (1./nm(Nx)).*Rh_outlet^(2/3)*I0(Nx)^0.5; % Normal depth at the outlet
763         flow_dir = 1;
764     else
765         wse_dif = y(Nx-1,2-1) + inv_el(Nx-1) - y(Nx,2) - inv_el(Nx); % Difference in wse
766         out_slope = abs(wse_dif)/dx; % Friction slope at the outlet as a diffusive model
767         if wse_dif < 0
768             ttt = 1;
769         end
770
771         u = (1./nm(Nx)).*Rh_outlet^(2/3)*out_slope^0.5; % Normal velocity at the outlet
772         if wse_dif > 0
773             flow_dir = 1; % Flowing towards the outlet

```

```

774         else
775             flow_dir = -1; % Flowing to inside of the channel
776         end
777     end
778 else
779     u = sqrt(8*g*Rh_outlet*I0(Nx)/f); % outlet velocity
780 end
781 % Outlet Flow
782 q2(Nx,2) = q1(Nx,2)*u*flow_dir; % Area x Velocity
783 if isnan(q2(Nx,2))
784     ttt = 1;
785 end
786 % Outlet Flow Under No Inflow Hydrograph & Not Enough WSE_dif
787 if flag_stage_hydrograph ~= 1 && flag_nash ~= 1 && flag_hydrograph ~= 1
788     % Only Outlet Tidal B.C.
789     if wse_dif > 0 && y(Nx-1,2-1) ≤ fv*1e-3
790         q2(Nx,2) = q1(Nx,2)*dx/(time_step); % Making sure all available depth becomes outflow ...
791         % in the outlet
792     end
793 end
794 % ybar
795 if flag_section ~= 4
796     ybar(Nx,2) = ybar_function(D,Z1,Z2,a,b,y(Nx,2));
797 else
798     % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
799     % [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
800     % ybar = y - ybar*
801     coll = 2; % A
802     % ybar(Nx,2) = Vlookup_leq(irr_table,coll,c,1) - Vlookup_leq(irr_table,coll,c,5);
803 % if q1(Nx,2) == 0
804 % ybar(Nx,2) = 0;
805 % else
806 % ybar(Nx,2) = Vlookup_l(irr_table,coll,q1(Nx,2),5);
807 % end
808 col_var = 5;
809 % Var* = Var(-) + alfa*(Var(+) - Var(-))
810 ybar(Nx,2) = var_outlet(col_var,1,1) + alfa_var_outlet(col_var,1)*(var_outlet(col_var,1,2) ...
811     - var_outlet(col_var,1,1));
812 end
813 % f1 and f2
814 f1(Nx,2) = q2(Nx,2); % f1 - Flow
815 zzz = q2(Nx,2).*abs(q2(Nx,2))./q1(Nx,2) + g*q1(Nx,2).*ybar(Nx,2); % f2 = (Qv + gAy_bar)
816 zzz(isnan(zzz)) = 0;
817 f2(Nx,2) = zzz; % f2 = (Qv + gAy_bar)
818 % J2
819 % Friction
820 if flag_friction == 1
821     J2(Nx,2) = g*q1(Nx,2).*(I0(Nx) - ...
822         q2(Nx,2).*abs(q2(Nx,2)).*nm(Nx)^2./(q1(Nx,2).^2*Rh_outlet.^(4/3))); % Manning --> ...
823         gA*(I0 - If), If = n^2*Q*abs*Q)/(Rh^(4/3)*A^2)
824 else
825     J2(Nx,2) = g*q1(Nx,2).*(I0(Nx) - f*q2(:,2).*abs(q2(Nx,2))./((q1(Nx,2).^2)*g*Rh_outlet));
826 end
827 J2(isnan(J2)) = 0; % Attention Here
828 % Froude
829 if flag_section ~= 4
830     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)))^0.5);
831     % Froude Number
832     % Courant
833     Hm = A_function(D,Z1,Z2,a,b,y(Nx,2))./B_function(D,Z1,Z2,a,b,y(Nx,2));
834     Cn(Nx,2)=(abs(q2(Nx,2)./q1(Nx,2))+(g*Hm).^0.5)/(dx/time_step); % Courant Number
835     if isnan(Cn(Nx,2)) || isinf(Cn(Nx,2))
836         Cn(Nx,2) = 0;
837     end
838 else
839     % Froude
840     % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
841     % [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
842     coll = 2; % Col with A
843     A_f_irr = c;
844     col_var = 9;
845     % Var* = Var(-) + alfa*(Var(+) - Var(-))
846     B_f_irr = var_outlet(col_var,1,1) + alfa_var_outlet(col_var,1)*(var_outlet(col_var,1,2) - ...
847         var_outlet(col_var,1,1));
848     % B_f_irr = Vlookup_l(irr_table,coll,c,9);

```

```

845     Fr(Nx,2) = abs(q2(Nx,2)./q1(Nx,2))./((g*A_f_irr./B_f_irr)^0.5);% Froude Number
846     % Courant
847     Hm = A_f_irr./B_f_irr;
848     if y(Nx,2) ≤ min_depth
849         Cn(Nx,2) = 0;
850     else
851         Cn(Nx,2)=(abs(q2(Nx,2)./q1(Nx,2))+(g*Hm).^0.5)/(dx/time_step);% Courant Number
852         if isnan(Cn(Nx,2)) || isinf(Cn(Nx,2))
853             Cn(Nx,2) = 0;
854         end
855     end
856 end
857
858 %% Main Loop for Non-Boundary Cells from 2 to (Nx - 1)
859 % vectorized calculations
860 q1_back = q1(1:(Nx-2),(2-1));
861 q1_forward = q1(3:(Nx),(2-1));
862 q2_back = q2(1:(Nx-2),(2-1));
863 q2_forward = q2(3:(Nx),(2-1));
864 f1_back = f1(1:(Nx-2),(2-1));
865 f1_forward = f1(3:(Nx),(2-1));
866 f2_back = f2(1:(Nx-2),(2-1));
867 f2_forward = f2(3:(Nx),(2-1));
868 J2_back = J2(1:(Nx-2),(2-1));
869 J2_forward = J2(3:(Nx),(2-1));
870
871 % Lax-Friedrichs Method
872 % Given a hyperbolic partial derivative system of equations described
873 % by:
874 %  $pq/pt + pF/px - S = 0$ , where p is the partial derivative, one can
875 % solve this equation by performing a forward discretization for q and a
876 % central discretization for F. Moreover,  $S = (S_{back} + S_{forward})/2$ 
877 % Expliciting the system of equations for q, it follows that:
878
879 q1(x_i,2) = 0.5.*(q1_forward + q1_back) - 0.5*time_step/dx*(f1_forward - f1_back); %% attention ...
880 % here in flforward
881 q2(x_i,2) = 0.5*(q2_forward + q2_back) - 0.5*time_step/dx*(f2_forward - f2_back) + ...
882 % 0.5*time_step*(J2_back + J2_forward);
883
884 if q1(Nx-1,2) > 0.0
885     ttt = 1;
886 end
887 % There is no such thing as a negative water depth, so we apply a
888 % constraint
889 if min(q1(x_i,2)) < 0
890     zzz = q1(x_i,2); zzz(zzz<0) = 0; q1(x_i,2) = zzz;
891     ttt = 1;
892 end
893 % Interpolating All Values from I_rr_table using q1 as basis
894 if flag_section == 4
895     for mm = 1:(length(irr_table(1,:))-1)
896         for hh = 1:length(x_i)
897             interp_base = q1(hh+1,2); % Value that will be used for interpolation (area)
898             if interp_base ≤ min_area
899                 area_smaller = 0;
900             else
901                 area_smaller = Vlookup_l(irr_table,2,interp_base,2); % Smaller values
902             end
903             area_larger = Vlookup_g(irr_table,2,interp_base,2); % Larger values
904             coll = 2; % Interpolating from area values
905             if interp_base ≤ min_area
906                 var_middle(mm,hh,1) = irr_table(2,mm); % Smaller values
907             else
908                 var_middle(mm,hh,1) = Vlookup_l(irr_table,coll,interp_base,mm); % Smaller values
909             end
910             var_middle(mm,hh,2) = Vlookup_g(irr_table,coll,interp_base,mm); % Larger values
911             alfa_var_middle(mm,hh,1) = sqrt((interp_base - area_smaller)/(area_larger - ...
912                 area_smaller));
913         end
914     end
915 end
916
917 if flag_section == 1
918     if Z1>0 || Z2>0
919         y(x_i,2) = max(h_function(D,Z1,Z2,a,b,q1(x_i,2)')); % water depth in terms of area q1
920     else
921         y(x_i,2)=q1(x_i,2)/b;
922     end
923 end

```

