Extraction of Stream Characteristics Using DEM

RnD Report
By

Jayabrata Das
193050085

Master of Technology
Computer Science and Engineering

Guided By
Prof. Milind Sohoni
Co-guide
Prof. Om Prakash Damani

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay

June 2019

Acknowledgments

I would like to express my special thanks of gratitude to Prof. Milind Sohoni and Prof. Om Damani for giving me this opportunity and for their help in the Seminar. I would like to give my special thanks to Mr Parth Gupta for his guidance and weekly meetings that helped me better understand the topic and kept me motivated. I am also very much grateful to my family members and friends for their support, help and encouragement throughout this course.

Contents

LIST OF FIGURES

1 Introduction

Watershed is an area in land that drains the rain water or snowmelt water into a specific waterbody like river and stream. These waterbodies later join larger waterbodies like bays and oceans. So, the watershed contributes to these large waterbodies indirectly by draining the water into smaller streams. Some portion of the rain water also get filtered into the ground and joins the underground water reservoirs, which is the major source of drinking water. Thus, watersheds also provide drinking water to us.

Watersheds are extremely important. A healthy watershed performs many functions. The main functions that a watershed performs are 1) Watershed collects the rainfall water and uses them in different functions. 2) Some part of the water gets filtered into the ground and thus stored in the underground water reservoir. It also stores water in minor waterbodies on the ground. 3) All precipitation that falls within a watershed, but is not used by existing vegetation, are released as runoff. This runoff water ultimately flows to the lowest points. These low points are bodies of water such as rivers, lakes, and the ocean.

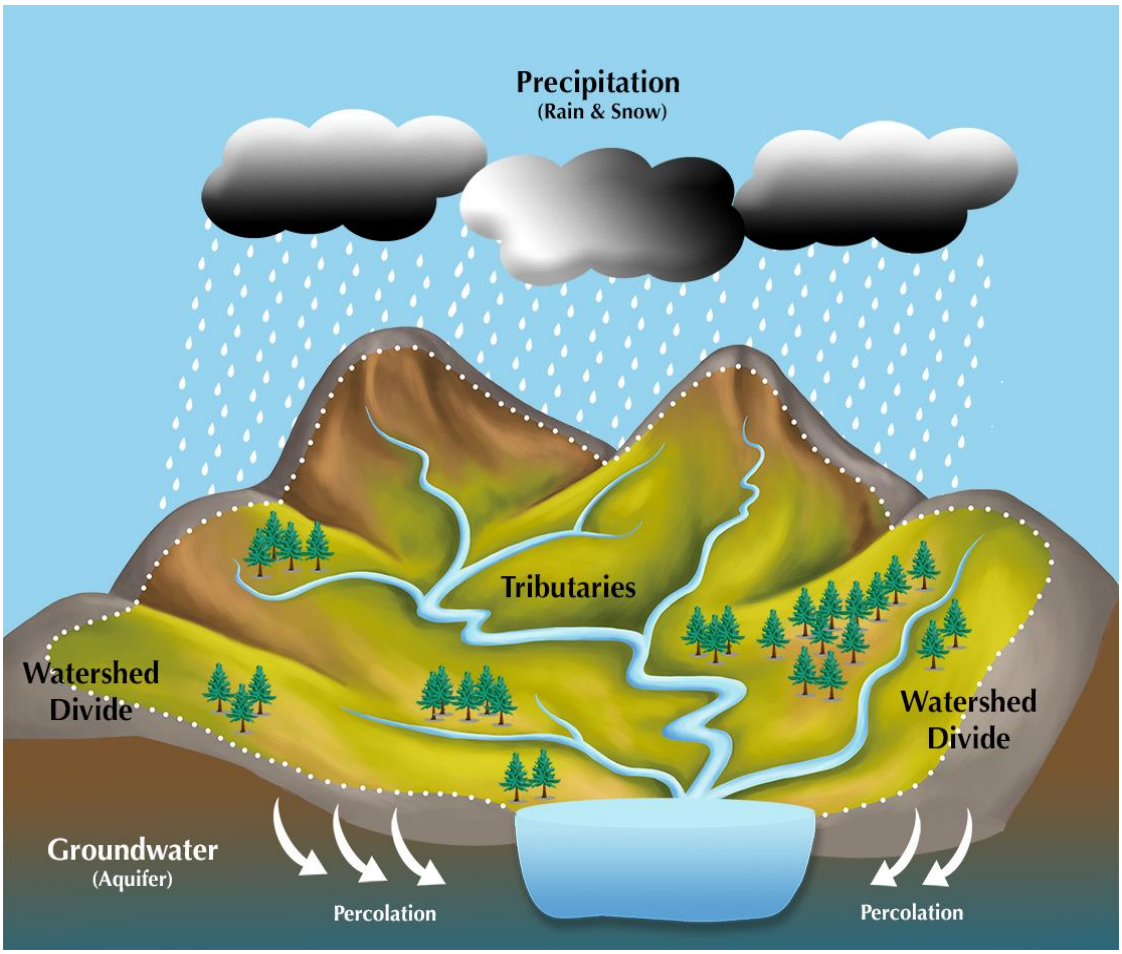


Figure 1: Example of watershed basin and hydrological cycle

2 Motivation

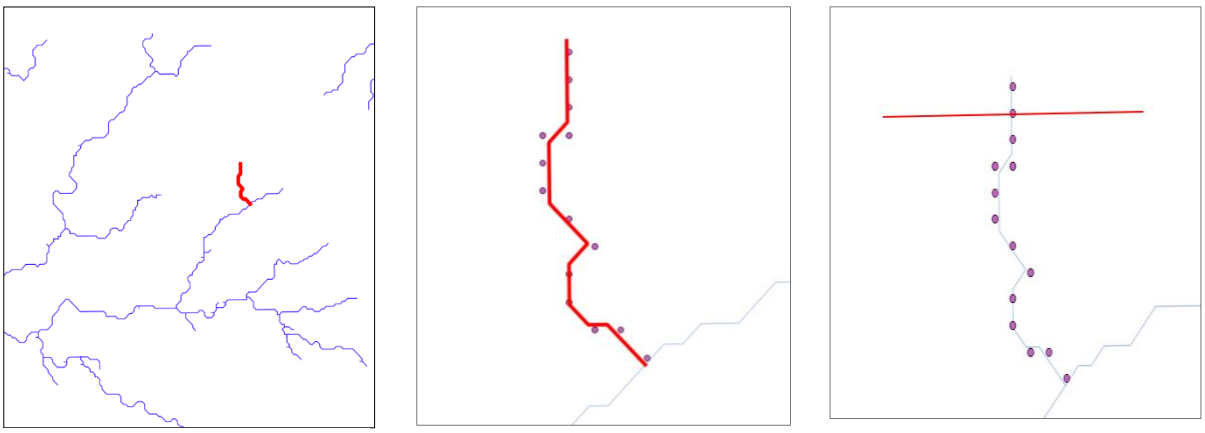
Watershed plays a major role hydrological cycle by collecting the precipitations and draining them into minor waterbodies. In an agricultural country like India these minor waterbodies are important part of agriculture. Also, in so rural areas the minor waterbodies are considered a major source of water. So, it is very important to keep track of these waterbodies and the watersheds that creates them. Using modern hydrological models, it is possible to compute various static and dynamic characteristics of such waterbodies. Based on these features, it can be possible to plan proper policies so that these water sources can be utilised efficiently. This can be beneficial for government to make agricultural policies if the available water amount is known.

In this project we try to compute these stream features from a DEM raster file. Some functions already exist using which we can compute stream network from DEM but measuring depth and width in more challenging as there is no existing function for this purpose. We first introduce an algorithm to compute depth of stream from a DEM raster. Next using this depth value and some other known parameters we compute the width value using equations from hydrological models. Finally, from these stream characteristics we compute stream flow features (dynamic).

3 Depth Computation Methods

Computing depth and width of a stream is a challenging task as there is no existing function that can compute the depth and width from the elevation data only. So, we have to apply our own algorithm for this task. Given a DEM file we can extract the stream network from it using existing functions (*'grass:r.streamrextract'* or *'saga:channelnetworkanddrainagebasins'*). By observing the elevation data across different on the stream line we may get some idea about the depth. We apply this approach in our algorithm.

