

# 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>, César Ambrogi Ferreira do Lago<sup>§</sup>, and Eduardo M. Mendiondo<sup>§</sup>

## I. SUPPLEMENTARY MATERIAL

This supplemental material presents the following:

- Cross-Section Data in Sec. I-A
- Data Derived from ANA in Sec. I-B
- Appendix 2 - Algorithm 2 for HP Estimation on Python language in Sec. I-C
- Matlab codes of (i) HP Estimator Sec. I-D1, (ii) Read Input Data for SVE Model Sec. I-D2, (iii) SVE Model in Sec. I-D3, and (iv) post-processing in Sec. I-D4 are presented in the end of this document, respectively.

### A. Cross-Section 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

```

<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.br).

<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.

```

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, typycally 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_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)

```

```

174 plot(x_triangular,y_triangular,'LineWidth',line_w,'color',c)
175 xlabel('x(m)','Interpreter','latex','FontSize',font)
176 ylabel('y(m)','Interpreter','latex','FontSize',font)
177 grid on
178 set(gca,'FontSize',font)
179 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')

```

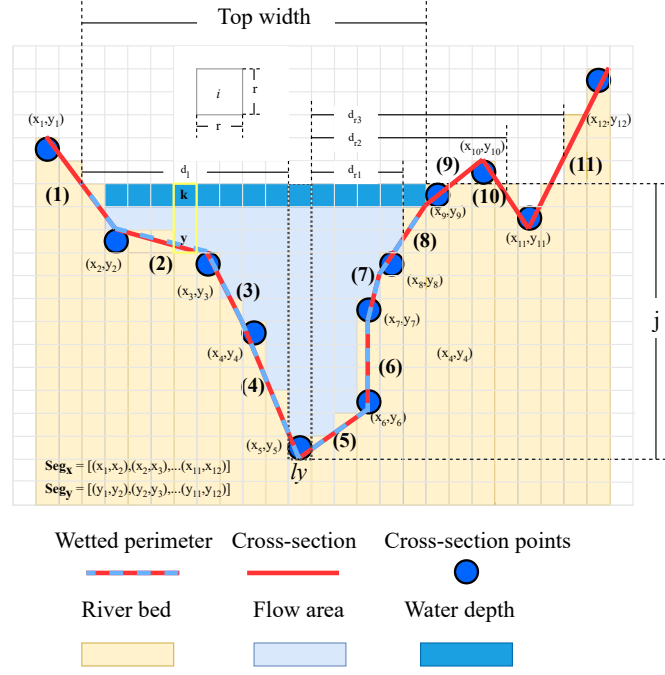
### B. Data derived from ANA

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

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

The algorithm begins by finding the lowest bottom elevation of the riverbed  $ly$ , then, two vectors are defined ( $seg_x$  and  $seg_y$ ) with consecutive pairs or coordinates for both axis, this aims to determine the flow area between the water depth  $j$  and the boundaries of the riverbed (see Fig. 1). The main loop is used to represent the water depth increasing, then, inside of this, three individual loops are used to 1) define the riverbed boundaries; 2) calculate the flow area, and 3) calculate the wet perimeter. The left HP are determined in terms of the aforementioned variables. Considering that the water depth is monotonically increasing from  $ly$  for every pixel in the mesh on the vertical axis, boundaries from the riverbed topography are identified for every  $j$



**Fig. 1:** Example of cross-section discretization with finite element and riverbed boundaries identification according to a water depth  $j$ .

iteration, hence defining new boundaries to be reached before the water can overflow to the next height of the cross-section for each side. To this end, first, in the vector  $seg_y$  is identified between of which pair of coordinates or segments  $j$  belongs to. It is worth mentioning that through this method many segments could be considered, as shown in Fig. 1 where  $j$  intersect segments 1, 8, 10, and 11, to solve this, the pairs of horizontal coordinates from those segments in  $seg_x$  are filtered by considering the closer distance of the average of those pair of coordinates related to the station of  $l_y$  for left and right sides, for instance, on the right side the distance  $d_{r1}$  is lower than  $d_{r2}$  and  $d_{r3}$ , for the left side there is just a segment to be considered.

### 1) Flow area and centroid

To calculate any HP is necessary to define which elements in the mesh belong to the flow area, for this, and considering the previous method to find boundaries in the riverbed, the value of 1 is assigned to elements in the flow area, otherwise, 0 is assigned to the left elements in the matrix. To this end, it was defined the function  $f_1$ , which returns the riverbed elevation for a specific station  $k$  within the cross-section, in this case, for every column in the matrix. According to Eq. (1) as shown in Fig. 1, 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. (2)) 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.

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

$$f_2 = x_{i+1} - \left( \frac{y_{i+1} - y}{y_i - y_{i+1}} \right) (x_{i+1} - x_i) \quad (2)$$

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. (1).

## 2) Wetted Perimeter

This procedure is divided into two steps: first, with the  $f_3$  (Eq. (3)) 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. (1)) and (Eq. (2)), thus, knowing the coordinates, the distances are calculated using (Eq. (3)).

$$f_3(i) = \sqrt{(y_i - y_{i+1})^2 + (x_i - x_{i+1})^2} \quad (3)$$

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.

---

### 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**

---

#### 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
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 # --- Preallocate HP --- #
35 area, top, = np.zeros((int(Ymax*res - Ymin*res), 1)), np.zeros((int(Ymax*res - Ymin*res), 1))
36 perimeter_2, y = np.zeros((int(Ymax*res - Ymin*res), 1)), np.zeros((int(Ymax*res - Ymin*res), 1))
37 RH, centroid = np.zeros((int(Ymax*res - Ymin*res), 1)), np.zeros((int(Ymax*res - Ymin*res), 1))
38 con, phi = np.zeros((int(Ymax*res - Ymin*res), 1)), np.zeros((int(Ymax*res - Ymin*res), 1))
39 Q, center = np.zeros((int(Ymax*res - Ymin*res), 1)), np.zeros((int(Ymax*res - Ymin*res), 1))
40 seg_x, seg_y = np.zeros((len(coors[:, 0]) - 1, 2)), np.zeros((len(coors[:, 0]) - 1, 2))
41
42 for i in range(len(coors) - 1):
43     seg_x[i, 0], seg_x[i, 1] = coors[i, 0], coors[i+1, 0]
44     seg_y[i, 0], seg_y[i, 1] = coors[i, 1], coors[i+1, 1]
45
46 def per(i):
47     return math.sqrt(pow(seg_y[i, 0] - seg_y[i, 1], 2) + pow(seg_x[i, 0] - seg_x[i, 1], 2))
48
49 def image_x(i, j): # Function that according to the horizontal position of K, returns the vertical ...
50     image of the segment
51     if seg_y[i, 0] == seg_y[i, 1]: # if there is a vertical wall
52         return (seg_x[i, 0]) - (((seg_y[i, 0] - j)*(seg_x[i, 0] - seg_x[i, 1])) / ((seg_y[i, 0] - seg_y[i, 1] + noise) - (seg_y[i, 1] + seg_y[i, 1] * noise)))
53     return (seg_x[i, 0]) - (((seg_y[i, 0] - j)*(seg_x[i, 0] - seg_x[i, 1])) / (seg_y[i, 0] - seg_y[i, 1]))
54
55 def image_y(i, j): # Function that according to the horizontal position of K, returns the vertical ...
56     image of the segment
57     if seg_x[i, 0] == seg_x[i, 1]: # if there is a horizontal wall
58         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)))
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

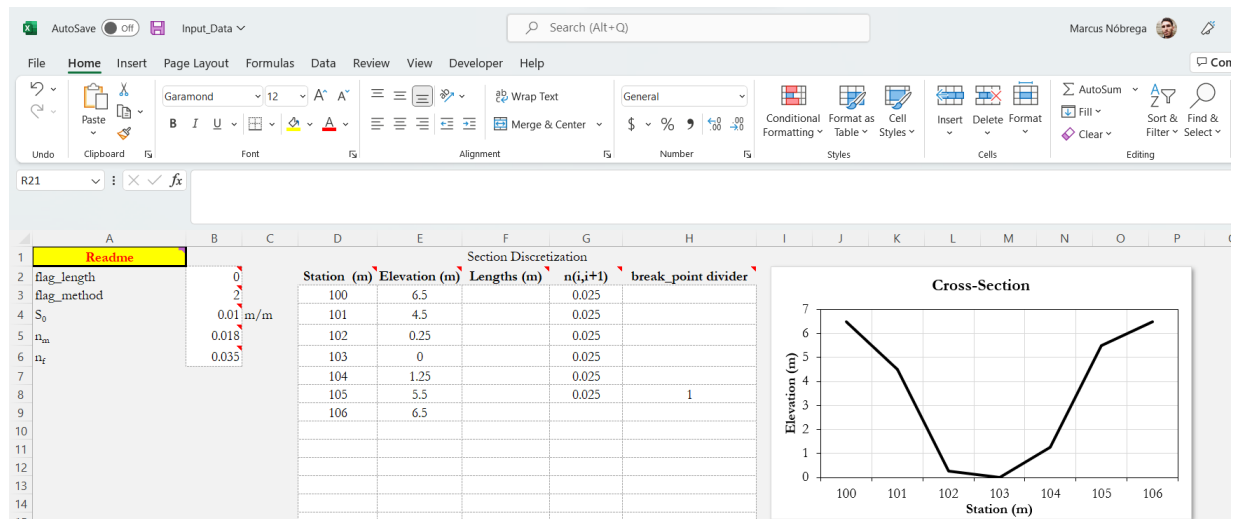
```

```

65         seg_x_2[i, 0] = (seg_x[i, 0]+seg_x[i, 1])/2 - middle
66         left_wall = np.where(seg_x_2 < 0, seg_x_2, -np.inf).argmax() # Finding the walls that contains ...
67         the current
68         right_wall = np.where(seg_x_2 > 0, seg_x_2, np.inf).argmin() # water level
69
70         if left_wall == right_wall: # this condition is meet when water level is higher the profile
71             break
72
73         for i in np.arange(round(image_x(left_wall, j/res)*res) - Xmin*res, # Looping thought the ...
74             horizontal axis
75             round(image_x(right_wall, j/res)*res) - Xmin*res): # Modifying the main grid
76             for k in range(len(seg_x)):
77                 if (seg_x[k, 0] ≤ (i / res + Xmin) < seg_x[k, 1]): # Looking for what segment "i" ...
78                     belongs to.
79                     break
80             mg[round(Ymax*res-j): int(round(Ymax*res)) - int(round(image_y(k, (i/res + Xmin))*res)), ...
81                 int(i)] = 1
82             center[int(j - Ymin*res), 0] = ((np.count_nonzero(mg[:, int(i)] == 1)/2)/res + (image_y(k, ...
83                 (i/res))) * (np.count_nonzero(mg[:, int(i)] == 1)/pow(res, 2))
84
85             perimeter = []
86             for i in range(left_wall,
87                 right_wall): # all segments between the walls but not including they selfs
88                 perimeter.append(per(i))
89                 perimeter.append(math.sqrt(pow(j/res - seg_y[left_wall, 1], 2) +
90                     pow(image_x(left_wall, j/res) - seg_x[left_wall, 1],
91                         2))) # perimeter for the left boundary
92                 perimeter.append(math.sqrt(pow(j/res - seg_y[right_wall, 0], 2) +
93                     pow(image_x(right_wall, j/res) - seg_x[right_wall, 0],
94                         2))) # perimeter for the right boundary
95
96             area[int(j - Ymin*res), 0] = np.sum(mg) / pow(res, 2)
97             y[int(j - Ymin*res), 0] = j / res - Ymin
98             perimeter_2[int(j - Ymin*res), 0] = np.sum(perimeter)
99             RH[int(j - Ymin*res), 0] = (np.sum(mg) / pow(res, 2))/np.sum(perimeter)
100             top[int(j - Ymin*res), 0] = image_x(right_wall, j/res)-image_x(left_wall, j/res)
101             centroid[int(j - Ymin*res), 0] = np.sum(center)/(np.sum(mg))
102             con[int(j - Ymin*res), 0] = (1/man)*(np.sum(mg) / pow(res, 2))*pow((np.sum(mg) / ...
103                 pow(res,2))/(np.sum(perimeter)), 2/3)
104             phi[int(j - Ymin*res), 0] = (np.sum(mg) / pow(res, 2))*pow((np.sum(mg) / ...
105                 pow(res,2))/(np.sum(perimeter)), 2/3)
106             Q[int(j - Ymin*res), 0] = (1/man)*(np.sum(mg) / pow(res, 2))*pow((np.sum(mg) / ...
107                 pow(res,2))/(np.sum(perimeter)), 2/3)*pow(s, 1/2)
108
109 # --- Filling with Nan all extra elements in the arrays --- #
110 area[int(j - Ymin*res): , 0], y[int(j - Ymin*res): , 0], perimeter_2[int(j - Ymin*res): , 0] = ...
111     math.nan, math.nan, math.nan
112 RH[int(j - Ymin*res): , 0], top[int(j - Ymin*res): , 0], centroid[int(j - Ymin*res): , 0] = ...
113     math.nan, math.nan, math.nan
114 con[int(j - Ymin*res): , 0], phi[int(j - Ymin*res): , 0], Q[int(j - Ymin*res): , 0] = math.nan, ...
115     math.nan, math.nan
116 plt.imshow(mg)
117
118 # --- Plotting the HP curves --- #
119 fig, (ax1, ax2, ax3, ax4, ax5, ax6, ax7, ax8) = plt.subplots(1, 8)
120 fig.suptitle('2b')
121 ax1.plot(area, y)
122 ax1.set_xlabel('Area $(m^2)$')
123 ax1.set_ylabel('water depth $(m)$')
124 ax2.plot(perimeter_2, y)
125 ax2.set_xlabel('Perimeter $(m)$')
126 ax3.plot(top, y)
127 ax3.set_xlabel('Top lenght $(m)$')
128 ax4.plot(RH, y)
129 ax4.set_xlabel('Hydraulic radius $(m)$')
130 ax5.plot(centroid, y)
131 ax5.set_xlabel('Centroid $(m)$')
132 ax6.plot(con, y)
133 ax6.set_xlabel('Conveyance $(m^3/s)$')
134 ax7.plot(phi, y)
135 ax7.set_xlabel('Phi $(m^3/s)$')
136 ax8.plot(Q, y)
137 ax8.set_xlabel('Flow $(m^3/s)$')

```





**Fig. 2:** 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.

#### D. 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  %%% STATUS %%%
7  % Model working
8
9  %% Read Input Data
10 clear all
11 input_table = xlsread('Input_Data.xlsx');
12 input_data = input_table(1:5,1);
13 input_data_coordinates = input_table(2:end,3:end);
14 flag_length = input_data(1,1); % If == 1, use lengths as main input data, otherwise use absolute ...
    values of x (m)
15 flag_method = input_data(2,1); % If == 1, SCM, else DCM
16 s0 = input_data(3,1); % Slope in m/m
17 nm = input_data(4,1); % Main channel roughness
18 nf = input_data(5,1); % Overbanks channel roughness
19
20 if flag_method == 1
21     n_channel = input_data_coordinates(1:(end-1),4);
22 end

```

```

23
24 % Retrieving Data
25 x_absolute = input_data_coordinates(:,1);
26 elevations = input_data_coordinates(:,2);
27 lengths = input_data_coordinates(1:(end-1),3);
28 break_point_divider = input_data_coordinates(1:(end),5);
29
30 Δ = zeros(length(elevations),1);
31 for i = 1:(length(elevations)-1)
32     Δ(i) = abs(elevations(i+1,1) - elevations(i,1));
33 end
34 Δ_h = min(Δ(Δ > 0));
35 tic
36
37 % Checking input data consistency
38 if length(elevations) ≤ 3
39     error('Please, enter at least 4 points for elevation and 3 points for manning and lengths. If ...
        you have a triangular shape, please enter the invert elevation twice and add a 0 length and ...
        0 manning, such that you have 4 points for elevation and 3 points for manning and lengths)');
40 end
41
42 points = (1:1:length(elevations))'; % stations from 1 to n
43
44 % Let's assume a maximum 1 cm difference in the depths
45 % Noise
46 dh = 1; % cm
47 noise = Δ_h/1000; % Noise in m
48 factor = 1; %precision = 1/factor * noise
49
50 [au,ia] = unique(elevations,'stable');
51 Same = ones(size(elevations));
52 Same(ia) = 0; % repetitive values
53 noise_i = rand(1,1)*noise;
54 small_number = noise/100;
55 % New Elevation and X_values
56 ii = 0;
57 for i = 1:(length(elevations) - 1)
58     e1 = elevations(i); e2 = elevations(i+1);
59     x1 = x_absolute(i); x2 = x_absolute(i+1);
60     if e1 == e2
61         elevations(i+1) = elevations(i+1) + noise;
62     end
63     if x1 == x2
64         x_absolute(i+1) = x_absolute(i+1) + noise;
65     end
66 end
67
68 % if max(isnan(n_channel)) > 0
69 %     error('Please, enter (n-1) data for Manning coefficient, where n is the number of break-points')
70 % end
71
72 % Roughness Boundary Condition
73 if flag_method == 1
74     n_channel(end+1,1) = 0; % adding last boundary condition
75 end
76
77 % Minimum elevation
78 min_el = min(elevations); % m
79 % y (bottom to up)
80 y = elevations - min_el;
81 pos_inv = find(y == 0); % position of invert elevation
82 % If we have more than 1 invert
83 pos_inv = pos_inv(1);
84
85 % x (left to right)
86 if flag_length == 1
87     for i = 1:length(y) % coordinates of each measured point
88         if i == 1
89             x_absolute(i,1) = 0 + noise;
90         else
91             x_absolute(i,1) = x_absolute(i-1) + lengths(i-1) + noise;
92         end
93     end
94 else % Lengths are already assumed from the input data table
95     for i = 1:length(y)
96         if i ≠ length(y)
97             lengths(i) = x_absolute(i+1) - x_absolute(i);

```