```

919     end
920 elseif flag_section > 1
921     y0_guess = y(x_i,2-1);
922     c = q1(x_i,2)*fv; % It has to be a line vector (area)
923     if flag_section ≠ 4
924         fun = @(y_) fun_solve(D,Z1,Z2,a,b,c,y_);
925         y(x_i,2) = fsolve(fun,y0_guess,options); % non-linear solver
926     else
927         % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
928         % [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
929         col1 = 2; % Col with A
930         for i = 1:length(x_i)
931             cc = c(i); % be careful here
932             col_var = 1;
933             % Var* = Var(-) + alfa*(Var(+) - Var(-))
934             y(i+1,2) = var_middle(col_var,i,1) + ...
935                 alfa_var_middle(col_var,i)*(var_middle(col_var,i,2) - var_middle(col_var,i,1));
936         end
937     end
938 % Hydraulic Radius
939 if flag_section ≠ 4
940     Rh_middle = Rh_function(D,Z1,Z2,a,b,y(x_i,2));
941     % ybar
942     ybar(x_i,2) = ybar_function(D,Z1,Z2,a,b,y(x_i,2));
943     % f1 and f2
944     f1(x_i,2) = q2(x_i,2);
945     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);
946     % Froude
947     Hm = A_function(D,Z1,Z2,a,b,y(x_i,2))./B_function(D,Z1,Z2,a,b,y(x_i,2));
948     Fr(x_i,2)=abs(q2(x_i,2)./q1(x_i,2))./((g*Hm).^0.5); % Froude Number
949     % Courant
950     Cn(x_i,2)=(abs(q2(x_i,2)./q1(x_i,2))+(g*Hm).^0.5)/(dx/time_step); % Courant Number
951     % Friction
952     if flag_friction == 1
953         J2(x_i,2) = g*q1(x_i,2).*(I0(x_i) - ...
954             q2(x_i,2).*abs(q2(x_i,2)).*nm(x_i).^2./(q1(x_i,2).^2.*Rh_middle.^(4/3)));
955     else
956         J2(x_i,2) = g*q1(x_i,2).*(I0(x_i) - ...
957             f*q2(x_i,2).*abs(q2(x_i,2))./((q1(x_i,2).^2)*g*Rh_midle));
958     end
959 % Stability Check
960 if max(Cn(:,2)) > 1
961     error('Please, decrease the time-step')
962 end
963 else
964     for jj = 1:length(x_i)
965         cc = c(jj); % Area
966         % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
967         % [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
968         col_var = 4;
969         % Var* = Var(-) + alfa*(Var(+) - Var(-))
970         Rh_middle(jj,1) = var_middle(col_var,jj,1) + ...
971             alfa_var_middle(col_var,jj)*(var_middle(col_var,jj,2) - var_middle(col_var,jj,1));
972         col_var = 5;
973         ybar(jj+1,2) = var_middle(col_var,jj,1) + ...
974             alfa_var_middle(col_var,jj)*(var_middle(col_var,jj,2) - var_middle(col_var,jj,1));
975         col_var = 6;
976         nm(jj+1,1) = var_middle(col_var,jj,1) + ...
977             alfa_var_middle(col_var,jj)*(var_middle(col_var,jj,2) - var_middle(col_var,jj,1));
978         % f1 and f2
979         f1(jj+1,2) = q2(jj+1,2);
980         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);
981         % Froude
982         A_f_irr = q1(jj+1,2);
983         col_var = 9;
984         B_f_irr = var_middle(col_var,jj,1) + ...
985             alfa_var_middle(col_var,jj)*(var_middle(col_var,jj,2) - var_middle(col_var,jj,1));
986         Hm = A_f_irr./B_f_irr;
987         Fr(jj+1,2) = abs(q2(jj+1,2)./q1(jj+1,2))./((g*Hm).^0.5); % Froude Number
988         % Courant
989         if y(jj+1,2) > 0.005 % 0.5 cm
990             Cn(jj+1,2) = (abs(q2(jj+1,2)./q1(jj+1,2))+(g*Hm).^0.5)/(dx/time_step); % Courant Number
991         else
992             Cn(jj+1,2) = 0;
993         end
994         if isinf(Cn(jj+1,2))

```

```

989         Cn(jj+1,2) = 0;
990     end
991     % Friction
992     if flag_friction == 1
993         J2(jj+1,2) = g*A_f_irr.*(I0(jj+1,1) - ...
994             q2(jj+1,2).*abs(q2(jj+1,2)).*nm(jj+1,1).^2./(A_f_irr.^2.*Rh_middle(jj,1)^(4/3)));
995     else
996         J2(jj+1,2) = g*q1(jj+1,2).*(I0(jj+1,2) - ...
997             f*q2(jj+1,2).*abs(q2(jj+1,2))./((q1(jj+1,2).^2)*8*g*Rh_midle(jj,1)));
998     end
999     % Stability Check
1000     if Cn(jj+1,2) > 1 && y(jj+1,2) ≥ min_depth && q1(jj+1,2) ≥ min_area
1001         error('Please, decrease the time-step')
1002     end
1003 end
1004
1005 % Constraint at dry areas
1006 % -- the idea is that dry cells have no hydraulic properties
1007 if min(q1(:,2)) ≤ min_area || min(y(:,2)) ≤ min_depth
1008     idx1 = q1(:,2) ≤ min_area; idx2 = y(:,2) ≤ min_area; idx = idx1 + idx2; % Both
1009     idx = logical([zeros(size(idx,1),1), idx]);
1010     q1(idx) = 0; q2(idx) = 0; f1(idx) = 0; f2(idx) = 0; J2(idx) = 0; y(idx) = 0;
1011     Fr(idx) = 0; Cn(idx) = 0;
1012 end
1013 % Adaptive Time-Step - Outlet not considered
1014 idx_courant = Cn(1:end-1,2) ≤ 0;
1015 zzz = Cn(1:end-1,2); zzz(idx_courant) = nan;
1016 dt_courant_1 = zzz/time_step; % Cn/time_step = dx/ (v + sqrt(Hm*g)), this the the time-step ...
1017     for Courant = 1
1018         time_step = min(alpha./dt_courant_1); % Calculated
1019         time_step = min(time_step,dtmax);
1020         time_step = max(time_step,dtmin);
1021         time_previous = time;
1022     if time_step < 1
1023         ttt = 1;
1024     end
1025
1026 % Saving hydrographs and depths with user defined recording time-step
1027 if n == 1
1028     % Do nothing, it is already solved, we just have to save the data
1029     % for the next time-step
1030     t_store = 1;
1031     time_save(1,1) = time;
1032 else
1033     t_store = find(time_store ≤ time,1,'last'); % Time that is being recorded in min
1034     if t_store > t_store_prev
1035         time_save(t_store,1) = time;
1036         Flow_Area(t_store,:) = q1(:,2); % m2
1037         Discharge(t_store,:) = q2(:,2); % m3/s
1038         Depth(t_store,:) = y(:,2); % m
1039         Velocity(t_store,:) = q2(:,2)./q1(:,1); % m/s
1040         Froude(t_store,:) = Fr(:,2);
1041         Courant(t_store,:) = Cn(:,2);
1042         t_store_prev = t_store;
1043     end
1044 end
1045 % Refreshing States
1046 % idx = y < 1e-3; q1(idx) = 0; q2(idx)
1047
1048 q1(:,1) = q1(:,2);
1049 q2(:,1) = q2(:,2);
1050 f1(:,1) = f1(:,2);
1051 f2(:,1) = f2(:,2);
1052 J2(:,1) = J2(:,2);
1053 y(:,1) = y(:,2);
1054 Cn(:,1) = Cn(:,2);
1055 Fr(:,1) = Fr(:,2);
1056
1057 if time > 2*1000
1058     ttt = 1;
1059 end
1060
1061 catch ME
1062     % If this condition is reached, we are reducing the time-step to

```

