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

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

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

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

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

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

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

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

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

B. Data derived from ANA

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

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

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

The algorithm begins by finding the lowest bottom elevation of the riverbed ly, then, two vectors are defined ($\boldsymbol{seg_x}$ and $\boldsymbol{seg_y}$) with consecutive pairs or coordinates for both axis, this aims to determine the flow area between the water depth j and the boundaries of the riverbed (see Fig. 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

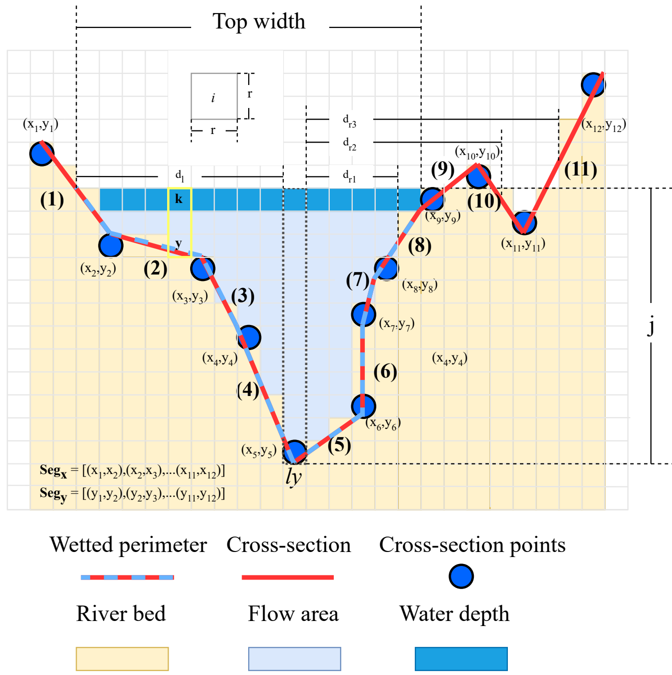

Fig. 1: Example of cross-section discretization with finite element and riverbed boundaries identification according to a water depth j.

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 $\boldsymbol{seg_y}$ is identified between of which pair of coordinates or segments j belongs to. It is worth mentioning that through this method many segments could be considered, as shown in Fig. 1 where j intersect segments 1, 8, 10, and 11, to solve this, the pairs of horizontal coordinates from those segments in $\boldsymbol{seg_x}$ are filtered by considering the closer distance of the average of those pair of coordinates related to the station of ly for left and right sides, for instance, on the right side the distance d_{r1} is lower than d_{r2} and d_{r3}, for the left side there is just a segment to be considered.

1) Flow area and centroid

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

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

$$f_2 = x_{\mathrm{i}+1} - \left(\frac{y_{\mathrm{i}+1} - y}{y_{\mathrm{i}} - y_{\mathrm{i}+1}}\right)(x_{\mathrm{i}+1} - x_{\mathrm{i}}) \tag{2}$$

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)).

$$f_3(i) = \sqrt{(y_\mathrm{i} - y_{\mathrm{i}+1})^2 + (x_\mathrm{i} - x_{\mathrm{i}+1})^2} \tag{3}$$

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, ϕ Eq. (3), flow, and top width are calculated. A pseudocode of the main algorithm is shown in Algorithm 1 to briefly introduce the algorithm structure.

Algorithm 1 Finite Element Procedure with nested loops

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

```
mg= matrix of zeros(((x_max − x_min) ∗ r, (y_max-y_min) ∗ r)
for j=y_min ∗ r + 1; y_max ∗ r do
    for i : seg_y do
        if seg_y[i][0] >= j/r > seg_y[i][1] or seg_y[i][0] <= j/r < seg_y[i][1] then
            seg_y2 = append(i)
            seg_x2 = append((seg_x[i][0] + seg_x[i][1])/2 − mid)
        end if
    end for
    seg_x3 = array(seg_x2)
    lw = max argument((where(seg_x3 < 0, seg_x3, -inf))
    rw = min argument((where(seg_x3 > 0, seg_x3, inf))
    if lf == rw then
        break the loop
    end if
    for i = f_2(lw, j/r) ∗ r − x_min ∗ r : f_2(rw, j/r) ∗ r − x_min ∗ r do
        for k : seg_x do
            compute: calculate flow area from the matrix.
            compute: calculate relative centroid for every column.
        end for
    end for
    for i = lw + 1 : rw do
        per=append(f_3(i))
    end for
    compute: calculate distance for the first segment intersected by j.
    compute: calculate distance for the last segment intersected by j.
    compute: sum the cumulated area, perimeter, top width and vertical centroid for the j water depth and then reset values.
    compute: hydraulic radius, centroid, convenyance, streamflow, flow velocity for the j water depth.
end for
```

4) Main Python Code

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

import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
from matplotlib import pyplot
from numpy import exp

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

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

Ymax, Ymin, Xmax, Xmin = max(coors[:, 1]), min(coors[:, 1]), max(coors[:, 0]), min(coors[:,0 ])   # ...
    Maximum and minimum values of the list of coordinates
for m in range(len(coors)):
    if coors[m][1] ≤ Ymin:  # Looking for the middle part of the cross-section
        middle = coors[m][0]
# --- Preallocate HP --- #
area, top, = np.zeros((int(Ymax*res - Ymin*res), 1)), np.zeros((int(Ymax*res - Ymin*res), 1))
perimeter_2, y = np.zeros((int(Ymax*res - Ymin*res), 1)), np.zeros((int(Ymax*res - Ymin*res), 1))
RH, centroid = np.zeros((int(Ymax*res - Ymin*res), 1)), np.zeros((int(Ymax*res - Ymin*res), 1))
con, phi = np.zeros((int(Ymax*res - Ymin*res), 1)), np.zeros((int(Ymax*res - Ymin*res), 1))
Q, center = np.zeros((int(Ymax*res - Ymin*res), 1)), np.zeros((int(Ymax*res - Ymin*res), 1))
seg_x, seg_y = np.zeros((len(coors[:, 0]) - 1, 2)), np.zeros((len(coors[:, 0]) - 1, 2))

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


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


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


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


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

for j in range(int(round(Ymin*res))+1, int(round(Ymax*res))):  # Looping thought the vertical axis
    seg_x_2, seg_y_2 = np.zeros((len(seg_y), 1)), np.zeros((len(seg_y), 1))
    for i in range(len(seg_y)):  # finding the upper boundary of the water deep
        if (seg_y[i, 0] ≥ j/res > seg_y[i, 1]) or (seg_y[i, 0] ≤ j/res ≤ seg_y[i, 1]):
            seg_y_2[i, 0] = i
```

```
            seg_x_2[i, 0] = (seg_x[i, 0]+seg_x[i, 1])/2 - middle
    left_wall = np.where(seg_x_2 < 0, seg_x_2, -np.inf).argmax()  # Finding the walls that contains ...
        the current
    right_wall = np.where(seg_x_2 > 0, seg_x_2, np.inf).argmin()   # water level

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

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

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

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

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

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

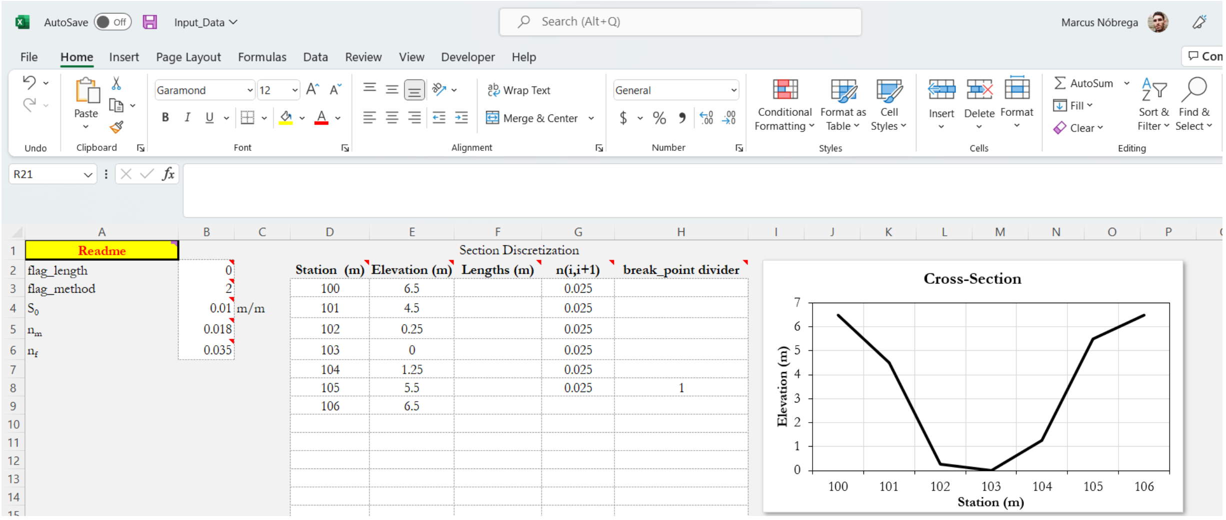

Fig. 2: Excel Spreadsheet input data file. Column B allows selecting the data entry method and the hydraulic assumption of the DCM or SCM model. Moreover, it allows entering the roughness coefficient for inbank and overbank areas. Columns D to H are relative to the cross-section. An automatic plot of 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)

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

%% Read Input Data
clear all
input_table = xlsread('Input_Data.xlsx');
input_data = input_table(1:5,1);
input_data_coordinates = input_table(2:end,3:end);
flag_length = input_data(1,1); % If == 1, use lengths as main input data, otherwise use absolute ...
    values of x (m)
flag_method = input_data(2,1); % If == 1, SCM, else DCM
s0 = input_data(3,1); % Slope in m/m
nm = input_data(4,1); % Main channel roughness
nf = input_data(5,1); % Overbanks channel roughness

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

```

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

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

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

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

% Let's assume a maximum 1 cm difference in the depths
% Noise
dh = 1; % cm
noise = ∆_h/1000; % Noise in m
factor = 1; %precision = 1/factor * noise

[au,ia] = unique(elevations,'stable');
Same = ones(size(elevations));
Same(ia) = 0; % repetitive values
noise_i = rand(1,1)*noise;
small_number = noise/100;
% New Elevation and X_values
ii = 0;
for i = 1:(length(elevations) - 1)
    el1 = elevations(i); el2 = elevations(i+1);
    x1 = x_absolute(i); x2 = x_absolute(i+1);
    if el1 == el2
        elevations(i+1) = elevations(i+1) + noise;
    end
    if x1 == x2
        x_absolute(i+1) = x_absolute(i+1) + noise;
    end
end

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

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

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

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