```

98         end
99     end
100 end
101
102 % Alfa min
103 alfa_min_bound = noise/max(lengths(lengths>1e-8));
104 big_n = 100000*atan(asin(1)); % big number making sure it is a multiple of 1 rad, so that ...
105     sin(atan(big_n)) = 1
106 min_length = min(lengths(lengths>0));
107
108 % Invert coordinates
109 x_invert = x_absolute(pos_inv,1);
110 y_invert = 0;
111
112 % Slopes (taking from x (left-right) y (down-up)
113 % For point 1 and for the last point
114 alfa_l = (y(1,1) - y(2,1))/lengths(1,1);
115
116 % Unsorted Values
117 x_left_unsorted = x_absolute(1:(pos_inv-1),1);
118 y_left_unsorted = y(1:(pos_inv-1),1);
119 x_right_unsorted = x_absolute(pos_inv + 1:end,1);
120 y_right_unsorted = y(pos_inv + 1:end,1);
121 if flag_method == 1
122     n_left_unsorted = n_channel(1:(pos_inv-1),1);
123     n_right_unsorted = n_channel(pos_inv:(end-1),1);
124 end
125
126 % Maximum depth (left and right)
127 max_left = max(y_left_unsorted); max_right = max(y_right_unsorted);
128 max_y = min(max_left, max_right);
129
130 % Refreshing values of ymax
131 pos_r = length(y_right_unsorted);
132 if max_left == max_right
133     if max_left > max_y % the maximum is located at left
134         z = sort(y_left_unsorted,1,'descend');
135         if length(z) == 1 % Case where we have a vertical wall
136             z(2,1) = y_invert;
137         end
138         x_left_first = round(x_absolute(2) - (max_y - z(2))/alfa_l,2);
139         % New values of x and y
140         x_absolute(1) = x_left_first;
141         y(1) = max_y;
142         pos_r = length(y_right_unsorted);
143     else
144         pos_r = find(y_right_unsorted > max_y ,1,'first');
145         alfa_r = (y_right_unsorted(pos_r) - y_right_unsorted(pos_r - ...
146             1))/lengths(length(y_left_unsorted) + 1 + pos_r-1);
147         z = sort(y_right_unsorted,1,'descend');
148         x_rigth_last = round(x_absolute(end-1) + (max_y - z(2))/alfa_r,2);
149         % New values of x and y
150         x_absolute(end) = x_rigth_last;
151         y(length(y_left_unsorted) + 1 + pos_r) = max_y;
152     end
153 end
154 dim = 1:(length(y_left_unsorted) + 1 + pos_r);
155 y = y(dim,1);
156 x_absolute = x_absolute(dim,1);
157 % n_channel = n_channel(dim,1);
158 points = points(dim);
159
160 % New Unsorted Values with New max
161 x_left_unsorted = x_absolute(1:(pos_inv-1),1);
162 y_left_unsorted = y(1:(pos_inv-1),1);
163 x_right_unsorted = x_absolute(pos_inv + 1:end,1);
164 y_right_unsorted = y(pos_inv + 1:end,1);
165 if flag_method == 1
166     n_left_unsorted = n_channel(1:(pos_inv-1),1);
167     n_right_unsorted = n_channel(pos_inv:(end-1),1);
168 end
169
170 % Main Matrix
171 % table = [points,x_absolute,y,n_channel];
172
173 % % Vlookup Function
174 % Vlookup_eq = @(data,col1,va11,col2) data((find(data(:,col1)==va11,1)),col2); %Vlookup function as ...

```

```

Excel
173 % Vlookup_leq = @(data,col1,vall,col2) data((find(data(:,col1)≤vall,1)),col2); %Vlookup function as ...
Excel
174
175 % Sections left
176 numb_left = length(find(y_left_unsorted ≥ y_left_unsorted(end)));
177 % Sections right
178 numb_right = length(find(y_right_unsorted ≥ y_right_unsorted(1)));
179 % Tot sections
180 tot_sections = numb_left + numb_right - 1; % take one out because both sides are equal
181
182 y_l_prev = y_left_unsorted(2:length(y_left_unsorted));
183 y_l_next = y_left_unsorted(1:(length(y_left_unsorted)-1));
184
185 %%% Precision
186 precision = 1/factor*noise; % m
187
188 %%% small number ≥ 1 < 1e-8 + 1
189 sm = (1e-8 + 1);
190
191 %%% Total_Noise
192 tot_noise = noise*sum(Same);
193 % Main loop
194 i = 0; int_n_p = 0; % integral of n*perimeter
195
196 %% Define Main Channel and Overbanks
197 pos_break = find(break_point_divider == 1); % Position where the divider occurs
198 % Main Channel Height
199 ym = y(pos_break); % Main channel height (m)
200 if pos_break > pos_inv % Left intersection
201     % Left intersection
202     posm_left = find(y_left_unsorted ≥ ym,1,'last');
203     ym_left_up = y_left_unsorted(posm_left);
204     xm_left_up = x_left_unsorted(posm_left);
205     ym_left_down = y_left_unsorted(min(posm_left+1,length(y_left_unsorted)));
206     xm_left_down = x_left_unsorted(min(posm_left+1,length(y_left_unsorted)));
207     % Angles
208     if (ym_left_up - ym_left_down ≤ length(y_left_unsorted)*noise)
209         alfa_m_l = big_n;
210     else
211         alfa_m_l = (ym_left_up - ym_left_down)/(xm_left_down - xm_left_up); % Slope
212     end
213     xm_left = xm_left_down - (ym - ym_left_down)/alfa_m_l;
214     ym_left = ym;
215     % Polygons (left - inv - right)
216     x_pol = [xm_left; x_left_unsorted((posm_left + 1:end),1); x_invert; ...
        x_right_unsorted(1:(pos_break-pos_inv),1)];
217     y_pol = [ym_left; y_left_unsorted((posm_left + 1:end),1); y_invert; ...
        y_right_unsorted(1:(pos_break-pos_inv),1)];
218     % Top-Width
219     bm = abs(x_pol(1) - x_pol(end));
220     % Area
221     am = polyarea(x_pol,y_pol);
222     % Perimeter
223     polyin = polyshape(x_pol,y_pol);
224     pm = perimeter(polyin) - bm; % Taking away the top width
225 else
226     % Right Intersection
227     posm_right = find(y_right_unsorted ≥ ym,1,'first');
228     ym_right_up = y_right_unsorted(posm_right);
229     xm_right_up = x_right_unsorted(posm_right);
230     ym_right_down = y_right_unsorted(max(posm_right-1,1));
231     xm_right_down = x_right_unsorted(max(posm_right-1,1));
232     % Angles
233     if (ym_right_up - ym_right_down < noise*length(y_right_unsorted)) % No depth
234         alfa_m_r = big_n;
235     else
236         alfa_m_r = (ym_right_up - ym_right_down)/(xm_right_up - xm_right_down); % Slope
237     end
238     xm_right = xm_right_down + (ym - ym_right_down)/alfa_m_r;
239     ym_right = ym;
240     % Polygons (left - inv - right)
241     x_pol = [x_left_unsorted(pos_break:end,1); x_invert; x_right_unsorted(1:(posm_right - 1),1); ...
        xm_right];
242     y_pol = [y_left_unsorted(pos_break:end,1); y_invert; y_right_unsorted(1:(posm_right - 1),1); ...
        ym_right];
243     % Top-Width

```

```

244     bm = abs(x_pol(1) - x_pol(end));
245     % Area
246     am = polyarea(x_pol,y_pol);
247     % Perimeter
248     polyin = polyshape(x_pol,y_pol);
249     pm = perimeter(polyin) - bm; % Taking away the top width
250 end
251 if flag_method ~= 1
252     % Number of floodplains
253     if pos_break == 1 || pos_break == length(y)
254         n_fp = 1;
255     else
256         n_fp = 2;
257     end
258 end
259 while i < big_n
260     %% Case where i == 1
261     i = i + 1;
262     n_P_left = 0;
263     n_P_right = 0;
264     n_P_left_extra = 0;
265     n_P_right_extra = 0;
266     B_extra = 0;
267     P_extra = 0;
268     P_extra_left = 0;
269     P_extra_right = 0;
270     if i == 1 % We are talking about the first point
271
272         %% Initializing variables
273         y_table = 0; h = 0; B = 0; A = 0; Rh = 0; P = 0; Phi = 0; K_c = 0;
274         % Look to both sides from pos_inv (invert point)
275
276         % Left Direction
277         pos_left = find(y_left_unsorted>sm*y_invert,1,'last');
278         y_left_point = y_left_unsorted(pos_left,1);
279         x_left_point = x_left_unsorted(pos_left,1);
280         if flag_method == 1
281             n_left_segment = n_left_unsorted(pos_right,1);
282         else
283             n_left_segment = nm; % Main channel
284         end
285
286         % Right Direction
287         pos_right = find(y_right_unsorted>sm*y_invert,1,'first');
288         y_right_point = y_right_unsorted(pos_right,1);
289         x_right_point = x_right_unsorted(pos_right,1);
290         if flag_method == 1
291             n_right_segment = n_right_unsorted(pos_right,1);
292         else
293             n_right_segment = nm; % Main channel
294         end
295
296         % Angles Calculations
297         % Alfa Left
298         % Case 01 - Vertical Point
299         if (x_invert - x_left_point <= tot_noise) && (y_left_point - y_invert > tot_noise)
300             alfa_l = big_n;
301             alfa_l_tang = big_n;
302         end
303         % Case 02 - Horizontal Point
304         if (x_invert - x_left_point > tot_noise) && (y_left_point - y_invert <= tot_noise)
305             alfa_l = big_n;
306             alfa_l_tang = big_n;
307         end
308         % Case 03 - Horizontal and Vertical Point
309         if (x_invert - x_left_point <= tot_noise) && (y_left_point - y_invert <= tot_noise)
310             alfa_l = big_n;
311             alfa_l_tang = big_n;
312         end
313         % Case 04 - Point with normal slopes
314         if (x_invert - x_left_point > tot_noise) && (y_left_point - y_invert > tot_noise)
315             alfa_l = (y_left_point - y_invert)/(x_invert - x_left_point);
316             alfa_l_tang = alfa_l;
317         end
318
319         % Alfa Right
320         % Case 01 - Vertical Point

```

```

321     if (x_right_point - x_invert ≤ tot_noise) && (y_right_point - y_invert > tot_noise)
322         alfa_r = big_n;
323         alfa_r_tang = big_n;
324     end
325     % Case 02 - Horizontal Point
326     if (x_right_point - x_invert > tot_noise) && (y_right_point - y_invert ≤ tot_noise)
327         alfa_r = big_n;
328         alfa_r_tang = big_n;
329     end
330     % Case 03 - Horizontal and Vertical Point
331     if (x_right_point - x_invert ≤ tot_noise) && (y_right_point - y_invert ≤ tot_noise)
332         alfa_r = big_n;
333         alfa_r_tang = big_n;
334     end
335     % Case 04 - Point with normal slopes
336     if (x_right_point - x_invert > tot_noise) && (y_right_point - y_invert > tot_noise)
337         alfa_r = (y_right_point - y_invert)/(x_right_point - x_invert);
338         alfa_r_tang = alfa_r;
339     end
340
341     % Min Angle
342     if alfa_l ≤ alfa_min_bound
343         alfa_l_tang = big_n;
344     end
345     if alfa_r ≤ alfa_min_bound
346         alfa_r_tang = big_n;
347     end
348
349     if y_left_point ≤ y_right_point
350         y_moving = y_left_point;
351         x_left_point = x_absolute(pos_inv - 1,1);
352         precision_section = min(y_left_point - y_invert, precision);
353         n_points = floor((y_left_point - y_invert)/(precision_section)); % number of ...
354         interpolated points
355         if n_points == 1 % only one point means no slope
356             if x_invert - x_left_point ≥ sm*noise && alfa_l == big_n
357                 P_extra_left = sqrt((x_invert - x_left_point)^2 + (y_invert - y_left_point)^2);
358                 n_P_left_extra = P_extra_left*n_left_segment^(3/2);
359                 B_extra = (x_invert - x_left_point);
360             else
361                 B_extra = 0;
362                 n_P_left_extra = 0;
363                 P_extra_left;
364             end
365         end
366         if n_points == 1 % only one point means no slope
367             if x_right_point - x_invert > 1.0001*noise && alfa_r == big_n
368                 P_extra_right = sqrt((x_invert - x_right_point)^2 + (y_invert - ...
369                     y_right_point)^2) + B_extra;
370                 B_extra = B_extra + (x_right_point - x_invert);
371                 n_P_right_extra = (P_extra_right)*n_right_segment^(3/2);
372             else
373                 n_P_left_extra = 0;
374                 P_extra_right = 0;
375             end
376         end
377         P_extra = P_extra_right + P_extra_left;
378
379         % Main loop for i == 1
380         for j = 1:(n_points)
381             h = precision_section;
382             y_table(j+1,1) = y_table(j,1) + h;
383             B(j+1,1) = h/alfa_l_tang + h/alfa_r_tang + B(j,1);
384             A(j+1,1) = (B(j+1,1) + B(j,1))*h/2 + A(j,1); % Trapezoid
385             P(j+1,1) = h/sin(atan(alfa_l_tang)) + h/sin(atan(alfa_r_tang)) + P(j,1);
386             Rh(j+1,1) = A(j+1,1)/P(j+1,1);
387             Phi(j+1,1) = A(j+1,1)*Rh(j+1,1)^(2/3);
388             int_n_p = n_P_left_extra + n_P_right_extra + ...
389                 n_left_segment^(3/2)*h/sin(atan(alfa_l_tang)) + ...
390                 n_right_segment^(3/2)*h/sin(atan(alfa_r_tang)) + int_n_p;
391             % Representative Roughness Coefficient
392             if flag_method == 1
393                 n_med(j+1,1) = (int_n_p/P(j+1,1))^(2/3);
394             else
395                 if y_table(j+1,1) > ym
396                     yf = max(y_table(j+1,1) - ym, 0); % Overbank depth
397                     af = max(A(j+1,1) - (am + bm*yf), 0); % Overbank flow area

```

```

394         pf = max(P(j+1,1) - pm,0); % Floodplain perimeter (m)
395         pm_star = max(pm + n_fp*yf,0);
396         am_star = max(am + bm*yf,0);
397         n_med(j+1,1) = (Phi(j+1,1))/(1/nf*af*(af/pf)^(2/3) + ...
398             1/nm*am_star*(am_star/pm_star)^(2/3));
399     else
400         yf = 0; % Overbank depth
401         af = 0; % Overbank flow area
402         pf = 0; % Floodplain perimeter (m)
403         pm_star = 0;
404         am_star = 0;
405         n_med(j+1,1) = nm;
406     end
407     K_c(j+1,1) = 1/n_med(j+1,1)*Phi(j+1,1);
408
409     if j == (n_points) % final point
410         % Final point - make sure you have the exact surveyed point at the end
411         h_ = y_right_point - y_table(j,1);
412         y_table(j+1,1) = y_table(j,1) + h_;
413         B(j+1,1) = h_/alfa_l_tang + h_/alfa_r_tang + B(j,1) + B_extra;
414         A(j+1,1) = (B(j+1,1) + B(j,1))*h/2 + A(j,1); % Trapezoid
415         P(j+1,1) = h_/sin(atan(alfa_l_tang)) + h_/sin(atan(alfa_r_tang)) + P(j,1) + ...
416             P_extra;
417         Rh(j+1,1) = A(j+1,1)/P(j+1,1);
418         Phi(j+1,1) = A(j+1,1)*Rh(j+1,1)^(2/3);
419         if n_points == 1
420             int_n_p = n_left_segment^(3/2)*h/sin(atan(alfa_l_tang)) + ...
421                 n_right_segment^(3/2)*h/sin(atan(alfa_r_tang)) + n_P_right_extra + ...
422                 n_P_left_extra;
423         else
424             int_n_p = n_left_segment^(3/2)*h/sin(atan(alfa_l_tang)) + ...
425                 n_right_segment^(3/2)*h/sin(atan(alfa_r_tang)) + int_n_p;
426         end
427         % Representative Roughness Coefficient
428         if flag_method == 1
429             n_med(j+1,1) = round((int_n_p/P(j+1,1))^(2/3),3);
430         else
431             if y_table(j+1,1) > ym
432                 yf = max(y_table(j+1,1) - ym,0); % Overbank depth
433                 af = max(A(j+1,1) - (am + bm*yf),0); % Overbank flow area
434                 pf = max(P(j+1,1) - pm,0); % Floodplain perimeter (m)
435                 pm_star = max(pm + n_fp*yf,0);
436                 am_star = max(am + bm*yf,0);
437                 n_med(j+1,1) = round((Phi(j+1,1))/(1/nf*af*(af/pf)^(2/3) + ...
438                     1/nm*am_star*(am_star/pm_star)^(2/3)),3);
439             else
440                 yf = 0; % Overbank depth
441                 af = 0; % Overbank flow area
442                 pf = 0; % Floodplain perimeter (m)
443                 pm_star = 0;
444                 am_star = 0;
445                 n_med(j+1,1) = nm;
446             end
447         end
448         K_c(j+1,1) = 1/n_med(j+1,1)*Phi(j+1,1);
449     end
450 end
451 else
452     x_right_point = x_absolute(pos_inv + 1,1);
453     precision_section = min(y_right_point - y_invert,precision);
454     n_points = floor((y_right_point - y_invert)/(precision_section)); % number of ...
455     interpolated points
456     if n_points == 1 % only one point means no slope
457         if x_right_point - x_invert >= sm*noise && alfa_r == big_n % Additional B_extra
458             P_extra = sqrt((x_right_point - x_invert)^2 + (y_right_point - y_invert)^2);
459             B_extra = x_right_point - x_invert;
460             n_P_right_extra = P_extra*n_right_segment^(3/2);
461         else
462             B_extra = 0;
463             n_P_right_extra = 0;
464             P_extra = 0;
465         end
466     end
467     y_moving = y_right_point;
468     % For loop to calculate functions
469     for j = 1:(n_points)

```

```

464 h = precision_section;
465 B(j+1,1) = h/alfa_l_tang + h/alfa_r_tang + B(j,1);
466 y_table(j+1,1) = y_table(j,1) + h;
467 A(j+1,1) = (B(j+1,1) + B(j,1))*h/2 + A(j,1); % Trapezoid
468 P(j+1,1) = h/sin(atan(alfa_l_tang)) + h/sin(atan(alfa_r_tang)) + P(j,1);
469 Rh(j+1,1) = A(j+1,1)/P(j+1,1);
470 Phi(j+1,1) = A(j+1,1)*Rh(j+1,1)^(2/3);
471 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;
472 % Representative Roughness Coefficient
473 if flag_method == 1
474     n_med(j+1,1) = round((int_n_p/P(j+1,1))^(2/3),3);
475 else
476     if y_table(j+1,1) > ym
477         yf = max(y_table(j+1,1) - ym,0); % Overbank depth
478         af = max(A(j+1,1) - (am + bm*yf),0); % Overbank flow area
479         pf = max(P(j+1,1) - pm,0); % Floodplain perimeter (m)
480         pm_star = max(pm + n_fp*yf,0);
481         am_star = max(am + bm*yf,0);
482         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));
483     else
484         yf = 0; % Overbank depth
485         af = 0; % Overbank flow area
486         pf = 0; % Floodplain perimeter (m)
487         pm_star = 0;
488         am_star = 0;
489         n_med(j+1,1) = nm;
490     end
491 end
492 K_c(j+1,1) = 1/n_med(j+1,1)*Phi(j+1,1);
493 if j == (n_points) % final point
494     % Final point - make sure you have the exact surveyed point at the end
495     h_ = y_right_point - y_table(j,1);
496     y_table(j+1,1) = y_table(j,1) + h_;
497     B(j+1,1) = h_/alfa_l_tang + h_/alfa_r_tang + B(j,1) + B_extra;
498     A(j+1,1) = (B(j+1,1) + B(j,1))*h/2 + A(j,1); % Trapezoid
499     P(j+1,1) = h_/sin(atan(alfa_l_tang)) + h_/sin(atan(alfa_r_tang)) + P(j,1) + ...
         P_extra;
500     Rh(j+1,1) = A(j+1,1)/P(j+1,1);
501     Phi(j+1,1) = A(j+1,1)*Rh(j+1,1)^(2/3);
502     if n_points == 1
503         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;
504     else
505         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;
506     end
507     % Representative Roughness Coefficient
508     if flag_method == 1
509         n_med(j+1,1) = (int_n_p/P(j+1,1))^(2/3);
510     else
511         if y_table(j+1,1) > ym
512             yf = max(y_table(j+1,1) - ym,0); % Overbank depth
513             af = max(A(j+1,1) - (am + bm*yf),0); % Overbank flow area
514             pf = max(P(j+1,1) - pm,0); % Floodplain perimeter (m)
515             pm_star = max(pm + n_fp*yf,0);
516             am_star = max(am + bm*yf,0);
517             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));
518         else
519             yf = 0; % Overbank depth
520             af = 0; % Overbank flow area
521             pf = 0; % Floodplain perimeter (m)
522             pm_star = 0;
523             am_star = 0;
524             n_med(j+1,1) = nm;
525         end
526     end
527     K_c(j+1,1) = 1/n_med(j+1,1)*Phi(j+1,1);
528 end
529 end
530 end
531 % Previous Positions
532 pos_left_previous = pos_left;

```