The idea is to find the depth on multiple points along the stream line and take their mean. For this first, points are generated along each line segment at regular interval as shown in ***figure 2b***. Next, for each point height is measured from the elevation data along a line perpendicular to the stream line ("depth- view line") and passing through the point as shown in ***figure 2c.*** Then, we apply the depth measuring algorithm, which is explained later in this section. Thus, we get a depth value for each point along the stream line. Finally, we take the average of the depth value along each line segments and add that value in the feature table. The details step wise description of this approach given below.



a) The red line indicates a stream line segment, one row in the data table

b) Points are generated along stream line segment at regular interval

c) Depth is calculated along the red line ("depth-view line"), line crossing the point

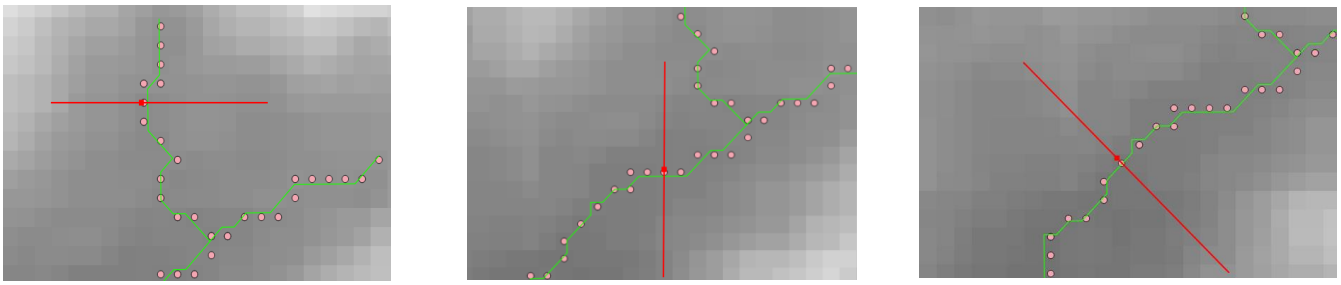
Figure 2: Generating points along stream line and depth-view line

- **Generating points along line stream:**

To generate the points along the stream lines at regular interval a vector processing tool is used named, generate points (pixel centroids) along line. This function maintains the source info of each generated point by including the *line_id* of its source line segment in the feature table, which later helps to calculate the average depth of stream segment from its corresponding points. Also, each point has a unique *point_id* and these ids are assigned serially along the line segments.

- **Finding Depth-view line type:**

To find elevation info along the line (depth-view line) perpendicular to the stream line we need to know whether the line segment is vertical or horizontal or at 45º angle. If the line is vertical then the depth-view line will be horizontal ***fig 3a*** and vice versa ***fig 3b***. For this, we compare the coordinate of a point with its previous point. If the point is found to move along X-axis only from the previous point then the point is part of horizontal line segment in that case the depth-view line is vertical and if it moves only along Y-axis then it is part of vertical line segment in which case, the depth-view line is horizontal. In case the point is found to move along both the axis (45º or 135º) then, the depth-view line will be 135º or 45º accordingly.



a) Horizontal depth view line (in red) for vertical part of the stream segment

b) Vertical depth view line (in red) for horizontal part of the stream segment

c) Depth view line at 135º angle (in red) for a stream segment at 45º angle

Figure 3: Different types of depth-view line for different line segment

- **Finding depth for stream points:**

To find the depth we start from the point on the line stream and apply an algorithm. We move perpendicularly to the line stream from the point in both direction by one unit and keep checking the height. The coordinates of the point are used for this (if the movement is horizontal then only x coordinate is updates by one unit after each iteration and only y is updated for vertical line). The height at a coordinate is retrieved from the DEM raster layer. In every iteration we start from the point on the stream line (denoted by blue dot in the figure below) and move to two points on both sides of the stream line (denoted by black dots in the figure below) and check the height on those points and compare it with previous points. Initially the height increases and after some point, (after few iterations)

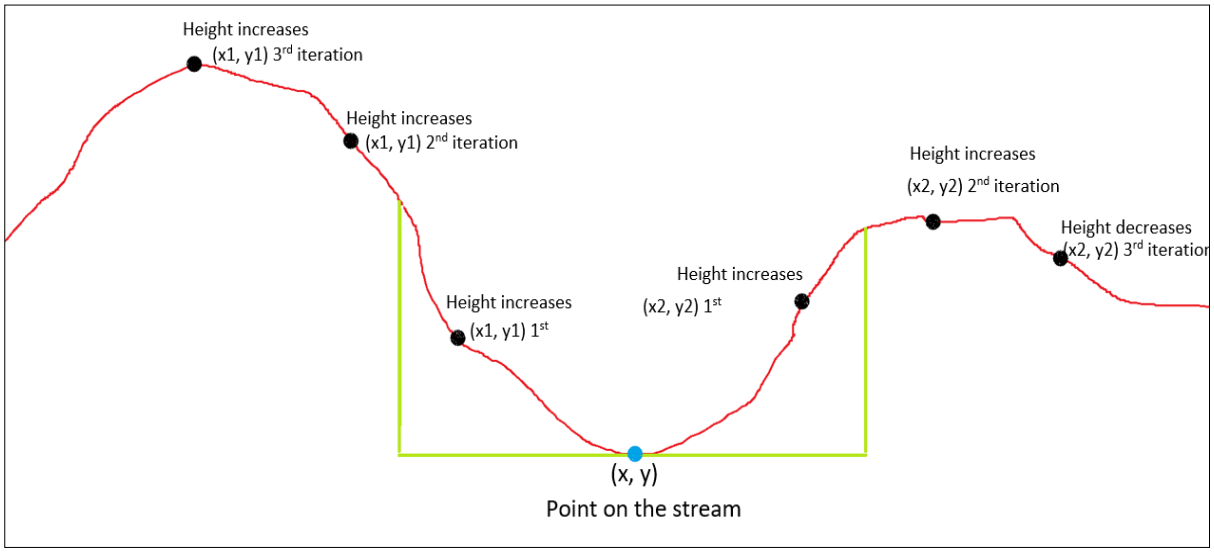


Figure 4: Height view along the depth-view line. Finding depth by checking height on multiple points on both sides of the stream line

the height decreases (or remains unchanged) at least on one side. In the ***fig 4*** the height increases for first two points on the left side but at the third point the height falls. We capture the highest point after which either the height start falling or remains unchanged (at least for two consecutive pixel values) and the difference of heights between this point (denoted by green line in the figure below) and the initial point is considered as the depth on that point.

3.1 Algorithm: Measure mean depth of a stream segment

INPUT: max_width (limiting depth-view line), stepSize (movement along depth-view line)

Function getDepth(streamSegment)

depth_sum = 0

count = 0

maxItr1 = max_width/stepSize

maxItr2 = max_width/(stepSize*√2)

for each point in the layer

x,y = coordinate of the point

if the point is not the first point

line = line connecting current point with previous point

if line is horizontal

xd, yd = 0, stepSize

maxItr = maxItr1

else if line is vertical

xd, yd = stepSize, 0

maxItr = maxItr1

else if line is 45º

xd, yd = stepSize, stepSize

maxItr = maxItr2

else if line is 135º

xd, yd = stepSize, -stepSize

maxItr = maxItr2

end if

height = height at (x, y)

flag_right = flag_left = 0

x1, y1 = x2, y2 = x, y

height1 = height2 = height

while flag_right is positive and iteration is < maxItr

height1 = height1’

```
            next right point, x1, y1 = x1+xd, y1+yd
            height1' = height at (x1, y1)
            if flag_right == 0 and height has increases
               minHeight = minimum(height, height1')
               flag_right = 1
            else if flag_right == 1 and height has not increased
               flag_right = -1
            end if
         end while
         while flag_left is positive and iteration is < maxItr and (flag_right>=0 or height2'<= height1)
            height2 = height2'
            next left point, x2, y2 = x2+xd, y2+yd
            height2' = height at (x2, y2)
            if flag_left == 0 and height has increases
               minHeight = minimum(height, height2')
               flag_left = 1
            else if flag_left == 1 and height has not increased
               flag_left = -1
            end if
         end while
         if both right and left flag are positive
            depth = minimum(height1', height2') – height
         else
            depth = minimum(height1, height2) – height
         end if
         depth_sum, count = depth_sum + depth, count + 1
      end if
   end for
   if depth is positive and count >0
      return depth
   else
      return 0
   end if
end function
```

3.2 Output of getDepth function

Following ***figure 5*** shows the depth on different points along the stream segment. Here 'id' refers to the unique point ids of points along a specific stream segment. The depths are close to the real depth values of the stream segments.

```
id - 0 : Depth:  1.0
id - 1 : Depth:  1.0
id - 2 : Depth:  3.0
id - 5 : Depth:  1.0
id - 6 : Depth:  5.0
id - 7 : Depth:  5.0
id - 8 : Depth:  5.0
id - 9 : Depth:  1.0
id - 11 : Depth:  3.0
id - 12 : Depth:  3.0
id - 13 : Depth:  2.0
id - 14 : Depth:  5.0
id - 15 : Depth:  5.0
id - 16 : Depth:  2.0
id - 17 : Depth:  0.0
id - 18 : Depth:  34.0
id - 19 : Depth:  27.0
id - 20 : Depth:  20.0
id - 21 : Depth:  2.0
id - 22 : Depth:  1.0
id - 23 : Depth:  32.0
id - 26 : Depth:  4.0
```

Figure 5*:* Output of getDepth function showing depth on different points along the stream segment

4 Width Computation Method

Once we get the depth values of a stream segment, one simple approach to find width can be to traverse along the depth-view line until the depth height is reaches on both sides of the stream point. This will give two points on the depth-view line and the distance between these two points can be considered as width of the stream segment. But this approach may fail due to the following reasons.