```
        end
    end
end

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

%% Define Main Channel and Overbanks
pos_break = find(break_point_divider == 1); % Position where the divider occurs
% Main Channel Height
ym = y(pos_break); % Main channel height (m)
if pos_break > pos_inv % Left intersection
    % Left intersection
    posm_left = find(y_left_unsorted ≥ ym,1,'last');
    ym_left_up = y_left_unsorted(posm_left);
    xm_left_up = x_left_unsorted(posm_left);
    ym_left_down = y_left_unsorted(min(posm_left+1,length(y_left_unsorted)));
    xm_left_down = x_left_unsorted(min(posm_left+1,length(y_left_unsorted)));
    % Angles
    if (ym_left_up - ym_left_down ≤ length(y_left_unsorted)*noise)
        alfa_m_l = big_n;
    else
        alfa_m_l = (ym_left_up - ym_left_down )/(xm_left_down - xm_left_up); % Slope
    end
    xm_left = xm_left_down - (ym - ym_left_down )/alfa_m_l;
    ym_left = ym;
    % Polygons (left - inv - right)
    x_pol = [xm_left; x_left_unsorted((posm_left + 1:end),1); x_invert; ...
        x_right_unsorted(1:(pos_break-pos_inv),1)];
    y_pol = [ym_left; y_left_unsorted((posm_left + 1:end),1); y_invert; ...
        y_right_unsorted(1:(pos_break-pos_inv),1)];
    % Top-Width
    bm = abs(x_pol(1) - x_pol(end));
    % Area
    am = polyarea(x_pol,y_pol);
    % Perimeter
    polyin = polyshape(x_pol,y_pol);
    pm = perimeter(polyin) - bm; % Taking away the top width
else
    % Right Intersection
    posm_right = find(y_right_unsorted ≥ ym,1,'first');
    ym_right_up = y_right_unsorted(posm_right);
    xm_right_up = x_right_unsorted(posm_right);
    ym_right_down = y_right_unsorted(max(posm_right-1,1));
    xm_right_down = x_right_unsorted(max(posm_right-1,1));
    % Angles
    if (ym_right_up - ym_right_down < noise*length(y_right_unsorted)) % No depth
        alfa_m_r = big_n;
    else
        alfa_m_r = (ym_right_up - ym_right_down )/(xm_right_up - xm_right_down); % Slope
    end
    xm_right = xm_right_down + (ym - ym_right_down )/alfa_m_r;
    ym_right = ym;
    % Polygons (left - inv - right)
    x_pol = [x_left_unsorted(pos_break:end,1); x_invert; x_right_unsorted(1:(posm_right - 1),1); ...
        xm_right];
    y_pol = [y_left_unsorted(pos_break:end,1); y_invert; y_right_unsorted(1:(posm_right - 1),1); ...
        ym_right];
    % Top-Width
```

```
    bm = abs(x_pol(1) - x_pol(end));
    % Area
    am = polyarea(x_pol,y_pol);
    % Perimeter
    polyin = polyshape(x_pol,y_pol);
    pm = perimeter(polyin) - bm; % Taking away the top width
end
if flag_method ≠ 1
    % Number of floodplains
    if pos_break == 1 || pos_break == length(y)
        n_fp = 1;
    else
        n_fp = 2;
    end
end
while i < big_n
    %% Case where i == 1
    i = i + 1;
    n_P_left = 0;
    n_P_right = 0;
    n_P_left_extra = 0;
    n_P_right_extra = 0;
    B_extra = 0;
    P_extra = 0;
    P_extra_left = 0;
    P_extra_right = 0;
    if i == 1 % We are talking about the first point

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

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

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

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

        %%%% Alfa Right %%%%
        % Case 01 - Vertical Point
```

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

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

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

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

```
                        pf = max(P(j+1,1) - pm,0); % Floodplain perimeter (m)
                        pm_star = max(pm + n_fp*yf,0);
                        am_star = max(am + bm*yf,0);
                        n_med(j+1,1) = (Phi(j+1,1))/(1/nf*af*(af/pf)^(2/3) + ...
                            1/nm*am_star*(am_star/pm_star)^(2/3));
                    else
                        yf = 0; % Overbank depth
                        af = 0; % Overbank flow area
                        pf = 0; % Floodplain perimeter (m)
                        pm_star = 0;
                        am_star = 0;
                        n_med(j+1,1) = nm;
                    end
                end
                K_c(j+1,1) = 1/n_med(j+1,1)*Phi(j+1,1);

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

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

```
        pos_right_previous = pos_right;
    else
        %% Case where i ≠ 1

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

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

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

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


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

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

        %%%% Alfa Left %%%%
        % Case 01 - Vertical Point
        if (x_prev_left - x_left_point ≤ tot_noise) && (y_left_point - y_prev_left > tot_noise)
            alfa_l = big_n;
            alfa_l_tang = big_n;
        end
        % Case 02 - Horizontal Point
        if (x_prev_left - x_left_point > tot_noise) && (y_left_point - y_prev_left ≤ tot_noise)
            alfa_l = big_n;
            alfa_l_tang = big_n;
        end
        % Case 03 - Horizontal and Vertical Point
        if (x_prev_left - x_left_point ≤ tot_noise) && (y_left_point - y_prev_left ≤ tot_noise)
```

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

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


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

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


```

```
        % Checking Discontinuities
        if (pos_right - pos_right_previous) > 1 % More than one movement
            % intersect
            if alfa_r_tang == 0
                x_intersect = x_right_unsorted(pos_right - 1,1);
            else
                x_intersect = x_right_unsorted(pos_right - 1,1) + (y_moving - ...
                    y_right_unsorted(pos_right - 1,1))/alfa_r;
            end
            x_pol = []; y_pol = [];
            for nn = 1:(pos_right - pos_right_previous)
                x_pol = [x_pol; x_right_unsorted(pos_right_previous + nn - 1)];
                y_pol = [y_pol; y_right_unsorted(pos_right_previous + nn - 1)];
            end
            % Adding intersection
            x_pol = [x_pol;x_intersect];
            y_pol = [y_pol;y_moving];
            % Delta B
            Delta_B_right = abs(x_pol(1) - x_pol(end));
            % Delta A
            Delta_Area_right = polyarea(x_pol,y_pol);
            % Delta P
            polyin = polyshape(x_pol,y_pol);
            Delta_P_right = perimeter(polyin) - Delta_B_right; % Taking away top width
            % Manning * Perimeter
            n_P_right = Delta_P_right*n_right_segment^(3/2);
        end
        y_moving_end = min(y_right_point,y_left_point);
        %         if (y_moving_end - y_moving)/(precision/100) < 1
        %             error('Please, increase precision. Instability!')
        %         end
        precision_section = min(y_moving_end - y_moving,precision); % meters
        if y_moving_end - y_moving < precision
            ttt = 1;
        end
        n_points = floor((y_moving_end - y_moving)/(precision_section)); % number of interpolated ...
            points
        % For loop to calculate functions
        if n_points == 1 % only one point means no slope
            if y_moving_end == y_right_point && y_moving_end == y_left_point && alfa_l == big_n && ...
                alfa_r == big_n
                B_extra = x_right_point - x_prev_right + x_prev_left - x_left_point;
                P_extra_left = sqrt((x_prev_left - x_left_point)^2 + (y_prev_left - y_left_point)^2);
                P_extra_right = sqrt((x_right_point - x_prev_right)^2 + (y_right_point - ...
                    y_prev_right)^2);
            elseif y_moving_end == y_right_point && alfa_r == big_n
                if pos_right == 1
                    P_extra_right = sqrt((x_right_point - x_invert)^2 + (y_right_point - y_invert)^2);
                    B_extra = x_right_point - x_invert;
                else
                    P_extra_right = sqrt((x_right_point - x_prev_right)^2 + (y_right_point -  ...
                        y_prev_right)^2);
                    B_extra = x_right_point - x_prev_right;
                end
            else % y_moving == y_left
                if pos_left + 1 > length(x_left_unsorted) && alfa_l == big_n
                    P_extra_left = sqrt((x_invert - x_left_point)^2 + (y_invert - y_left_point)^2);
                    B_extra = x_invert - x_left_point;
                elseif alfa_l == big_n
                    P_extra_left = sqrt((x_prev_left - x_left_point)^2 + (y_prev_left - ...
                        y_left_point)^2);
                    B_extra = x_prev_left - x_left_point;
                end
                % Right
                if pos_right == 1 && alfa_r == big_n
                    P_extra_right = sqrt((x_invert - x_right_point)^2 + (y_invert - y_right_point)^2);
                    B_extra = x_right_point - x_invert + B_extra;
                elseif alfa_r == big_n
                    P_extra_left = sqrt((x_prev_right - x_right_point)^2 + (y_right_point -  ...
                        y_prev_right^2));
                    B_extra = x_right_point - x_prev_right + B_extra;
                end

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

```
        else
            B_extra = 0;
            n_P_right_extra = 0;
            n_P_left_extra = 0;
            P_extra = 0;
            P_extra_left = 0;
            P_extra_right = 0;
        end

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

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

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

            if j == (n_points) % final point
                % Final point - make sure you have the exact surveyed point at the end
                h_ = y_moving_end - y_table(k-1,1);
                y_table(k,1) = y_table(k-1,1) + h_;
                % Roughness
                if y_table(k,1) ≤ ym % Inside of the channel
                    if flag_method == 1
                        n_left_segment = n_left_unsorted(pos_left,1);
                        n_right_segment = n_right_unsorted(pos_right,1);
                    else
                        if (abs(y_left_unsorted(pos_left) - ym) ≤ noise*length(y_left_unsorted))
                            n_left_segment = nf; % Attention here
                        else
```

```
                            n_left_segment = nm; % Attention here
                        end
                        if (abs(y_right_unsorted(pos_right) - ym) ≤ noise*length(y_right_unsorted))
                            n_right_segment = nf;  % Attention here
                        else
                            n_right_segment = nm;  % Attention here
                        end
                    end
                else % Overbanks
                    if flag_method == 1
                        n_left_segment = n_left_unsorted(pos_left,1);
                        n_right_segment = n_right_unsorted(pos_right,1);
                    elseif y_left_unsorted(pos_left) - ym < noise*length(y_left_unsorted)% Check Noises
                        n_left_segment = nm; % Attention here
                        n_right_segment = nm;  % Attention here
                    else
                        n_left_segment = nf; % Attention here
                        n_right_segment = nf;  % Attention here
                    end
                end
                B(k,1) = h_/alfa_l_tang + h_/alfa_r_tang + B(k-1,1) + B_extra;
                A(k,1) = (B(k,1) + B(k-1,1))*h_/2 + A(k-1,1); % Trapezoid
                P(k,1) = h_/sin(atan(alfa_l_tang)) + h_/sin(atan(alfa_r_tang)) + P(k-1,1) + P_extra;
                Rh(k,1) = A(k,1)/P(k,1);
                Phi(k,1) = A(k,1)*Rh(k,1)^(2/3);
                int_n_p = n_left_segment^(3/2)*h/sin(atan(alfa_l_tang)) + ...
                    n_right_segment^(3/2)*h/sin(atan(alfa_r_tang)) + int_n_p;
                % Representative Roughness Coefficient
                if flag_method == 1
                    n_med(k,1) = (int_n_p/P(k,1))^(2/3);
                else
                    if y_table(k,1) ≥ ym
                        yf = max(y_table(k,1) - ym,0); % Overbank depth
                        af = max(A(k,1) - (am + bm*yf),0); % Overbank flow area
                        pf = max(P(k,1) - pm,0); % Floodplain perimeter (m)
                        pm_star = max(pm + n_fp*yf,0);
                        am_star = max(am + bm*yf,0);
                        n_med(k,1) = (Phi(k,1))/(1/nf*af*(af/pf)^(2/3) + ...
                            1/nm*am_star*(am_star/pm_star)^(2/3));
                    else
                        yf = 0; % Overbank depth
                        af = 0; % Overbank flow area
                        pf = 0; % Floodplain perimeter (m)
                        pm_star = 0;
                        am_star = 0;
                        n_med(k,1) = nm;
                    end
                end
                K_c(k,1) = 1/n_med(k,1)*Phi(k,1);
            end
        end
        % Previous Positions
        pos_left_previous = pos_left;
        pos_right_previous = pos_right;
    end
    % Checking i
    if round(y_table(end),3) == round(max_y,3) % Stop de algorithm
        i = big_n;
    end
end

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

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

% Beta - Boussinesq factor
```