```

533     pos_right_previous = pos_right;
534 else
535     %% Case where i ≠ 1
536
537     % Look to left sides from x_point_left and from right side of
538     % x_point_right
539     y_moving = y_table(end,1); % actual water depth
540
541     % Left Direction
542     pos_left = find(y_left_unsorted>sm*y_moving,1,'last');
543     y_left_point = y_left_unsorted(pos_left,1);
544     x_left_point = x_left_unsorted(pos_left,1);
545
546     % Right Direction
547     pos_right = find(y_right_unsorted>sm*y_moving,1,'first');
548     y_right_point = y_right_unsorted(pos_right,1);
549     x_right_point = x_right_unsorted(pos_right,1);
550
551     % Roughness
552     if y_moving ≤ ym % Inside of the channel
553         if flag_method == 1
554             n_left_segment = n_left_unsorted(pos_left,1);
555             n_right_segment = n_right_unsorted(pos_right,1);
556         else
557             if (abs(y_left_unsorted(pos_left) - ym) ≤ noise*length(y_left_unsorted))
558                 n_left_segment = nf; % Attention here
559             else
560                 n_left_segment = nm; % Attention here
561             end
562             if (abs(y_right_unsorted(pos_right) - ym) ≤ noise*length(y_right_unsorted))
563                 n_right_segment = nf; % Attention here
564             else
565                 n_right_segment = nm; % Attention here
566             end
567         end
568     else % Overbanks
569         if flag_method == 1
570             n_left_segment = n_left_unsorted(pos_left,1);
571             n_right_segment = n_right_unsorted(pos_right,1);
572         elseif y_left_unsorted(pos_left) - ym < noise*length(y_left_unsorted) % Check Noises
573             n_left_segment = nm; % Attention here
574             n_right_segment = nm; % Attention here
575         else
576             n_left_segment = nf; % Attention here
577             n_right_segment = nf; % Attention here
578         end
579     end
580
581     % Checking Discontinuities
582     %% Initializing Variables
583     Delta_Area_left = 0; Delta_Area_right = 0;
584     Delta_B_left = 0; Delta_B_right = 0;
585     Delta_P_left = 0; Delta_P_right = 0;
586
587     % Angles Calculation
588     if pos_left + 1 > length(y_left_unsorted)
589         x_prev_left = x_invert;
590         y_prev_left = y_invert;
591     else
592         x_prev_left = (x_left_unsorted(pos_left + 1,1));
593         y_prev_left = (y_left_unsorted(pos_left + 1,1));
594     end
595
596     %% Alfa Left
597     % Case 01 - Vertical Point
598     if (x_prev_left - x_left_point ≤ tot_noise) && (y_left_point - y_prev_left > tot_noise)
599         alfa_l = big_n;
600         alfa_l_tang = big_n;
601     end
602     % Case 02 - Horizontal Point
603     if (x_prev_left - x_left_point > tot_noise) && (y_left_point - y_prev_left ≤ tot_noise)
604         alfa_l = big_n;
605         alfa_l_tang = big_n;
606     end
607     % Case 03 - Horizontal and Vertical Point
608     if (x_prev_left - x_left_point ≤ tot_noise) && (y_left_point - y_prev_left ≤ tot_noise)
609

```

```

610         alfa_l = big_n;
611         alfa_l_tang = big_n;
612     end
613     % Case 04 - Poit with normal slopes
614     if (x_prev_left - x_left_point > tot_noise) && (y_left_point - y_prev_left > tot_noise)
615         alfa_l = (y_left_point - y_prev_left)/(x_prev_left - x_left_point);
616         alfa_l_tang = alfa_l;
617     end
618     if pos_right == 1
619         x_prev_right = x_invert;
620         y_prev_right = y_invert;
621     else
622         x_prev_right = x_right_unsorted(pos_right - 1,1);
623         y_prev_right = y_right_unsorted(pos_right - 1,1);
624     end
625     %%% Alfa Right %%%
626     % Case 01 - Vertical Point
627     if (x_right_point - x_prev_right ≤ tot_noise) && (y_right_point - y_prev_right > tot_noise)
628         alfa_r = big_n;
629         alfa_r_tang = big_n;
630     end
631     % Case 02 - Horizontal Point
632     if (x_right_point - x_prev_right > tot_noise) && (y_right_point - y_prev_right ≤ tot_noise)
633         alfa_r = big_n;
634         alfa_r_tang = big_n;
635     end
636     % Case 03 - Horizontal and Vertical Point
637     if (x_right_point - x_prev_right ≤ tot_noise) && (y_right_point - y_prev_right ≤ tot_noise)
638         alfa_r = big_n;
639         alfa_r_tang = big_n;
640     end
641     % Case 04 - Poit with normal slopes
642     if (x_right_point - x_prev_right > tot_noise) && (y_right_point - y_prev_right > tot_noise)
643         alfa_r = (y_right_point - y_prev_right)/(x_right_point - x_prev_right);
644         alfa_r_tang = alfa_r;
645     end
646
647     % Min Angle
648     if alfa_l ≤ alfa_min_bound
649         alfa_l_tang = big_n;
650     end
651     if alfa_r ≤ alfa_min_bound
652         alfa_r_tang = big_n;
653     end
654
655
656     if (pos_left_previous - pos_left) > 1 % More than one movement
657
658         % intersect
659         if alfa_l_tang == 0
660             x_intersect = x_left_unsorted(pos_left + 1,1);
661         else
662             x_intersect = x_left_unsorted(pos_left + 1,1) - (y_moving - ...
663                 y_left_unsorted(pos_left + 1,1))/alfa_l;
664
665         end
666         x_pol = []; y_pol = [];
667         for nn = 1:(pos_left_previous - pos_left)
668             x_pol = [x_pol; x_left_unsorted(pos_left_previous - nn + 1)];
669             y_pol = [y_pol; y_left_unsorted(pos_left_previous - nn + 1)];
670         end
671         % Adding intersection
672         x_pol = [x_pol; x_intersect];
673         y_pol = [y_pol; y_moving];
674         % Delta B
675         Delta_B_left = abs(x_pol(1) - x_pol(end));
676         % Delta A
677         Delta_Area_left = polyarea(x_pol,y_pol);
678         % Delta P
679         polyin = polyshape(x_pol,y_pol);
680         Delta_P_left = perimeter(polyin) - Delta_B_left; % Taking away top width
681         n_P_left = Delta_P_left*n_left_segment^(3/2);
682         % Delta Rh left
683         % Phi left
684         % Conductance Left
685     end

```

```

686 % Checking Discontinuities
687 if (pos_right - pos_right_previous) > 1 % More than one movement
688     % intersect
689     if alfa_r_tang == 0
690         x_intersect = x_right_unsorted(pos_right - 1,1);
691     else
692         x_intersect = x_right_unsorted(pos_right - 1,1) + (y_moving - ...
693             y_right_unsorted(pos_right - 1,1))/alfa_r;
694     end
695     x_pol = []; y_pol = [];
696     for nn = 1:(pos_right - pos_right_previous)
697         x_pol = [x_pol; x_right_unsorted(pos_right_previous + nn - 1)];
698         y_pol = [y_pol; y_right_unsorted(pos_right_previous + nn - 1)];
699     end
700     % Adding intersection
701     x_pol = [x_pol;x_intersect];
702     y_pol = [y_pol;y_moving];
703     % Delta B
704     Delta_B_right = abs(x_pol(1) - x_pol(end));
705     % Delta A
706     Delta_A_right = polyarea(x_pol,y_pol);
707     % Delta P
708     polyin = polyshape(x_pol,y_pol);
709     Delta_P_right = perimeter(polyin) - Delta_B_right; % Taking away top width
710     % Manning * Perimeter
711     n_P_right = Delta_P_right*n_right_segment^(3/2);
712 end
713 y_moving_end = min(y_right_point,y_left_point);
714 % if (y_moving_end - y_moving)/(precision/100) < 1
715 %     error('Please, increase precision. Instability!')
716 % end
717 precision_section = min(y_moving_end - y_moving,precision); % meters
718 if y_moving_end - y_moving < precision
719     ttt = 1;
720 end
721 n_points = floor((y_moving_end - y_moving)/(precision_section)); % number of interpolated ...
722 points
723 % For loop to calculate functions
724 if n_points == 1 % only one point means no slope
725     if y_moving_end == y_right_point && y_moving_end == y_left_point && alfa_l == big_n && ...
726         alfa_r == big_n
727         B_extra = x_right_point - x_prev_right + x_prev_left - x_left_point;
728         P_extra_left = sqrt((x_prev_left - x_left_point)^2 + (y_prev_left - y_left_point)^2);
729         P_extra_right = sqrt((x_right_point - x_prev_right)^2 + (y_right_point - ...
730             y_prev_right)^2);
731     elseif y_moving_end == y_right_point && alfa_r == big_n
732         if pos_right == 1
733             P_extra_right = sqrt((x_right_point - x_invert)^2 + (y_right_point - y_invert)^2);
734             B_extra = x_right_point - x_invert;
735         else
736             P_extra_right = sqrt((x_right_point - x_prev_right)^2 + (y_right_point - ...
737                 y_prev_right)^2);
738             B_extra = x_right_point - x_prev_right;
739         end
740     else % y_moving == y_left
741         if pos_left + 1 > length(x_left_unsorted) && alfa_l == big_n
742             P_extra_left = sqrt((x_invert - x_left_point)^2 + (y_invert - y_left_point)^2);
743             B_extra = x_invert - x_left_point;
744         elseif alfa_l == big_n
745             P_extra_left = sqrt((x_prev_left - x_left_point)^2 + (y_prev_left - ...
746                 y_left_point)^2);
747             B_extra = x_prev_left - x_left_point;
748         end
749     % Right
750     if pos_right == 1 && alfa_r == big_n
751         P_extra_right = sqrt((x_invert - x_right_point)^2 + (y_invert - y_right_point)^2);
752         B_extra = x_right_point - x_invert + B_extra;
753     elseif alfa_r == big_n
754         P_extra_left = sqrt((x_prev_right - x_right_point)^2 + (y_right_point - ...
755             y_prev_right)^2);
756         B_extra = x_right_point - x_prev_right + B_extra;
757     end
758 end
759 P_extra = P_extra_left + P_extra_right;
760 n_P_right_extra = P_extra_right*n_right_segment^(3/2);
761 n_P_left_extra = P_extra_left*n_left_segment^(3/2);

```

```

756     else
757         B_extra = 0;
758         n_P_right_extra = 0;
759         n_P_left_extra = 0;
760         P_extra = 0;
761         P_extra_left = 0;
762         P_extra_right = 0;
763     end
764
765     dim_table = length(y_table);
766     %%%%%%%%%%% Main loop for i ≠ 1 %%%%%%%%%%%
767
768     for j = 1:(n_points)
769         k = dim_table + j;
770         if j == 1 % We have to add values from discontinuity (Deltas)
771             h = precision_section; % meters
772             y_table(k,1) = y_table(k-1,1) + h;
773             % Roughness
774             if y_table(k,1) ≤ ym % Inside of the channel
775                 if flag_method == 1
776                     n_left_segment = n_left_unsorted(pos_left,1);
777                     n_right_segment = n_right_unsorted(pos_right,1);
778                 else
779                     if (abs(y_left_unsorted(pos_left) - ym) ≤ noise*length(y_left_unsorted))
780                         n_left_segment = nf; % Attention here
781                     else
782                         n_left_segment = nm; % Attention here
783                     end
784                     if (abs(y_right_unsorted(pos_right) - ym) ≤ noise*length(y_right_unsorted))
785                         n_right_segment = nf; % Attention here
786                     else
787                         n_right_segment = nm; % Attention here
788                     end
789                 end
790             else % Overbanks
791                 if flag_method == 1
792                     n_left_segment = n_left_unsorted(pos_left,1);
793                     n_right_segment = n_right_unsorted(pos_right,1);
794                 elseif y_left_unsorted(pos_left) - ym < noise*length(y_left_unsorted) % Check Noises
795                     n_left_segment = nm; % Attention here
796                     n_right_segment = nm; % Attention here
797                 else
798                     n_left_segment = nf; % Attention here
799                     n_right_segment = nf; % Attention here
800                 end
801             end
802             B(k,1) = B(k-1,1) + Delta_B_left + Delta_B_right + h/alfa_l_tang + h/alfa_r_tang;
803             A(k,1) = A(k-1,1) + (B(k,1) + B(k-1,1))*h/2 + Delta_Area_left + Delta_Area_right;
804             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;
805             Rh(k,1) = A(k,1)/P(k,1);
806             Phi(k,1) = A(k,1)*Rh(k,1)^(2/3);
807             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;
808             % Representative Roughness Coefficient
809             if flag_method == 1
810                 n_med(k,1) = (int_n_p/P(k,1))^(2/3);
811             else
812                 if y_table(k,1) > ym
813                     yf = max(y_table(k,1) - ym,0); % Overbank depth
814                     af = max(A(k,1) - (am + bm*yf),0); % Overbank flow area
815                     pf = max(P(k,1) - pm,0); % Floodplain perimeter (m)
816                     pm_star = max(pm + n_fp*yf,0);
817                     am_star = max(am + bm*yf,0);
818                     n_med(k,1) = (Phi(k,1))/(1/nf*af*(af/pf)^(2/3) + ...
                        1/nm*am_star*(am_star/pm_star)^(2/3));
819                 else
820                     yf = 0; % Overbank depth
821                     af = 0; % Overbank flow area
822                     pf = 0; % Floodplain perimeter (m)
823                     pm_star = 0;
824                     am_star = 0;
825                     n_med(k,1) = nm;
826                 end
827             end
828             K_c(k,1) = 1/n_med(k,1)*Phi(k,1);

```

```

829 else
830     % Functions in terms of depth
831     h = precision_section;
832     y_table(k,1) = h + y_table(k-1,1);
833     % Roughness
834     if y_table(k,1) ≤ ym % Inside of the channel
835         if flag_method == 1
836             n_left_segment = n_left_unsorted(pos_left,1);
837             n_right_segment = n_right_unsorted(pos_right,1);
838         else
839             if (abs(y_left_unsorted(pos_left) - ym) ≤ noise*length(y_left_unsorted))
840                 n_left_segment = nf; % Attention here
841             else
842                 n_left_segment = nm; % Attention here
843             end
844             if (abs(y_right_unsorted(pos_right) - ym) ≤ noise*length(y_right_unsorted))
845                 n_right_segment = nf; % Attention here
846             else
847                 n_right_segment = nm; % Attention here
848             end
849         end
850     else % Overbanks
851         if flag_method == 1
852             n_left_segment = n_left_unsorted(pos_left,1);
853             n_right_segment = n_right_unsorted(pos_right,1);
854         elseif y_left_unsorted(pos_left) - ym < noise*length(y_left_unsorted) % Check Noises
855             n_left_segment = nm; % Attention here
856             n_right_segment = nm; % Attention here
857         else
858             n_left_segment = nf; % Attention here
859             n_right_segment = nf; % Attention here
860         end
861     end
862     B(k,1) = h/alfa_l_tang + h/alfa_r_tang + B(k-1,1);
863     A(k,1) = (B(k,1) + B(k-1,1))*h/2 + A(k-1,1); % Trapezoid
864     P(k,1) = h/sin(atan(alfa_l_tang)) + h/sin(atan(alfa_r_tang)) + P(k-1,1);
865     Rh(k,1) = A(k,1)/P(k,1);
866     Phi(k,1) = A(k,1)*Rh(k,1)^(2/3);
867     int_n_p = n_left_segment^(3/2)*h/sin(atan(alfa_l_tang)) + ...
868             n_right_segment^(3/2)*h/sin(atan(alfa_r_tang)) + int_n_p;
869     % Representative Roughness Coefficient
870     if flag_method == 1
871         n_med(k,1) = (int_n_p/P(k,1))^(2/3);
872     else
873         if y_table(k,1) > ym
874             yf = max(y_table(k,1) - ym, 0); % Overbank depth
875             af = max(A(k,1) - (am + bm*yf), 0); % Overbank flow area
876             pf = max(P(k,1) - pm, 0); % Floodplain perimeter (m)
877             pm_star = max(pm + n_fp*yf, 0);
878             am_star = max(am + bm*yf, 0);
879             n_med(k,1) = (Phi(k,1))/(1/nf*af*(af/pf)^(2/3) + ...
880                 1/nm*am_star*(am_star/pm_star)^(2/3));
881         else
882             yf = 0; % Overbank depth
883             af = 0; % Overbank flow area
884             pf = 0; % Floodplain perimeter (m)
885             pm_star = 0;
886             am_star = 0;
887             n_med(k,1) = nm;
888         end
889     end
890     K_c(k,1) = 1/n_med(k,1)*Phi(k,1);
891 end
892
893 if j == (n_points) % final point
894     % Final point - make sure you have the exact surveyed point at the end
895     h_ = y_moving_end - y_table(k-1,1);
896     y_table(k,1) = y_table(k-1,1) + h_;
897     % Roughness
898     if y_table(k,1) ≤ ym % Inside of the channel
899         if flag_method == 1
900             n_left_segment = n_left_unsorted(pos_left,1);
901             n_right_segment = n_right_unsorted(pos_right,1);
902         else
903             if (abs(y_left_unsorted(pos_left) - ym) ≤ noise*length(y_left_unsorted))
904                 n_left_segment = nf; % Attention here
905             else

```