```

1063         % 50% and doing the calculations again
1064         idx = q1(:,2) ≤ min_area;
1065         vel = abs(q2(:,2)./q1(:,2)) + sqrt(g*y(:,2)); vel(idx) = 0;
1066         dtnew = min(alpha*dx./(vel));
1067         time = time - time_step; % Seconds
1068         time_step = dtnew; % Halving the time-step
1069         n = n - 1;
1070     end
1071 end
1072 %% 7.0 - Post-Processing
1073 water_depths = Depth;
1074 %%% Post Processing Figures %%%
1075 % Call function
1076 warning('on');
1077 post_processing
1078 close all
1079 toc
1080 disp(['Thank you for using HyProSWE. If you have any questions, please contact me at ...
        marcusnobrega.engcivil@gmail.com.'])
1081 disp(['Also, please check your current matlab folder. The outputs are there.'])

```

#### 4) SVE Post Processing

```

1  %%% ----- HyProSWE Model ----- %%%
2  % Post-Processing Routine
3  % Developer: Marcus Nobrega Gomes Junior
4  % 5/1/2023
5  % Goal: Solution of 1-D SVE for given cross-section functions of Area, Perimeter, and
6  % top Width
7  % If you have any issues, please contact me at
8  % marcusnobrega.engcivil@gmail.com
9
10 %% Creating Modeling Results Folder
11 % Create the folder name
12 folderName = 'Modeling_Results';
13
14 % Check if the folder already exists
15 if ~exist(folderName, 'dir')
16     % If it doesn't exist, create the folder
17     mkdir(folderName);
18     disp('Folder "Modeling_Results" created successfully!');
19 else
20     disp('Data sucessfully exported in Modeling_Results Folder');
21 end
22
23 %% Post Processing Graphs
24 clf
25 close all
26
27
28 color_plot = [21, 179, 196]/255; % You can change it if you want
29
30 % Surfplot
31 t_save = [0:Nat:tt/dt];
32 t_save(1,1) = 1;
33 set(gcf, 'units', 'inches', 'position', [2,0,8,10])
34 subplot(3,1,1)
35 surf(x,tint(t_save)/3600,Froude);
36 view(0,90);
37 kk = colorbar ; colormap('jet')
38 shading interp
39 xlabel('x (m)', 'Interpreter', 'latex')
40 ylabel('t (h)', 'Interpreter', 'latex')
41 ylabel(kk, 'Froude Number', 'Interpreter', 'latex')
42 zlabel ('Froude Number', 'Interpreter', 'Latex');
43 xlim([0 L]);
44 ylim([0 tt/60/60]);
45 set(gca, 'FontName', 'Garamond', 'FontSize', 12, 'FontWeight', 'Bold', 'LineWidth', 1.5);
46 set(gca, 'TickLength', [0.02 0.01])
47 set(gca, 'TickDir', 'out')
48
49 subplot(3,1,2)
50 surf(x,tint(t_save)/60/60,Depth);

```

```

51 view(0,90);
52 kk = colorbar ; colormap('jet')
53 shading interp
54 xlabel('x (m)','Interpreter','latex')
55 ylabel('t (h)','Interpreter','latex')
56 ylabel(kk,'y (m)','Interpreter','latex')
57 zlabel ('y (m)','Interpreter','Latex');
58 xlim([0 L]);
59 ylim([0 tt/60/60]);
60 set(gca,'FontName','Garamond','FontSize',12,'FontWeight','Bold','LineWidth', 1.5);
61 set(gca,'TickLength',[0.02 0.01])
62 set(gca,'TickDir','out')
63
64 subplot(3,1,3)
65 wse = Depth + repmat(inv_el',[size(Depth,1),1]);
66 surf(x,tint(t_save)/60/60,wse);
67 view(0,90);
68 kk = colorbar ; colormap('jet')
69 shading interp
70 xlabel('x (m)','Interpreter','latex')
71 ylabel('t (h)','Interpreter','latex')
72 ylabel(kk,'WSE (m)','Interpreter','latex')
73 zlabel ('WSE (m)','Interpreter','Latex');
74 xlim([0 L]);
75 ylim([0 tt/60/60]);
76 set(gca,'FontName','Garamond','FontSize',12,'FontWeight','Bold','LineWidth', 1.5);
77 set(gca,'TickLength',[0.02 0.01])
78 set(gca,'TickDir','out')
79 exportgraphics(gcf,fullfile(folderName,'Surf_Plots.pdf'),'ContentType','image','Colorspace','rgb','Resolution',600
80 clf
81 close all
82
83 if flag_section == 2 % circular
84 % Video
85 obj = VideoWriter('Circular_Depth.avi','Motion JPEG AVI');
86 obj.Quality = 100;
87 obj.FrameRate = 20;
88 open(obj)
89 set(gcf,'units','inches','position',[2,2,10,3])
90 for n=1:1:(Nt/Nat)
91     if n == 1
92         t = 1;
93         pos = 1;
94     else
95         t=time_save(n);
96         pos = n;
97     end
98     % Circle Function
99     xcir = linspace(0,2*pi,100); % 100 points within 0 and 360 deg
100     cir = @(r,ctr) [r*cos(xcir)+ctr(1); r*sin(xcir)+ctr(2)];
101     c1 = cir(D/2, [D/2; D/2]);
102
103     % Boundary Circle
104     % (x - xc)^2 + (y - yc)^2 = D^2/4
105     % where xc = D/2 and yc = D/2
106     xc = D/2; yc = D/2;
107     y01 = Depth(pos,1);
108     y02 = Depth(pos,ceil(ceil(Nx/2)));
109     y03 = Depth(pos,Nx);
110     y0_c = [y01; y02; y03];
111     % For a given known y, we have to find two xs, such that
112     % x^2 + (-2xc)x + ( (y0 - yc)^2 - xc^2 - D^2/4 )
113     % or ax^2 + bx + c, with
114     % a = 1; b = -2xc; c = (y0 - yc)^2 - xc^2 - D^2/4
115     % x = (- b +/- sqrt(b^2 - 4ac)) / (2a)
116     a = 1;
117     b = -2*xc;
118     c = xc^2 + (y0_c - yc).^2 - D^2/4;
119     Delta = b^2 - 4*a.*c;
120     x1 = (-b + sqrt(Delta))/(2*a);
121     x2 = (-b - sqrt(Delta))/(2*a);
122     % Now we found the intersection of the circle and a line with know
123     % depth
124     subplot(1,3,1)
125     title(['t = ',num2str(round(round(t,2),0)), ' [sec]'])
126     ylim([0 D]);
127     xlim([0 D]);

```