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

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


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

```
grid on
xlabel('Flow discharge ($m^3/s$)','Interpreter','latex');
ylabel('y ($m$)','Interpreter','latex');
exportgraphics(gcf,'Hydraulic_Properties.pdf','ContentType','vector')
toc

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

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

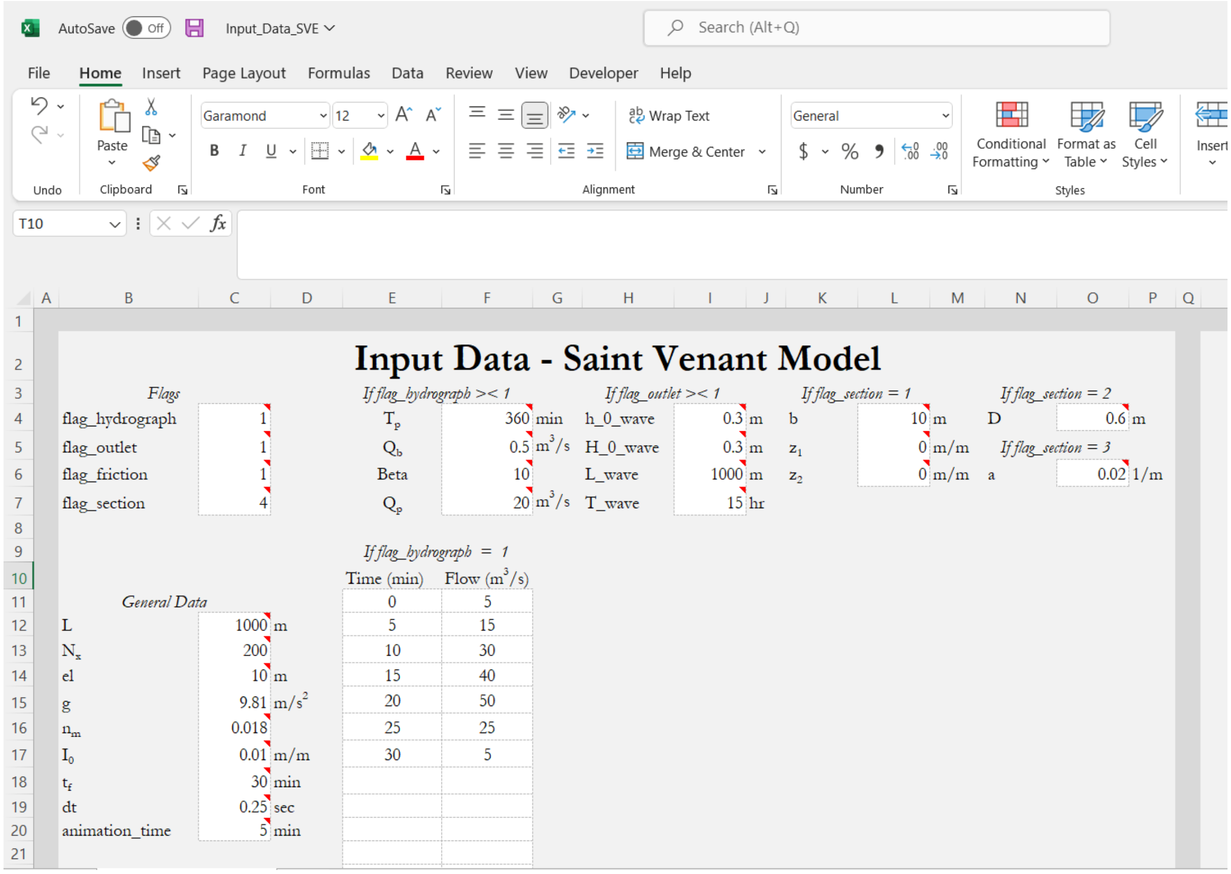

Fig. 3: Input data for the SVE model. All variables have a full explanation in the excel file.

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:

```
%% Read Input Data %%
data = xlsread('Input_Data_SVE.xlsx');

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

% Flags
flag_hydrograph = data(1,1);
flag_outlet = data(2,1);
flag_friction = data(3,1);
flag_section = data(4,1);

if flag_hydrograph ≠ 1
    % Hydrograph
    Tp = data(1,4);
    Qb = data(2,4);
    Beta = data(3,4);
    Qp = data(4,4);
else
    % Input Hydrograph
     time_ = data(8:end,3);
     Qe1_ = data(8:end,4);
     Qe1 = zeros(size(Qe1_,1) - sum(isnan(Qe1_)),1);
     time = zeros(size(time_,1) - sum(isnan(time_)),1);
     % Taking away nans
     for i = 1:length(Qe1)
        if isnan(Qe1_(i)) || isnan(time_(i))
```

```
                break
            else
                Qe1(i,1) = Qe1_(i,1);
                time(i,1) = time_(i,1);
            end
         end
         clear Qe1_ time_
end

% Outlet
if flag_outlet ≠1
    h_0_wave = data(1,7);
    H_0_wave = data(2,7);
    L_wave = data(3,7);
    T_wave = data(4,7);
end

% Section
if flag_section == 1
    b = data(1,10);
    Z1 = data(2,10);
    Z2 = data(3,10);
elseif flag_section == 2
    D = data(1,13);
elseif flag_section == 3
    a = data(3,13);
else
    % Read HP estimator data
    [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr, x_cross, ...
        y_cross] = HP_estimator;
    irr_table = [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
end

% General Data
L = data(9,1);
Nx = data(10,1);
el = data(11,1);
g = data(12,1);
nm = data(13,1);
I0 = data(14,1);
tf = data(15,1);
dt = data(16,1);
animation_time = data(17,1);

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

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.

```
%%% Saint Venant Equations Solver %%%
% Developer: Marcus Nobrega Gomes Junior
% 4/10/2022
% Solution of SVE for given cross-section functions of Area, Perimeter, and
% top Width
%%% Still misses the introduction of Beta in F2 (check the paper)

%% Pre-Processing
clear all
clc
warning('off')

```

```
% Reading the Input Data
Read_Input_Data

% Inflow Hydrograph
% flag_hydrograph = 2; % If 1, enter the hydrograph, if 2, model using nash function
% 1st option - Enter Qe1 and Time below
if flag_hydrograph == 1
    % We already read the hydrograph in Read_Input_Data file
else
    % 2nd option - Model the hydrograph using a nash function
    %%% Q(t) = Qb(t) + (Qp(t) - Qb(t))*(t/TP*EXP(1 - t/TP))^Beta
    Inflow_Hydrograph_fun = @(t)(Qb + (Qp - Qb).*(t/(Tp*60).*exp(1 - (t)/(Tp*60))).^Beta);
    time = [0 tf/60]'; % begin and end in minutes
end

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

% Time Calculations
time = time*60; % time in seconds
[a1,¬] = size(time);
tt_h = time(a1,1); % end of hydrograph in seconds
tt = min(tf,tt_h)*60; % End of simulation in seconds
Nt = tt/dt; % Number of time-steps
Nat = animation_time*60/dt; % Number of time-steps within an animation time
tint = linspace(0,tt/60,Nt); % Generate Nt points within 0 and tt(min)
if flag_hydrograph == 1
    Qe1int = max(interp1(time/60,Qe1(:,1),tint,'pchip'),0); % Interpolated flow % N o utilizar spline;
    % pchip, que corresponde ao Hermite c bico ,    uma o p   o prefer vel.
    % Assuming no negative flows
    Qe1int = Qe1int';
else
    Qe1int = Inflow_Hydrograph_fun(tint)';
end


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

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

% Manning
nm = repmat(nm,Nx,1); % Bottom slope in m/m for all reaches
% Darcy Weisbach
f = 0; % not implemented

% Pre-allocating arrays
% Matrices
x = (0:dx:L)'; % x discretization in meters
y = zeros(Nx,Nt);
q1 = zeros(Nx,Nt);
q2 = zeros(Nx,Nt);
f1 = zeros(Nx,Nt);
f2 = zeros(Nx,Nt);
J2 = zeros(Nx,Nt);
q1_back = q1(1:(Nx-2),Nt);
q1_foward = zeros(Nx-2,Nt);
q2_back = zeros(Nx-2,Nt);
q2_foward = zeros(Nx-2,Nt);
f1_back = zeros(Nx-2,Nt);
f1_foward = zeros(Nx-2,Nt);
f2_back = zeros(Nx-2,Nt);
f2_foward = zeros(Nx-2,Nt);
J2_back = zeros(Nx-2,Nt);
J2_foward = zeros(Nx-2,Nt);
ybar = zeros(Nx,Nt);
Fr = zeros(Nx,Nt);
Cn = zeros(Nx,Nt);

```

```
%% Channel Data (Cross Section)
% Slope
I0 = repmat(I0,Nx,1); % Bottom slope in m/m for all reaches

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

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

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

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

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

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

```matlab

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

%% Initial Boundary Conditions
Q0 = Qe1int(1,1); % Flow at inlet section at time 0
if flag_outlet == 1
    if flag_friction == 1
        if flag_section ≠ 4
            if Q0 == 0
                Q0 = sm; % Numerical Constraint
            end
            y0 = uniformeM(nm,Q0,b,Z1,Z2,a,D,I0,P,A,y0_guess)   ; % normal depth using manning equation
                % More Initial Boundary Conditions for Area, Velocity, Perimeter and Rh
            A0 = A_function(D,Z1,Z2,a,b,y0); % Cross section area in m
            u0 = (Q0./A0)'; % Initial velocity in m/s
            P0 = P_function(D,Z1,Z2,a,b,y0); % Hydraulic perimeter in m
            Rh0 = A0./P0; % Hydraulic radius at time 0
            % Boundary Conditions
            y(:,1) = y0; % all sub-reaches with y0 at the beginning
            q1(:,1) = A0; % all sub-reaches with same area A0 at the beginning
            q2(:,1) = Q0; % Assuming permanent conditions at the beginning
            f1(:,1) = q2(:,1);
            % f2 depends on ybar
        else % Irregular Cross-Section
            % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
            % [   1,     2,      3,       4,     5,             6,           7,      8,       9,    10]
            col1 = 10; % Col with Q
            y0 = Vlookup_l(irr_table,col1,Q0*fv,1);
            A0 = Vlookup_l(irr_table,col1,Q0*fv,2);
            P0 = Vlookup_l(irr_table,col1,Q0*fv,3);
            Rh0 = Vlookup_l(irr_table,col1,Q0*fv,4);
            % Boundary Conditions
            y(:,1) = y0; % all sub-reaches with y0 at the beginning
            q1(:,1) = A0; % all sub-reaches with same area A0 at the beginning
            q2(:,1) = Q0; % Assuming permanent conditions at the beginning
            f1(:,1) = q2(:,1);
        end
    else
        % % normal depth using Darcy-Weisbach equation not implemented yet
    end
else
    steady_depth = uniformeM(nm,Q0,b,Z1,Z2,a,D,I0,P,A);
    y0(1:(Nx-1),1) = steady_depth(1:(Nx-1),1);
    % Stage Hydrograph Boundary Condition
    time_wave = 0; % time in seconds
    y0(Nx,1) = h_wave_function(time_wave);
    % More Initial Boundary Conditions for Area, Velocity, Perimeter and Rh
    A0 = A_function(D,Z1,Z2,a,b,y0); % Cross section area in m
    u0 = (Q0./A0)'; % Initial velocity in m/s
    P0 = P_function(D,Z1,Z2,a,b,y0); % Hydraulic perimeter in m
    Rh0 = A0./P0; % Hydraulic radius at time 0
    y(:,1) = y0(:,1); % all sub-reaches with y0 at the beginning
    q1(:,1) = A0(:,1); % all sub-reaches with same area A0 at the beginning
    q2(:,1) = Q0(:,1); % Assuming permanent conditions at the beginning
    f1(:,1) = q2(:,1);
    % f2 depends on ybar, we will calculate below
end