```

904         n_left_segment = nm; % Attention here
905     end
906     if (abs(y_right_unsorted(pos_right) - ym) ≤ noise*length(y_right_unsorted))
907         n_right_segment = nf; % Attention here
908     else
909         n_right_segment = nm; % Attention here
910     end
911 end
912 else % Overbanks
913     if flag_method == 1
914         n_left_segment = n_left_unsorted(pos_left,1);
915         n_right_segment = n_right_unsorted(pos_right,1);
916     elseif y_left_unsorted(pos_left) - ym < noise*length(y_left_unsorted) % Check Noises
917         n_left_segment = nm; % Attention here
918         n_right_segment = nm; % Attention here
919     else
920         n_left_segment = nf; % Attention here
921         n_right_segment = nf; % Attention here
922     end
923 end
924 B(k,1) = h_/alfa_l_tang + h_/alfa_r_tang + B(k-1,1) + B_extra;
925 A(k,1) = (B(k,1) + B(k-1,1))*h_/2 + A(k-1,1); % Trapezoid
926 P(k,1) = h_/sin(atan(alfa_l_tang)) + h_/sin(atan(alfa_r_tang)) + P(k-1,1) + P_extra;
927 Rh(k,1) = A(k,1)/P(k,1);
928 Phi(k,1) = A(k,1)*Rh(k,1)^(2/3);
929 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;
930 % Representative Roughness Coefficient
931 if flag_method == 1
932     n_med(k,1) = (int_n_p/P(k,1))^(2/3);
933 else
934     if y_table(k,1) ≥ ym
935         yf = max(y_table(k,1) - ym,0); % Overbank depth
936         af = max(A(k,1) - (am + bm*yf),0); % Overbank flow area
937         pf = max(P(k,1) - pm,0); % Floodplain perimeter (m)
938         pm_star = max(pm + n_fp*yf,0);
939         am_star = max(am + bm*yf,0);
940         n_med(k,1) = (Phi(k,1))/(1/nf*af*(af/pf)^(2/3) + ...
          1/nm*am_star*(am_star/pm_star)^(2/3));
941     else
942         yf = 0; % Overbank depth
943         af = 0; % Overbank flow area
944         pf = 0; % Floodplain perimeter (m)
945         pm_star = 0;
946         am_star = 0;
947         n_med(k,1) = nm;
948     end
949 end
950 K_c(k,1) = 1/n_med(k,1)*Phi(k,1);
951 end
952 end
953 % Previous Positions
954 pos_left_previous = pos_left;
955 pos_right_previous = pos_right;
956 end
957 % Checking i
958 if round(y_table(end),3) == round(max_y,3) % Stop de algorithm
959     i = big_n;
960 end
961 end
962
963 % Centroid Coordinates
964 int_a_y = 0; % Integral of A(y)dy
965 for i = 1:(length(A))
966     if i == 1
967         y_bar(i) = 0;
968         int_a_y(i) = 0;
969     else
970         int_a_y(i) = (A(i) - A(i-1))*(y_table(i) + y_table(i-1))/2 + int_a_y(i-1);
971         y_bar(i) = int_a_y(i)/A(i);
972     end
973 end
974
975 % Flow Discharge Calculations
976 Q = K_c*sqrt(s0);
977
978 % Beta - Boussinesq factor

```

```

979 kappa = 0.41;
980 g = 9.81; % m/s2
981 Beta = (1 + (g*n_med.^2)./(Rh.^(1/3)*kappa^2));
982
983 %% Plotting Results
984 % Plotting Channel
985 close all
986 subplot(1,2,1)
987 set(gcf,'units','inches','position',[4,4,6.5,4])
988 mark_size = 5;
989 plot(x_absolute,y,'linewidth',2,'color','black')
990 xlabel('x ($m$)','Interpreter','latex');
991 ylabel('y ($m$)','Interpreter','latex');
992 xlim([min(x_absolute) max(x_absolute)])
993 grid on
994 hold on
995 scatter(x_absolute,y,'black')
996 subplot(1,2,2)
997 n_med(1,1) = inf;
998 plot(n_med(2:end,1),y_table(2:end,1),'linewidth',2,'color','black')
999 xlabel('Manning`s coefficient (SI)','Interpreter','latex');
1000 ylabel('y ($m$)','Interpreter','latex');
1001 xlim([0.9*min(n_med) 1.1*max(n_med(-isinf(n_med)))])
1002 grid on
1003 exportgraphics(gcf,'Cross_Section.pdf','ContentType','vector')
1004
1005
1006 subplot(2,4,1)
1007 set(gcf,'units','inches','position',[4,2,7.5,5])
1008 sz = 5;
1009 c = linspace(1,sz,length(y_table));
1010 scatter(A,y_table,sz,c,'filled')
1011 grid on
1012 grid on
1013 xlabel('Area ($m^2$)','Interpreter','latex');
1014 ylabel('y ($m$)','Interpreter','latex');
1015 % xlim([0 4])
1016 subplot(2,4,2)
1017 grid on
1018 scatter(P,y_table,sz,c,'filled')
1019 grid on
1020 xlabel('Perimeter ($m$)','Interpreter','latex');
1021 ylabel('y ($m$)','Interpreter','latex');
1022 % xlim([0 4])
1023 subplot(2,4,3)
1024 grid on
1025 scatter(Rh,y_table,sz,c,'filled')
1026 grid on
1027 xlabel('Hydraulic Radius ($m$)','Interpreter','latex');
1028 ylabel('y ($m$)','Interpreter','latex');
1029 % xlim([0 4])
1030 subplot(2,4,4)
1031 grid on
1032 scatter(B,y_table,sz,c,'filled')
1033 grid on
1034 xlabel('Top width ($m$)','Interpreter','latex');
1035 ylabel('y ($m$)','Interpreter','latex');
1036 subplot(2,4,5)
1037 grid on
1038 scatter(K_c,y_table,sz,c,'filled')
1039 grid on
1040 xlabel('Conveyance ($m^3/s$)','Interpreter','latex');
1041 ylabel('y ($m$)','Interpreter','latex');
1042 subplot(2,4,6)
1043 sz = 5;
1044 c = linspace(1,sz,length(y_table));
1045 scatter(Phi,y_table,sz,c,'filled')
1046 grid on
1047 xlabel('$\Phi$ ($m^{5/3}$)','Interpreter','latex');
1048 ylabel('y ($m$)','Interpreter','latex');
1049 subplot(2,4,7)
1050 scatter(y_bar,y_table,sz,c,'filled')
1051 grid on
1052 xlabel('$\bar{y}$ (m)','Interpreter','latex');
1053 ylabel('y ($m$)','Interpreter','latex');
1054 subplot(2,4,8)
1055 scatter(Q,y_table,sz,c,'filled')

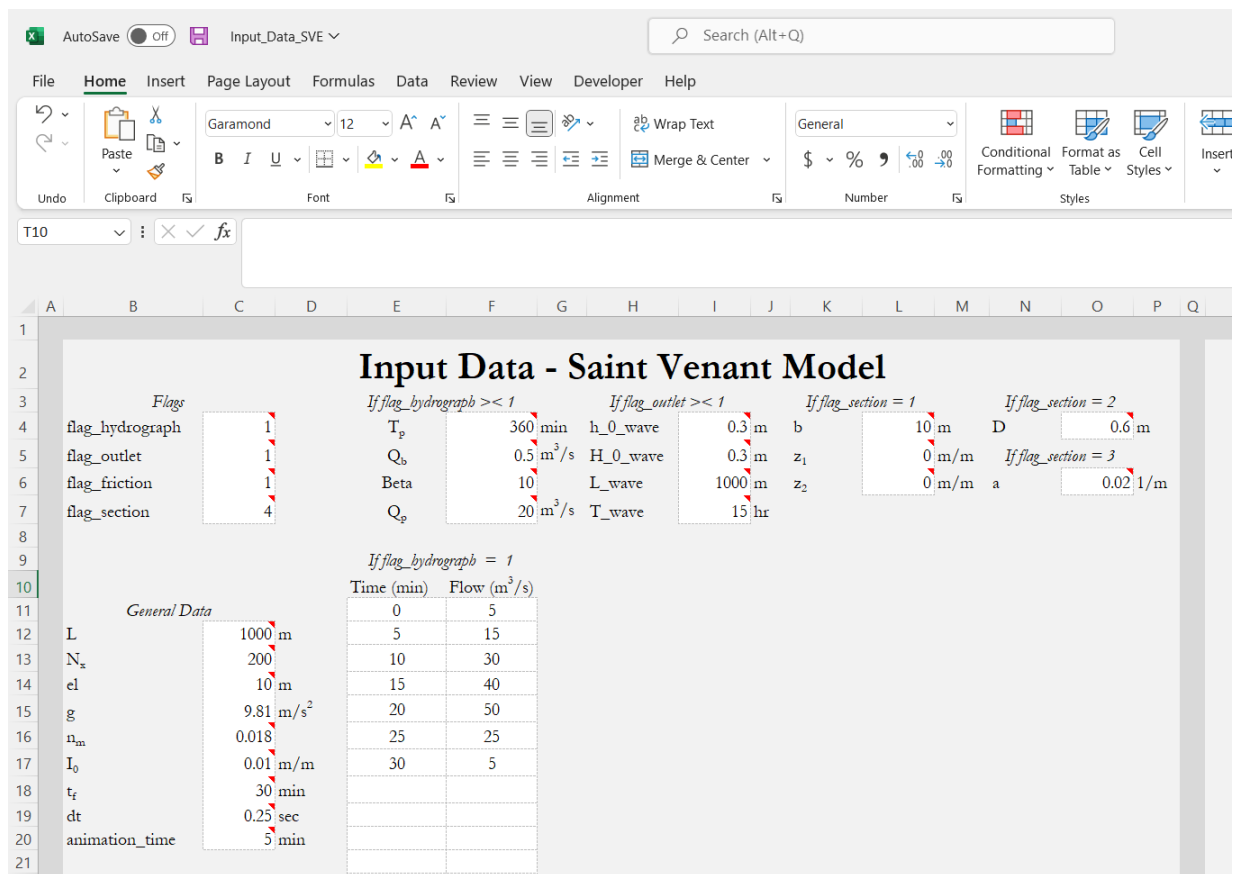
```

```

1056 grid on
1057 xlabel('Flow discharge ( $m^3/s$ )', 'Interpreter', 'latex');
1058 ylabel('y ( $m$ )', 'Interpreter', 'latex');
1059 exportgraphics(gcf, 'Hydraulic_Properties.pdf', 'ContentType', 'vector')
1060 toc
1061
1062 % Rating Curve
1063 close all
1064 subplot(3,1,1)
1065 set(gcf, 'units', 'inches', 'position', [4,4,6.5,4])
1066 mark_size = 5;
1067 plot(x_absolute, y, 'linewidth', 2, 'color', 'black')
1068 xlabel('x ( $m$ )', 'Interpreter', 'latex');
1069 ylabel('y ( $m$ )', 'Interpreter', 'latex');
1070 xlim([min(x_absolute) max(x_absolute)])
1071 grid on
1072 subplot(3,1,2)
1073 scatter(Q, y_table, sz, c, 'filled')
1074 xlabel('Flow discharge ( $m^3/s$ )', 'Interpreter', 'latex');
1075 ylabel('y ( $m$ )', 'Interpreter', 'latex');
1076 grid on
1077 box on
1078 % Velocity
1079 subplot(3,1,3)
1080 scatter(Q./A, y_table, sz, c, 'filled')
1081 xlabel('Velocity ( $m/s$ )', 'Interpreter', 'latex');
1082 ylabel('y ( $m$ )', 'Interpreter', 'latex');
1083 grid on
1084 box on
1085 exportgraphics(gcf, 'Rating_Curve.pdf', 'ContentType', 'vector')
1086
1087 % Plotting Normalized Values
1088 set(gcf, 'units', 'inches', 'position', [4,2,8,4])
1089 subplot(1,5,1)
1090 scatter(Q/max(Q), y_table/max(y_table), sz, c, 'filled')
1091 xlabel('$Q/Q_{p}$', 'Interpreter', 'latex');
1092 ylabel('$y/y_{\{max\}}$', 'Interpreter', 'latex');
1093 title(['$Q_p$ ( $m^3/s$ ) = $ ', num2str(round(max(Q), 2))], 'interpreter', 'latex')
1094 axis equal
1095 grid on
1096 xlim([0 1]); ylim([0 1]);
1097 subplot(1,5,2)
1098 scatter(A/max(A), y_table/max(y_table), sz, c, 'filled')
1099 xlabel('$A/A_{\{max\}}$', 'Interpreter', 'latex');
1100 ylabel('$y/y_{\{max\}}$', 'Interpreter', 'latex');
1101 title(['$A_{\{max\}}$ ( $m^2$ ) = $ ', num2str(round(max(A), 2))], 'interpreter', 'latex')
1102 axis equal
1103 grid on
1104 xlim([0 1]); ylim([0 1]);
1105 subplot(1,5,3)
1106 scatter(Phi/max(Phi), y_table/max(y_table), sz, c, 'filled')
1107 xlabel('$\Phi/\Phi_{\{max\}}$', 'Interpreter', 'latex');
1108 ylabel('$y/y_{\{max\}}$', 'Interpreter', 'latex');
1109 title(['$\Phi_{\{max\}}$ ( $m^2$ ) = $ ', num2str(round(max(Phi), 2))], 'interpreter', 'latex')
1110 axis equal
1111 grid on
1112 xlim([0 1]); ylim([0 1]);
1113 subplot(1,5,4)
1114 scatter(K_c/max(K_c), y_table/max(y_table), sz, c, 'filled')
1115 xlabel('$K_c/K_{\{c,max\}}$', 'Interpreter', 'latex');
1116 ylabel('$y/y_{\{max\}}$', 'Interpreter', 'latex');
1117 title(['$K_{\{c,max\}}$ ( $m^3/s$ ) = $ ', num2str(round(max(K_c), 2))], 'interpreter', 'latex')
1118 axis equal
1119 grid on
1120 xlim([0 1]); ylim([0 1]);
1121 subplot(1,5,5)
1122 scatter((Q./A)/(max(Q./A)), y_table/max(y_table), sz, c, 'filled')
1123 xlabel('$v/v_{\{c,max\}}$', 'Interpreter', 'latex');
1124 ylabel('$y/y_{\{max\}}$', 'Interpreter', 'latex');
1125 title(['$v_{\{max\}}$ ( $m/s$ ) = $ ', num2str(round(max(Q./A), 2))], 'interpreter', 'latex')
1126 axis equal
1127 grid on
1128 xlim([0 1]); ylim([0 1]);
1129 exportgraphics(gcf, 'Normalized_Values.pdf', 'ContentType', 'vector')
1130 close all

```





**Fig. 3:** Input data for the SVE model. All variables have a full explanation in the excel file.

## 2) Read Input Data - SVE

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

The excel input data is given a spreadsheet as follows:

```

1 %% Read Input Data %%
2 data = xlsread('Input_Data_SVE.xlsx');
3
4 b = 0; Z1 = 0; Z2 = 0; a = 0; D = 0;
5
6 % Flags
7 flag_hydrograph = data(1,1);
8 flag_outlet = data(2,1);
9 flag_friction = data(3,1);
10 flag_section = data(4,1);
11
12 if flag_hydrograph ~= 1
13     % Hydrograph
14     Tp = data(1,4);
15     Qb = data(2,4);
16     Beta = data(3,4);
17     Qp = data(4,4);
18 else
19     % Input Hydrograph
20     time_ = data(8:end,3);
21     Qe1_ = data(8:end,4);
22     Qe1 = zeros(size(Qe1_,1) - sum(isnan(Qe1_)),1);
23     time = zeros(size(time_,1) - sum(isnan(time_)),1);
24     % Taking away nans
25     for i = 1:length(Qe1)
26         if isnan(Qe1_(i)) || isnan(time_(i))

```

```

27         break
28     else
29         Qel(i,1) = Qel_(i,1);
30         time(i,1) = time_(i,1);
31     end
32 end
33 clear Qel_ time_
34 end
35
36 % Outlet
37 if flag_outlet ~= 1
38     h_0_wave = data(1,7);
39     H_0_wave = data(2,7);
40     L_wave = data(3,7);
41     T_wave = data(4,7);
42 end
43
44 % Section
45 if flag_section == 1
46     b = data(1,10);
47     Z1 = data(2,10);
48     Z2 = data(3,10);
49 elseif flag_section == 2
50     D = data(1,13);
51 elseif flag_section == 3
52     a = data(3,13);
53 else
54     % Read HP estimator data
55     [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr, x_cross, ...
56      y_cross] = HP_estimator;
57     irr_table = [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
58 end
59
60 % General Data
61 L = data(9,1);
62 Nx = data(10,1);
63 el = data(11,1);
64 g = data(12,1);
65 nm = data(13,1);
66 I0 = data(14,1);
67 tf = data(15,1);
68 dt = data(16,1);
69 animation_time = data(17,1);
70
71 % Constraint at observed flow
72 if flag_hydrograph == 1
73     if max(time) ~= tf
74         z = round(tf - max(time),0);
75         for i = 1:z
76             Qel(end + 1,1) = 0;
77             time(end+1,1) = time(end,1) + 1;
78         end
79     end
80 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  %%% Saint Venant Equations Solver %%%
2  % Developer: Marcus Nobrega Gomes Junior
3  % 4/10/2022
4  % Solution of SVE for given cross-section functions of Area, Perimeter, and
5  % top Width
6  %%% Still misses the introduction of Beta in F2 (check the paper)
7
8  %% Pre-Processing
9  clear all
10 clc
11 warning('off')
12

```