```

128     viscircles([D/2 D/2],D/2,'Color','black');
129 %     plot(c1(1,:),c1(2,:), 'Color','black');
130     hold on
131     x_water = linspace(x2(1),x1(1),100);
132     y_water = repmat(y01,1,100);
133     plot(x_water,y_water,'Color',color_plot,'linewidth',2);
134 %     fill([c1(1,:) fliplr(c1(1,:))], [y_water fliplr(c2(1,:))],color_plot)
135     ylabel('y(m)','Interpreter','latex')
136     xlabel('B(m)','Interpreter','latex')
137     legend('Entrance','interpreter','latex')
138     hold off
139     grid on
140     set(gca,'FontName','Garamond','FontSize',12);
141     set(gca,'TickLength',[0.02 0.01])
142     set(gca,'TickDir','out');
143     box on
144     % second section
145     subplot(1,3,2)
146     title(['t = ',num2str(round(round(t,2),0)), ' [sec]'])
147     ylim([0 D]);
148     xlim([0 D]);
149     viscircles([D/2 D/2],D/2,'Color','black');
150     hold on
151     x_water = linspace(x2(2),x1(2),100);
152     y_water = repmat(y02,1,100);
153     plot(x_water,y_water,'Color',color_plot,'linewidth',2);
154     ylabel('y(m)','Interpreter','latex')
155     xlabel('B(m)','Interpreter','latex')
156     legend('x = L/2','interpreter','latex')
157     hold off
158     legend('L/2','interpreter','latex')
159     % third section
160     grid on
161     set(gca,'FontName','Garamond','FontSize',12);
162     set(gca,'TickLength',[0.02 0.01])
163     set(gca,'TickDir','out');
164     box on
165     subplot(1,3,3)
166     title(['t = ',num2str(round(round(t/60),0)), ' [sec]'])
167     ylim([0 D]);
168     xlim([0 D]);
169     viscircles([D/2 D/2],D/2,'Color','black');
170     hold on
171     x_water = linspace(x2(3),x1(3),100);
172     y_water = repmat(y03,1,100);
173     plot(x_water,y_water,'color',color_plot,'linewidth',2);
174     hold off
175     ylabel('y(m)','Interpreter','latex')
176     xlabel('B(m)','Interpreter','latex')
177     legend('Exit','interpreter','latex')
178     grid on
179     set(gca,'FontName','Garamond','FontSize',12);
180     set(gca,'TickLength',[0.02 0.01])
181     set(gca,'TickDir','out');
182     box on
183     % Save frame
184     title(['t = ',num2str(round(round(t,2),0)), ' [sec]'])
185     f = getframe(gcf);
186     writeVideo(obj,f);
187     hold off
188     clf
189     end
190 obj.close();
191 end
192
193 if flag_section == 3 % paraboloid
194 % Video
195 obj = VideoWriter('Parabolic_Depth.avi','Motion JPEG AVI');
196 obj.Quality = 100;
197 obj.FrameRate = 20;
198 open(obj)
199 set(gcf,'units','inches','position',[2,2,10,3])
200 for n=1:1:(Nt/Nat)
201     if n == 1
202         t = 1;
203         pos = 1;
204     else

```

```

205         t=time_save(n);
206         pos = n;
207     end
208     % Save frame
209     Plot_Title = 'Time = %d (sec)';
210     sgtitle(sprintf(Plot_Title, time_store(n)), 'fontsize',18,'interpreter','latex')
211     % Parabolic Function
212     %  $y = a \cdot x^2 \Rightarrow x_{\max} = \sqrt{(y_{\max}/a)}$ 
213     ymax = max(max(Depth));
214     xmax = sqrt(ymax/a); % x to left and right directions
215     xpar = linspace(-xmax,xmax,100); % 100 points within -xmax and xmax deg
216     ypar = a.*xpar.^2;
217     % Now we found bottom of the channel
218     % We still need to find xleft and xright for a given y
219     y01 = Depth(pos,1);
220     y02 = Depth(pos,ceil(Nx/2));
221     y03 = Depth(pos,Nx);
222     y0_c = [y01; y02; y03];
223     xright = sqrt(y0_c/a);
224     xleft = - xright;
225     subplot(1,3,1)
226     title(['t = ', num2str(round(round(t/60,2),0)), ' [min]'])
227     ylim([0 ymax]);
228     xlim([0 ymax]);
229     plot(xpar,ypar, 'Color','black','LineWidth',2);
230     hold on
231     x_water = linspace(xleft(1),xright(1),100);
232     y_water = linspace(y01,y01,100);
233     plot(x_water,y_water, 'color',color_plot, 'linewidth',2);
234     ylabel('y(m)', 'Interpreter','latex')
235     xlabel('B(m)', 'Interpreter','latex')
236     legend('Entrance', 'interpreter','latex')
237     grid on
238     set(gca, 'FontName','Garamond','FontSize',12);
239     set(gca, 'TickLength',[0.02 0.01])
240     set(gca, 'TickDir','out');
241     hold off
242     % second section
243     subplot(1,3,2)
244     title(['t = ', num2str(round(round(t/60,2),0)), ' [min]'])
245     ylim([0 ymax]);
246     xlim([0 ymax]);
247     plot(xpar,ypar, 'Color','black','LineWidth',2);
248     hold on
249     x_water = linspace(xleft(2),xright(2),100);
250     y_water = linspace(y02,y02,100);
251     plot(x_water,y_water, 'color',color_plot, 'linewidth',2);
252     ylabel('y(m)', 'Interpreter','latex')
253     xlabel('B(m)', 'Interpreter','latex')
254     legend('x = L/2', 'interpreter','latex')
255     grid on
256     set(gca, 'FontName','Garamond','FontSize',12);
257     set(gca, 'TickLength',[0.02 0.01])
258     set(gca, 'TickDir','out');
259     hold off
260     % third section
261     subplot(1,3,3)
262     title(['t = ', num2str(round(round(t/60,2),0)), ' [min]'])
263     ylim([0 ymax]);
264     xlim([0 ymax]);
265     plot(xpar,ypar, 'Color','black','LineWidth',2);
266     hold on
267     x_water = linspace(xleft(3),xright(3),100);
268     y_water = linspace(y03,y03,100);
269     plot(x_water,y_water, 'Color',color_plot, 'linewidth',2);
270     ylabel('y(m)', 'Interpreter','latex')
271     xlabel('B(m)', 'Interpreter','latex')
272     legend('Outlet', 'interpreter','latex')
273     grid on
274     set(gca, 'FontName','Garamond','FontSize',12);
275     set(gca, 'TickLength',[0.02 0.01])
276     set(gca, 'TickDir','out');
277     f = getframe(gcf);
278     writeVideo(obj,f);
279     hold off
280     clf
281 end

```