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

% ybar = y - int(A(y)) / A(y) from y = 0 to y = y0
if flag_section ≠ 4
    ybar = ybar_function(D,Z1,Z2,a,b,y0);
else
    % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
```

```
    % [   1,     2,      3,       4,     5,              6,           7,      8,       9,
    % ybar = y - ybar*
    %      ybar(:,1) =  Vlookup_leq(irr_table,col1,Q0*fv,1) - Vlookup_leq(irr_table,col1,Q0*fv,5);
    ybar(:,1) = Vlookup_l(irr_table,col1,Q0*fv,5);
end
f2(:,1) = q2(:,1).*abs(q2(:,1))./q1(:,1) + g*q1(:,1).*ybar(:,1);

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

% Froude Number
if flag_section ≠ 4
    Fr(:,1)=abs(q2(:,1)./q1(:,1))./((g*A_function(D,Z1,Z2,a,b,y0)./B_function(D,Z1,Z2,a,b,y0)).^0.5);% N mero de Froude
else
    % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
    % [   1,     2,      3,       4,     5,              6,           7,      8,       9,
    A_f_irr = Vlookup_l(irr_table,col1,Q0*fv,2)*ones(length(q1(:,1)),1);
    B_f_irr = Vlookup_l(irr_table,col1,Q0*fv,9)*ones(length(q1(:,1)),1);
    Fr(:,1)=abs(q2(:,1)./q1(:,1))./((g*A_f_irr./B_f_irr).^0.5);% N mero de Froude
end
% Courant Number
% Cn = c / (dx / dt), where c = v + sqrt(g.Hm), where Hm = A / B
if flag_section ≠ 4
    Hm = A_function(D,Z1,Z2,a,b,y0)./B_function(D,Z1,Z2,a,b,y0);
    Cn(:,1)=(abs(q2(:,1)./q1(:,1))+(g*Hm).^0.5)/(dx/dt);% Courant Number
else
    Hm = A_f_irr./B_f_irr;
    Cn(:,1) = (abs(q2(:,1)./q1(:,1))+(g*Hm).^0.5)/(dx/dt);
end

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

syms c_
if flag_section ≠ 4
    fun_solve = (A - c_); % with c = area, we solve for y.
    options = optimoptions ('fsolve', 'Display', ...
        'none','FunctionTolerance',1e-2,'MaxFunctionEvaluations',Nx*10);
end
if flag_section == 1
    % We have an analytical solution for this case
    z = solve(fun_solve,y_); % solving for y_ = y and c = A(y)
    h_function = matlabFunction(z); % h(A) = z;
else
    % Non-linear set of equations for circular pipe, we need to use fsolve
end
if flag_section ≠ 4
    fun_solve = matlabFunction(fun_solve); % Transforming into an equation
end
%% Main Loop %%
n = 1; % initializing counter
tic % starts measuring time

for t = dt:dt:(tt - dt) % Main loop
    n = n + 1;
    percentage_complete = [t/(tt - dt)*100, max(Cn(:,n-1)), max(q2(:,n-1))]
    % Stop Program if Complex Number Occurs
    if imag(max(Cn(:,n-1))) > 0 || imag(max(q2(:,n-1)))
        error('Complex number possibly due to changing the regime from free flow to pressurized flow.')
    end
    %%%%% -  Boundary Conditions -  %%%%%
    %% Channel's begin (INLET)
    q1(1,n) = q1(2,n-1); % Area at section 1 is equals area of section 2 from previous time-step
    q2(1,n) = Qe1int(n,1); % Flow at section 1 is the inflow hydrograph

    % Interpolating All Values from I_rr_table using q1 as basis
    % Explanation: area is given in m2. P, Rh, and other variables are
    % in m. So we have a quadratically similar triangle relationship
    if flag_section == 4
        for mm = 1:(length(irr_table(1,:))-1)
            interp_base = q1(1,n); % Value that will be used for interpolation (area)
```

```
            area_smaller = Vlookup_l(irr_table,2,interp_base,2); % Smaller values
            area_larger = Vlookup_g(irr_table,2,interp_base,2); % Larger values
            col1 = 2; % Interpolating from area values
            var_inlet(mm,1,1) = Vlookup_l(irr_table,col1,interp_base,mm); % Smaller values
            var_inlet(mm,1,2) = Vlookup_g(irr_table,col1,interp_base,mm); % Larger values
            alfa_var_inlet(mm,1) = sqrt((interp_base - area_smaller)/(area_larger - area_smaller));
        end
    end