```

13 % Reading the Input Data
14 Read_Input_Data
15
16 % Inflow Hydrograph
17 % flag_hydrograph = 2; % If 1, enter the hydrograph, if 2, model using nash function
18 % 1st option - Enter Qel and Time below
19 if flag_hydrograph == 1
20     % We already read the hydrograph in Read_Input_Data file
21 else
22     % 2nd option - Model the hydrograph using a nash function
23     %% Q(t) = Qb(t) + (Qp(t) - Qb(t))*(t/TP*EXP(1 - t/TP))^Beta
24     Inflow_Hydrograph_fun = @(t)(Qb + (Qp - Qb).*(t/(Tp*60).*exp(1 - (t)/(Tp*60))).^Beta);
25     time = [0 tf/60]'; % begin and end in minutes
26 end
27
28 % Outlet Boundary Condition
29 % flag_outlet = 1; % 1 = normal depth, 2 = stage_hydrograph
30 if flag_outlet ~= 1
31     %% Wave Properties for Outlet Stage Hydrograph
32     x_wave = L_wave/1; % point position in wave x direction;
33     k_wave = 2*pi/L_wave;
34     sigma_wave = 2*pi/(T_wave*3600);
35     h_wave_function = @(t)(h_0_wave + H_0_wave/2.*cos(k_wave.*x_wave - sigma_wave*t));
36 end
37
38 % Time Calculations
39 time = time*60; % time in seconds
40 [a1,-] = size(time);
41 tt_h = time(a1,1); % end of hydrograph in seconds
42 tt = min(tf,tt_h)*60; % End of simulation in seconds
43 Nt = tt/dt; % Number of time-steps
44 Nat = animation_time*60/dt; % Number of time-steps within an animation time
45 tint = linspace(0,tt/60,Nt); % Generate Nt points within 0 and tt(min)
46 if flag_hydrograph == 1
47     Qelint = max(interpl(time/60,Qel(:,1),tint,'pchip'),0); % Interpolated flow % N o utilizar spline;
48     % pchip, que corresponde ao Hermite c bico, uma op o prefervel.
49     % Assuming no negative flows
50     Qelint = Qelint';
51 else
52     Qelint = Inflow_Hydrograph_fun(tint)';
53 end
54
55
56 % Channel Discretization
57 dx = L/(Nx-1); % Channel discretization length in meters
58
59 % Friction Data
60 flag_friction = 1; % If 1, Manning, otherwise DW
61
62 % Manning
63 nm = repmat(nm,Nx,1); % Bottom slope in m/m for all reaches
64 % Darcy Weisbach
65 f = 0; % not implemented
66
67 % Pre-allocating arrays
68 % Matrices
69 x = (0:dx:L)'; % x discretization in meters
70 y = zeros(Nx,Nt);
71 q1 = zeros(Nx,Nt);
72 q2 = zeros(Nx,Nt);
73 f1 = zeros(Nx,Nt);
74 f2 = zeros(Nx,Nt);
75 J2 = zeros(Nx,Nt);
76 q1_back = q1(1:(Nx-2),Nt);
77 q1_foward = zeros(Nx-2,Nt);
78 q2_back = zeros(Nx-2,Nt);
79 q2_foward = zeros(Nx-2,Nt);
80 f1_back = zeros(Nx-2,Nt);
81 f1_foward = zeros(Nx-2,Nt);
82 f2_back = zeros(Nx-2,Nt);
83 f2_foward = zeros(Nx-2,Nt);
84 J2_back = zeros(Nx-2,Nt);
85 J2_foward = zeros(Nx-2,Nt);
86 ybar = zeros(Nx,Nt);
87 Fr = zeros(Nx,Nt);
88 Cn = zeros(Nx,Nt);
89

```

```

90 %% Channel Data (Cross Section)
91 % Slope
92 I0 = repmat(I0,Nx,1); % Bottom slope in m/m for all reaches
93
94 % Initializing channel data
95 sm = 1e-12; % Small number
96 b = sm + b; Z1 = sm + Z1; Z2 = sm + Z2; D = sm + D; a = sm + a;
97 % flag_section - If 1, trapezoid, if 2, circular, if 3, paraboloid, if 4 - Irregular
98
99 % Invert Elevations
100 inv_el = zeros(Nx,1);
101 for i = 1:Nx
102     if i == 1
103         inv_el(i) = el;
104     else
105         inv_el(i) = inv_el(i-1) - (I0(i)*dx);
106     end
107 end
108
109 % Geometrical Functions
110 syms b_ y_ Z1_ Z2_ Q_ I0_ D_ a_
111 dim_all = 1e-6*(y_ + Z1_ + Z2_ + a_ + D_ + b_);
112 if flag_section == 1
113     B = b_ + y_*(Z1_ + Z2_) + dim_all; % user defined function (top width)
114     B_function = matlabFunction(B);
115     P = b_ + y_*(sqrt(1 + Z1_^2) + sqrt(1 + Z2_^2)) + dim_all; % Perimeter Function % user defined ...
        function
116     P_function = matlabFunction(P);
117     A = (2*b_ + y_*(Z1_ + Z2_))*y_/2 + dim_all; % Area function % user defined function
118     A_function = matlabFunction(A); % Function describing the area in terms of y
119     centroid = y_ - int(A,y_)./A + dim_all; % 1st order momentum
120     ybar_function = matlabFunction(centroid); % Function describing ybar in terms of y
121 end
122 if flag_section == 2
123     % Circular Section
124     theta = 2*acos(1 - 2.*y_./D_) + dim_all;
125     B = D_.*sin(theta/2) ; % top width
126     B_function = matlabFunction(B);
127     P = theta.*D_/2 ; % perimeter
128     P_function = matlabFunction(P);
129     A = D_.^2/8.*(theta - sin(theta)) ; % area
130     A_function = matlabFunction(A); % Function describing the area in terms of y
131     Ybar = y_ - (D_.*(- cos(theta/2)/2 + 2.*sin(theta/2).^3./(3*(theta - sin(theta))))); % Very ...
        much attention here
132     ybar_function = matlabFunction(Ybar);
133 end
134
135 if flag_section == 3
136     % Parabolic Section
137     % Area Function
138     A = 4.*(y_.^3/2)./(3*sqrt(a_)) + dim_all; % m2
139     A_function = matlabFunction(A); % Function describing the area in terms of y
140     % Top Width
141     B = 3/2.*A./y_ + dim_all; % m
142     B_function = matlabFunction(B);
143     % Hydraulic Perimeter
144     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_))));
145     P_function = matlabFunction(P);
146     Y_bar = y_ - 2/5*y_ + dim_all;
147     ybar_function = matlabFunction(Y_bar);
148 end
149
150 if flag_section == 4
151     %%%%%% Hydraulic Radius %%%%%%
152     Rh = A/P; % Hydraulic Radius Function
153     Rh_function = matlabFunction(Rh); % Function describing the hydraulic radius in terms of y
154 end
155
156 % Vlookup Function
157 Vlookup_eq = @(data,col1,vall,col2) data((find(data(:,col1)==vall,1,'first')),col2); %Vlookup ...
        function as Excel
158 Vlookup_l = @(data,col1,vall,col2) data((find(data(:,col1)<vall,1,'last')),col2); %Vlookup function ...
        as Excel]
159 Vlookup_g = @(data,col1,vall,col2) data((find(data(:,col1)>vall,1,'first')),col2); %Vlookup ...
        function as Excel
160 fv = 1 + 1e-4; % Factor to avoid fails in vlookup function

```

```

161
162 % Initial Guess
163 if flag_section == 1
164     y0_guess = 1;
165 elseif flag_section == 2
166     y0_guess = D/2;
167 elseif flag_section == 3
168     y0_guess = 1;
169 end
170
171 %% Initial Boundary Conditions
172 Q0 = Qelint(1,1); % Flow at inlet section at time 0
173 if flag_outlet == 1
174     if flag_friction == 1
175         if flag_section ≠ 4
176             if Q0 == 0
177                 Q0 = sm; % Numerical Constraint
178             end
179             y0 = uniformeM(nm,Q0,b,Z1,Z2,a,D,I0,P,A,y0_guess) ; % normal depth using manning equation
180             % More Initial Boundary Conditions for Area, Velocity, Perimeter and Rh
181             A0 = A_function(D,Z1,Z2,a,b,y0); % Cross section area in m
182             u0 = (Q0./A0)'; % Initial velocity in m/s
183             P0 = P_function(D,Z1,Z2,a,b,y0); % Hydraulic perimeter in m
184             Rh0 = A0./P0; % Hydraulic radius at time 0
185             % Boundary Conditions
186             y(:,1) = y0; % all sub-reaches with y0 at the beginning
187             q1(:,1) = A0; % all sub-reaches with same area A0 at the beginning
188             q2(:,1) = Q0; % Assuming permanent conditions at the beginning
189             f1(:,1) = q2(:,1);
190             % f2 depends on ybar
191         else % Irregular Cross-Section
192             % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
193             % [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
194             coll = 10; % Col with Q
195             y0 = Vlookup_1(irr_table,coll,Q0*fv,1);
196             A0 = Vlookup_1(irr_table,coll,Q0*fv,2);
197             P0 = Vlookup_1(irr_table,coll,Q0*fv,3);
198             Rh0 = Vlookup_1(irr_table,coll,Q0*fv,4);
199             % Boundary Conditions
200             y(:,1) = y0; % all sub-reaches with y0 at the beginning
201             q1(:,1) = A0; % all sub-reaches with same area A0 at the beginning
202             q2(:,1) = Q0; % Assuming permanent conditions at the beginning
203             f1(:,1) = q2(:,1);
204         end
205     else
206         % % normal depth using Darcy-Weisbach equation not implemented yet
207     end
208 else
209     steady_depth = uniformeM(nm,Q0,b,Z1,Z2,a,D,I0,P,A);
210     y0(1:(Nx-1),1) = steady_depth(1:(Nx-1),1);
211     % Stage Hydrograph Boundary Condition
212     time_wave = 0; % time in seconds
213     y0(Nx,1) = h_wave_function(time_wave);
214     % More Initial Boundary Conditions for Area, Velocity, Perimeter and Rh
215     A0 = A_function(D,Z1,Z2,a,b,y0); % Cross section area in m
216     u0 = (Q0./A0)'; % Initial velocity in m/s
217     P0 = P_function(D,Z1,Z2,a,b,y0); % Hydraulic perimeter in m
218     Rh0 = A0./P0; % Hydraulic radius at time 0
219     y(:,1) = y0(:,1); % all sub-reaches with y0 at the beginning
220     q1(:,1) = A0(:,1); % all sub-reaches with same area A0 at the beginning
221     q2(:,1) = Q0(:,1); % Assuming permanent conditions at the beginning
222     f1(:,1) = q2(:,1);
223     % f2 depends on ybar, we will calculate below
224 end
225
226 %%% State Space Format %%%
227 % dq/dt + dF/dx = S, we solve for A(x,t) and Q(x,t)
228 % q = [A Q]' = [q1 q2]'
229 % F [Q (Qv + gAybar)]' = [q2 (q2^2)/q1 + g.q1.ybar]' = [f1 f2]'
230 % where ybar is the centroid depth from the top
231 % S = [0 gA(I0 - If)]'
232
233 % ybar = y - int(A(y)) / A(y) from y = 0 to y = y0
234 if flag_section ≠ 4
235     ybar = ybar_function(D,Z1,Z2,a,b,y0);
236 else
237     % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];

```

```

238 % [ 1, 2, 3, 4, 5, 6, 7, 8, 9,
239 % ybar = y - ybar*
240 % ybar(:,1) = Vlookup_leq(irr_table,coll,Q0*fv,1) - Vlookup_leq(irr_table,coll,Q0*fv,5);
241 ybar(:,1) = Vlookup_1(irr_table,coll,Q0*fv,5);
242 end
243 f2(:,1) = q2(:,1).*abs(q2(:,1))./q1(:,1) + g*q1(:,1).*ybar(:,1);
244
245 % Friction S = [J1 J2]' with J1 = 0 and J2 calculated as follows:
246 if flag_friction == 1
247     J2(:,1) = g*q1(:,1).*(I0(:) - q2(:,1).*abs(q2(:,1)).*nm(:)./(q1(:,1).^2.*Rh0.^(4/3))); % Manning
248 else
249     J2(:,1) = g*q1(:,1).*(I0(:) - f*q2(:,1).*abs(q2(:,1))./((q1(:,1).^2).*g.*Rh0)); % Darcy
250 end
251
252 % Froude Number
253 if flag_section == 4
254     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);% ...
255     N_mero de Froude
256 else
257     % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
258     % [ 1, 2, 3, 4, 5, 6, 7, 8, 9,
259     A_f_irr = Vlookup_1(irr_table,coll,Q0*fv,2)*ones(length(q1(:,1)),1);
260     B_f_irr = Vlookup_1(irr_table,coll,Q0*fv,9)*ones(length(q1(:,1)),1);
261     Fr(:,1)=abs(q2(:,1)./q1(:,1))./((g*A_f_irr./B_f_irr).^0.5);% N_mero de Froude
262 end
263 % Courant Number
264 % Cn = c / (dx / dt), where c = v + sqrt(g.Hm), where Hm = A / B
265 if flag_section == 4
266     Hm = A_function(D,Z1,Z2,a,b,y0)./B_function(D,Z1,Z2,a,b,y0);
267     Cn(:,1)=(abs(q2(:,1)./q1(:,1))+(g*Hm).^0.5)/(dx/dt);% Courant Number
268 else
269     Hm = A_f_irr./B_f_irr;
270     Cn(:,1) = (abs(q2(:,1)./q1(:,1))+(g*Hm).^0.5)/(dx/dt);
271 end
272
273 % Depth in terms of Area function
274 % let c be the area in terms of Z1,Z2,b, and y, such that A(y) = c
275 % we want to solve y for A(y) = c
276
277 syms c_
278 if flag_section == 4
279     fun_solve = (A - c_); % with c = area, we solve for y.
280     options = optimoptions('fsolve','Display',...
281         'none','FunctionTolerance',1e-2,'MaxFunctionEvaluations',Nx*10);
282 end
283 if flag_section == 1
284     % We have an analytical solution for this case
285     z = solve(fun_solve,y_); % solving for y_ = y and c = A(y)
286     h_function = matlabFunction(z); % h(A) = z;
287 else
288     % Non-linear set of equations for circular pipe, we need to use fsolve
289 end
290 if flag_section == 4
291     fun_solve = matlabFunction(fun_solve); % Transforming into an equation
292 end
293 %% Main Loop %%
294 n = 1; % initializing counter
295 tic % starts measuring time
296
297 for t = dt:dt:(tt - dt) % Main loop
298     n = n + 1;
299     percentage_complete = [t/(tt - dt)*100, max(Cn(:,n-1)), max(q2(:,n-1))]
300     % Stop Program if Complex Number Occurs
301     if imag(max(Cn(:,n-1))) > 0 || imag(max(q2(:,n-1)))
302         error('Complex number possibly due to changing the regime from free flow to pressurized flow.')
303     end
304     % Boundary Conditions - %
305     % Channel's begin (INLET)
306     q1(1,n) = q1(2,n-1); % Area at section 1 is equals area of section 2 from previous time-step
307     q2(1,n) = Qelint(n,1); % Flow at section 1 is the inflow hydrograph
308
309     % Interpolating All Values from I_rr_table using q1 as basis
310     % Explanation: area is given in m2. P, Rh, and other variables are
311     % in m. So we have a quadratically similar triangle relationship
312     if flag_section == 4
313         for mm = 1:(length(irr_table(1,:))-1)
314             interp_base = q1(1,n); % Value that will be used for interpolation (area)

```

```

313     area_smaller = Vlookup_l(irr_table,2,interp_base,2); % Smaller values
314     area_larger = Vlookup_g(irr_table,2,interp_base,2); % Larger values
315     coll = 2; % Interpolating from area values
316     var_inlet(mm,1,1) = Vlookup_l(irr_table,coll,interp_base,mm); % Smaller values
317     var_inlet(mm,1,2) = Vlookup_g(irr_table,coll,interp_base,mm); % Larger values
318     alfa_var_inlet(mm,1) = sqrt((interp_base - area_smaller)/(area_larger - area_smaller));
319 end
320 end
321
322 if flag_section == 1 % Trapezoid or Rectangular
323     if Z1 > 0 || Z2 > 0 % Trapezoidal channel
324         y(1,n) = max(h_function(D,Z1,Z2,a,b,q1(1,n)')); % water depth in terms of area q1
325         % In this previous function, we solve h = y in terms of A = q1 = c
326     else
327         y(1,n) = q1(1,n)/b; % water depth in terms of area q1 for rectangular channels
328     end
329 elseif flag_section > 1 % circular or paraboloid or irregular
330     y0_guess = y(1,n-1);
331     c = q1(1,n)*fv; % WEIRDO. I HAVE TO CHECK IT OUT ... ISNT IT (n-1)?
332     if flag_section ≠ 4
333         fun = @(y_) fun_solve(D,Z1,Z2,a,b,c,y_);
334         y(1,n) = fsolve(fun,y0_guess,options); % non-linear solver
335     else % Irregular section
336         % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
337         % [ 1, 2, 3, 4, 5, 6, 7, 8, 9,
338         coll = 2; % Col with A
339         col_var = 1;
340         % Var* = Var(-) + alfa*(Var(+) - Var(-))
341         y(1,n) = var_inlet(col_var,1,1) + alfa_var_inlet(col_var,1)*(var_inlet(col_var,1,2) - ...
342             var_inlet(col_var,1,1));
343         % y(1,n) = Vlookup_leq(irr_table,coll,c,1);
344     end
345 end
346 % ybar
347 if flag_section ≠ 4
348     ybar(1,n) = ybar_function(D,Z1,Z2,a,b,y(1,n));
349     % f1 and f2
350     f1(1,n) = q2(1,n);
351     f2(1,n) = q2(1,n).*abs(q2(1,n))./q1(1,n) + g*q1(1,n).*ybar(1,n);
352     % Hydraulic Radius
353     Rh_inlet = Rh_function(D,Z1,Z2,a,b,y(1,n));
354     % Friction
355     if flag_friction == 1
356         J2(1,n) = g*q1(1,n).*(I0(1) - ...
357             q2(1,n).*abs(q2(1,n)).*nm(1).^2./(q1(1,n).^2*Rh_inlet.^(4/3))); % Manning
358     else
359         J2(1,n) = (I0(1) - f*q2(1,n).*abs(q2(1,n))./((q1(1,n).^2)*g.*Rh_inlet));
360     end
361     % Froude
362     Fr(1,n)=abs(q2(1,n)./q1(1,n))./((g*A_function(D,Z1,Z2,a,b,y(1,n))./B_function(D,Z1,Z2,a,b,y(1,n)))^0.5);%
363     N mero de Froude
364     % Courant
365     Hm = A_function(D,Z1,Z2,a,b,y(1,n))./B_function(D,Z1,Z2,a,b,y(1,n));
366     Cn(1,n)=(abs(q2(1,n)./q1(1,n))+(g.*Hm).^0.5)/(dx/dt);% Courant Number
367 else
368     % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
369     % [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
370     coll = 2; % Col with A
371     col_var = 5;
372     % Var* = Var(-) + alfa*(Var(+) - Var(-))
373     ybar(1,n) = var_inlet(col_var,1,1) + alfa_var_inlet(col_var,1)*(var_inlet(col_var,1,2) - ...
374         var_inlet(col_var,1,1));
375     % f1 and f2
376     f1(1,n) = q2(1,n);
377     f2(1,n) = q2(1,n).*abs(q2(1,n))./q1(1,n) + g*q1(1,n).*ybar(1,n);
378     % Hydraulic Radius
379     col_var = 4;
380     % Var* = Var(-) + alfa*(Var(+) - Var(-))
381     Rh_inlet = var_inlet(col_var,1,1) + alfa_var_inlet(col_var,1)*(var_inlet(col_var,1,2) - ...
382         var_inlet(col_var,1,1));
383     % Friction
384     if flag_friction == 1
385         col_var = 6;
386         % Var* = Var(-) + alfa*(Var(+) - Var(-))
387         nm(1) = var_inlet(col_var,1,1) + alfa_var_inlet(col_var,1)*(var_inlet(col_var,1,2) - ...
388             var_inlet(col_var,1,1));
389         if isnan(nm(1,1))

```