```

282 obj.close();
283 end
284
285 %% Plots
286 % Time Scale
287 if flag_elapsed_time == 1
288     close all
289     flag_date = 3; % 1 min, 2 hour, 3 day, 4 month
290     date_string = {'Elapsed time (min)', 'Elapsed time (h)', 'Elapsed time (days)', 'Elapsed time ...
291                   (months)'};
292     if flag_date == 1
293         time_scale = 1;
294     elseif flag_date == 2
295         time_scale = 1/60;
296     elseif flag_date == 3
297         time_scale = 1/60/24;
298     else
299         time_scale = 1/60/24/30;
300     end
301     set(gcf, 'units', 'inches', 'position', [2,0,8,10])
302     subplot(3,2,1)
303     % Flows
304     plot(time_save/60, Discharge(:,1), 'LineStyle', '--', 'LineWidth', 2, 'Color', 'k')
305     hold on
306     plot(time_save/60, Discharge(:,ceil(Nx/2)), 'LineStyle', ':', 'LineWidth', 2, 'Color', 'k')
307     hold on
308     plot(time_save/60, Discharge(:,Nx), 'LineStyle', '-', 'LineWidth', 2, 'Color', 'k')
309     hold on
310     xlabel(date_string(flag_date), 'interpreter', 'latex');
311     ylabel('Flow Discharge (m\textsuperscript{3}/s)', 'Interpreter', 'latex');
312     legend('Entrance', 'L/2', 'Outlet', 'Interpreter', 'Latex', 'Location', 'best')
313     % Velocity
314     subplot(3,2,2)
315     plot(time_save/60, Velocity(:,1), 'LineStyle', '--', 'LineWidth', 2, 'Color', 'k')
316     hold on
317     plot(time_save/60, Velocity(:,ceil(Nx/2)), 'LineStyle', ':', 'LineWidth', 2, 'Color', 'k')
318     hold on
319     plot(time_save/60, Velocity(:,Nx), 'LineStyle', '-', 'LineWidth', 2, 'Color', 'k')
320     %%% Normal Depth Velocity %%%
321     xlabel(date_string(flag_date), 'interpreter', 'latex');
322     ylabel('Velocity (m/s)', 'Interpreter', 'latex');
323     legend('Entrance', 'L/2', 'Outlet', 'Interpreter', 'Latex', 'Location', 'best')
324     % Water Depth
325     subplot(3,2,3)
326     plot(time_save/60, Depth(:,1), 'LineStyle', '--', 'LineWidth', 2, 'Color', 'k')
327     hold on
328     plot(time_save/60, Depth(:,ceil(Nx/2)), 'LineStyle', ':', 'LineWidth', 2, 'Color', 'k')
329     hold on
330     plot(time_save/60, Depth(:,Nx), 'LineStyle', '-', 'LineWidth', 2, 'Color', 'k')
331     xlabel(date_string(flag_date), 'interpreter', 'latex');
332     ylabel('Water Depths (m)', 'Interpreter', 'latex');
333     legend('Entrance', 'L/2', 'Outlet', 'Interpreter', 'Latex', 'Location', 'best')
334     % Froude Number
335     subplot(3,2,4)
336     plot(time_save/60, Froude(:,1), 'LineStyle', '--', 'LineWidth', 2, 'Color', 'k')
337     hold on
338     plot(time_save/60, Froude(:,ceil(Nx/2)), 'LineStyle', ':', 'LineWidth', 2, 'Color', 'k')
339     hold on
340     plot(time_save/60, Froude(:,Nx), 'LineStyle', '-', 'LineWidth', 2, 'Color', 'k')
341     xlabel(date_string(flag_date), 'interpreter', 'latex');
342     ylabel('Froude Number', 'Interpreter', 'latex');
343     legend('Entrance', 'L/2', 'Outlet', 'Interpreter', 'Latex', 'Location', 'best')
344
345     % Courant Number
346     subplot(3,2,5)
347     plot(time_save/60, Courant(:,1), 'LineStyle', '--', 'LineWidth', 2, 'Color', 'k')
348     hold on
349     plot(time_save/60, Courant(:,ceil(Nx/2)), 'LineStyle', ':', 'LineWidth', 2, 'Color', 'k')
350     hold on
351     plot(time_save/60, Courant(:,Nx), 'LineStyle', '-', 'LineWidth', 2, 'Color', 'k')
352     xlabel(date_string(flag_date), 'interpreter', 'latex');
353     ylabel('Courant Number', 'Interpreter', 'latex');
354     legend('Entrance', 'L/2', 'Outlet', 'Interpreter', 'Latex', 'Location', 'best')
355
356     % Rating Curve
357     % Solving for normal Depth

```

```

358     ymin = min(min(Depth));
359     ymax = max(max(Depth));
360     hs = 1; % 1 node
361     % hs = ceil(1);
362     if flag_section ≠ 4
363         y_m = [ymin:0.01:ymax]'; % meters
364         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;
365     else
366         % [y_table, A, P, Rh, y_bar, n_med, Beta, v, B, Q]
367         % [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
368         coll = 2; % Col with A
369         for jj = 1:length(Flow_Area(:,1))
370             Qn(jj,1) = Vlookup_g(irr_table,coll,Flow_Area(jj,hs),10); % Attention here
371             y_m(jj,1) = Vlookup_g(irr_table,coll,Flow_Area(jj,hs),1);
372             rh_i = Vlookup_g(irr_table,coll,Flow_Area(jj,hs),4);
373         end
374     end
375     subplot(3,2,6)
376     tbegin = 30; % (steps), considering initial stabilization of the domain
377     plot(Discharge(2:end,hs),Depth(2:end,hs),'LineStyle','--','LineWidth',2,'Color','k')
378     hold on
379     plot(Discharge(2:end,ceil(Nx/2)),Depth(2:end,ceil(Nx/2)),'LineStyle',':', 'LineWidth',2,'Color','k')
380     hold on
381     plot(Qn,y_m,'LineStyle','-','LineWidth',2,'Color','k')
382     xlabel('Flow Discharge (m\textsuperscript{3}/s)','Interpreter','latex');
383     ylabel('Water Depth (m)','Interpreter','latex');
384     ylim([ymin 1.1*max([max(y_m),max(y(ceil(Nx)))])]);
385     legend('Q(Inlet)','Q(Nx/2)','$Q_{n}$ (L)','Interpreter','Latex','Location','best')
386     hold off
387     exportgraphics(gcf,fullfile(folderName,'Summary_Charts.pdf'),'ContentType','vector')
388     clf
389     close all
390 else
391     close all
392     % Time Calculation
393     time_duration = time_save/3600/24 + Date_Begin;
394     set(gcf,'units','inches','position',[2,0,8,10])
395     date_string = {''};
396     flag_date = 1;
397     subplot(3,2,1)
398     % Flows
399     plot(time_duration,Discharge(:,1),'LineStyle','--','LineWidth',2,'Color','k')
400     hold on
401     plot(time_duration,Discharge(:,ceil(Nx/2)),'LineStyle',':', 'LineWidth',2,'Color','k')
402     hold on
403     plot(time_duration,Discharge(:,Nx),'LineStyle','-','LineWidth',2,'Color','k')
404     hold on
405     xlabel(date_string(flag_date),'interpreter','latex');
406     ylabel('Flow Discharge (m\textsuperscript{3}/s)','Interpreter','latex');
407     legend('Entrance','L/2','Outlet','Interpreter','Latex','location','best')
408     % Velocity
409     subplot(3,2,2)
410     plot(time_duration,Velocity(:,1),'LineStyle','--','LineWidth',2,'Color','k')
411     hold on
412     plot(time_duration,Velocity(:,ceil(Nx/2)),'LineStyle',':', 'LineWidth',2,'Color','k')
413     hold on
414     plot(time_duration,Velocity(:,Nx),'LineStyle','-','LineWidth',2,'Color','k')
415     %%% Normal Depth Velocity %%%
416     xlabel(date_string(flag_date),'interpreter','latex');
417     ylabel('Velocity (m/s)','Interpreter','latex');
418     legend('Entrance','L/2','Outlet','Interpreter','Latex','Location','best')
419     % Water Depth
420     subplot(3,2,3)
421     plot(time_duration,Depth(:,1),'LineStyle','--','LineWidth',2,'Color','k')
422     hold on
423     plot(time_duration,Depth(:,ceil(Nx/2)),'LineStyle',':', 'LineWidth',2,'Color','k')
424     hold on
425     plot(time_duration,Depth(:,Nx),'LineStyle','-','LineWidth',2,'Color','k')
426     xlabel(date_string(flag_date),'interpreter','latex');
427     ylabel('Water Depths (m)','Interpreter','latex');
428     legend('Entrance','L/2','Outlet','Interpreter','Latex','Location','best')
429     % Froude Number
430     subplot(3,2,4)
431     plot(time_duration,Froude(:,1),'LineStyle','--','LineWidth',2,'Color','k')
432     hold on
433     plot(time_duration,Froude(:,ceil(Nx/2)),'LineStyle',':', 'LineWidth',2,'Color','k')

```