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

```
                nm = irr_table(2,6)*ones(length(q1(:,1)),1);
            end
            J2(1,n) = g*c.*(I0(1) - q2(1,n).*abs(q2(1,n)).*nm(1).^2./(c.^2*Rh_inlet.^(4/3))); % Manning
        else
            J2(1,n) = (I0(1) - f*q2(1,n).*abs(q2(1,n))./((q1(1,n).^2)*8*g.*Rh_inlet));
        end
        % Froude
        % Var* = Var(-) + alfa*(Var(+) - Var(-))
        A_f_irr = q1(1,n);
        col_var = 9;
        B_f_irr = var_inlet(col_var,1,1) + alfa_var_inlet(col_var,1)*(var_inlet(col_var,1,2) - ...
            var_inlet(col_var,1,1));
        %         B_f_irr = Vlookup_leq(irr_table,col1,c,9);
        Fr(1,n) = abs(q2(1,n)./q1(1,n))./((g*A_f_irr./B_f_irr)^0.5);% N mero de Froude
        % Courant
        Hm = A_f_irr./B_f_irr;
        Cn(1,n) = (abs(q2(1,n)./q1(1,n))+(g*Hm).^0.5)/(dx/dt);% Courant Number
    end

    %% Right side of the channel (outlet)
    if flag_outlet == 1 % Normal Depth
        q1(Nx,n) = q1(Nx-1,n-1); % Boundary Condition
        % Interpolating All Values from I_rr_table using q1 as basis
        % Explanation: area is given in m2. P, Rh, and other variables are
        % in m. So we have a quadratically similar triangle relationship
        if flag_section == 4
            for mm = 1:(length(irr_table(1,:))-1)
                interp_base = q1(Nx,n); % Value that will be used for interpolation (area)
                area_smaller = Vlookup_l(irr_table,2,interp_base,2); % Smaller values
                area_larger = Vlookup_g(irr_table,2,interp_base,2); % Larger values
                col1 = 2; % Interpolating from area values
                var_outlet(mm,1,1) = Vlookup_l(irr_table,col1,interp_base,mm); % Smaller values
                var_outlet(mm,1,2) = Vlookup_g(irr_table,col1,interp_base,mm); % Larger values
                alfa_var_outlet(mm,1) = sqrt((interp_base - area_smaller)/(area_larger - ...
                    area_smaller));
            end
        end
        if flag_section == 1
            q1(Nx,n) = q1(Nx-1,n-1); % Area of outlet is the area of previous cell at previous ...
                time-step
            if Z1 > 0 || Z2 > 0
                y(Nx,n) = max(h_function(D,Z1,Z2,a,b,q1(Nx,n)')); % water depth in terms of area q1
            else
                y(Nx,n) = q1(Nx,n)/b; % water depth in terms of area q1 for rectangular channels
            end
        elseif flag_section ≥ 2 % circular or paraboloid or irregular
            % If we do not have an stage-hydrograph boundary condition
            y0_guess = y(Nx,n-1);
            if flag_section ≠ 4
                fun = @(y_)fun_solve(D,Z1,Z2,a,b,c,y_); % non-linear solver
                y(Nx,n) = fsolve(fun,y0_guess,options); % non-linear solver
            else
                % [y_table, A, P, Rh, y_bar, n_med, Beta, v, B, Q]
                % [   1,     2, 3, 4,     5,     6,      7,  8,  9, 10]
                col_var = 1;
                % Var* = Var(-) + alfa*(Var(+) - Var(-))
                y(Nx,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));
            end
        end
    else
        % Stage Hydrograph Boundary Condition
        time_wave = n*dt; % time in seconds
        y(Nx,n) = h_wave_function(time_wave);
        q1(Nx,n) = A_function(D,Z1,Z2,a,b,y(Nx,n));
        %         q1(Nx,n) = q1(Nx-1,n-1)
    end
    % Hydraulic Radius
    if flag_section ≠ 4
        Rh_outlet = Rh_function(D,Z1,Z2,a,b,y(Nx,n));
    else
        % [y_table, A, P, Rh, y_bar, n_med, Beta, v, B, Q]
        % [   1,     2, 3, 4,     5,     6,      7,  8,  9, 10]
        %         col1 = 2; % Col with A
        %         Rh_outlet = Vlookup_l(irr_table,col1,c,4);
        col_var = 4;
        % Var* = Var(-) + alfa*(Var(+) - Var(-))
```

```
        Rh_outlet = var_outlet(col_var,1,1) + alfa_var_outlet(col_var,1)*(var_outlet(col_var,1,2) - ...
            var_outlet(col_var,1,1));
        % Var* = Var(-) + alfa*(Var(+) - Var(-))
        col_var = 6;
        nm(end,1) = var_inlet(col_var,1,1) + alfa_var_inlet(col_var,1)*(var_inlet(col_var,1,2) - ...
            var_inlet(col_var,1,1));
    end
    if flag_friction == 1
        if flag_outlet == 1
            u = (1./nm(Nx)).*Rh_outlet^(2/3)*I0(Nx)^0.5; % Normal depth at the outlet
            flow_dir = 1;
        else
            depth_dif = y(Nx-1,n-1) + inv_el(Nx-1) - y(Nx,n) - inv_el(Nx); % Diference in wse
            out_slope = abs(depth_dif)/dx; % Flow slope at the outlet
            u = (1./nm(Nx)).*Rh_outlet^(2/3)*out_slope^0.5; % Normal depth at the outlet
            if depth_dif > 0
                flow_dir = 1; % Flowing towards the outlet
            else
                flow_dir = -1; % Flowing to inside of the channel
            end
        end
    else
        u = sqrt(8*g*Rh_outlet*I0(Nx)/f); % outlet velocity
    end
    % Outlet Flow
    q2(Nx,n) = q1(Nx,n)*u*flow_dir; % Area x Velocity
    if isnan(q2(Nx,n))
        ttt = 1;
    end
    % ybar
    if flag_section ≠ 4
        ybar(Nx,n) = ybar_function(D,Z1,Z2,a,b,y(Nx,n));
    else
        % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
        % [   1,     2,      3,       4,     5,             6,          7,   8,      9,   10]
        % ybar = y - ybar*
        col1 = 2; % A
        %          ybar(Nx,n) =  Vlookup_leq(irr_table,col1,c,1) - Vlookup_leq(irr_table,col1,c,5);
        ybar(Nx,n) = Vlookup_l(irr_table,col1,c,5);
        col_var = 5;
        % Var* = Var(-) + alfa*(Var(+) - Var(-))
        ybar(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));
    end
    % f1 and f2
    f1(Nx,n) = q2(Nx,n); % f1 - Area
    f2(Nx,n) = q2(Nx,n).*abs(q2(Nx,n))./q1(Nx,n) + g*q1(Nx,n).*ybar(Nx,n); % f2 = (Qv + gAy_bar)
    % J2
    % Friction
    if flag_friction == 1
        J2(Nx,n) = g*q1(Nx,n).*(I0(Nx) - ...
            q2(Nx,n).*abs(q2(Nx,n)).*nm(Nx)^2./(q1(Nx,n).^2*Rh_outlet.^(4/3))); % Manning --> ...
            gA*(I0 - If), If = n^2*Q*abs*Q)/(Rh^(4/3)*A^2)
    else
        J2(Nx,n) = g*q1(Nx,n).*(I0(Nx) - f*q2(:,n).*abs(q2(Nx,n))./((q1(Nx,n).^2)*8*g*Rh_outlet));
    end
    % Froude
    if flag_section ≠ 4
        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.
            N mero de Froude
        % Courant
        Hm = A_function(D,Z1,Z2,a,b,y(Nx,n))./B_function(D,Z1,Z2,a,b,y(Nx,n));
        Cn(Nx,n)=(abs(q2(Nx,n)./q1(Nx,n))+(g*Hm).^0.5)/(dx/dt);% Courant Number
    else
        % Froude
        % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
        % [   1,     2,      3,       4,     5,             6,          7,   8,      9,   10]
        col1 = 2; % Col with A
        A_f_irr = c;
        col_var = 9;
        % Var* = Var(-) + alfa*(Var(+) - Var(-))
        B_f_irr = var_outlet(col_var,1,1) + alfa_var_outlet(col_var,1)*(var_outlet(col_var,1,2) - ...
            var_outlet(col_var,1,1));
        %          B_f_irr = Vlookup_l(irr_table,col1,c,9);
        Fr(Nx,n) = abs(q2(Nx,n)./q1(Nx,n))./((g*A_f_irr./B_f_irr)^0.5);% N mero de Froude
        % Courant
        Hm = A_f_irr./B_f_irr;
```

```
        Cn(Nx,n)=(abs(q2(1,n)./q1(1,n))+(g*Hm).^0.5)/(dx/dt);% Courant Number
    end

    %% Main Loop for Non Boundary Cells from 2 to (Nx - 1)
    % vectorized calculations
    q1_back = q1(1:(Nx-2),(n-1));
    q1_foward = q1(3:(Nx),(n-1));
    q2_back = q2(1:(Nx-2),(n-1));
    q2_foward = q2(3:(Nx),(n-1));
    f1_back = f1(1:(Nx-2),(n-1));
    f1_foward = f1(3:(Nx),(n-1));
    f2_back = f2(1:(Nx-2),(n-1));
    f2_foward = f2(3:(Nx),(n-1));
    J2_back = J2(1:(Nx-2),(n-1));
    J2_foward = J2(3:(Nx),(n-1));

    x_i = 2:(Nx-1); % vector for interior sections varying from 2 to (Nx - 1)
    % Lax-Friedrichs Method
    % Given an hyperbolic partial derivative system of equations described
    % by:
    % pq/pt + pF/px - S = 0, where p is the partial derivative, one can
    % solve this equation by performing a foward discretization for q and a
    % central discretization for F. Moreover, S = (Sback + Sfoward)/2
    % Expliciting the system of equations for q1, it follows that

    q1(x_i,n) = 0.5.*(q1_foward + q1_back) - 0.5*dt/dx*(f1_foward - f1_back); %% attention here in ...
        f1foward
    q2(x_i,n) = 0.5*(q2_foward + q2_back) - 0.5*dt/dx*(f2_foward - f2_back) + 0.5*dt*(J2_back + ...
        J2_foward);

    % Interpolating All Values from I_rr_table using q1 as basis
    if flag_section == 4
        for mm = 1:(length(irr_table(1,:))-1)
            for hh = 1:length(x_i)
                interp_base = q1(hh+1,n); % Value that will be used for interpolation (area)
                area_smaller = Vlookup_l(irr_table,2,interp_base,2); % Smaller values
                area_larger = Vlookup_g(irr_table,2,interp_base,2); % Larger values
                col1 = 2; % Interpolating from area values
                var_middle(mm,hh,1) = Vlookup_l(irr_table,col1,interp_base,mm); % Smaller values
                var_middle(mm,hh,2) = Vlookup_g(irr_table,col1,interp_base,mm); % Larger values
                alfa_var_middle(mm,hh,1) = sqrt((interp_base - area_smaller)/(area_larger - ...
                    area_smaller));
            end
        end
    end

    if flag_section == 1
        if Z1>0 || Z2>0
            y(x_i,n) = max(h_function(D,Z1,Z2,a,b,q1(x_i,n)')); % water depth in terms of area q1
        else
            y(x_i,n)=q1(x_i,n)/b;
        end
    elseif flag_section > 1
        y0_guess = y(x_i,n-1);
        c = q1(x_i,n)*fv; % It has to be a line vector (area)
        if flag_section ≠ 4
            fun = @(y_)fun_solve(D,Z1,Z2,a,b,c,y_);
            y(x_i,n) = fsolve(fun,y0_guess,options); % non-linear solver
        else
            % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
            % [   1,     2,      3,       4,     5,             6,           7,   8,       9,   10]
            col1 = 2; % Col with A
            for i = 1:length(x_i)
                cc = c(i); % be careful here
                col_var = 1;
                % Var* = Var(-) + alfa*(Var(+) - Var(-))
                y(i+1,n) = var_middle(col_var,i,1) + ...
                    alfa_var_middle(col_var,i)*(var_middle(col_var,i,2) - var_middle(col_var,i,1));
            end
        end
    end
    % Hydraulic Radius
    if flag_section ≠ 4
        Rh_middle = Rh_function(D,Z1,Z2,a,b,y(x_i,n));
        % ybar
        ybar(x_i,n) = ybar_function(D,Z1,Z2,a,b,y(x_i,n));
        % f1 and f2
```

```
        f1(x_i,n) = q2(x_i,n);
        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);
        % Froude
        Hm = A_function(D,Z1,Z2,a,b,y(x_i,n))./B_function(D,Z1,Z2,a,b,y(x_i,n));
        Fr(x_i,n)=abs(q2(x_i,n)./q1(x_i,n))./((g*Hm).^0.5);% N mero de Froude
        % Courant
        Cn(x_i,n)=(abs(q2(x_i,n)./q1(x_i,n))+(g*Hm).^0.5)/(dx/dt);% Courant Number
        % Friction
        if flag_friction == 1
            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))));
        else
            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));
        end
        % Stability Check
        if max(Cn(:,n)) > 1
            error('Please, decrease the time-step')
        end
    else
        for jj = 1:length(x_i)
            cc = c(jj); % Area
            % [y_irr, A_irr, P_irr, Rh_irr, y_bar_irr, n_med_irr, Beta_irr, u_irr, B_irr, Q_irr];
            % [   1,     2,      3,       4,     5,              6,          7,   8,       9,   10]
            col_var = 4;
            % Var* = Var(-) + alfa*(Var(+) - Var(-))
            Rh_middle(jj,1) = var_middle(col_var,jj,1) + ...
                alfa_var_middle(col_var,jj)*(var_middle(col_var,jj,2) - var_middle(col_var,jj,1));
            col_var = 5;
            ybar(jj+1,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));
            col_var = 6;
            nm(jj+1,1) = var_middle(col_var,jj,1) + ...
                alfa_var_middle(col_var,jj)*(var_middle(col_var,jj,2) - var_middle(col_var,jj,1));
            % f1 and f2
            f1(jj+1,n) = q2(jj+1,n);
            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);
            % Froude
            A_f_irr = q1(jj+1,n);
            col_var = 9;
            B_f_irr = var_middle(col_var,jj,1) + ...
                alfa_var_middle(col_var,jj)*(var_middle(col_var,jj,2) - var_middle(col_var,jj,1));
            Hm = A_f_irr./B_f_irr;
            Fr(jj+1,n) = abs(q2(jj+1,n)./q1(jj+1,n))./((g*Hm).^0.5);% N mero de Froude
            % Courant
            Cn(jj+1,n) = (abs(q2(jj+1,n)./q1(jj+1,n))+(g*Hm).^0.5)/(dx/dt);% Courant Number
            % Friction
            if flag_friction == 1
                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))));
            else
                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)));
            end
            % Stability Check
            if Cn(jj+1,n) > 1
                error('Please, decrease the time-step')
            end
        end
    end
end

zzz = [y(ceil(Nx/2),:)', q2(ceil(Nx/2),:)',tint' , q2(1,:)'];
%%% Post Processing Figures %%%
% Call function
warning('on');
post_processing
toc
```