- The pixel size of the DEM raster layer used in the program is 12.5 x 12.5 (meters) but in the region for which the watershed is computed, some streams may have less than 12.5 m width. In such cases we will not be able to get the actual width value which will cause farther computation errors in the next stages.

- There are some cases, (***fig 6***) where we do not get two points on the depth-view line within the minimum width range. In such cases we have to assign the minimum width value (used in getDepth function) to such points. Thus, we end up over estimating the width value.

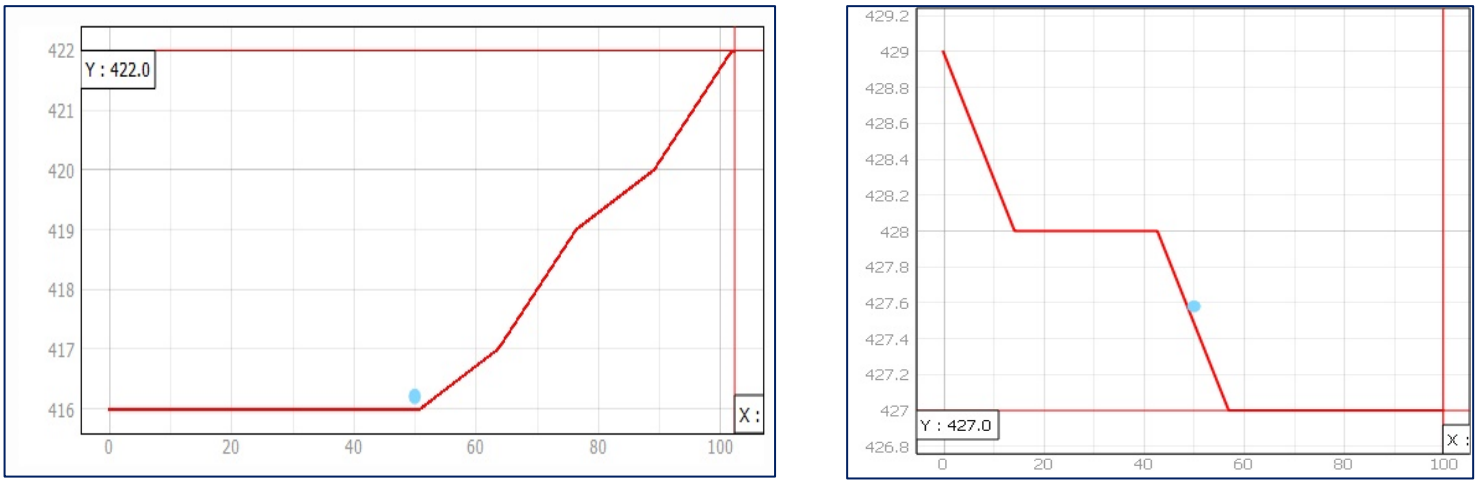


Figure 6*:* Two cases where width can not be found within the minimum width range

5 Computing Width Using Hydrological Model

After the failure of the width computing algorithm we tried some alternative methods. Using the formulas in hydrological model it is possible to compute the width of a stream segment if other parameters are known. Following are the parameters and their formulas that are used to compute the width.

Parameter list:

- qch = discharge (m^3/s)
- peak runoff (mm)
- Area = Watershed Area (Hectare)
- Depth (meter)
- Ach = cross section area
- Wtop = top width
- Wbottom = bottom width (unknown)
- Slp = mean slope in the stream
- Pch = wetted parameter
- Rch = Hydraulic radius
- n = Manning = 0.05
- Sideslope = 1

List of formulas used:

$$\text{qch} = \frac{Peak\ Runoff * Area * 10}{3600} \quad \text{.....(1)}$$

$$\text{qch} = \frac{Ach * Rch^{\frac{2}{3}} * slp^{\frac{1}{2}}}{n} \quad \text{.....(2)}$$

$$\text{Wtop} = Wbottom + 2 * SideSlop * depth \quad \text{.....(3)}$$

$$\text{Ach} = \frac{1}{2} * (Wtop + Wbottom) * depth \quad \text{.....(4)}$$

$$\text{Pch} = Wbottom + 2 * depth * (1 + SideSlop^2)^{\frac{1}{2}} \quad \text{.....(5)}$$

$$\text{Rch} = \frac{Ach}{Pch} \quad \text{.....(6)}$$

Putting value of eq(3) in eq(4):

$$\text{Ach} = (Wbottom * depth + SideSlop * depth^2) \quad \text{.....(7)}$$

Equating equation (1) and (2):

$$\frac{Peak\ Runoff * Area * 10}{3600} = \frac{Ach * Rch^{\frac{2}{3}} * slp^{\frac{1}{2}}}{n} \quad \text{.....(8)}$$

Putting values of equation (6) in equation (8):

$$\frac{Peak\ Runoff * Area * 10 * n}{3600 * slp^{\frac{1}{2}}} = \frac{Ach^{\frac{5}{3}}}{Pch^{\frac{2}{3}}} \quad \text{.....(9)}$$

Putting values of eq(5) and (7) in eq(9):

$$\frac{(Wbottom * depth + SideSlop * depth^2)^{\frac{5}{3}}}{\left(Wbottom + 2 * depth * (1 + SideSlop^2)^{\frac{1}{2}}\right)^{\frac{2}{3}}} = \frac{Peak\ Runoff * Area * 10 * n}{3600 * slp^{0.5}} \quad \text{.....(10)}$$

In equation (10) all parameters are known excepting for wbottom (bottom width). Solving this equation will give the width of a stream segment.

6 Description of the Plugin

The plugin generates a vector stream layer containing the following stream features for each segment – Stream Order, Stream Length (in meter), Depth (in meter), Top Width(in meter), Bottom Width (in meters), Watershed Area (in meter^2), Mean slope of the stream segment and Discharge (meter^3/second).
The plugin works in the following steps:

6.1 Inputs:

It requires the following inputs from the user –

a) a DEM file (to generate stream and extract the stream features from it)
b) a csv file with runoff information for the region in different timestamps
c) a coordinate on the DEM file (to compute watershed w.r.t that point)
d) different parameter values:
    i) Threshold (for stream extraction, default: 500)
    ii) Mannigs (to compute discharge, default: 0.05)
    iii) Friction Deep Aquifer (default: 0.5)
    iv) Hydraulic Conductivity (default: 5)
    v) Bank Flow Recession (Default: 0.3)
    vi) Potential Evaporation (Default: 1)
    vii) Side Slope (default: 1)
e) output file name and location