```

434 hold on
435 plot(time_duration,Froude(:,Nx),'LineStyle','-', 'LineWidth',2, 'Color','k')
436 xlabel(date_string(flag_date), 'interpreter','latex');
437 ylabel('Froude Number','Interpreter','latex');
438 legend('Entrance','L/2','Outlet','Interpreter','Latex','Location','best')
439
440 % Courant Number
441 subplot(3,2,5)
442 plot(time_duration,Courant(:,1),'LineStyle','--', 'LineWidth',2, 'Color','k')
443 hold on
444 plot(time_duration,Courant(:,ceil(Nx/2)),'LineStyle',':', 'LineWidth',2, 'Color','k')
445 hold on
446 plot(time_duration,Courant(:,Nx),'LineStyle','-', 'LineWidth',2, 'Color','k')
447 xlabel(date_string(flag_date), 'interpreter','latex');
448 ylabel('Courant Number','Interpreter','latex');
449 legend('Entrance','L/2','Outlet','Interpreter','Latex','Location','best')
450
451 % Rating Curve
452 % Solving for normal Depth
453 ymin = min(min(Depth));
454 ymax = max(max(Depth));
455 hs = 1; % 1 node
456 % hs = ceil(1);
457 if flag_section ≠ 4
458     y_m = [ymin:0.01:ymax]'; % meters
459     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;
460 else
461     % [y_table, A, P, Rh, y_bar, n_med, Beta, v, B, Q]
462     % [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
463     coll = 2; % Col with A
464     for jj = 1:length(Flow_Area(:,1))
465         Qn(jj,1) = Vlookup_g(irr_table,coll,Flow_Area(jj,hs),10); % Attention here
466         y_m(jj,1) = Vlookup_g(irr_table,coll,Flow_Area(jj,hs),1);
467         rh_i = Vlookup_g(irr_table,coll,Flow_Area(jj,hs),4);
468     end
469 end
470 subplot(3,2,6)
471 tbegin = 30; % (steps), considering initial stabilization of the domain
472 plot(Discharge(2:end,hs),Depth(2:end,hs), 'LineStyle','--', 'LineWidth',2, 'Color','k')
473 hold on
474 plot(Discharge(2:end,ceil(Nx/2)),Depth(2:end,ceil(Nx/2)), 'LineStyle',':', 'LineWidth',2, 'Color','k')
475 hold on
476 plot(Qn,y_m,'LineStyle','-', 'LineWidth',2, 'Color','k')
477 xlabel('Flow Discharge (m\textsuperscript{3}/s)', 'Interpreter','latex');
478 ylabel('Water Depth (m)', 'Interpreter','latex');
479 ylim([ymin 1.1*max([max(y_m),max(y(ceil(Nx)))])]);
480 legend('Q(Inlet)','Q(Nx/2)','$Q_{n}$ (L)', 'Interpreter','Latex','Location','best')
481 hold off
482 exportgraphics(gcf,fullfile(folderName,'Summary_Charts.pdf'),'ContentType','vector')
483 clf
484 close all
485 end
486
487 %% States Post-Processing
488 states_post_processing
489 %% Cross-Section Post-Processing
490 if flag_section == 4
491     cross_section_post_processing
492 end
493
494 %% Lateral Profiles
495 if flag_section ≠ 4
496     wse_top_width_regular
497 end
498 %% Detailed Output
499 Detailed_Output_Script

```

## 5) Cross-Section Post Processing

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

```

1 % Post-Processing Routine
2 % Model: HyPro-SWE
3 % Developer: Marcus Nobrega
4 % Last Update: 4/29/2023
5 % Goal: Create animations of water depth, top width, and water surface
6 % elevation
7
8 close all
9 close(video);
10
11 Video_Name = 'Depth_WSE_Top_Width.mp4';
12
13 % Set up video
14 video = VideoWriter(Video_Name, 'MPEG-4');
15 open(video);
16
17 % Define water depths for each time
18 depths = Depth(:,1)';
19
20 % Preallocate Top Width
21 B2 = zeros(size(Flow_Area));
22
23 % Time
24 t = time_save; % Sec
25
26 % Define tick size
27 ticksize = [0.015 0.01];
28
29 % Define Tick Position
30 tickposition = 'in';
31
32
33 % Define polygon for the cross-section
34 polygon = polyshape(x_cross,y_cross);
35
36 % Water Surface Elevation
37 wse = Depth + repmat(inv_el', [size(Depth,1),1]);
38
39 % Color
40 color_plot = [21, 179, 196]/255;
41 set(gcf, 'units', 'inches', 'position', [2,0,8,10])
42
43 if flag_elapsed_time ~= 1
44     % Time Calculation
45     time_duration = time_save/3600/24 + Date_Begin;
46 end
47
48 % Iterate through all time steps
49 set(gca, 'FontSize', 14, 'FontName', 'Garamond')
50 for i=1:length(t)
51
52     if flag_elapsed_time == 1
53         Plot_Title = 'Time = %d (sec)';
54         sgtitle(sprintf(Plot_Title, time_store(i)), 'fontsize', 18, 'interpreter', 'latex')
55     else
56         sgtitle(string(time_duration(i)), 'fontsize', 18, 'interpreter', 'latex');
57     end
58     for j = 1:3 % 3 Cross-sections
59         if j == 1
60             sec = 1;
61         elseif j == 2
62             sec = ceil(Nx/2);
63         else
64             sec = Nx;
65         end
66         depths = Depth(i,sec)';
67         hold on
68         subplot(3,3,(j))
69         % Set title with time and water depth
70         % Define the water depth for this time step
71         depth_line = depths*ones(1,length(x_cross));
72         plot(x_cross, y_cross, '-k', 'LineWidth', 2, Marker='*'); hold on
73         % Find where depth line intersects cross-section polygon
74         [x_intersect, y_intersect] = polyxpoly(x_cross,y_cross,x_cross,depth_line);
75         if length(x_intersect) > 1
76             % Finding Inside Values
77             idx1 = x_cross > x_intersect(1);

```