4) SVE Post Processing

```
%% Post Processing Graphs
clf
```

```
close all


% Surfplot
t_save = [0:Nat:tt/dt];
t_save(1,1) = 1;
set(gcf,'units','inches','position',[2,2,4,5])
subplot(3,1,1)
surf(x,tint(t_save)/60,Fr(:,t_save)');
view(0,90);
kk = colorbar ; colormap('jet')
shading interp
xlabel('x (m)','Interpreter','latex')
ylabel('t (h)','Interpreter','latex')
ylabel(kk,'Froude Number','Interpreter','latex')
zlabel ('Froude Number','Interpreter','Latex');
xlim([0 L]);
ylim([0 tt/60/60]);

subplot(3,1,2)
surf(x,tint(t_save)/60/60,y(:,t_save)');
view(0,90);
kk = colorbar ; colormap('jet')
shading interp
xlabel('x (m)','Interpreter','latex')
ylabel('t (h)','Interpreter','latex')
ylabel(kk,'y (m)','Interpreter','latex')
zlabel ('y (m)','Interpreter','Latex');
xlim([0 L]);
ylim([0 tf/60/60]);
subplot(3,1,3)
wse = y + inv_el;
surf(x,tint(t_save)/60/60,wse(:,t_save)');
view(0,90);
kk = colorbar ; colormap('jet')
shading interp
xlabel('x (m)','Interpreter','latex')
ylabel('t (h)','Interpreter','latex')
ylabel(kk,'WSE (m)','Interpreter','latex')
zlabel ('WSE (m)','Interpreter','Latex');
xlim([0 L]);
ylim([0 tf/60/60]);
exportgraphics(gcf,'Surf_Plots.pdf','ContentType','image','Colorspace','rgb','Resolution',600)
clf
close all

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

        % Boundary Circle
        % (x - xc)^2 + (y - yc)^2 = D^2/4
        % where xc = D/2 and yc = D/2
        xc = D/2; yc = D/2;
        y01 = y(1,pos);
        y02 = y(ceil(ceil(Nx/2)),pos);
        y03 = y(Nx,pos);
        y0_c = [y01; y02; y03];
        % For a given known y, we have to find two xs, such that
        % x^2 + (-2xc)x + ( (y0 - yc)^2 - xc^2 - D^2/4 )
        % or ax^2 + bx + c, with
```

```
        % a = 1; b = -2xc; c = (y0 - yc)^2 - xc^2 - D^2/4
        % x = (- b +- sqrt(b^2 - 4ac)) / (2a)
        a = 1;
        b = -2*xc;
        c = xc^2 + (y0_c - yc).^2 - D^2/4;
        Delta = b^2 - 4*a.*c;
        x1 = (-b + sqrt(Delta))/(2*a);
        x2 = (-b - sqrt(Delta))/(2*a);
        % Now we found the intersection of the circle and a line with know
        % depth
        subplot(1,3,1)
        title(['t = ',num2str(round(round(t/60,2),2)),' [min]'])
        ylim([0 D]);
        xlim([0 D]);
        viscircles([D/2 D/2],D/2,'Color','black');
%         plot(c1(1,:),c1(2,:),'Color','black');
        hold on
        x_water = linspace(x2(1),x1(1),100);
        y_water = repmat(y01,1,100);
        plot(x_water,y_water,'blue','linewidth',2);
%         fill([c1(1,:) fliplr(c1(1,:))], [y_water fliplr(c2(1,:))],'blue')
        ylabel('y(m)','Interpreter','latex')
        xlabel('B(m)','Interpreter','latex')
        legend('Entrance','interpreter','latex')
        hold off
        % second section
        subplot(1,3,2)
        title(['t = ',num2str(round(round(t/60,2),2)),' [min]'])
        ylim([0 D]);
        xlim([0 D]);
        viscircles([D/2 D/2],D/2,'Color','black');
        hold on
        x_water = linspace(x2(2),x1(2),100);
        y_water = repmat(y02,1,100);
        plot(x_water,y_water,'blue','linewidth',2);
        ylabel('y(m)','Interpreter','latex')
        xlabel('B(m)','Interpreter','latex')
        legend('x = L/2','interpreter','latex')
        hold off
        legend('L/2','interpreter','latex')
        % third section
        subplot(1,3,3)
        title(['t = ',num2str(round(round(t/60,2),2)),' [min]'])
        ylim([0 D]);
        xlim([0 D]);
        viscircles([D/2 D/2],D/2,'Color','black');
        hold on
        x_water = linspace(x2(3),x1(3),100);
        y_water = repmat(y03,1,100);
        plot(x_water,y_water,'blue','linewidth',2);             hold off
        ylabel('y(m)','Interpreter','latex')
        xlabel('B(m)','Interpreter','latex')
        legend('Exit','interpreter','latex')
        % Save frame
        title(['t = ',num2str(round(round(t/60,2),2)),' [min]'])
        f = getframe(gcf);
        writeVideo(obj,f);
        hold off
        clf
    end
obj.close();
end

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

```
        end
        % Parabolic Function
        % y = a*x^2 => xmax = sqrt((ymax/a))
        ymax = max(max(y));
        xmax = sqrt(ymax/a); % x to left and right directions
        xpar = linspace(-xmax,xmax,100); % 100 points within -xmax and xmax deg
        ypar = a.*xpar.^2;
        % Now we found bottom of the channel
        % We still need to find xleft and xright for a given y
        y01 = y(1,pos);
        y02 = y(ceil(Nx/2),pos);
        y03 = y(Nx,pos);
        y0_c = [y01; y02; y03];
        xright = sqrt(y0_c/a);
        xleft = - xright;
        subplot(1,3,1)
        title(['t = ',num2str(round(round(t/60,2),2)),' [min]'])
        ylim([0 ymax]);
        xlim([0 ymax]);
        plot(xpar,ypar,'Color','black');
        hold on
        x_water = linspace(xleft(1),xright(1),100);
        y_water = linspace(y01,y01,100);
        plot(x_water,y_water,'blue','linewidth',2);
        ylabel('y(m)','Interpreter','latex')
        xlabel('B(m)','Interpreter','latex')
        legend('Entrance','interpreter','latex')
        hold off
        % second section
        subplot(1,3,2)
        title(['t = ',num2str(round(round(t/60,2),2)),' [min]'])
        ylim([0 ymax]);
        xlim([0 ymax]);
        plot(xpar,ypar,'Color','black');
        hold on
        x_water = linspace(xleft(2),xright(2),100);
        y_water = linspace(y02,y02,100);
        plot(x_water,y_water,'blue','linewidth',2);
        ylabel('y(m)','Interpreter','latex')
        xlabel('B(m)','Interpreter','latex')
        legend('x = L/2','interpreter','latex')
        hold off
        % third section
        subplot(1,3,3)
        title(['t = ',num2str(round(round(t/60,2),2)),' [min]'])
        ylim([0 ymax]);
        xlim([0 ymax]);
        plot(xpar,ypar,'Color','black');
        hold on
        x_water = linspace(xleft(3),xright(3),100);
        y_water = linspace(y03,y03,100);
        plot(x_water,y_water,'blue','linewidth',2);
        ylabel('y(m)','Interpreter','latex')
        xlabel('B(m)','Interpreter','latex')
        legend('Outlet','interpreter','latex')
        % Save frame
        title(['t = ',num2str(round(round(t/60,2),2)),' [min]'])
        f = getframe(gcf);
        writeVideo(obj,f);
        hold off
        clf
    end
obj.close();
end


% Plots
set(gcf,'units','inches','position',[0,0,7,12])
subplot(3,2,1)
% Flows
plot(tint/60,q2(1,:),'LineStyle','--','LineWidth',2,'Color','k')
hold on
plot(tint/60,q2(ceil(Nx/2),:),'LineStyle',':','LineWidth',2,'Color','k')
hold on
plot(tint/60,q2(Nx,:),'LineStyle','-','LineWidth',2,'Color','k')
hold on
xlabel('Elapsed Time (min)','Interpreter','latex');
```

```
ylabel('Flow Discharge (m\textsuperscript{3}/s)','Interpreter','latex');
legend('Entrance','L/2','Outlet','Interpreter','Latex','location','best')
% Velocity
subplot(3,2,2)
plot(tint/60,q2(1,:)./q1(1,:),'LineStyle','--','LineWidth',2,'Color','k')
hold on
plot(tint/60,q2(ceil(Nx/2),:)./q1(ceil(Nx/2),:),'LineStyle',':','LineWidth',2,'Color','k')
hold on
plot(tint/60,q2(Nx,:)./q1(Nx,:),'LineStyle','-','LineWidth',2,'Color','k')
%%% Normal Depth Velocity %%%
xlabel('Elapsed Time (min)','Interpreter','latex');
ylabel('Velocity (m/s)','Interpreter','latex');
legend('Entrance','L/2','Outlet','Interpreter','Latex','Location','best')
% Water Depth
subplot(3,2,3)
plot(tint/60,y(1,:),'LineStyle','--','LineWidth',2,'Color','k')
hold on
plot(tint/60,y(ceil(Nx/2),:),'LineStyle',':','LineWidth',2,'Color','k')
hold on
plot(tint/60,y(Nx,:),'LineStyle','-','LineWidth',2,'Color','k')
xlabel('Elapsed Time (min)','Interpreter','latex');
ylabel('Water Depths (m)','Interpreter','latex');
legend('Entrance','L/2','Outlet','Interpreter','Latex','Location','best')
% Froude Number
subplot(3,2,4)
plot(tint/60,Fr(1,:),'LineStyle','--','LineWidth',2,'Color','k')
hold on
plot(tint/60,Fr(ceil(Nx/2),:),'LineStyle',':','LineWidth',2,'Color','k')
hold on
plot(tint/60,Fr(Nx,:),'LineStyle','-','LineWidth',2,'Color','k')
xlabel('Elapsed Time (min)','Interpreter','latex');
ylabel('Froude Number','Interpreter','latex');
legend('Entrance','L/2','Outlet','Interpreter','Latex','Location','best')

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

% Rating Curve
% Solving for normal Depth
ymin = min(min(y));
ymax = max(max(y));
y_m = [ymin:0.01:ymax]'; % meters
hs = 1; % half section
% hs = ceil(1);
if flag_section ≠ 4
        Qn = 1/nm(hs).*A_function(D,Z1,Z2,a,b,y_m).*Rh_function(D,Z1,Z2,a,b,y_m).^(2/3).*I0(hs)^0.5;
else
    % [y_table, A, P, Rh, y_bar, n_med, Beta, v, B, Q]
    % [   1,    2, 3, 4,    5,    6,      7,  8,  9, 10]
    col1 = 2; % Col with A
    for jj = 1:length(q1(hs,:))
        Qn(jj,1) = Vlookup_g(irr_table,col1,q1(hs,jj),10); % Attention here
        y_m(jj,1) = Vlookup_g(irr_table,col1,q1(hs,jj),1);
        rh_i = Vlookup_g(irr_table,col1,q1(hs,jj),4);
    end
end
subplot(3,2,6)
tbegin = 30; % (steps), considering initial stabilization of the domain
plot(q2(hs,tbegin:end),y(hs,tbegin:end),'LineStyle','--','LineWidth',2,'Color','k')
hold on
plot(q2(ceil(Nx/2),tbegin:end),y(ceil(Nx/2),tbegin:end),'LineStyle',':','LineWidth',2,'Color','k')
hold on
plot(Qn,y_m,'LineStyle','-','LineWidth',2,'Color','k')
xlabel('Flow Discharge (m\textsuperscript{3}/s)','Interpreter','latex');
ylabel('Water Depth (m)','Interpreter','latex');
ylim([ymin 1.1*max([max(y_m),max(y(ceil(Nx)))])]);
legend('Q(Inlet)','Q(Nx/2)','$Q_{n}$ (L)','Interpreter','Latex','Location','best')
hold off
exportgraphics(gcf,'Summary_Charts.pdf','ContentType','vector')
```