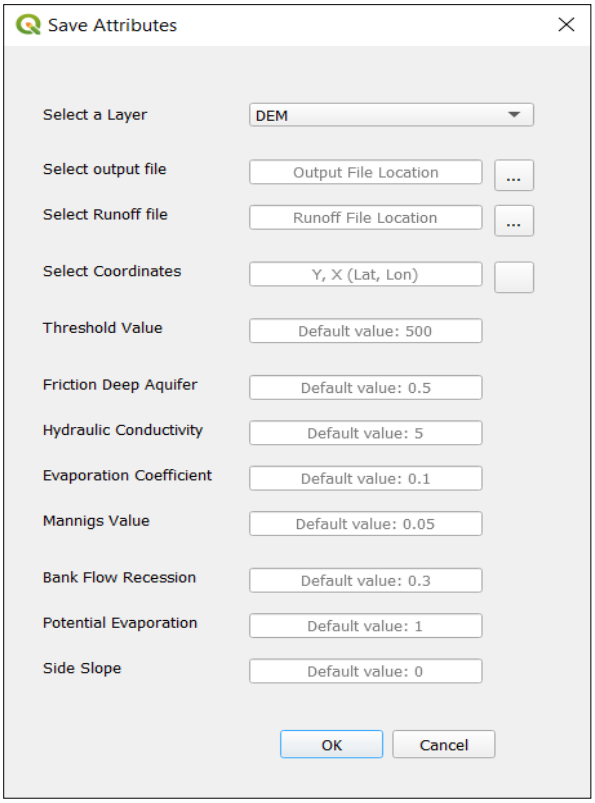
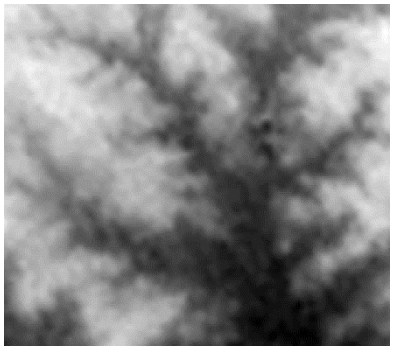


fig 7: Plugin dialog Box

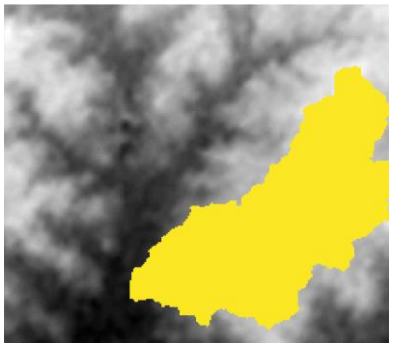
6.2 Generating Stream Network:

Following are the steps to generate stream network from DEM

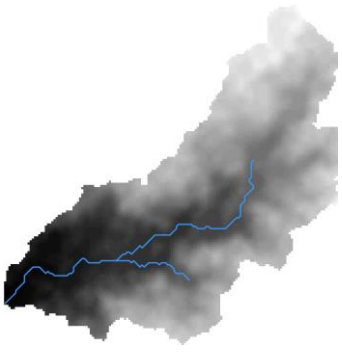
- First, a directory is generated in the given output address named '_temp_' to save all layers that are generated throughout the process.
- A new DEM file (fillDEM.sdat) is generated from the old one after removing all sinks. The *'saga:fillsinksqmofesp'* function is used here. This step is important for generating stream network.
- Next, using *'grass7:r.watershed'* function Drainage Direction raster file (DrainageDirection.tif) is generated. This file is required in order to generate the watershed basin.
- Watershed Basin raster (output file name) is generated for the given point (coordinates) using *'grass7:r.water.outlet'* function. Stream network is generated only for this watershed region.
- Next, the basin raster is converted into vector (BasinArea.shp) using *'grass7:r.water.outlet'*. This is to check the area of the watershed. If very small watershed is generated (or the point is outside the DEM), then the process stops here because for very small watershed no stream network can be generated. A message box is displayed ("invalid point selected") in this case.
- In order to generate stream network only for the watershed area it is required to clip (BasinRaster.tif) the DEM (fillDEM.sdat) by the basin area (BasinArea.shp). For this purpose *'gdal:cliprasterbymasklayer'* function is used.
- Finally, using *'saga:channelnetworkanddrainagebasins'* function stream network layer (Output Filename_Stream.shp) is generated. This function also generates a point vector layer (JunctionPoints.shp) containing the starting points, end points and junction points. These junction points are useful for next steps.



a) Input DEM Raster layer



b) Watershed basin is generated for the given point



c) Stream network is generated from clipped Raster layer

Figure 8: Generating stream network from DEM raster layer

6.3 Generating points along the stream segments:

- As mentioned in the previous section, in order to compute the depth of the stream we generate points (vector point layer: StreamPoints.shp) along the stream lines and measure the depth across all these points and take the mean. The function *'qgis:generatepointspixelcentroidsalongline'* is used for this purpose.
- The layer generated in the previous step may contain some duplicate points (points on same coordinate). To remove such points *'native:delete-duplicategeometries'* function is used that produced a new vector point layer (CleanStreamPoints.shp).

6.4 Adding New Attributes to the Stream Network vector layer:

Following attributes are added to the stream vector layer to save data for each stream segment –

a) "DEPTH" (type: double) – to save depth value
b) "BASIN AREA" (type: double) – to save watershed area value
c) "SLOPE" (type: double) – to save slope value
d) "B_WIDTH" (type: double) – to save bottom width value
e) "T_WIDTH" (type: double) – to save top width value
f) "SOURCES" (type: text) – to save the source "SEGMENT ID"s in a list
g) "Q" (type: double) – to save discharge value

6.5 Finding depth, slope, basin area for each segment:

The getFeatures() is used to measure these values.

a) Finding Depth – To find the mean depth for each segments the algorithm mentioned in the previous section is used. First, the points along the stream lines are selected segment-wise and for each segment the function is called. In case the segment has two or less points on it its depth is considered zero.
b) Finding Slope – To measure the slope for each segment the elevations on two end points are checked. The slope is calculated by the formula –

$$Slope = \frac{|Elevation\ Difference\ at\ two\ end\ points|}{length\ of\ the\ stream}$$

c) Finding Basin Area – The following steps are followed measure the area.

- For each segment find coordinates of two specific points –

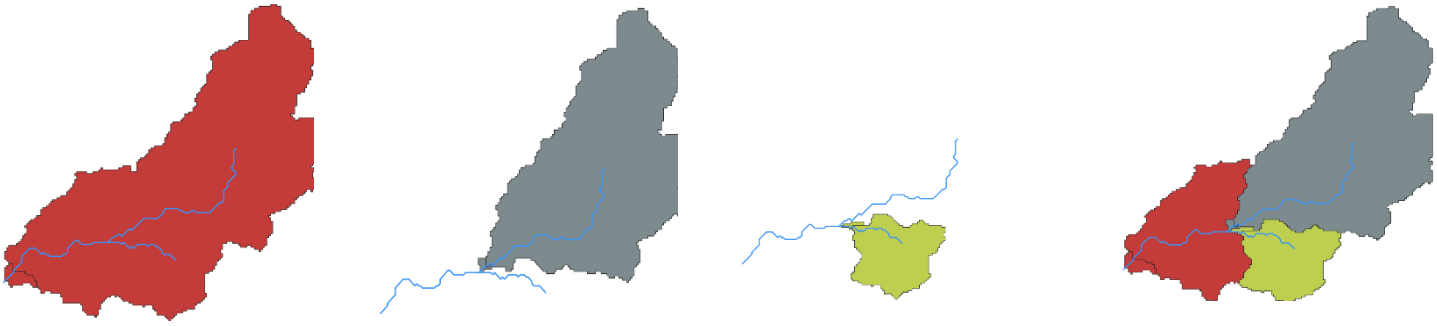
  i) The end point (P_{end}), it is the point on the segment just before merging into another segment. In case this point is a junction point (JunctionPoints.shp) we take the point before that.

ii) The last point P_{ddc} before the P_{end} point at which the drainage direction value changes i.e starting from the P_{end} point the point at which the drainage direction value changes for the first time.

- Next, we compute watershed ("RBasin[*SEGMENT ID*].tif" and "secRBasin[*SEGMENT ID*].tif") w.r.t these two points (P_{end} & P_{ddc}) using *'grass7:r.water.outlet'* function.
- Then these two raster layers are converted into vector layer ("VBasin[*SEGMENT ID*].shp" and "secVBasin[*SEGMENT ID*] .shp") using *'grass7:r.to.vect'* function.
- We take the union of these tow vector layer to compute the final watershed area (Basin[*SEGMENT ID*].shp) using *'native:union'* function. The area of this final vector layer is the watershed area value for a stream segment.
- We take union of watershed areas for two different points because it has been observed that in some cases using a single point (any one of these) may not give the entire watershed area. So, we take the union of these two watersheds to confirm that no region is missed.

6.6 Find Sources and subtracting sub-basin area:

- A stream segment is called source of another stream segment if its end point ("NODE_B") is the starting point ("NODE_A") of the other.
- The stream segment sources are saved in list format in a list named 'sourcesList'.



a) Watershed basin for order 2 stream segment

b) Watershed basin for (1st source) order1 stream segment

c) Watershed basin for (2nd source) order1 stream segment

d) Subtracting basin area of source streams from order 2 stream network

Figure 9: Subtracting basin areas of source streams (order 1) from basin area of higher order (order2) stream

- In the previous step we computed the basin area for each segment but in that basin area a stream segment with higher order (>1) also included the basin area of its sources. But to compute discharge of each segment separately it is required to subtract the basin areas of source segment from the its corresponding higher order stream segment basin area.