```

78     idx2 = x_cross ≤ x_intersect(end);
79     idx = logical(idx1.*idx2); % Both cases
80     x_pol = [x_intersect(1), x_cross(idx)', x_intersect(end)];
81     y_pol = [y_intersect(1), y_cross(idx)', y_intersect(2)];
82     hold on
83     % If the depth line intersects the polygon, plot it
84     if ~isempty(x_intersect) && ~isempty(y_intersect)
85         depth_plot = depth_line(1)*ones(size(x_pol));
86         fill([x_pol fliplr(x_pol)], [y_pol fliplr(depth_plot)],color_plot)
87     else
88         error('Call developer')
89     end
90 end
91 box on
92 if j == 1
93     ylabel('Depth [m]','Interpreter','latex')
94 end
95 xlabel('Station [m]','Interpreter','latex')
96 title(sprintf('x = %0.2f m, h = %0.2f m', round((sec-1)*dx,2), ...
97     depths),'fontsize',16,'interpreter','latex');
98 set(gca,'FontSize',12,'FontName','Garamond')
99 % Set Tick Postion and Tick Size
100 set(gca,'TickLength',ticksize)
101 set(gca,'TickDir',tickposition)
102 end
103 % ----- Plotting Channel Width ----- %
104 subplot(3,3,[4 5 6]);
105 if flag_section ≠ 4
106     B2 = B_function(D,Z1,Z2,a,b,y);
107 else
108     for pos_b = 1:length(Flow_Area(1,:))
109         % [y_table, A, P, Rh, y_bar, n_med, Beta, v, B, Q]
110         % [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
111         B2(i,pos_b) = Vlookup_g(irr_table, coll, Flow_Area(i,pos_b),9);
112     end
113 end
114 offset = max(x_cross)/2; % From station data
115 right_margin = B2(i,:)/2 + offset; left_margin = -B2(i,:)/2 + offset;
116 plot(x,right_margin,'k','LineWidth',2); set(gca,'YDir','reverse');
117 hold on
118 plot(x,left_margin,'k','LineWidth',2); set(gca,'YDir','reverse');
119 hold on
120 fill([x' fliplr(x')], [left_margin fliplr(right_margin)],color_plot)
121 xlabel('$x$ [m]','Interpreter','latex');
122 ylabel('Station [m]','Interpreter','latex');
123 ylim([0, max(x_cross)]);
124 grid on
125 title(sprintf('$B_{\{max\}}(t)$ = %0.2f m', max(right_margin - ...
126     left_margin)), 'fontsize',16,'interpreter','latex');
127 set(gca,'FontSize',12,'FontName','Garamond')
128 % Set Tick Postion and Tick Size
129 set(gca,'TickLength',ticksize)
130 set(gca,'TickDir',tickposition)
131 % ----- Ploting Water Surface Elevation ----- %
132 subplot(3,3,[7 8 9]);
133 plot(x,inv_el,'LineWidth',4,'LineStyle','-','Color','k');
134 hold on
135 plot(x,wse(i,:), 'k','LineWidth',2,'LineStyle','-','Color',color_plot);
136 fill([x' fliplr(x')], [inv_el' fliplr(wse(i,:))],color_plot)
137 xlabel('$x$ [m]','Interpreter','latex');
138 ylabel('Water Surface Elevation [m]','Interpreter','latex');
139 ylim([0.98*min(min(wse - Depth)) max(max(1.01*wse))])
140 grid on
141 title(sprintf('$WSE_{\{max\}}(t)$ = %0.2f m', max(wse(i,:))), 'fontsize',16,'interpreter','latex');
142 set(gca,'FontSize',12,'FontName','Garamond')
143 % Set Tick Postion and Tick Size
144 set(gca,'TickLength',ticksize)
145 set(gca,'TickDir',tickposition)
146
147 % Save the frame for the video
148 % Set background color and write to video
149 frame = getframe(gcf);
150 writeVideo(video,frame);
151 hold off
152 clf

```

```

153 end
154 % Close video writer
155 close(video);
156 close all

```

## 6) Water Surface Elevation Profiles

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

```

1 % Post-Processing Routine
2 % Model: HyPro-SWE
3 % Developer: Marcus Nobrega
4 % Last Update: 4/29/2023
5 % Goal: Create animations of WSE and Top Width for regular sections
6
7 close all
8
9 Video_Name = 'WSE_Top_Width.avi';
10
11 % Set up video
12 video = VideoWriter(Video_Name, 'MPEG-4');
13 open(video);
14
15 % Define water depths for each time
16 depths = Depth(:,1)';
17
18 % Preallocate Top Width
19 B2 = zeros(size(Flow_Area));
20
21 % Time
22 t = time_save; % Sec
23
24 % Define tick size
25 ticksize = [0.02 0.01];
26
27
28
29 % Water Surface Elevation
30 wse = Depth + repmat(inv_el', [size(Depth,1),1]);
31
32 % Color
33 color_plot = [21, 179, 196]/255;
34 set(gcf, 'units', 'inches', 'position', [2,0,8,10])
35
36 % Iterate through all time steps
37 set(gca, 'FontSize', 14, 'FontName', 'Garamond')
38 for i=1:length(t)
39     Plot_Title = 'Time = %d (sec)';
40     sgtitle(sprintf(Plot_Title, time_store(i)), 'fontsize', 18, 'interpreter', 'latex')
41     % ----- Plotting Channel Width ----- %
42     subplot(2,3,[1 2 3]);
43     if flag_section ~= 4
44         B2 = B_function(D, Z1, Z2, a, b, Depth);
45     else
46         for pos_b = 1:length(Flow_Area(1,:))
47             % [y_table, A, P, Rh, y_bar, n_med, Beta, v, B, Q]
48             % [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
49             B2(i, pos_b) = Vlookup_g(irr_table, col1, Flow_Area(i, pos_b), 9);
50         end
51     end
52     if flag_section == 1
53         offset = b/2 + (Z1 + Z2)/2*max(max(depths));
54         xmax_plot = (Z1 + Z2)*max(max(depths)) + b;
55     elseif flag_section == 2
56         offset = D/2;
57         xmax_plot = D;
58     elseif flag_section == 3
59         offset = xmax/2;
60         xmax_plot = xmax;
61     else
62         offset = max(x_cross)/2; % From station data
63         xmax_plot = max(x_cross);

```



```

64     end
65     right_margin = B2(i,:)/2 + offset; left_margin = -B2(i,:)/2 + offset;
66     plot(x,right_margin,'k','LineWidth',2); set(gca,'YDir','reverse');
67     hold on
68     plot(x,left_margin,'k','LineWidth',2); set(gca,'YDir','reverse');
69     hold on
70     fill([x' fliplr(x)], [left_margin fliplr(right_margin)],color_plot)
71     xlabel('$x$ [m]','Interpreter','latex');
72     ylabel('Station [m]','Interpreter','latex');
73     ylim([0, xmax_plot]);
74     xlim([0, max(x)]);
75     grid on
76     title(sprintf('$B_{\{max\}}(t)$ = %.2f m', max(right_margin - ...
77         left_margin)), 'fontsize',16,'interpreter','latex');
78     set(gca,'FontSize',12,'FontName','Garamond')
79
80     % ----- Plotting Water Surface Elevation ----- %
81     subplot(2,3,[4 5 6])
82     plot(x,inv_el,'LineWidth',4,'LineStyle','-','Color','k');
83     hold on
84     plot(x,wse(i,:), 'k','LineWidth',2,'LineStyle','-','Color',color_plot);
85     fill([x' fliplr(x)], [inv_el' fliplr(wse(i,:))],color_plot)
86     xlabel('$x$ [m]','Interpreter','latex');
87     ylabel('Water Surface Elevation [m]','Interpreter','latex');
88     ylim([0.98*min(min(wse - Depth)) max(max(1.01*wse))])
89     grid on
90     title(sprintf('$WSE_{\{max\}}(t)$ = %.2f m', max(wse(i,:))), 'fontsize',16,'interpreter','latex');
91
92     % Save the frame for the video
93     set(gca,'FontSize',12,'FontName','Garamond')
94     % Set background color and write to video
95     frame = getframe(gcf);
96     writeVideo(video,frame);
97     hold off
98 end
99 % Close video writer
100 close(video);
101 close all

```

## 7) Detailed Output

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