```
clf
close all

% Channel Width Chart
h_f = figure;
axis tight manual % this ensures that getframe() returns a consistent size
filename = 'channel_width.gif';
B2 = zeros(size(q1));
if flag_section ≠ 4
    B2 = B_function(D,Z1,Z2,a,b,y);
else
    for pos_b = 1:length(q1(:,1))
        for time_b = 1:length(q1(1,:))
            % [y_table, A, P, Rh, y_bar, n_med, Beta, v, B, Q]
            % [   1,     2, 3, 4,     5,     6,       7,   8,   9, 10]
            B2(pos_b,time_b) = Vlookup_g(irr_table,col1,q1(pos_b,time_b),9);
        end
    end
end
for n=1:1:(Nt/Nat)
    if n == 1
        t = 1;
        pos = 1;
    else
        t=(n-1)*Nat*dt;
        pos = t/dt;
    end
    left_margin = B2(:,pos)/2; right_margin = -B2(:,pos)/2;
    plot(x,left_margin,'k','LineWidth',2);
    hold on
    plot(x,right_margin,'k','LineWidth',2);
    hold on
    fill([x' fliplr(x')],[right_margin' fliplr(left_margin')],'blue')
    xlabel('x [m]');
    ylabel('B [m]');
    ylim([1.1*min(min(-B2/2)), max(max(1.1*(B2/2)))]);
    grid on
    %legend('y(x,t)',2)
    title(['t = ',num2str(round(t/60,2)),' [min]'])
    drawnow
    % Capture the plot as an image
    frame = getframe(h_f);
    im = frame2im(frame);
    [imind,cm] = rgb2ind(im,256);
    % Write to the GIF File
    if n == 1
        imwrite(imind,cm,filename,'gif', 'Loopcount',inf);
    else
        imwrite(imind,cm,filename,'gif','WriteMode','append');
    end
    hold off
end
clf
close all

% Water Depth Profile
h_f = figure;
axis tight manual % this ensures that getframe() returns a consistent size
filename = 'channel_wse_profile.gif';
wse = inv_el + y; % water surface elevation in meters
for n=1:1:(Nt/Nat)
     if n == 1
        t = 1;
        pos = 1;
    else
        t=(n-1)*Nat*dt;
        pos = t/dt;
    end
    plot(x,inv_el,'LineWidth',4,'LineStyle','-','Color','k')
    hold on
    plot(x,wse(:,pos),'k','LineWidth',2,'LineStyle','-','Color','blue')
    fill([x' fliplr(x')], [inv_el' fliplr(wse(:,pos)')],'blue')
    xlabel('x [m]','Interpreter','latex');
    ylabel('Water surface Elevation [m]','Interpreter','latex');
    ylim([0.98*min(min(wse - y)) max(max(1.01*wse))])
    grid on
    %legend('y(x,t)',2)
```

```
    title(['t = ',num2str(round(t/60,2)),' [min]'])
    drawnow
    % Capture the plot as an image
    frame = getframe(h_f);
    im = frame2im(frame);
    [imind,cm] = rgb2ind(im,256);
    % Write to the GIF File
    if n == 1
        imwrite(imind,cm,filename,'gif', 'Loopcount',inf);
    else
        imwrite(imind,cm,filename,'gif','WriteMode','append');
    end
    hold off
    %mov=addframe(mov,Mo); %Para gravar o v deo , n o comentar (excluir %)
end
close all
% 3D Channel Water Surface Elevation

% Video
obj = VideoWriter('WSE.avi','Motion JPEG AVI');
obj.Quality = 100;
obj.FrameRate = 20;
open(obj)
for n=1:1:(Nt/Nat)
    if n == 1
        t = 1;
        pos = 1;
    else
        t=(n-1)*Nat*dt;
        pos = t/dt;
    end
    plot(x,inv_el,'LineWidth',4,'LineStyle','-','Color','k')
    hold on
    plot(x,wse(:,pos),'k','LineWidth',2,'LineStyle','-','Color','blue')
    fill([x' fliplr(x')], [inv_el' fliplr(wse(:,pos)')],'blue')
    xlabel('x [m]','Interpreter','latex');
    ylabel('Water surface Elevation [m]','Interpreter','latex');
    ylim([0.98*min(min(wse - y)) max(max(1.01*wse))])
    grid on
    title(['t = ',num2str(round(round(t/60,2),2)),' [min]'])
    f = getframe(gcf);
    writeVideo(obj,f);
    hold off
end
obj.close();

%% Ploting Irregular Cross-section
% Cross-section Depths
% Water Depth Profile
% Video
%%%%%%%%%%% Figure Without Fill %%%%%%%%%%%%%%%%
% sm = (1e-8 + 1);
% obj = VideoWriter('Cross_section_outlet.avi','Motion JPEG AVI');
% obj.Quality = 100;
% obj.FrameRate = 20;
% open(obj)
% set(gcf,'units','inches','position',[2,0,6.5,8])
% for n=1:1:(Nt/Nat)
%     if n == 1
%         t = 1;
%     else
%         t=(n-1)*Nat*dt;
%     end
%     subplot(3,1,1)
%     pos = Nx; % Position where the Plots will be made (Outlet)
%     y_cs = [y(pos,t) y(pos,t)]; % Vector of Water Depth (m)
%     plot(x_cross,y_cross,'LineWidth',1.5,'Color','k') % Plotting Cross-Section
%     hold on
%     scatter(x_cross,y_cross,'o','b') % Plotting Break Points
%     % Determine x_inv e y_inv
%     min_el = min(y_cross); % Minimum elevation (m)
%     pos_inv = find(y_cross == min_el); % Position where it occurs
%     x_inv = x_cross(pos_inv); y_inv = y_cross(pos_inv); % x coordinate of invert (m)
%     % Determine x_left
%     x_left_unsorted = x_cross(1:(pos_inv-1),1);  % Left values of x
%     y_left_unsorted = y_cross(1:(pos_inv-1),1); % Right values of x
%     pos_left_up = find(y_left_unsorted>sm*y_cs(1),1,'last'); % left postion with y > ym
```

```
%     pos_left_down = (pos_left_up + 1); % down postion of y > ym
%     % x and y for left points
%     x_left_up = x_left_unsorted(pos_left_up,1); % x(m)
%     x_left_down = x_left_unsorted(pos_left_down,1); % x(m)
%     y_left_up = y_left_unsorted(pos_left_up,1); % y(m)
%     y_left_down = y_left_unsorted(pos_left_down,1); % y(m)
%     alfa_l = (y_left_up - y_left_down)/(abs(x_left_up - x_left_down)); % Left angle
%     dy = (abs(y_cs(1,1) - y_left_down)); % Difference in water depth
%     x_begin = x_left_down - dy/alfa_l; % Intersection with left bank
%     x_end = x_begin + B2(pos,t); % Intersection with right bank
%     x_b = [x_begin x_end]; % Vector of x
%     plot(x_b,y_cs,'b','LineWidth',2); % Plot of top-width
%     hold off
%     grid on
%     xlabel('x(m)','Interpreter','latex')
%     ylabel('y(m)','Interpreter','latex')
%     title(['Inlet (t) = ',num2str(round(round(t/60,2),2)),' [min]'])
%
%     subplot(3,1,2)
%     pos = ceil(Nx/2); % Position where the Plots will be made (Half)
%     y_cs = [y(pos,t) y(pos,t)]; % Vector of Water Depth (m)
%     plot(x_cross,y_cross,'LineWidth',1.5,'Color','k') % Plotting Cross-Section
%     hold on
%     scatter(x_cross,y_cross,'o','b') % Plotting Break Points
%     % Determine x_inv e y_inv
%     min_el = min(y_cross); % Minimum elevation (m)
%     pos_inv = find(y_cross == min_el); % Position where it occurs
%     x_inv = x_cross(pos_inv); y_inv = y_cross(pos_inv); % x coordinate of invert (m)
%     % Determine x_left
%     x_left_unsorted = x_cross(1:(pos_inv-1),1);  % Left values of x
%     y_left_unsorted = y_cross(1:(pos_inv-1),1); % Right values of x
%     pos_left_up = find(y_left_unsorted>sm*y_cs(1),1,'last'); % left postion with y > ym
%     pos_left_down = (pos_left_up + 1); % down postion of y > ym
%     % x and y for left points
%     x_left_up = x_left_unsorted(pos_left_up,1); % x(m)
%     x_left_down = x_left_unsorted(pos_left_down,1); % x(m)
%     y_left_up = y_left_unsorted(pos_left_up,1); % y(m)
%     y_left_down = y_left_unsorted(pos_left_down,1); % y(m)
%     alfa_l = (y_left_up - y_left_down)/(abs(x_left_up - x_left_down)); % Left angle
%     dy = (abs(y_cs(1,1) - y_left_down)); % Difference in water depth
%     x_begin = x_left_down - dy/alfa_l; % Intersection with left bank
%     x_end = x_begin + B2(pos,t); % Intersection with right bank
%     x_b = [x_begin x_end]; % Vector of x
%     plot(x_b,y_cs,'b','LineWidth',2); % Plot of top-width
%     hold off
%     grid on
%     xlabel('x(m)','Interpreter','latex')
%     ylabel('y(m)','Interpreter','latex')
%     title(['L/2 (t) = ',num2str(round(round(t/60,2),2)),' [min]'])
%
%     subplot(3,1,3)
%     pos = Nx; % Position where the Plots will be made (Half)
%     y_cs = [y(pos,t) y(pos,t)]; % Vector of Water Depth (m)
%     plot(x_cross,y_cross,'LineWidth',1.5,'Color','k') % Plotting Cross-Section
%     hold on
%     scatter(x_cross,y_cross,'o','b') % Plotting Break Points
%     % Determine x_inv e y_inv
%     min_el = min(y_cross); % Minimum elevation (m)
%     pos_inv = find(y_cross == min_el); % Position where it occurs
%     x_inv = x_cross(pos_inv); y_inv = y_cross(pos_inv); % x coordinate of invert (m)
%     % Determine x_left
%     x_left_unsorted = x_cross(1:(pos_inv-1),1);  % Left values of x
%     y_left_unsorted = y_cross(1:(pos_inv-1),1); % Right values of x
%     pos_left_up = find(y_left_unsorted>sm*y_cs(1),1,'last'); % left postion with y > ym
%     pos_left_down = (pos_left_up + 1); % down postion of y > ym
%     % x and y for left points
%     x_left_up = x_left_unsorted(pos_left_up,1); % x(m)
%     x_left_down = x_left_unsorted(pos_left_down,1); % x(m)
%     y_left_up = y_left_unsorted(pos_left_up,1); % y(m)
%     y_left_down = y_left_unsorted(pos_left_down,1); % y(m)
%     alfa_l = (y_left_up - y_left_down)/(abs(x_left_up - x_left_down)); % Left angle
%     dy = (abs(y_cs(1,1) - y_left_down)); % Difference in water depth
%     x_begin = x_left_down - dy/alfa_l; % Intersection with left bank
%     x_end = x_begin + B2(pos,t); % Intersection with right bank
%     x_b = [x_begin x_end]; % Vector of x
%     plot(x_b,y_cs,'b','LineWidth',2); % Plot of top-width
%     hold off
```