```

384     nm = irr_table(2,6)*ones(length(q1(:,1)),1);
385     end
386     J2(1,n) = g*c.*(I0(1) - q2(1,n).*abs(q2(1,n)).*nm(1).^2./(c.^2*Rh_inlet.^(4/3))); % Manning
387 else
388     J2(1,n) = (I0(1) - f*q2(1,n).*abs(q2(1,n))./((q1(1,n).^2)*8*g.*Rh_inlet));
389 end
390 % Froude
391 % Var* = Var(-) + alfa*(Var(+) - Var(-))
392 A_f_irr = q1(1,n);
393 col_var = 9;
394 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));
395 % B_f_irr = Vlookup_leq(irr_table,col1,c,9);
396 Fr(1,n) = abs(q2(1,n)./q1(1,n))./((g*A_f_irr./B_f_irr)^0.5); % N mero de Froude
397 % Courant
398 Hm = A_f_irr./B_f_irr;
399 Cn(1,n) = (abs(q2(1,n)./q1(1,n)) + (g*Hm).^0.5)/(dx/dt); % Courant Number
400 end
401
402 %% Right side of the channel (outlet)
403 if flag_outlet == 1 % Normal Depth
404     q1(Nx,n) = q1(Nx-1,n-1); % Boundary Condition
405     % Interpolating All Values from I_irr_table using q1 as basis
406     % Explanation: area is given in m2. P, Rh, and other variables are
407     % in m. So we have a quadratically similar triangle relationship
408     if flag_section == 4
409         for mm = 1:(length(irr_table(1,:))-1)
410             interp_base = q1(Nx,n); % Value that will be used for interpolation (area)
411             area_smaller = Vlookup_l(irr_table,2,interp_base,2); % Smaller values
412             area_larger = Vlookup_g(irr_table,2,interp_base,2); % Larger values
413             coll = 2; % Interpolating from area values
414             var_outlet(mm,1,1) = Vlookup_l(irr_table,coll,interp_base,mm); % Smaller values
415             var_outlet(mm,1,2) = Vlookup_g(irr_table,coll,interp_base,mm); % Larger values
416             alfa_var_outlet(mm,1) = sqrt((interp_base - area_smaller)/(area_larger - ...
                area_smaller));
417         end
418     end
419     if flag_section == 1
420         q1(Nx,n) = q1(Nx-1,n-1); % Area of outlet is the area of previous cell at previous ...
            time-step
421         if Z1 > 0 || Z2 > 0
422             y(Nx,n) = max(h_function(D,Z1,Z2,a,b,q1(Nx,n)')); % water depth in terms of area q1
423         else
424             y(Nx,n) = q1(Nx,n)/b; % water depth in terms of area q1 for rectangular channels
425         end
426     elseif flag_section > 2 % circular or paraboloid or irregular
427         % If we do not have an stage-hydrograph boundary condition
428         y0_guess = y(Nx,n-1);
429         if flag_section ≠ 4
430             fun = @(y_) fun_solve(D,Z1,Z2,a,b,c,y_);
431             y(Nx,n) = fsolve(fun,y0_guess,options); % non-linear solver
432         else
433             % [y_table, A, P, Rh, y_bar, n_med, Beta, v, B, Q]
434             % [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
435             col_var = 1;
436             % Var* = Var(-) + alfa*(Var(+) - Var(-))
437             y(Nx,n) = var_outlet(col_var,1,1) + ...
                alfa_var_outlet(col_var,1)*(var_outlet(col_var,1,2) - var_outlet(col_var,1,1));
438         end
439     end
440 else
441     % Stage Hydrograph Boundary Condition
442     time_wave = n*dt; % time in seconds
443     y(Nx,n) = h_wave_function(time_wave);
444     q1(Nx,n) = A_function(D,Z1,Z2,a,b,y(Nx,n));
445     % q1(Nx,n) = q1(Nx-1,n-1)
446 end
447 % Hydraulic Radius
448 if flag_section ≠ 4
449     Rh_outlet = Rh_function(D,Z1,Z2,a,b,y(Nx,n));
450 else
451     % [y_table, A, P, Rh, y_bar, n_med, Beta, v, B, Q]
452     % [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
453     % coll = 2; % Col with A
454     % Rh_outlet = Vlookup_l(irr_table,coll,c,4);
455     col_var = 4;
456     % Var* = Var(-) + alfa*(Var(+) - Var(-))

```



```

457     Rh_outlet = var_outlet(col_var,1,1) + alfa_var_outlet(col_var,1)*(var_outlet(col_var,1,2) - ...
458         var_outlet(col_var,1,1));
459     % Var* = Var(-) + alfa*(Var(+) - Var(-))
460     col_var = 6;
461     nm(end,1) = var_inlet(col_var,1,1) + alfa_var_inlet(col_var,1)*(var_inlet(col_var,1,2) - ...
462         var_inlet(col_var,1,1));
463 end
464 if flag_friction == 1
465     if flag_outlet == 1
466         u = (1./nm(Nx)).*Rh_outlet^(2/3)*I0(Nx)^0.5; % Normal depth at the outlet
467         flow_dir = 1;
468     else
469         depth_dif = y(Nx-1,n-1) + inv_el(Nx-1) - y(Nx,n) - inv_el(Nx); % Difference in wse
470         out_slope = abs(depth_dif)/dx; % Flow slope at the outlet
471         u = (1./nm(Nx)).*Rh_outlet^(2/3)*out_slope^0.5; % Normal depth at the outlet
472         if depth_dif > 0
473             flow_dir = 1; % Flowing towards the outlet
474         else
475             flow_dir = -1; % Flowing to inside of the channel
476         end
477     end
478 end
479 else
480     u = sqrt(8*g*Rh_outlet*I0(Nx)/f); % outlet velocity
481 end
482 % Outlet Flow
483 q2(Nx,n) = q1(Nx,n)*u*flow_dir; % Area x Velocity
484 if isnan(q2(Nx,n))
485     ttt = 1;
486 end
487 % ybar
488 if flag_section ≠ 4
489     ybar(Nx,n) = ybar_function(D,Z1,Z2,a,b,y(Nx,n));
490 else
491     % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
492     % [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
493     % ybar = y - ybar*
494     coll = 2; % A
495     % ybar(Nx,n) = Vlookup_leq(irr_table,coll,c,1) - Vlookup_leq(irr_table,coll,c,5);
496     ybar(Nx,n) = Vlookup_l(irr_table,coll,c,5);
497     col_var = 5;
498     % Var* = Var(-) + alfa*(Var(+) - Var(-))
499     ybar_outlet(col_var,1,1) + alfa_var_outlet(col_var,1)*(var_outlet(col_var,1,2) ...
500         - var_outlet(col_var,1,1));
501 end
502 % f1 and f2
503 f1(Nx,n) = q2(Nx,n); % f1 - Area
504 f2(Nx,n) = q2(Nx,n).*abs(q2(Nx,n))./q1(Nx,n) + g*q1(Nx,n).*ybar(Nx,n); % f2 = (Qv + gAy_bar)
505 % J2
506 % Friction
507 if flag_friction == 1
508     J2(Nx,n) = g*q1(Nx,n).*(I0(Nx) - ...
509         q2(Nx,n).*abs(q2(Nx,n)).*nm(Nx)^2./(q1(Nx,n).^2*Rh_outlet^(4/3))); % Manning --> ...
510         gA*(I0 - If), If = n^2*Q*abs*Q)/(Rh^(4/3)*A^2)
511 else
512     J2(Nx,n) = g*q1(Nx,n).*(I0(Nx) - f*q2(:,n).*abs(q2(Nx,n))./((q1(Nx,n).^2)*8*g*Rh_outlet));
513 end
514 % Froude
515 if flag_section ≠ 4
516     Fr(Nx,n)=abs(q2(Nx,n)./q1(Nx,n))./((g*A_function(D,Z1,Z2,a,b,y(Nx,n))./B_function(D,Z1,Z2,a,b,y(Nx,n)))^0.5).
517         N mero de Froude
518     % Courant
519     Hm = A_function(D,Z1,Z2,a,b,y(Nx,n))./B_function(D,Z1,Z2,a,b,y(Nx,n));
520     Cn(Nx,n)=(abs(q2(Nx,n)./q1(Nx,n))+(g*Hm).^0.5)/(dx/dt); % Courant Number
521 else
522     % Froude
523     % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
524     % [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
525     coll = 2; % Col with A
526     A_f_irr = c;
527     col_var = 9;
528     % Var* = Var(-) + alfa*(Var(+) - Var(-))
529     B_f_irr = var_outlet(col_var,1,1) + alfa_var_outlet(col_var,1)*(var_outlet(col_var,1,2) - ...
530         var_outlet(col_var,1,1));
531     % B_f_irr = Vlookup_l(irr_table,coll,c,9);
532     Fr(Nx,n) = abs(q2(Nx,n)./q1(Nx,n))./((g*A_f_irr./B_f_irr)^0.5); % N mero de Froude
533     % Courant
534     Hm = A_f_irr./B_f_irr;

```

```

527     Cn(Nx,n)=(abs(q2(1,n)./q1(1,n))+(g*Hm).^0.5)/(dx/dt); % Courant Number
528 end
529
530 %% Main Loop for Non Boundary Cells from 2 to (Nx - 1)
531 % vectorized calculations
532 q1_back = q1(1:(Nx-2),(n-1));
533 q1_foward = q1(3:(Nx),(n-1));
534 q2_back = q2(1:(Nx-2),(n-1));
535 q2_foward = q2(3:(Nx),(n-1));
536 f1_back = f1(1:(Nx-2),(n-1));
537 f1_foward = f1(3:(Nx),(n-1));
538 f2_back = f2(1:(Nx-2),(n-1));
539 f2_foward = f2(3:(Nx),(n-1));
540 J2_back = J2(1:(Nx-2),(n-1));
541 J2_foward = J2(3:(Nx),(n-1));
542
543 x_i = 2:(Nx-1); % vector for interior sections varying from 2 to (Nx - 1)
544 % Lax-Friedrichs Method
545 % Given an hyperbolic partial derivative system of equations described
546 % by:
547 %  $pq/pt + pF/px - S = 0$ , where p is the partial derivative, one can
548 % solve this equation by performing a foward discretization for q and a
549 % central discretization for F. Moreover,  $S = (Sback + Sfoward)/2$ 
550 % Expliciting the system of equations for q1, it follows that
551
552 q1(x_i,n) = 0.5.*(q1_foward + q1_back) - 0.5*dt/dx*(f1_foward - f1_back); %% attention here in ...
553     flfoward
554 q2(x_i,n) = 0.5*(q2_foward + q2_back) - 0.5*dt/dx*(f2_foward - f2_back) + 0.5*dt*(J2_back + ...
555     J2_foward);
556
557 % Interpolating All Values from I_rr_table using q1 as basis
558 if flag_section == 4
559     for mm = 1:(length(irr_table(1,:))-1)
560         for hh = 1:length(x_i)
561             interp_base = q1(hh+1,n); % Value that will be used for interpolation (area)
562             area_smaller = Vlookup_l(irr_table,2,interp_base,2); % Smaller values
563             area_larger = Vlookup_g(irr_table,2,interp_base,2); % Larger values
564             coll = 2; % Interpolating from area values
565             var_middle(mm,hh,1) = Vlookup_l(irr_table,coll,interp_base,mm); % Smaller values
566             var_middle(mm,hh,2) = Vlookup_g(irr_table,coll,interp_base,mm); % Larger values
567             alfa_var_middle(mm,hh,1) = sqrt((interp_base - area_smaller)/(area_larger - ...
568                 area_smaller));
569         end
570     end
571 end
572
573 if flag_section == 1
574     if Z1>0 || Z2>0
575         y(x_i,n) = max(h_function(D,Z1,Z2,a,b,q1(x_i,n)')); % water depth in terms of area q1
576     else
577         y(x_i,n)=q1(x_i,n)/b;
578     end
579 elseif flag_section > 1
580     y0_guess = y(x_i,n-1);
581     c = q1(x_i,n)*fv; % It has to be a line vector (area)
582     if flag_section ≠ 4
583         fun = @(y_) fun_solve(D,Z1,Z2,a,b,c,y_);
584         y(x_i,n) = fsolve(fun,y0_guess,options); % non-linear solver
585     else
586         % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
587         % [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
588         coll = 2; % Col with A
589         for i = 1:length(x_i)
590             cc = c(i); % be careful here
591             col_var = 1;
592             % Var* = Var(-) + alfa*(Var(+) - Var(-))
593             y(i+1,n) = var_middle(col_var,i,1) + ...
594                 alfa_var_middle(col_var,i)*(var_middle(col_var,i,2) - var_middle(col_var,i,1));
595         end
596     end
597 end
598
599 % Hydraulic Radius
600 if flag_section ≠ 4
601     Rh_middle = Rh_function(D,Z1,Z2,a,b,y(x_i,n));
602     % ybar
603     ybar(x_i,n) = ybar_function(D,Z1,Z2,a,b,y(x_i,n));
604     % f1 and f2

```

```

600     f1(x_i,n) = q2(x_i,n);
601     f2(x_i,n) = q2(x_i,n).*abs(q2(x_i,n))./q1(x_i,n) + g*q1(x_i,n).*ybar(x_i,n);
602     % Froude
603     Hm = A_function(D,Z1,Z2,a,b,y(x_i,n))./B_function(D,Z1,Z2,a,b,y(x_i,n));
604     Fr(x_i,n)=abs(q2(x_i,n)./q1(x_i,n))./((g*Hm).^0.5);% N mero de Froude
605     % Courant
606     Cn(x_i,n)=(abs(q2(x_i,n)./q1(x_i,n))+(g*Hm).^0.5)/(dx/dt);% Courant Number
607     % Friction
608     if flag_friction == 1
609         J2(x_i,n) = g*q1(x_i,n).*(I0(x_i) - ...
            q2(x_i,n).*abs(q2(x_i,n)).*nm(x_i).^2./(q1(x_i,n).^2.*Rh_middle.(4/3)));
610     else
611         J2(x_i,n) = g*q1(x_i,n).*(I0(x_i) - ...
            f*q2(x_i,n).*abs(q2(x_i,n))./((q1(x_i,n).^2)*8*g*Rh_midle));
612     end
613     % Stability Check
614     if max(Cn(:,n)) > 1
615         error('Please, decrease the time-step')
616     end
617 else
618     for jj = 1:length(x_i)
619         cc = c(jj); % Area
620         % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
621         % [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
622         col_var = 4;
623         % Var* = Var(-) + alfa*(Var(+) - Var(-))
624         Rh_middle(jj,1) = var_middle(col_var,jj,1) + ...
            alfa_var_middle(col_var,jj)*(var_middle(col_var,jj,2) - var_middle(col_var,jj,1));
625         col_var = 5;
626         ybar(jj+1,n) = var_middle(col_var,jj,1) + ...
            alfa_var_middle(col_var,jj)*(var_middle(col_var,jj,2) - var_middle(col_var,jj,1));
627         col_var = 6;
628         nm(jj+1,1) = var_middle(col_var,jj,1) + ...
            alfa_var_middle(col_var,jj)*(var_middle(col_var,jj,2) - var_middle(col_var,jj,1));
629         % f1 and f2
630         f1(jj+1,n) = q2(jj+1,n);
631         f2(jj+1,n) = q2(jj+1,n).*abs(q2(jj+1,n))./q1(jj+1,n) + g*q1(jj+1,n).*ybar(jj+1,n);
632         % Froude
633         A_f_irr = q1(jj+1,n);
634         col_var = 9;
635         B_f_irr = var_middle(col_var,jj,1) + ...
            alfa_var_middle(col_var,jj)*(var_middle(col_var,jj,2) - var_middle(col_var,jj,1));
636         Hm = A_f_irr./B_f_irr;
637         Fr(jj+1,n) = abs(q2(jj+1,n)./q1(jj+1,n))./((g*Hm).^0.5);% N mero de Froude
638         % Courant
639         Cn(jj+1,n) = (abs(q2(jj+1,n)./q1(jj+1,n))+(g*Hm).^0.5)/(dx/dt);% Courant Number
640         % Friction
641         if flag_friction == 1
642             J2(jj+1,n) = g*A_f_irr.*(I0(jj+1,1) - ...
                q2(jj+1,n).*abs(q2(jj+1,n)).*nm(jj+1,1).^2./(A_f_irr.^2.*Rh_middle(jj,1)^(4/3)));
643         else
644             J2(jj+1,n) = g*q1(jj+1,n).*(I0(jj+1,n) - ...
                f*q2(jj+1,n).*abs(q2(jj+1,n))./((q1(jj+1,n).^2)*8*g*Rh_midle(jj,1)));
645         end
646         % Stability Check
647         if Cn(jj+1,n) > 1
648             error('Please, decrease the time-step')
649         end
650     end
651 end
652 end
653
654 zzz = [y(ceil(Nx/2),:)', q2(ceil(Nx/2),:)',tint' , q2(1,:)]';
655 %%% Post Processing Figures %%%
656 % Call function
657 warning('on');
658 post_processing
659 toc

```

#### 4) SVE Post Processing

```

1  %% Post Processing Graphs
2  clf

```

```

3 close all
4
5
6 % Surfplot
7 t_save = [0:Nat:tt/dt];
8 t_save(1,1) = 1;
9 set(gcf,'units','inches','position',[2,2,4,5])
10 subplot(3,1,1)
11 surf(x,tint(t_save)/60,Fr(:,t_save));
12 view(0,90);
13 kk = colorbar ; colormap('jet')
14 shading interp
15 xlabel('x (m)','Interpreter','latex')
16 ylabel('t (h)','Interpreter','latex')
17 ylabel(kk,'Froude Number','Interpreter','latex')
18 zlabel('Froude Number','Interpreter','Latex');
19 xlim([0 L]);
20 ylim([0 tt/60/60]);
21
22 subplot(3,1,2)
23 surf(x,tint(t_save)/60/60,y(:,t_save));
24 view(0,90);
25 kk = colorbar ; colormap('jet')
26 shading interp
27 xlabel('x (m)','Interpreter','latex')
28 ylabel('t (h)','Interpreter','latex')
29 ylabel(kk,'y (m)','Interpreter','latex')
30 zlabel('y (m)','Interpreter','Latex');
31 xlim([0 L]);
32 ylim([0 tf/60/60]);
33 subplot(3,1,3)
34 wse = y + inv_el;
35 surf(x,tint(t_save)/60/60,wse(:,t_save));
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,'WSE (m)','Interpreter','latex')
42 zlabel('WSE (m)','Interpreter','Latex');
43 xlim([0 L]);
44 ylim([0 tf/60/60]);
45 exportgraphics(gcf,'Surf_Plots.pdf','ContentType','image','Colorspace','rgb','Resolution',600)
46 clf
47 close all
48
49 if flag_section == 2 % circular
50 % Video
51 obj = VideoWriter('Circular_Depth.avi','Motion JPEG AVI');
52 obj.Quality = 100;
53 obj.FrameRate = 20;
54 open(obj)
55 set(gcf,'units','inches','position',[2,2,10,3])
56 for n=1:1:(Nt/Nat)
57     if n == 1
58         t = 1;
59         pos = 1;
60     else
61         t=(n-1)*Nat*dt;
62         pos = t/dt;
63     end
64     % Circle Function
65     xcir = linspace(0,2*pi,100); % 100 points within 0 and 360 deg
66     cir = @(r,ctr) [r*cos(xcir)+ctr(1); r*sin(xcir)+ctr(2)];
67     c1 = cir(D/2, [D/2; D/2]);
68
69     % Boundary Circle
70     %  $(x - xc)^2 + (y - yc)^2 = D^2/4$ 
71     % where xc = D/2 and yc = D/2
72     xc = D/2; yc = D/2;
73     y01 = y(1,pos);
74     y02 = y(ceil(ceil(Nx/2)),pos);
75     y03 = y(Nx,pos);
76     y0_c = [y01; y02; y03];
77     % For a given known y, we have to find two xs, such that
78     %  $x^2 + (-2xc)x + ((y0 - yc)^2 - xc^2 - D^2/4)$ 
79     % or  $ax^2 + bx + c$ , with

```