```

1     % HyProSWE Model
2 % Output .csv script
3 % Developer: Marcus Nobrega
4 % Goal: Create a detailed output from modeling results
5 % Last updated: 4/30/2023
6
7
8 %%% ----- All rights reserved ----- %%
9
10 % Number of states
11 ns = 6;
12 % 0 - time, 1 - flow, 2 - depth, 3 - velocity, 4 - Courant, 5 - Froude, 6,
13 % 7 WSE
14
15 % Concatenate data
16 t = time_store; % time vector
17 h = Depth; % water level matrix
18 q = Discharge; % flow rate matrix
19 v = Velocity; % velocity matrix
20 f = Froude; % Froude number matrix
21 c = Courant; % Courant number matrix
22 z = x; % distance matrix
23
24 % Round Data
25 decimal_places = 3;
26
27 data = zeros(size(Depth,1),size(Depth,2),ns);
28
29 data(:,:,1) = Depth;

```

```

30 data(:, :, 2) = Discharge;
31 data(:, :, 3) = Velocity;
32 data(:, :, 4) = Froude;
33 data(:, :, 5) = Courant;
34 data(:, :, 6) = wse;
35
36
37 if flag_output == 1
38     for i = 1:(Nx*ns)
39         j = floor((i-1)/ns);
40         x_cell = j*dx;
41         if mod(i-1,ns) == 0 || (i-1)/ns == 1
42             states_title(1,i) = cellstr(sprintf('Depth (m), x(m) = %0.2f',x_cell));
43         elseif mod(i-1,ns) == 1 || (i-1)/ns == 2
44             states_title(1,i) = cellstr(sprintf('Discharge (m^3/s), x(m) = %0.2f',x_cell));
45         elseif mod(i-1,ns) == 2 || (i-1)/ns == 3
46             states_title(1,i) = cellstr(sprintf('Velocity (m/s), x(m) = %0.2f',x_cell));
47         elseif mod(i-1,ns) == 3 || (i-1)/ns == 4
48             states_title(1,i) = cellstr(sprintf('Froude (-), x(m) = %0.2f',x_cell));
49         elseif mod(i-1,ns) == 4 || (i-1)/ns == 5
50             states_title(1,i) = cellstr(sprintf('Courant Number (-), x(m) = %0.2f',x_cell));
51         elseif mod(i-1,ns) == 5 || (i-1)/ns == 6
52             states_title(1,i) = cellstr(sprintf('Water Surface Elevation (m), x(m) = %0.2f',x_cell));
53         end
54     end
55 else
56     for i = 1:(Nx*ns)
57         if mod(i,Nx) ~= 0
58             j = mod(i,Nx);
59             x_cell = (j-1)*dx; % m
60         else
61             j = Nx;
62             x_cell = (j-1)*dx; % m
63         end
64         if floor(i/Nx) == 0 || i/Nx == 1
65             states_title(1,i) = cellstr(sprintf('Depth (m), x(m) = %0.2f',x_cell));
66         elseif floor(i/Nx) == 1 || i/Nx == 2
67             states_title(1,i) = cellstr(sprintf('Discharge (m^3/s), x(m) = %0.2f',x_cell));
68         elseif floor(i/Nx) == 2 || i/Nx == 3
69             states_title(1,i) = cellstr(sprintf('Velocity (m/s), x(m) = %0.2f',x_cell));
70         elseif floor(i/Nx) == 3 || i/Nx == 4
71             states_title(1,i) = cellstr(sprintf('Froude (-), x(m) = %0.2f',x_cell));
72         elseif floor(i/Nx) == 4 || i/Nx == 5
73             states_title(1,i) = cellstr(sprintf('Courant Number (-), x(m) = %0.2f',x_cell));
74         elseif floor(i/Nx) == 5 || i/Nx == 6
75             states_title(1,i) = cellstr(sprintf('Water Surface Elevation (m), x(m) = %0.2f',x_cell));
76         end
77     end
78 end
79 % states_title(1,end+1) = cellstr(sprintf('Water Surface Elevation (m), x(m) = %0.2f',dx*(Nx-1)));
80 time_string = {'Time (sec)'};
81 % Table Headers
82 table_headers = [time_string, states_title];
83 data_save = zeros(length(time_store),ns*Nx);
84
85
86 if flag_output == 1
87     % Detailed Output for each section with all states together
88     for i = 1:length(time_store)
89         % For all time
90         for j = 1:ns
91             if j == 2
92                 ttt = 1;
93             end
94             % For all states
95             for k = 1:Nx
96                 % For all nodes
97                 data_table = round(data(i,k,j),decimal_places);
98                 % data_save(i,ns*(k-1) + j) = data_table;
99                 data_table = round(data(i,k,j),decimal_places);
100                data_save(i,ns*(k-1) + j) = data_table;
101            end
102        end
103    end
104 else
105     % Detailed Output for each state for each section
106     for i = 1:length(time_store)

```

```

107         % For all time
108         for j = 1:ns
109             % For all states
110             for k = 1:Nx
111                 % For all nodes
112                 data_table = round(data(i,k,j),decimal_places);
113                 data_save(i,k + (j-1)*Nx) = data_table;
114             end
115         end
116     end
117 end
118
119
120
121 data_save = [time_save, data_save]; % Concatenating dataset to the time
122 T = array2table(data_save,'VariableNames',table_headers);
123 writetable(T,'Detailed_Output.csv','Delimiter',' ');
124 disp('Attention: Data exported in .CSV');
125
126
127 %% Detailed Output per Cross-Section (Similarly as HEC-RAS)
128 i_prev = 1;
129 if flag_elapsed_time == 1
130     time_str = 'Elapsed Time (sec)';
131 else
132     time_str = 'Time';
133 end
134 Titles_Section = {'x(m)',time_str,' Depth (m)','Discharge (m3/s)','Velocity (m/s)','Froude ...
135                  (-)','Courant Number (-)','WSE (m)'};
136 for i = 1:Nx
137     % Through each section
138     for j = 1:length(time_store)
139         % Through each time
140         for k = 1:ns
141             row = length(time_store)*(i-1) + j;
142             data_save_XS(row,k) = data(j,i,k);
143         end
144     end
145 end
146 zzz = data_save_XS;
147 clear data_table data_save_XS
148 for i = 1:Nx
149     x_cell = (i-1)*dx;
150     if i == 1
151         section(1,1) = x_cell;
152     end
153     row = length(time_store)*(i-1) + 1;
154     row_i = length(time_store)*(i);
155     data_save_XS((row + i-1):(row_i + i-1),:) = zzz(row:row_i,:);
156     data_save_XS(row_i+1 + i - 1,:) = nan;
157
158     section((row + i-1):(row_i + i-1),:) = x_cell;
159     section(row_i+1 + i - 1,:) = NaN;
160 end
161 section(size(data_save_XS,1),1) = x_cell;
162 % section(end:size(data_save_XS,1)) = [];
163 % section(section == 0) = NaN;
164 if flag_elapsed_time == 1
165     time_vector = time_save;
166 else
167     time_vector = time_begin + time_save/86400; % Days minutes and seconds
168 end
169
170 Δ = 0;
171 for i = 1:Nx
172     row = length(time_store)*(i-1) + 1;
173     row_i = length(time_store)*(i);
174     time_vector_total((row + i-1):(row_i + i-1),1) = time_vector;
175     time_vector_total(row_i+1 + i - 1,:) = nan;
176 end
177
178 data_save = [section, time_vector_total, data_save_XS]; % Concatenating dataset to the time
179 T = array2table(data_save,'VariableNames',Titles_Section);
180
181 T.Properties.VariableNames(1:size(data_save,2)) = Titles_Section;
182 writetable(T,'Detailed_Output_XS.csv','Delimiter',' ');

```

```
183 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](http://dx.doi.org/10.1007/978-981-15-4577-1_32)