- This is implemented simply by subtracting basin areas of source segments (from source list) from each segment (***Figure 9***). These area values are saved into a list named 'bAreaList2'.

6.7 Finding Peak Runoff from the input csv file:

The getPeakRunoff() is used to get the peak runoff from the input csv file. This function simply computes the sum of primary runoff and secondary runoff (column 9 and 12 in input csv file) and take the maximum value of it. In the example runoff file, the value of peak runoff is 12.44 mm.

6.8 Computing Discharge for each stream segment:

- The getQValue() is a simple recursive function that computes the discharge for each stream segment.
- To compute the discharge value for a segment first we need to convert the area into hectare and then use the following equation

$$Discharge = \frac{Peak\ Runoff * Basin\ Area * 10}{3600}$$

- Also, for each segment we need to add the discharge values of its source segments to compute the actual discharge.
- These values are saved into a list named 'qList'.

6.9 Computing Top Width and Bottom Width:

- The function computeFeatures() computes the bottom width and top width using the formula mention in previous section.
- To solve such equation, we used the *sympy* package. In case there is no real solution of the equation or the slope is zero we consider bottom width and top width as 1 to avoid error.

6.10 Adding attribute values into the stream vector layer:

- Once all stream feature values are computed, we add these values into the feature table of the stream vector layer.
- The attribute values are stored into the table in the following format-
    - LENGTH (meter)
    - DEPTH (meter)
    - BASIN AREA (meter^2)
    - SLOPE (no unit)
    - B_WIDTH (meter)
    - T_WIDTH (meter)
    - Q (meter^3/second)

6.11 Creating Stream Features csv file:

- While adding the attribute values into the vector layer, we also save the feature values for each segment into an ordered dictionary and keep them in a list named ‘streamFeaturesList’.
- The constant parameters taken as input from the user are also included into the dictionary.
- Finally, a csv file is created with the name “[*input filename*]_Stream Features.csv” in which the stream feature values are written from the list of ordered dictionary.
- Here the value of 'watershed_area' is in hectare unit and 'length’ is in km. This done to make these values ready for next step where these are need to be used in these specific units.

6.12 Creating Drainage object:

The models.py file defines Drainage class structure representing a surface drainage-network. This module also contains a function ‘run_stream_model_for_time_step()’ to compute the dynamin features of the stream segments. It computes the following features for each stream segment in every time stamp - swat runoff, total volume stored, volume in, discharge, Transmission loss, bank in, return flow from bank, evaporation loss, total loss, Volume after loss, volume out, volume stored end timestep, cross section, Depth water level, width water level, wetted perimeter, hydraulic radius, velocity, travel time, fraction time step and storage coefficient.

In order to compute these dynamic features using the function we create an object of the Drainage class and assign the stream feature values properly. We use the values from ‘streamFeaturesList’ for this assignment.

6.13 Extracting the Runoff values from input file:

Next we create a list containing runoff values at different time stamp from the input runoff files by taking the sum of primary and secondary runoffs (column 9 and 12). These values are used in the next step to compute the dynamic features of the stream segments for each time stamp.

6.14 Computing Dynamic stream features:

In this step we use the runoff values and stream feature values as input to the ‘run_stream_model_for_time_step()’ function that computes the dynamic features of the stream segments and store them in Transient class attributes.

6.15 Creating csv file with time stamp-wise dynamic features:

Finally, the stream dynamic features are saved into the csv file named “[*input filename*]_Stream Flow Data.csv”. The features are written in segment id order i.e. first the features for segment id 1 are written for each time stamp and then segment id 2 and so on.

6.16 Output:

(considering input filename is test)

- test_Stream.shp (stream vector layer containing the stream characteristics)
- test_Stream Features.csv (constant stream characteristics in csv file)
- test_Stream Flow Data.csv (dynamic stream features for each time stamp in csv file)
- Also, a "_temp_" file is created that contains all layers used in this process.

7 Sample Results

Following are some sample of output files –

- Output stream vector layer:

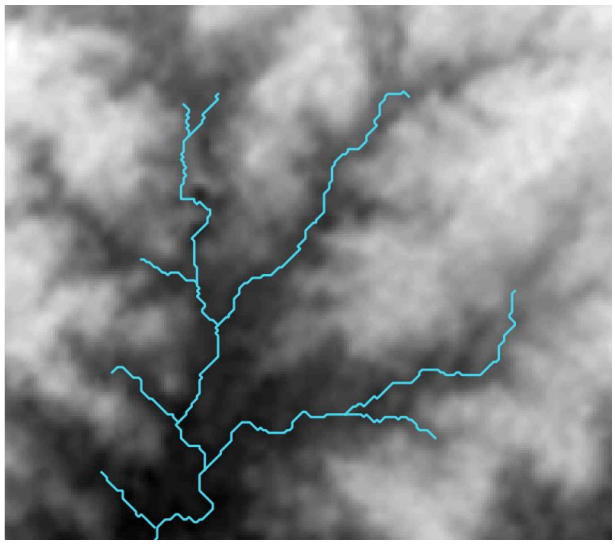


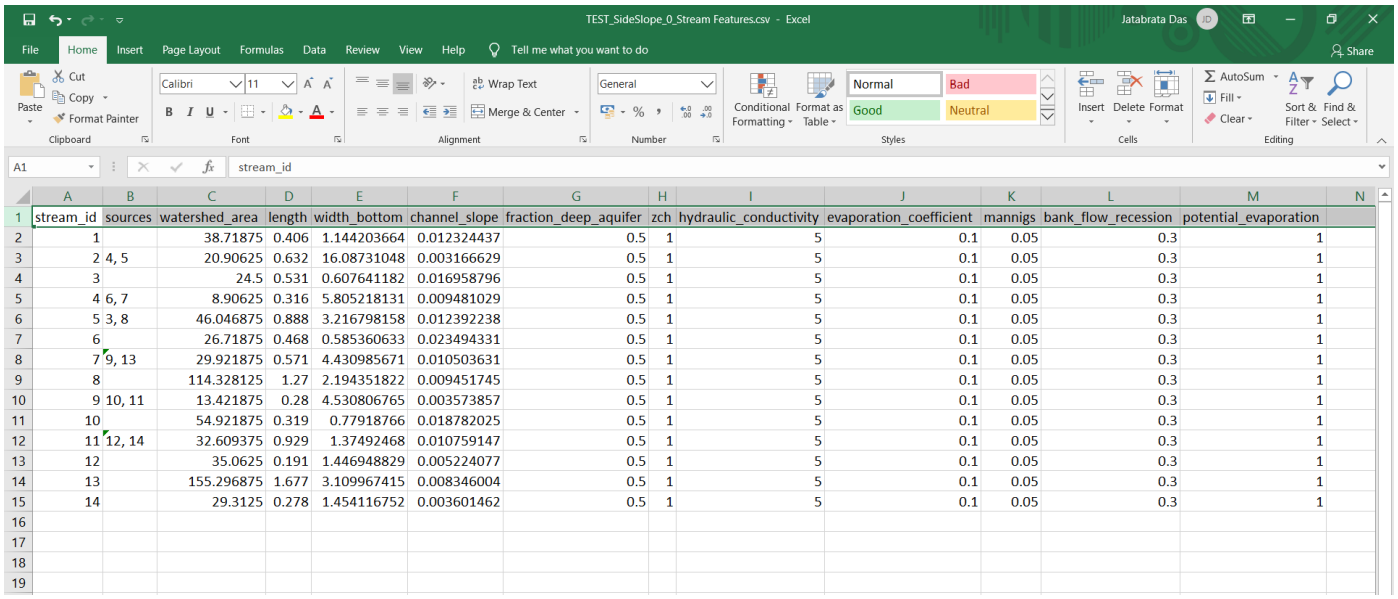
Figure 10: Stream vector layer generated from DEM raster layer

- Attribute table of output stream vectoe:

	SEGMENT_ID	NODE_A	NODE_B	BASIN	ORDER	ORDER_CELL	LENGTH	DEPTH	BASIN AREA	SLOPE	B_WIDTH	T_WIDTH	SOURCES	Q
1	1	3	2	2	1	6	405.6980515300	1.00000	387187.50000	0.01232	1.14420	1.14420	NULL	1.29493
2	2	4	1	1	3	8	631.5863991800	1.11111	209062.50000	0.00317	16.08731	16.08731	4, 5	19.79754
3	3	5	7	1	1	6	530.6980515300	1.31579	245000.00000	0.01696	0.60764	0.60764	NULL	0.81939
4	4	6	4	1	2	7	316.4213562400	1.25000	89062.50000	0.00948	5.80522	5.80522	6, 7	12.91530
5	5	7	4	1	2	7	887.6524163600	1.13333	460468.75000	0.01239	3.21680	3.21680	3, 8	6.18304
6	6	8	6	1	1	6	468.1980515300	1.29412	267187.50000	0.02349	0.58536	0.58536	NULL	0.89359
7	7	9	6	1	2	7	571.2310601200	1.42105	299218.75000	0.0105	4.43099	4.43099	9, 13	11.72385
8	8	10	7	1	1	6	1269.6067812000	1.27451	1143281.25000	0.00945	2.19435	2.19435	NULL	3.82364
9	9	11	9	1	2	7	279.8097038900	1.20000	134218.75000	0.00357	4.53081	4.53081	10, 11	5.52931
10	10	12	11	1	1	6	319.4543648300	1.83333	549218.75000	0.01878	0.77919	0.77919	NULL	1.83683
11	11	13	11	1	2	7	929.4417382400	1.80952	326093.75000	0.01076	1.37492	1.37492	12, 14	3.24359
12	12	14	13	1	1	6	191.4213562400	1.00000	350625.00000	0.00522	1.44695	1.44695	NULL	1.17265
13	13	15	9	1	1	6	1677.4494937000	1.18919	1552968.75000	0.00835	3.10997	3.10997	NULL	5.19382
14	14	16	13	1	1	6	277.6650429400	1.00000	293125.00000	0.0036	1.45412	1.45412	NULL	0.98034

Figure 11: Attribute table showing stream segment features for generated stream network

- Stream features stored in csv file:



stream_id	sources	watershed_area	length	width_bottom	channel_slope	fraction_deep_aquifer	zch	hydraulic_conductivity	evaporation_coefficient	mannigs	bank_flow_recession	potential_evaporation
1		38.71875	0.406	1.144203664	0.012324437	0.5	1	5	0.1	0.05	0.3	1
2	4, 5	20.90625	0.632	16.08731048	0.003166629	0.5	1	5	0.1	0.05	0.3	1
3		24.5	0.531	0.607641182	0.016958796	0.5	1	5	0.1	0.05	0.3	1
4	6, 7	8.90625	0.316	5.805218131	0.009481029	0.5	1	5	0.1	0.05	0.3	1
5	3, 8	46.046875	0.888	3.216798158	0.012392238	0.5	1	5	0.1	0.05	0.3	1
6		26.71875	0.468	0.585360633	0.023494331	0.5	1	5	0.1	0.05	0.3	1
7	9, 13	29.921875	0.571	4.430985671	0.010503631	0.5	1	5	0.1	0.05	0.3	1
8		114.328125	1.27	2.194351822	0.009451745	0.5	1	5	0.1	0.05	0.3	1
9	10, 11	13.421875	0.28	4.530806765	0.003573857	0.5	1	5	0.1	0.05	0.3	1
10		54.921875	0.319	0.77918766	0.018782025	0.5	1	5	0.1	0.05	0.3	1
11	12, 14	32.609375	0.929	1.37492468	0.010759147	0.5	1	5	0.1	0.05	0.3	1
12		35.0625	0.191	1.446948829	0.005224077	0.5	1	5	0.1	0.05	0.3	1
13		155.296875	1.677	3.109967415	0.008346004	0.5	1	5	0.1	0.05	0.3	1
14		29.3125	0.278	1.454116752	0.003601462	0.5	1	5	0.1	0.05	0.3	1

Figure 12: Stream Segment features for the generated stream network saved in csv file

APPENDIX

Description of the Updated Plugin

The plugin generates a vector stream layer containing the following stream features for each segment – Stream Order, Stream Length (in meter), Depth (in meter), Top Width(in meter), Bottom Width (in meters), Watershed Area (in meter^2), Mean slope of the stream segment and Discharge (meter^3/second).
The plugin works in the following steps:

Inputs:
It requires the following inputs from the user –
1) a DEM file (to generate stream and extract the stream features from it)
2) output file location to save results
3) a CSV file with runoff information for the region in different timestamps
4) a CSV file to give multiple coordinates (in same CRS) as input.
5) Single coordinate: It allows to select a coordinate on the DEM.
*(**Note**: One of the above two input is allowed in one execution.)*
6) Threshold for Drainage (default: 500): This threshold for GRASS stream network and drainage direction layer (*grass: r.watershed*).
7) Threshold for Stream (default: 5): This threshold is for SAGA stream network (*saga: channelnetworkanddrainagebasins*).
8) Max Width (default: 1 pixel): to find depth on each point on the stream, check on both sides of the stream point for elevation difference. Max Width defines the range on both sides of the stream point to check elevation difference. (Ex: for Max Width = 1, check elevation difference between the stream point and 1 pixel on both sides of the stream point)
9) Min Length (default: 200 m): remove first order streams of length < Min Length.
10) different parameter values:
a) Friction Deep Aquifer (default: 0.5)
b) Hydraulic Conductivity (default: 5)
c) Evaporation Coefficient (default: 0.1)
d) Mannigs (default: 0.05)
e) Bank Flow Recession (Default: 0.3)
f) Potential Evaporation (Default: 1)
g) Side Slope (default: 1)

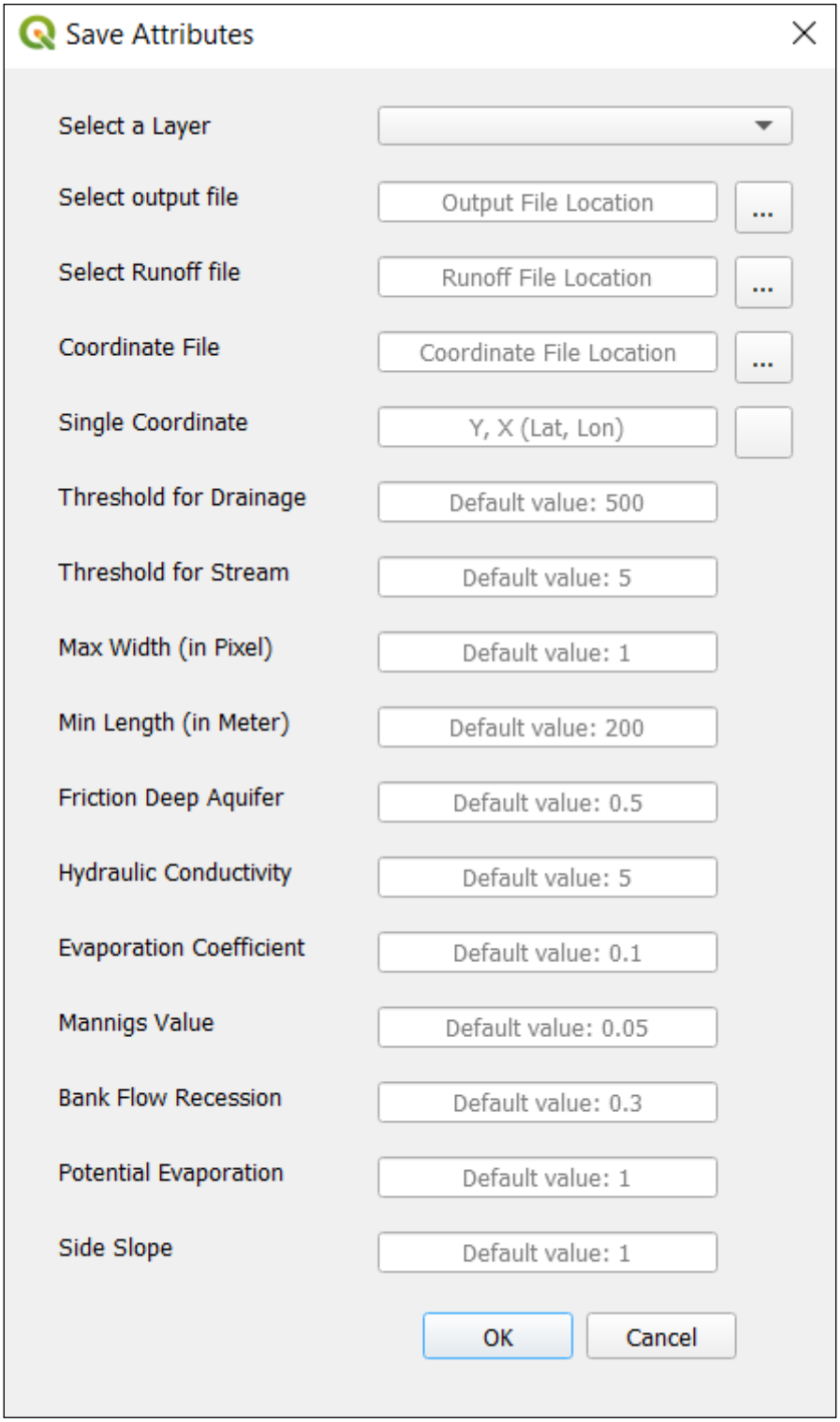


fig 1: Plugin dialog Box

Checking Input Parameters and Creating Main Output Directory:

- First, all inputs are stored in local variables and all mandatory input variables (DEM, output location, coordinate and runoff CSV) are checked for validity. In case any one of these parameters are missing or invalid, the execution is ended and an error message is shown.
- Next the input coordinates are converted into desired tuple format using the function "getInputCoordinates" and stored in a list named "input_coordinate_list".
- Once the inputs are found in order, a directory is created in the given output location and output directory name (say, *"output_dir"*).
- Inside this main output directory results for each input coordinate (different watersheds) are saved in directories with name *"output_dir-n"* where n is the coordinate serial. These directories are created later during the iterative process of generating results for each input coordinate.
- Before creating these directories, another directory is created inside this main output folder named *"_temp_"* to store the initial layers that are common for all input coordinate (like - FillDEM.tif, DrainageDirection.tif)

Generating Initial Layers that are Common for all Watershed:

- A new DEM file (fillDEM.sdat) is generated from the old one after removing all sinks. The *'saga:fillsinksqmofesp'* function is used here. This step is important for generating stream network.
- Next, using *'grass7:r.watershed'* function Drainage Direction raster (DrainageDirection.tif) and Stream Network raster file (GrassStream-Raster.tif) are generated. The Drainage Direction file is required in order to generate the watershed basin.
  Note: *Here the input "Threshold for Drainage" is used as threshold.*
- The stream raster generated in the previous step is converted into vector layer (GrassStreamVector.shp) using *"grass7:r.to.vect"*.

Projecting the Input Coordinates on the GRASS Stream Network:

- In most cases the input coordinates are not on the stream network. Using these coordinates will generate wrong watershed.
- To resolve this each input coordinate is replaced by the coordinate of its closest point on the grass stream vector.
- The function "snap" is used for this.
- This function stores the new coordinates in string format in a list named "finalCoordinateList" and returns it.
- The coordinates are converted into string format because in next step the coordinates need to be in this format.

The Iterative Process of Stream Features Extraction for Watersheds Generated from Different Coordinates:

In this step watershed is generated for each coordinate generated in previous step, stream is generated for each watershed and its features are extracted. The following steps are repeated in each iteration for different coordinates.

Generate Result Specific Output Folder:

- First, a sub-folder is created inside the main output directory with name *"output_dir-n"* where n is the coordinate serial.
- Inside this folder results are saved for the corresponding coordinate.
- Also, a "_temp_" folder is created inside this folder to save all layers that are used during the computation.
- The function "makeDirectory" generates all this result specific directories (also set result file names) in each iteration.

Generating Necessary Layers:

- Watershed Basin raster (SourceBasin.tif) is generated for one the coordinate using *'grass7:r.water.outlet'* function. Stream network will be generated only for this watershed region (it sets the boundary of the stream network).
- Next, the basin raster is converted into vector (BasinArea.shp) using *'grass7:r.water.outlet'*. This is to check the area of the watershed. If very small watershed is generated, then the process stops here because for very small watershed no stream network can be generated. A message box is displayed ("invalid point selected") in this case. The threshold area is currently set to zero, it should be changed according to requirements.
- In order to generate stream network only for the watershed area it is required to clip the DEM (fillDEM.sdat) by the basin area (BasinArea.shp). For this purpose *'gdal:cliprasterbymasklayer'* function is used.
- Finally, using *'saga:channelnetworkanddrainagebasins'* function stream network layer (*"Stream-n.shp"* where n is the serial number) is generated. This function also generates a point vector layer (JunctionPoints.shp) containing the starting points, end points and junction points. These junction points are useful for next steps.

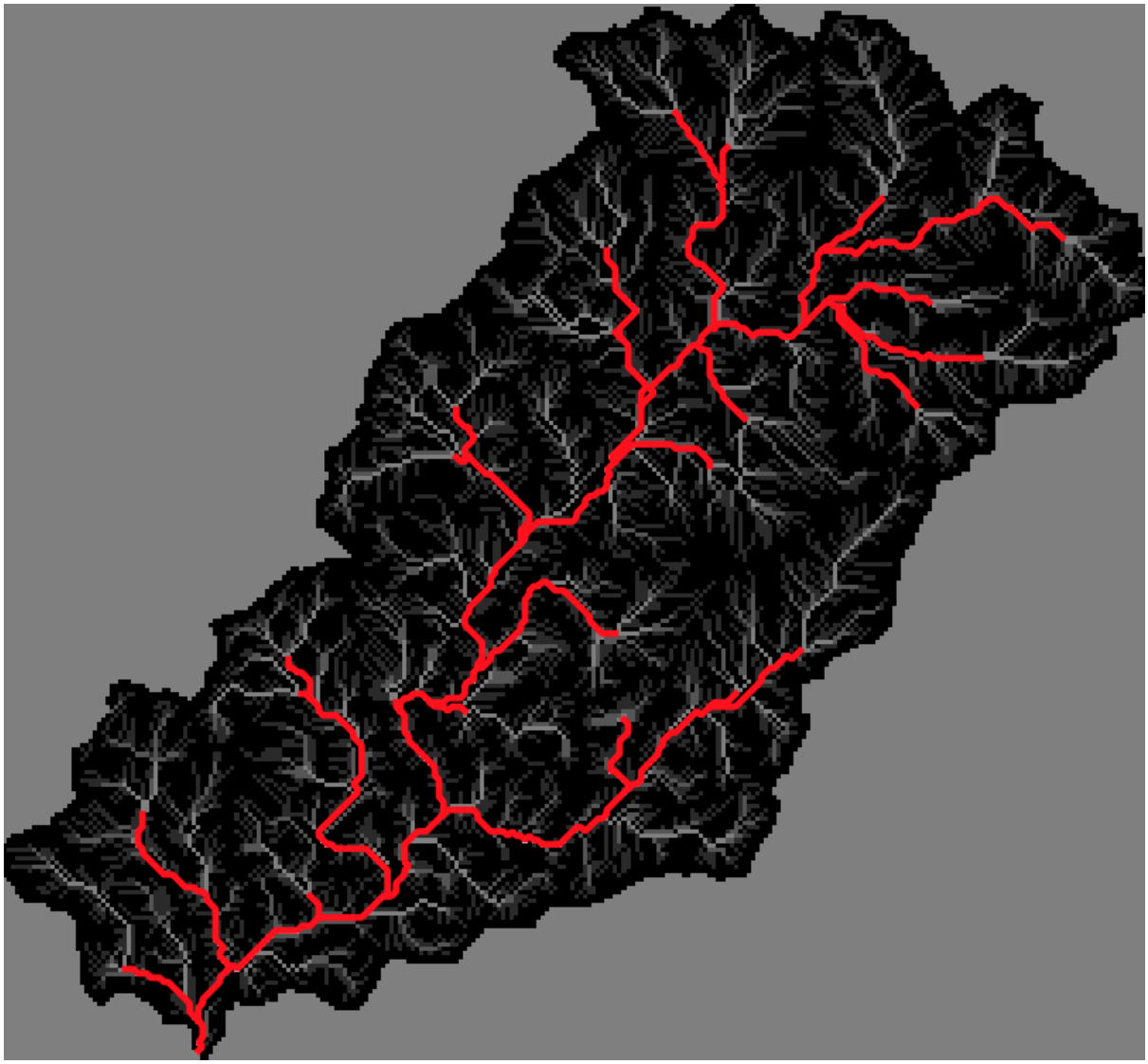
a) Input DEM Raster layer

b) Watershed basin is generated for the given point

c) Stream network is generated from clipped Raster layer

Figure : Generating stream network from DEM raster layer

Note: *Here the input "Threshold for Stream" is used as threshold. It actually defines the minimum segment order in the stream network. This process ('saga:channelnetworkanddrainagebasins') actually generates a stream order raster first which includes even finer segments. The threshold filters from these segments by stream order i.e. segments with stream order >= threshold are allowed and rest are removed. Finally, in the resultant stream network ("Stream-n.shp") the stream orders are decreased by (threshold-1) so that the stream order is maintained. The following image shows this filtering process. `*



In the stream order raster, there are segments of order 1-7. The threshold used here is 5. So, first segments of order 1-4 are removed keeping segments of order – 5, 6 and 7. Next the minimum segment order in the stream is set to 1 i.e. 5 becomes 1. Similarly, 6 becomes 2 and 7 becomes

3. Thus, the segment orders of the generated stream network are changed to keep the segment order integrity. This change can be observed in the image of attribute table of the stream below (column: ORDER_CELL and ORDER).