```
%     grid on
%     xlabel('x(m)','Interpreter','latex')
%     ylabel('y(m)','Interpreter','latex')
%     title(['L (t) = ',num2str(round(round(t/60,2),2)),' [min]'])
%
%     % Writting Video
%     f = getframe(gcf);
%     writeVideo(obj,f);
%     hold off
%     clf
%
% end
% close all
if flag_section == 4
sm = (1e-8 + 1);
obj = VideoWriter('Cross_section_outlet.avi','Motion JPEG AVI');
obj.Quality = 100;
obj.FrameRate = 5;
open(obj)
set(gcf,'units','inches','position',[2,0,6.5,8])
for n=1:1:(Nt/Nat)
    if n == 1
        t = 1;
        pos = 1;
    else
        t=(n-1)*Nat*dt;
        pos = t/dt;
    end
    subplot(3,1,1)
    pos_x = Nx; % Position where the Plots will be made (Inlet)
    y_cs = [y(pos_x,pos) y(pos_x,pos)]; % Vector of Water Depth (m)
    plot(x_cross,y_cross,'LineWidth',1.5,'Color','k') % Plotting Cross-Section
    hold on
    scatter(x_cross,y_cross,'o','b') % Plotting Break Points
    % Determine x_inv e y_inv
    min_el = min(y_cross); % Minimum elevation (m)
    pos_inv = find(y_cross == min_el); % Position where it occurs
    x_inv = x_cross(pos_inv); y_inv = y_cross(pos_inv); % x coordinate of invert (m)
    % Determine x_left
    x_left_unsorted = x_cross(1:(pos_inv-1),1);  % Left values of x
    y_left_unsorted = y_cross(1:(pos_inv-1),1); % Right values of x
    y_right_unsorted = y_cross(pos_inv + 1:end,1); % Right values of x
    x_right_unsorted = x_cross(pos_inv + 1:end,1); % Right values of x
    pos_left_up = find(y_left_unsorted>sm*y_cs(1),1,'last'); % left postion with y > ym
    if pos_left_up == length(y_left_unsorted)
        pos_left_down = pos_left_up;
        x_left_up = x_left_unsorted(pos_left_up,1); % x(m)
        x_left_down = x_inv;
        y_left_up = y_left_unsorted(pos_left_up,1); % y(m)
        y_left_down = y_inv;
        alfa_l = (y_left_up - y_left_down)/(abs(x_left_up - x_left_down)); % Left angle
        dy = (abs(y_cs(1,1) - y_left_down)); % Difference in water depth
    else
       pos_left_down = (pos_left_up + 1); % down postion of y > ym
        % x and y for left points
        x_left_up = x_left_unsorted(pos_left_up,1); % x(m)
        x_left_down = x_left_unsorted(pos_left_down,1); % x(m)
        y_left_up = y_left_unsorted(pos_left_up,1); % y(m)
        y_left_down = y_left_unsorted(pos_left_down,1); % y(m)
        alfa_l = (y_left_up - y_left_down)/(abs(x_left_up - x_left_down)); % Left angle
        dy = (abs(y_cs(1,1) - y_left_down)); % Difference in water depth
    end
    pos_right_end = find(y_right_unsorted>sm*y_cs(1),1,'first'); % left postion with y > ym

    if isnan(alfa_l)
        x_begin = x_left_down;
    else
        x_begin = x_left_down - dy/alfa_l; % Intersection with left bank
    end
    x_end = x_begin + B2(pos_x,pos); % Intersection with right bank
    x_b = [x_begin x_end]; % Vector of x
    plot(x_b,y_cs,'b','LineWidth',2); % Plot of top-width
    x_flip = linspace(x_b(1,1),x_b(1,2),length(x_cross))';
    %%%
    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)];
```

```
    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)];
%     y_section = linspace(y_cs(1,1),y_cs(1,2),length(x_cross))';
    %%%
    y_flip = linspace(y_cs(1,1),y_cs(1,2),length(y_section))';
    p = fill([x_section' fliplr(x_section')],[y_section' fliplr(y_flip')],'blue');
    p.EdgeColor = [1 1 1];
    hold off
    grid on
    xlabel('x(m)','Interpreter','latex')
    ylabel('y(m)','Interpreter','latex')
    xlim([min(x_cross) max(x_cross)])
    title(['Outlet (t) = ',num2str(round(round(t/60,2),2)),' [min]'])

    subplot(3,1,2)
    pos_x = ceil(Nx/2); % Position where the Plots will be made (Inlet)
    y_cs = [y(pos_x,pos) y(pos_x,pos)]; % Vector of Water Depth (m)
    plot(x_cross,y_cross,'LineWidth',1.5,'Color','k') % Plotting Cross-Section
    hold on
    scatter(x_cross,y_cross,'o','b') % Plotting Break Points
    % Determine x_inv e y_inv
    min_el = min(y_cross); % Minimum elevation (m)
    pos_inv = find(y_cross == min_el); % Position where it occurs
    x_inv = x_cross(pos_inv); y_inv = y_cross(pos_inv); % x coordinate of invert (m)
    % Determine x_left
    x_left_unsorted = x_cross(1:(pos_inv-1),1);  % Left values of x
    y_left_unsorted = y_cross(1:(pos_inv-1),1); % Right values of x
    y_right_unsorted = y_cross(pos_inv + 1:end,1); % Right values of x
    x_right_unsorted = x_cross(pos_inv + 1:end,1); % Right values of x
    pos_left_up = find(y_left_unsorted>sm*y_cs(1),1,'last'); % left postion with y > ym
    if pos_left_up == length(y_left_unsorted)
        pos_left_down = pos_left_up;
        x_left_up = x_left_unsorted(pos_left_up,1); % x(m)
        x_left_down = x_inv;
        y_left_up = y_left_unsorted(pos_left_up,1); % y(m)
        y_left_down = y_inv;
        alfa_l = (y_left_up - y_left_down)/(abs(x_left_up - x_left_down)); % Left angle
        dy = (abs(y_cs(1,1) - y_left_down)); % Difference in water depth
    else
       pos_left_down = (pos_left_up + 1); % down postion of y > ym
        % x and y for left points
        x_left_up = x_left_unsorted(pos_left_up,1); % x(m)
        x_left_down = x_left_unsorted(pos_left_down,1); % x(m)
        y_left_up = y_left_unsorted(pos_left_up,1); % y(m)
        y_left_down = y_left_unsorted(pos_left_down,1); % y(m)
        alfa_l = (y_left_up - y_left_down)/(abs(x_left_up - x_left_down)); % Left angle
        dy = (abs(y_cs(1,1) - y_left_down)); % Difference in water depth
    end
    pos_right_end = find(y_right_unsorted>sm*y_cs(1),1,'first'); % left postion with y > ym

    if isnan(alfa_l)
        x_begin = x_left_down;
    else
        x_begin = x_left_down - dy/alfa_l; % Intersection with left bank
    end
    x_end = x_begin + B2(pos_x,pos); % Intersection with right bank
    x_b = [x_begin x_end]; % Vector of x
    plot(x_b,y_cs,'b','LineWidth',2); % Plot of top-width
    x_flip = linspace(x_b(1,1),x_b(1,2),length(x_cross))';
    %%%
    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)];
    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)];
%     y_section = linspace(y_cs(1,1),y_cs(1,2),length(x_cross))';
    %%%
    y_flip = linspace(y_cs(1,1),y_cs(1,2),length(y_section))';
    p = fill([x_section' fliplr(x_section')],[y_section' fliplr(y_flip')],'blue');
    p.EdgeColor = [1 1 1];
    hold off
    grid on
    xlabel('x(m)','Interpreter','latex')
    ylabel('y(m)','Interpreter','latex')
    xlim([min(x_cross) max(x_cross)])
    title(['L/2 (t) = ',num2str(round(round(t/60,2),2)),' [min]'])

    subplot(3,1,3)
```

```
    pos_x = Nx; % Position where the Plots will be made (Inlet)
    y_cs = [y(pos_x,pos) y(pos_x,pos)]; % Vector of Water Depth (m)
    plot(x_cross,y_cross,'LineWidth',1.5,'Color','k') % Plotting Cross-Section
    hold on
    scatter(x_cross,y_cross,'o','b') % Plotting Break Points
    % Determine x_inv e y_inv
    min_el = min(y_cross); % Minimum elevation (m)
    pos_inv = find(y_cross == min_el); % Position where it occurs
    x_inv = x_cross(pos_inv); y_inv = y_cross(pos_inv); % x coordinate of invert (m)
    % Determine x_left
    x_left_unsorted = x_cross(1:(pos_inv-1),1);  % Left values of x
    y_left_unsorted = y_cross(1:(pos_inv-1),1); % Right values of x
    y_right_unsorted = y_cross(pos_inv + 1:end,1); % Right values of x
    x_right_unsorted = x_cross(pos_inv + 1:end,1); % Right values of x
    pos_left_up = find(y_left_unsorted>sm*y_cs(1),1,'last'); % left postion with y > ym
    if pos_left_up == length(y_left_unsorted)
        pos_left_down = pos_left_up;
        x_left_up = x_left_unsorted(pos_left_up,1); % x(m)
        x_left_down = x_inv;
        y_left_up = y_left_unsorted(pos_left_up,1); % y(m)
        y_left_down = y_inv;
        alfa_l = (y_left_up - y_left_down)/(abs(x_left_up - x_left_down)); % Left angle
        dy = (abs(y_cs(1,1) - y_left_down)); % Difference in water depth
    else
       pos_left_down = (pos_left_up + 1); % down postion of y > ym
        % x and y for left points
        x_left_up = x_left_unsorted(pos_left_up,1); % x(m)
        x_left_down = x_left_unsorted(pos_left_down,1); % x(m)
        y_left_up = y_left_unsorted(pos_left_up,1); % y(m)
        y_left_down = y_left_unsorted(pos_left_down,1); % y(m)
        alfa_l = (y_left_up - y_left_down)/(abs(x_left_up - x_left_down)); % Left angle
        dy = (abs(y_cs(1,1) - y_left_down)); % Difference in water depth
    end
    pos_right_end = find(y_right_unsorted>sm*y_cs(1),1,'first'); % left postion with y > ym

    if isnan(alfa_l)
        x_begin = x_left_down;
    else
        x_begin = x_left_down - dy/alfa_l; % Intersection with left bank
    end
    x_end = x_begin + B2(pos_x,pos); % Intersection with right bank
    x_b = [x_begin x_end]; % Vector of x
    plot(x_b,y_cs,'b','LineWidth',2); % Plot of top-width
    x_flip = linspace(x_b(1,1),x_b(1,2),length(x_cross))';
    %%%
    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)];
    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)];
%     y_section = linspace(y_cs(1,1),y_cs(1,2),length(x_cross))';
    %%%
    y_flip = linspace(y_cs(1,1),y_cs(1,2),length(y_section))';
    p = fill([x_section' fliplr(x_section')],[y_section' fliplr(y_flip')],'blue');
    p.EdgeColor = [1 1 1];
    hold off
    grid on
    xlabel('x(m)','Interpreter','latex')
    ylabel('y(m)','Interpreter','latex')
    xlim([min(x_cross) max(x_cross)])
    title(['Inlet (t) = ',num2str(round(round(t/60,2),2)),' [min]'])

    % Writting Video
    f = getframe(gcf);
    writeVideo(obj,f);
    hold off
    clf
end
end
obj.close();
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