```

80     % a = 1; b = -2xc; c = (y0 - yc)^2 - xc^2 - D^2/4
81     % x = (-b +/- sqrt(b^2 - 4ac)) / (2a)
82     a = 1;
83     b = -2*xc;
84     c = xc^2 + (y0_c - yc).^2 - D^2/4;
85     Delta = b^2 - 4*a.*c;
86     x1 = (-b + sqrt(Delta))/(2*a);
87     x2 = (-b - sqrt(Delta))/(2*a);
88     % Now we found the intersection of the circle and a line with know
89     % depth
90     subplot(1,3,1)
91     title(['t = ',num2str(round(round(t/60,2),2)), ' [min]'])
92     ylim([0 D]);
93     xlim([0 D]);
94     viscircles([D/2 D/2],D/2,'Color','black');
95     % plot(c1(1,:),c1(2:,:), 'Color','black');
96     hold on
97     x_water = linspace(x2(1),x1(1),100);
98     y_water = repmat(y01,1,100);
99     plot(x_water,y_water,'blue','linewidth',2);
100    % fill([c1(1,:) fliplr(c1(1:))], [y_water fliplr(c2(1:))], 'blue')
101    ylabel('y(m)','Interpreter','latex')
102    xlabel('B(m)','Interpreter','latex')
103    legend('Entrance','interpreter','latex')
104    hold off
105    % second section
106    subplot(1,3,2)
107    title(['t = ',num2str(round(round(t/60,2),2)), ' [min]'])
108    ylim([0 D]);
109    xlim([0 D]);
110    viscircles([D/2 D/2],D/2,'Color','black');
111    hold on
112    x_water = linspace(x2(2),x1(2),100);
113    y_water = repmat(y02,1,100);
114    plot(x_water,y_water,'blue','linewidth',2);
115    ylabel('y(m)','Interpreter','latex')
116    xlabel('B(m)','Interpreter','latex')
117    legend('x = L/2','interpreter','latex')
118    hold off
119    legend('L/2','interpreter','latex')
120    % third section
121    subplot(1,3,3)
122    title(['t = ',num2str(round(round(t/60,2),2)), ' [min]'])
123    ylim([0 D]);
124    xlim([0 D]);
125    viscircles([D/2 D/2],D/2,'Color','black');
126    hold on
127    x_water = linspace(x2(3),x1(3),100);
128    y_water = repmat(y03,1,100);
129    plot(x_water,y_water,'blue','linewidth',2);
130    ylabel('y(m)','Interpreter','latex')
131    xlabel('B(m)','Interpreter','latex')
132    legend('Exit','interpreter','latex')
133    % Save frame
134    title(['t = ',num2str(round(round(t/60,2),2)), ' [min]'])
135    f = getframe(gcf);
136    writeVideo(obj,f);
137    hold off
138    clf
139    end
140    obj.close();
141    end
142
143    if flag_section == 3 % paraboloid
144    % Video
145    obj = VideoWriter('Parabolic_Depth.avi','Motion JPEG AVI');
146    obj.Quality = 100;
147    obj.FrameRate = 20;
148    open(obj)
149    set(gcf,'units','inches','position',[2,2,10,3])
150    for n=1:1:(Nt/Nat)
151        if n == 1
152            t = 1;
153            pos = 1;
154        else
155            t=(n-1)*Nat*dt;
156            pos = t/dt;

```

```

157     end
158     % Parabolic Function
159     %  $y = a \cdot x^2 \Rightarrow x_{\max} = \sqrt{(y_{\max}/a)}$ 
160     ymax = max(max(y));
161     xmax = sqrt(ymax/a); % x to left and right directions
162     xpar = linspace(-xmax,xmax,100); % 100 points within -xmax and xmax deg
163     ypar = a.*xpar.^2;
164     % Now we found bottom of the channel
165     % We still need to find xleft and xright for a given y
166     y01 = y(1,pos);
167     y02 = y(ceil(Nx/2),pos);
168     y03 = y(Nx,pos);
169     y0_c = [y01; y02; y03];
170     xright = sqrt(y0_c/a);
171     xleft = - xright;
172     subplot(1,3,1)
173     title(['t = ',num2str(round(round(t/60,2),2)), ' [min]'])
174     ylim([0 ymax]);
175     xlim([0 ymax]);
176     plot(xpar,ypar,'Color','black');
177     hold on
178     x_water = linspace(xleft(1),xright(1),100);
179     y_water = linspace(y01,y01,100);
180     plot(x_water,y_water,'blue','linewidth',2);
181     ylabel('y(m)','Interpreter','latex')
182     xlabel('B(m)','Interpreter','latex')
183     legend('Entrance','interpreter','latex')
184     hold off
185     % second section
186     subplot(1,3,2)
187     title(['t = ',num2str(round(round(t/60,2),2)), ' [min]'])
188     ylim([0 ymax]);
189     xlim([0 ymax]);
190     plot(xpar,ypar,'Color','black');
191     hold on
192     x_water = linspace(xleft(2),xright(2),100);
193     y_water = linspace(y02,y02,100);
194     plot(x_water,y_water,'blue','linewidth',2);
195     ylabel('y(m)','Interpreter','latex')
196     xlabel('B(m)','Interpreter','latex')
197     legend('x = L/2','interpreter','latex')
198     hold off
199     % third section
200     subplot(1,3,3)
201     title(['t = ',num2str(round(round(t/60,2),2)), ' [min]'])
202     ylim([0 ymax]);
203     xlim([0 ymax]);
204     plot(xpar,ypar,'Color','black');
205     hold on
206     x_water = linspace(xleft(3),xright(3),100);
207     y_water = linspace(y03,y03,100);
208     plot(x_water,y_water,'blue','linewidth',2);
209     ylabel('y(m)','Interpreter','latex')
210     xlabel('B(m)','Interpreter','latex')
211     legend('Outlet','interpreter','latex')
212     % Save frame
213     title(['t = ',num2str(round(round(t/60,2),2)), ' [min]'])
214     f = getframe(gcf);
215     writeVideo(obj,f);
216     hold off
217     clf
218     end
219 obj.close();
220 end
221
222
223 % Plots
224 set(gcf,'units','inches','position',[0,0,7,12])
225 subplot(3,2,1)
226 % Flows
227 plot(tint/60,q2(1,:), 'LineStyle','--', 'LineWidth',2, 'Color','k')
228 hold on
229 plot(tint/60,q2(ceil(Nx/2),:), 'LineStyle',':', 'LineWidth',2, 'Color','k')
230 hold on
231 plot(tint/60,q2(Nx,:), 'LineStyle','-', 'LineWidth',2, 'Color','k')
232 hold on
233 xlabel('Elapsed Time (min)','Interpreter','latex');

```

```

234 ylabel('Flow Discharge (m\textsuperscript{3}/s)', 'Interpreter', 'latex');
235 legend('Entrance', 'L/2', 'Outlet', 'Interpreter', 'Latex', 'location', 'best')
236 % Velocity
237 subplot(3,2,2)
238 plot(tint/60, q2(1,:) ./ q1(1,:), 'LineStyle', '--', 'LineWidth', 2, 'Color', 'k')
239 hold on
240 plot(tint/60, q2(ceil(Nx/2),:) ./ q1(ceil(Nx/2),:), 'LineStyle', ':', 'LineWidth', 2, 'Color', 'k')
241 hold on
242 plot(tint/60, q2(Nx,:) ./ q1(Nx,:), 'LineStyle', '-', 'LineWidth', 2, 'Color', 'k')
243 %%% Normal Depth Velocity %%%
244 xlabel('Elapsed Time (min)', 'Interpreter', 'latex');
245 ylabel('Velocity (m/s)', 'Interpreter', 'latex');
246 legend('Entrance', 'L/2', 'Outlet', 'Interpreter', 'Latex', 'Location', 'best')
247 % Water Depth
248 subplot(3,2,3)
249 plot(tint/60, y(1,:), 'LineStyle', '--', 'LineWidth', 2, 'Color', 'k')
250 hold on
251 plot(tint/60, y(ceil(Nx/2),:), 'LineStyle', ':', 'LineWidth', 2, 'Color', 'k')
252 hold on
253 plot(tint/60, y(Nx,:), 'LineStyle', '-', 'LineWidth', 2, 'Color', 'k')
254 xlabel('Elapsed Time (min)', 'Interpreter', 'latex');
255 ylabel('Water Depths (m)', 'Interpreter', 'latex');
256 legend('Entrance', 'L/2', 'Outlet', 'Interpreter', 'Latex', 'Location', 'best')
257 % Froude Number
258 subplot(3,2,4)
259 plot(tint/60, Fr(1,:), 'LineStyle', '--', 'LineWidth', 2, 'Color', 'k')
260 hold on
261 plot(tint/60, Fr(ceil(Nx/2),:), 'LineStyle', ':', 'LineWidth', 2, 'Color', 'k')
262 hold on
263 plot(tint/60, Fr(Nx,:), 'LineStyle', '-', 'LineWidth', 2, 'Color', 'k')
264 xlabel('Elapsed Time (min)', 'Interpreter', 'latex');
265 ylabel('Froude Number', 'Interpreter', 'latex');
266 legend('Entrance', 'L/2', 'Outlet', 'Interpreter', 'Latex', 'Location', 'best')
267
268 % Courant Number
269 subplot(3,2,5)
270 plot(tint/60, Cn(1,:), 'LineStyle', '--', 'LineWidth', 2, 'Color', 'k')
271 hold on
272 plot(tint/60, Cn(ceil(Nx/2),:), 'LineStyle', ':', 'LineWidth', 2, 'Color', 'k')
273 hold on
274 plot(tint/60, Cn(Nx,:), 'LineStyle', '-', 'LineWidth', 2, 'Color', 'k')
275 xlabel('Elapsed Time (min)', 'Interpreter', 'latex');
276 ylabel('Courant Number', 'Interpreter', 'latex');
277 legend('Entrance', 'L/2', 'Outlet', 'Interpreter', 'Latex', 'Location', 'best')
278
279 % Rating Curve
280 % Solving for normal Depth
281 ymin = min(min(y));
282 ymax = max(max(y));
283 y_m = [ymin:0.01:ymax]'; % meters
284 hs = 1; % half section
285 % hs = ceil(1);
286 if flag_section ~= 4
287     Qn = 1/nm(hs) .* A_function(D, Z1, Z2, a, b, y_m) .* Rh_function(D, Z1, Z2, a, b, y_m) .^(2/3) .* IO(hs)^0.5;
288 else
289     % [y_table, A, P, Rh, y_bar, n_med, Beta, v, B, Q]
290     % [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
291     coll = 2; % Col with A
292     for jj = 1:length(q1(hs,:))
293         Qn(jj,1) = Vlookup_g(irr_table, coll, q1(hs, jj), 10); % Attention here
294         y_m(jj,1) = Vlookup_g(irr_table, coll, q1(hs, jj), 1);
295         rh_i = Vlookup_g(irr_table, coll, q1(hs, jj), 4);
296     end
297 end
298 subplot(3,2,6)
299 tbegin = 30; % (steps), considering initial stabilization of the domain
300 plot(q2(hs, tbegin:end), y(hs, tbegin:end), 'LineStyle', '--', 'LineWidth', 2, 'Color', 'k')
301 hold on
302 plot(q2(ceil(Nx/2), tbegin:end), y(ceil(Nx/2), tbegin:end), 'LineStyle', ':', 'LineWidth', 2, 'Color', 'k')
303 hold on
304 plot(Qn, y_m, 'LineStyle', '-', 'LineWidth', 2, 'Color', 'k')
305 xlabel('Flow Discharge (m\textsuperscript{3}/s)', 'Interpreter', 'latex');
306 ylabel('Water Depth (m)', 'Interpreter', 'latex');
307 ylim([ymin 1.1*max([max(y_m), max(y(ceil(Nx)))])]);
308 legend('Q(Inlet)', 'Q(Nx/2)', '$Q_{n}$ (L)', 'Interpreter', 'Latex', 'Location', 'best')
309 hold off
310 exportgraphics(gcf, 'Summary_Charts.pdf', 'ContentType', 'vector')

```

```

311 clf
312 close all
313
314 % Channel Width Chart
315 h_f = figure;
316 axis tight manual % this ensures that getframe() returns a consistent size
317 filename = 'channel_width.gif';
318 B2 = zeros(size(q1));
319 if flag_section ≠ 4
320     B2 = B_function(D,Z1,Z2,a,b,y);
321 else
322     for pos_b = 1:length(q1(:,1))
323         for time_b = 1:length(q1(1,:))
324             % [y_table, A, P, Rh, y_bar, n_med, Beta, v, B, Q]
325             % [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
326             B2(pos_b,time_b) = Vlookup_g(irr_table,coll,q1(pos_b,time_b),9);
327         end
328     end
329 end
330 for n=1:1:(Nt/Nat)
331     if n == 1
332         t = 1;
333         pos = 1;
334     else
335         t=(n-1)*Nat*dt;
336         pos = t/dt;
337     end
338     left_margin = B2(:,pos)/2; right_margin = -B2(:,pos)/2;
339     plot(x,left_margin,'k','LineWidth',2);
340     hold on
341     plot(x,right_margin,'k','LineWidth',2);
342     hold on
343     fill([x' fliplr(x')],[right_margin' fliplr(left_margin)],'blue')
344     xlabel('x [m]');
345     ylabel('B [m]');
346     ylim([1.1*min(min(-B2/2)), max(max(1.1*(B2/2)))]);
347     grid on
348     %legend('y(x,t)',2)
349     title(['t = ',num2str(round(t/60,2)), ' [min]'])
350     drawnow
351     % Capture the plot as an image
352     frame = getframe(h_f);
353     im = frame2im(frame);
354     [imind,cm] = rgb2ind(im,256);
355     % Write to the GIF File
356     if n == 1
357         imwrite(imind,cm,filename,'gif','Loopcount',inf);
358     else
359         imwrite(imind,cm,filename,'gif','WriteMode','append');
360     end
361     hold off
362 end
363 clf
364 close all
365
366 % Water Depth Profile
367 h_f = figure;
368 axis tight manual % this ensures that getframe() returns a consistent size
369 filename = 'channel_wse_profile.gif';
370 wse = inv_el + y; % water surface elevation in meters
371 for n=1:1:(Nt/Nat)
372     if n == 1
373         t = 1;
374         pos = 1;
375     else
376         t=(n-1)*Nat*dt;
377         pos = t/dt;
378     end
379     plot(x,inv_el,'LineWidth',4,'LineStyle','-','Color','k')
380     hold on
381     plot(x,wse(:,pos),'k','LineWidth',2,'LineStyle','-','Color','blue')
382     fill([x' fliplr(x')],[inv_el' fliplr(wse(:,pos))],'blue')
383     xlabel('x [m]','Interpreter','latex');
384     ylabel('Water surface Elevation [m]','Interpreter','latex');
385     ylim([0.98*min(min(wse - y)) max(max(1.01*wse))])
386     grid on
387     %legend('y(x,t)',2)

```



```

388     title(['t = ',num2str(round(t/60,2)), ' [min]'])
389     drawnow
390     % Capture the plot as an image
391     frame = getframe(h_f);
392     im = frame2im(frame);
393     [imind,cm] = rgb2ind(im,256);
394     % Write to the GIF File
395     if n == 1
396         imwrite(imind,cm,filename,'gif', 'Loopcount',inf);
397     else
398         imwrite(imind,cm,filename,'gif','WriteMode','append');
399     end
400     hold off
401     %mov=addframe(mov,Mo); %Para gravar o v deo, n o comentar (excluir %)
402 end
403 close all
404 % 3D Channel Water Surface Elevation
405
406 % Video
407 obj = VideoWriter('WSE.avi','Motion JPEG AVI');
408 obj.Quality = 100;
409 obj.FrameRate = 20;
410 open(obj)
411 for n=1:1:(Nt/Nat)
412     if n == 1
413         t = 1;
414         pos = 1;
415     else
416         t=(n-1)*Nat*dt;
417         pos = t/dt;
418     end
419     plot(x,inv_el,'LineWidth',4,'LineStyle','-','Color','k')
420     hold on
421     plot(x,wse(:,pos),'k','LineWidth',2,'LineStyle','-','Color','blue')
422     fill([x' fliplr(x')], [inv_el' fliplr(wse(:,pos)')], 'blue')
423     xlabel('x [m]','Interpreter','latex');
424     ylabel('Water surface Elevation [m]','Interpreter','latex');
425     ylim([0.98*min(min(wse - y)) max(max(1.01*wse))])
426     grid on
427     title(['t = ',num2str(round(round(t/60,2),2)), ' [min]'])
428     f = getframe(gcf);
429     writeVideo(obj,f);
430     hold off
431 end
432 obj.close();
433
434 %% Ploting Irregular Cross-section
435 % Cross-section Depths
436 % Water Depth Profile
437 % Video
438 %%%%%%%%%%% Figure Without Fill %%%%%%%%%%%
439 % sm = (1e-8 + 1);
440 % obj = VideoWriter('Cross_section_outlet.avi','Motion JPEG AVI');
441 % obj.Quality = 100;
442 % obj.FrameRate = 20;
443 % open(obj)
444 % set(gcf,'units','inches','position',[2,0,6.5,8])
445 % for n=1:1:(Nt/Nat)
446 %     if n == 1
447 %         t = 1;
448 %     else
449 %         t=(n-1)*Nat*dt;
450 %     end
451 %     subplot(3,1,1)
452 %     pos = Nx; % Position where the Plots will be made (Outlet)
453 %     y_cs = [y(pos,t) y(pos,t)]; % Vector of Water Depth (m)
454 %     plot(x_cross,y_cross,'LineWidth',1.5,'Color','k') % Plotting Cross-Section
455 %     hold on
456 %     scatter(x_cross,y_cross,'o','b') % Plotting Break Points
457 %     % Determine x_inv e y_inv
458 %     min_el = min(y_cross); % Minimum elevation (m)
459 %     pos_inv = find(y_cross == min_el); % Position where it occurs
460 %     x_inv = x_cross(pos_inv); y_inv = y_cross(pos_inv); % x coordinate of invert (m)
461 %     % Determine x_left
462 %     x_left_unsorted = x_cross(1:(pos_inv-1),1); % Left values of x
463 %     y_left_unsorted = y_cross(1:(pos_inv-1),1); % Right values of x
464 %     pos_left_up = find(y_left_unsorted>sm*y_cs(1),1,'last'); % left postion with y > ym

```

```

465 % pos_left_down = (pos_left_up + 1); % down postion of y > ym
466 % % x and y for left points
467 % x_left_up = x_left_unsorted(pos_left_up,1); % x(m)
468 % x_left_down = x_left_unsorted(pos_left_down,1); % x(m)
469 % y_left_up = y_left_unsorted(pos_left_up,1); % y(m)
470 % y_left_down = y_left_unsorted(pos_left_down,1); % y(m)
471 % alfa_l = (y_left_up - y_left_down)/(abs(x_left_up - x_left_down)); % Left angle
472 % dy = (abs(y_cs(1,1) - y_left_down)); % Difference in water depth
473 % x_begin = x_left_down - dy/alfa_l; % Intersection with left bank
474 % x_end = x_begin + B2(pos,t); % Intersection with right bank
475 % x_b = [x_begin x_end]; % Vector of x
476 % plot(x_b,y_cs,'b','LineWidth',2); % Plot of top-width
477 % hold off
478 % grid on
479 % xlabel('x(m)','Interpreter','latex')
480 % ylabel('y(m)','Interpreter','latex')
481 % title(['Inlet (t) = ',num2str(round(round(t/60,2),2)), ' [min]'])
482 %
483 % subplot(3,1,2)
484 % pos = ceil(Nx/2); % Position where the Plots will be made (Half)
485 % y_cs = [y(pos,t) y(pos,t)]; % Vector of Water Depth (m)
486 % plot(x_cross,y_cross,'LineWidth',1.5,'Color','k') % Plotting Cross-Section
487 % hold on
488 % scatter(x_cross,y_cross,'o','b') % Plotting Break Points
489 % % Determine x_inv e y_inv
490 % min_el = min(y_cross); % Minimum elevation (m)
491 % pos_inv = find(y_cross == min_el); % Position where it occurs
492 % x_inv = x_cross(pos_inv); y_inv = y_cross(pos_inv); % x coordinate of invert (m)
493 % % Determine x_left
494 % x_left_unsorted = x_cross(1:(pos_inv-1),1); % Left values of x
495 % y_left_unsorted = y_cross(1:(pos_inv-1),1); % Right values of x
496 % pos_left_up = find(y_left_unsorted>sm*y_cs(1,1),'last'); % left postion with y > ym
497 % pos_left_down = (pos_left_up + 1); % down postion of y > ym
498 % % x and y for left points
499 % x_left_up = x_left_unsorted(pos_left_up,1); % x(m)
500 % x_left_down = x_left_unsorted(pos_left_down,1); % x(m)
501 % y_left_up = y_left_unsorted(pos_left_up,1); % y(m)
502 % y_left_down = y_left_unsorted(pos_left_down,1); % y(m)
503 % alfa_l = (y_left_up - y_left_down)/(abs(x_left_up - x_left_down)); % Left angle
504 % dy = (abs(y_cs(1,1) - y_left_down)); % Difference in water depth
505 % x_begin = x_left_down - dy/alfa_l; % Intersection with left bank
506 % x_end = x_begin + B2(pos,t); % Intersection with right bank
507 % x_b = [x_begin x_end]; % Vector of x
508 % plot(x_b,y_cs,'b','LineWidth',2); % Plot of top-width
509 % hold off
510 % grid on
511 % xlabel('x(m)','Interpreter','latex')
512 % ylabel('y(m)','Interpreter','latex')
513 % title(['L/2 (t) = ',num2str(round(round(t/60,2),2)), ' [min]'])
514 %
515 % subplot(3,1,3)
516 % pos = Nx; % Position where the Plots will be made (Half)
517 % y_cs = [y(pos,t) y(pos,t)]; % Vector of Water Depth (m)
518 % plot(x_cross,y_cross,'LineWidth',1.5,'Color','k') % Plotting Cross-Section
519 % hold on
520 % scatter(x_cross,y_cross,'o','b') % Plotting Break Points
521 % % Determine x_inv e y_inv
522 % min_el = min(y_cross); % Minimum elevation (m)
523 % pos_inv = find(y_cross == min_el); % Position where it occurs
524 % x_inv = x_cross(pos_inv); y_inv = y_cross(pos_inv); % x coordinate of invert (m)
525 % % Determine x_left
526 % x_left_unsorted = x_cross(1:(pos_inv-1),1); % Left values of x
527 % y_left_unsorted = y_cross(1:(pos_inv-1),1); % Right values of x
528 % pos_left_up = find(y_left_unsorted>sm*y_cs(1,1),'last'); % left postion with y > ym
529 % pos_left_down = (pos_left_up + 1); % down postion of y > ym
530 % % x and y for left points
531 % x_left_up = x_left_unsorted(pos_left_up,1); % x(m)
532 % x_left_down = x_left_unsorted(pos_left_down,1); % x(m)
533 % y_left_up = y_left_unsorted(pos_left_up,1); % y(m)
534 % y_left_down = y_left_unsorted(pos_left_down,1); % y(m)
535 % alfa_l = (y_left_up - y_left_down)/(abs(x_left_up - x_left_down)); % Left angle
536 % dy = (abs(y_cs(1,1) - y_left_down)); % Difference in water depth
537 % x_begin = x_left_down - dy/alfa_l; % Intersection with left bank
538 % x_end = x_begin + B2(pos,t); % Intersection with right bank
539 % x_b = [x_begin x_end]; % Vector of x
540 % plot(x_b,y_cs,'b','LineWidth',2); % Plot of top-width
541 % hold off

```

```

542 % grid on
543 % xlabel('x(m)','Interpreter','latex')
544 % ylabel('y(m)','Interpreter','latex')
545 % title(['L (t) = ',num2str(round(round(t/60,2),2)), ' [min]'])
546 %
547 % % Writting Video
548 % f = getframe(gcf);
549 % writeVideo(obj,f);
550 % hold off
551 % clf
552 %
553 % end
554 % close all
555 if flag_section == 4
556 sm = (1e-8 + 1);
557 obj = VideoWriter('Cross_section_outlet.avi','Motion JPEG AVI');
558 obj.Quality = 100;
559 obj.FrameRate = 5;
560 open(obj)
561 set(gcf,'units','inches','position',[2,0,6.5,8])
562 for n=1:1:(NT/Nat)
563     if n == 1
564         t = 1;
565         pos = 1;
566     else
567         t=(n-1)*Nat*dt;
568         pos = t/dt;
569     end
570 subplot(3,1,1)
571 pos_x = Nx; % Position where the Plots will be made (Inlet)
572 y_cs = [y(pos_x,pos) y(pos_x,pos)]; % Vector of Water Depth (m)
573 plot(x_cross,y_cross,'LineWidth',1.5,'Color','k') % Plotting Cross-Section
574 hold on
575 scatter(x_cross,y_cross,'o','b') % Plotting Break Points
576 % Determine x_inv e y_inv
577 min_el = min(y_cross); % Minimum elevation (m)
578 pos_inv = find(y_cross == min_el); % Position where it occurs
579 x_inv = x_cross(pos_inv); y_inv = y_cross(pos_inv); % x coordinate of invert (m)
580 % Determine x_left
581 x_left_unsorted = x_cross(1:(pos_inv-1),1); % Left values of x
582 y_left_unsorted = y_cross(1:(pos_inv-1),1); % Right values of x
583 y_right_unsorted = y_cross(pos_inv + 1:end,1); % Right values of x
584 x_right_unsorted = x_cross(pos_inv + 1:end,1); % Right values of x
585 pos_left_up = find(y_left_unsorted>sm*y_cs(1),1,'last'); % left postion with y > ym
586 if pos_left_up == length(y_left_unsorted)
587     pos_left_down = pos_left_up;
588     x_left_up = x_left_unsorted(pos_left_up,1); % x(m)
589     x_left_down = x_inv;
590     y_left_up = y_left_unsorted(pos_left_up,1); % y(m)
591     y_left_down = y_inv;
592     alfa_l = (y_left_up - y_left_down)/(abs(x_left_up - x_left_down)); % Left angle
593     dy = (abs(y_cs(1,1) - y_left_down)); % Difference in water depth
594 else
595     pos_left_down = (pos_left_up + 1); % down postion of y > ym
596     % x and y for left points
597     x_left_up = x_left_unsorted(pos_left_up,1); % x(m)
598     x_left_down = x_left_unsorted(pos_left_down,1); % x(m)
599     y_left_up = y_left_unsorted(pos_left_up,1); % y(m)
600     y_left_down = y_left_unsorted(pos_left_down,1); % y(m)
601     alfa_l = (y_left_up - y_left_down)/(abs(x_left_up - x_left_down)); % Left angle
602     dy = (abs(y_cs(1,1) - y_left_down)); % Difference in water depth
603 end
604 pos_right_end = find(y_right_unsorted>sm*y_cs(1),1,'first'); % left postion with y > ym
605
606 if isnan(alfa_l)
607     x_begin = x_left_down;
608 else
609     x_begin = x_left_down - dy/alfa_l; % Intersection with left bank
610 end
611 x_end = x_begin + B2(pos_x,pos); % Intersection with right bank
612 x_b = [x_begin x_end]; % Vector of x
613 plot(x_b,y_cs,'b','LineWidth',2); % Plot of top-width
614 x_flip = linspace(x_b(1,1),x_b(1,2),length(x_cross));
615 %%%
616 y_section = [y_cs(1,1) ; y_left_unsorted(pos_left_up:end,1); y_inv; ...
    y_right_unsorted(1:(pos_right_end));y_cs(1,2)];

```

```

617 x_section = [x_b(1,1) ; x_left_unsorted(pos_left_up:end,1); x_inv; ...
        x_right_unsorted(1:(pos_right_end));x_b(1,2)];
618 % y_section = linspace(y_cs(1,1),y_cs(1,2),length(x_cross));
619 %%%
620 y_flip = linspace(y_cs(1,1),y_cs(1,2),length(y_section));
621 p = fill([x_section' fliplr(x_section')],[y_section' fliplr(y_flip')],'blue');
622 p.EdgeColor = [1 1 1];
623 hold off
624 grid on
625 xlabel('x(m)','Interpreter','latex')
626 ylabel('y(m)','Interpreter','latex')
627 xlim([min(x_cross) max(x_cross)])
628 title(['Outlet (t) = ',num2str(round(round(t/60,2),2)), ' [min]'])
629
630 subplot(3,1,2)
631 pos_x = ceil(Nx/2); % Position where the Plots will be made (Inlet)
632 y_cs = [y(pos_x,pos) y(pos_x,pos)]; % Vector of Water Depth (m)
633 plot(x_cross,y_cross,'LineWidth',1.5,'Color','k') % Plotting Cross-Section
634 hold on
635 scatter(x_cross,y_cross,'o','b') % Plotting Break Points
636 % Determine x_inv e y_inv
637 min_el = min(y_cross); % Minimum elevation (m)
638 pos_inv = find(y_cross == min_el); % Position where it occurs
639 x_inv = x_cross(pos_inv); y_inv = y_cross(pos_inv); % x coordinate of invert (m)
640 % Determine x_left
641 x_left_unsorted = x_cross(1:(pos_inv-1),1); % Left values of x
642 y_left_unsorted = y_cross(1:(pos_inv-1),1); % Right values of x
643 y_right_unsorted = y_cross(pos_inv + 1:end,1); % Right values of x
644 x_right_unsorted = x_cross(pos_inv + 1:end,1); % Right values of x
645 pos_left_up = find(y_left_unsorted>sm*y_cs(1),1,'last'); % left postion with y > ym
646 if pos_left_up == length(y_left_unsorted)
647     pos_left_down = pos_left_up;
648     x_left_up = x_left_unsorted(pos_left_up,1); % x(m)
649     x_left_down = x_inv;
650     y_left_up = y_left_unsorted(pos_left_up,1); % y(m)
651     y_left_down = y_inv;
652     alfa_l = (y_left_up - y_left_down)/(abs(x_left_up - x_left_down)); % Left angle
653     dy = (abs(y_cs(1,1) - y_left_down)); % Difference in water depth
654 else
655     pos_left_down = (pos_left_up + 1); % down postion of y > ym
656     % x and y for left points
657     x_left_up = x_left_unsorted(pos_left_up,1); % x(m)
658     x_left_down = x_left_unsorted(pos_left_down,1); % x(m)
659     y_left_up = y_left_unsorted(pos_left_up,1); % y(m)
660     y_left_down = y_left_unsorted(pos_left_down,1); % y(m)
661     alfa_l = (y_left_up - y_left_down)/(abs(x_left_up - x_left_down)); % Left angle
662     dy = (abs(y_cs(1,1) - y_left_down)); % Difference in water depth
663 end
664 pos_right_end = find(y_right_unsorted>sm*y_cs(1),1,'first'); % left postion with y > ym
665
666 if isnan(alfa_l)
667     x_begin = x_left_down;
668 else
669     x_begin = x_left_down - dy/alfa_l; % Intersection with left bank
670 end
671 x_end = x_begin + B2(pos_x,pos); % Intersection with right bank
672 x_b = [x_begin x_end]; % Vector of x
673 plot(x_b,y_cs,'b','LineWidth',2); % Plot of top-width
674 x_flip = linspace(x_b(1,1),x_b(1,2),length(x_cross));
675 %%%
676 y_section = [y_cs(1,1) ; y_left_unsorted(pos_left_up:end,1); y_inv; ...
        y_right_unsorted(1:(pos_right_end));y_cs(1,2)];
677 x_section = [x_b(1,1) ; x_left_unsorted(pos_left_up:end,1); x_inv; ...
        x_right_unsorted(1:(pos_right_end));x_b(1,2)];
678 % y_section = linspace(y_cs(1,1),y_cs(1,2),length(x_cross));
679 %%%
680 y_flip = linspace(y_cs(1,1),y_cs(1,2),length(y_section));
681 p = fill([x_section' fliplr(x_section')],[y_section' fliplr(y_flip')],'blue');
682 p.EdgeColor = [1 1 1];
683 hold off
684 grid on
685 xlabel('x(m)','Interpreter','latex')
686 ylabel('y(m)','Interpreter','latex')
687 xlim([min(x_cross) max(x_cross)])
688 title(['L/2 (t) = ',num2str(round(round(t/60,2),2)), ' [min]'])
689
690 subplot(3,1,3)

```

```

691 pos_x = Nx; % Position where the Plots will be made (Inlet)
692 y_cs = [y(pos_x,pos) y(pos_x,pos)]; % Vector of Water Depth (m)
693 plot(x_cross,y_cross,'LineWidth',1.5,'Color','k') % Plotting Cross-Section
694 hold on
695 scatter(x_cross,y_cross,'o','b') % Plotting Break Points
696 % Determine x_inv e y_inv
697 min_el = min(y_cross); % Minimum elevation (m)
698 pos_inv = find(y_cross == min_el); % Position where it occurs
699 x_inv = x_cross(pos_inv); y_inv = y_cross(pos_inv); % x coordinate of invert (m)
700 % Determine x_left
701 x_left_unsorted = x_cross(1:(pos_inv-1),1); % Left values of x
702 y_left_unsorted = y_cross(1:(pos_inv-1),1); % Right values of x
703 y_right_unsorted = y_cross(pos_inv + 1:end,1); % Right values of x
704 x_right_unsorted = x_cross(pos_inv + 1:end,1); % Right values of x
705 pos_left_up = find(y_left_unsorted>sm*y_cs(1),1,'last'); % left postion with y > ym
706 if pos_left_up == length(y_left_unsorted)
707     pos_left_down = pos_left_up;
708     x_left_up = x_left_unsorted(pos_left_up,1); % x(m)
709     x_left_down = x_inv;
710     y_left_up = y_left_unsorted(pos_left_up,1); % y(m)
711     y_left_down = y_inv;
712     alfa_l = (y_left_up - y_left_down)/(abs(x_left_up - x_left_down)); % Left angle
713     dy = (abs(y_cs(1,1) - y_left_down)); % Difference in water depth
714 else
715     pos_left_down = (pos_left_up + 1); % down postion of y > ym
716     % x and y for left points
717     x_left_up = x_left_unsorted(pos_left_up,1); % x(m)
718     x_left_down = x_left_unsorted(pos_left_down,1); % x(m)
719     y_left_up = y_left_unsorted(pos_left_up,1); % y(m)
720     y_left_down = y_left_unsorted(pos_left_down,1); % y(m)
721     alfa_l = (y_left_up - y_left_down)/(abs(x_left_up - x_left_down)); % Left angle
722     dy = (abs(y_cs(1,1) - y_left_down)); % Difference in water depth
723 end
724 pos_right_end = find(y_right_unsorted>sm*y_cs(1),1,'first'); % left postion with y > ym
725
726 if isnan(alfa_l)
727     x_begin = x_left_down;
728 else
729     x_begin = x_left_down - dy/alfa_l; % Intersection with left bank
730 end
731 x_end = x_begin + B2(pos_x,pos); % Intersection with right bank
732 x_b = [x_begin x_end]; % Vector of x
733 plot(x_b,y_cs,'b','LineWidth',2); % Plot of top-width
734 x_flip = linspace(x_b(1,1),x_b(1,2),length(x_cross));
735 %%%
736 y_section = [y_cs(1,1) ; y_left_unsorted(pos_left_up:end,1); y_inv; ...
737             y_right_unsorted(1:(pos_right_end));y_cs(1,2)];
738 x_section = [x_b(1,1) ; x_left_unsorted(pos_left_up:end,1); x_inv; ...
739             x_right_unsorted(1:(pos_right_end));x_b(1,2)];
740 % y_section = linspace(y_cs(1,1),y_cs(1,2),length(x_cross));
741 %%%
742 y_flip = linspace(y_cs(1,1),y_cs(1,2),length(y_section));
743 p = fill([x_section' fliplr(x_section')],[y_section' fliplr(y_flip')],'blue');
744 p.EdgeColor = [1 1 1];
745 hold off
746 grid on
747 xlabel('x(m)','Interpreter','latex')
748 ylabel('y(m)','Interpreter','latex')
749 xlim([min(x_cross) max(x_cross)])
750 title(['Inlet (t) = ',num2str(round(round(t/60,2),2)), ' [min]'])
751
752 % Writting Video
753 f = getframe(gcf);
754 writeVideo(obj,f);
755 hold off
756 clf
757 end
758 end
759 obj.close();
760 close all

```

## 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)