Channels :: Features Total: 47, Filtered: 47, Selected: 0

	SEGMENT_ID	NODE_A	NODE_B	BASIN	ORDER	ORDER_CELL	LENGTH
1	1	2	1	1	3	7	108.2106781200
2	2	3	2	1	1	5	289.2766953000
3	3	4	2	1	3	7	186.2436867100
4	4	5	4	1	3	7	334.9873734200
5	5	6	5	1	1	5	72.8553390590
6	6	7	5	1	3	7	225.8883476500
7	7	8	4	1	1	5	610.8757210600
8	8	9	7	1	3	7	337.1320343600
9	9	10	9	1	2	6	697.1194077700
10	10	11	10	1	1	5	253.9213562400
11	11	13	9	1	3	7	351.7766953000
12	12	12	14	1	1	5	120.7106781200
13	13	15	10	1	2	6	375.5203820000
14	14	16	13	1	1	5	30.1776695300
15	15	14	13	1	3	7	133.2106781200
16	16	18	7	1	2	6	862.6524163600
17	17	17	18	1	1	5	30.1776695300
18	18	19	15	1	1	5	90.5330085890
19	19	20	14	1	3	7	208.2106781200
20	20	21	18	1	1	5	153.0330085900
21	21	22	15	1	1	5	347.4873734200
22	22	23	20	1	1	5	631.5863991800

Editing the Attribute Table:

- The 1st order streams with length less than input value "Min Length" are deleted from the attribute table of the newly generated stream network.
- After this it is required to change the "SEGMANT_ID" field values as it may not be continuous after deleting the segments.
- We serially assign "SEGMENT_ID" values to each segment.
- Next, the unnecessary fields ("BASIN" and "ORDER_CELL") in the attribute table are deleted.
- After deleting some of the segments, some segments will be there that should be merged. This will give better results. Currently, this portion is not implemented but the code for determining the pairs to be merged is there in comment. Only the merging is to be left to be done.
- Also, in this phase "stream_network" objects are initialized.
- A segment is invalid if it has order > 2 and it has length <= pixelsize*(√2).
- Because in such cases the segment would have 0 or 1 stream point and with one stream point its depth and slope cannot be computed.
- So, these segments cannot be processed. It's better to fully ignore these segments.
- To mark such segments the valid attribute of stream_network object is set to False, making them invalid.
- Latter, we need to make connection between the stream segments on both ends of such invalid segments by transferring its sources to its child segment.

Generating points along the stream segments:

- In order to compute the depth of the stream we generate points (vector point layer: StreamPoints.shp) along the stream lines and measure the depth across all these points and take the mean. The function *'qgis:generatepointspixelcentroidsalongline'* is used for this purpose.
- The layer generated in the previous step may contain some duplicate points (points on same coordinate). To remove such points *'native:delete-duplicategeometries'* function is used that produced a new vector point layer (CleanStreamPoints.shp).
- **Some important properties of this layer:**
    - It has fields – "id", "line_id" and "point_id".
    - The "id" field stores unique ids for all points in the layer
    - The "lind_id" field stores the ("SEGMENT_ID"-1) of its source segment. So, by filtering the points by "line_id" we can select the

points along a specific segment. In next stage this method will be utilized for depth and slope computation.

- The "point_id" field is the unique ids for points inside a segment (i.e. for "line_id"s)
- The "point_id" values generally starts from the end point of a segment (destination point where it meets other segments) and keep increasing along the segment line and ends at the starting point of the segment (i.e. source point).
- So, the end point has smallest "point_id" value and starting point has largest "point_id" value.
- In such cases the points can be accessed sequentially along the segment line. Which helps to iterate over these points in a sequential manner.
- But there are 2 problems – first, in very few cases the points are not ordered in this way (ex: point_id 92 can come before point_id 90) and second iterating over the points along a segments the points are not always accessed by their "point_id" values (i.e. point_id 5 can be accessed before point_id 1).
- But These problems don't occur too often. So, they can be ignored.
- Another problem are the junction points. It may have any of the three "line_id" values (parent1, parent2 and child). Hence it can pa part of any of the three segments.
- Also, these allocations are not same always i.e. for the same stream network it may produce different results in different execution. In three execution the junction point can be allocated in three different segments.

Adding New Attributes to the Stream Network vector layer:

Following attributes are added to the stream vector layer to save data for each stream segment –

a) "DEPTH" (type: double) – to save depth value
b) "BASIN AREA" (type: double) – to save watershed area value
c) "SLOPE" (type: double) – to save slope value
d) "B_WIDTH" (type: double) – to save bottom width value
e) "T_WIDTH" (type: double) – to save top width value
f) "SOURCES" (type: text) – to save the source "SEGMENT ID"s in a list
g) "Q" (type: double) – to save discharge value

Computing depth, slope, basin area for each segment:
The getFeatures() is used to compute these values. As mentioned earlier, the points along a segment can be selected by filtering it by "line_id". This segment-wise selected points are passed as parameter to the getFeatures().
a) Finding Depth – To find the mean depth for each segments the following algorithm is used.

- First, the points along the stream lines are selected segment-wise and for each segment the function is called. In case the segment has two or less points on it its depth is considered 0.5.
- For each point along a segment first the alignment is determined by comparing its coordinate with the coordinates of the previous point. This will determine whether the segment is vertical or horizontal or 45º angle or 135º angle.
  Note: *for this method the stream points need to be accessed sequentially along the segment line. But as mentioned earlier it may not happen always. Failing this criterion will give wrong cross-section line. But it shouldn't affect the results much.*
  Note2: *We cannot determine the cross-section line type at the first point as it has no previous point. So, we have to ignore the first point for each segment.*
- Based on this the type of cross-section line need to be determined. It should be perpendicular to the segment line determined in the previous step.
- Finally, along this cross-section line elevation is checked on both sides of the stream point for Max Width number of pixels (these elevation data are stored in lists – "height_a" and "height_b").
- For each sides of the stream point the depth is computed as –
  depth_a = max(height_a) - baseHeight
  depth_b = max(height_b) – baseHeight
  where "baseHeight" is the elevation at the stream point.
- For the depth at the stream point the following rule is followed-
  depth = min(depth_a, depth_b) if, depth_a ≠ 0 and depth_b ≠ 0
  depth = max(depth_a, depth_b) else
- Finally, the depth for the segment is computed by taking the mean of the depths on each stream point along the segment.
- In case there is not enough stream points (i.e. the segment is too short and the mean depth is 0) depth value 0.5 is returned.

b) Finding Slope – To measure the slope for each segment the elevations on two end points are checked. The slope is calculated by the formula –

$$Slope = \frac{|Elevation\ Difference\ between\ 'firstPoint'\ \&\ 'lastPoint'|}{length\ of\ the\ stream}$$

- As mentioned earlier the points may not have sequential "point_id" values. In such cases the best way to determine the end points is by checking the distance of each point from the junction point or end points.
- Each segment has "end_node" and "source_node" which can be a junction point. The coordinates of these points are stored in "end_node_coord" and "source_node_coord" variables.
- In case these points belong to the stream points set for a segment we ignore those points i.e. junction points are ignored (because these points can be part of different segments – parent1, parent2 or child in different execution, giving different slope results in every execution). So, to maintain consistent result better to ignore junction points.
- To do this we compare the coordinate of each point with "end_node_coord" and "source_node_coord" if it matches then ignore.
- By computing the distance between each point coordinate and "end_node_coord" and "source_node_coord" we find the points that are closest to "end_node_coord" and "source_node_coord". These two points are the "lastPoint" and "firstPoint" accordingly.
- Elevation difference between these two points are used in slope computation.
- In case the elevation difference is 0, slope value 0.001 is used.

Note: *the current formula for slope computation is completely dependent on the elevation values at the 'firstPoint' and 'lastPoint' which doesn't depict the elevation change along the segment line. It is possible that the elevation difference between these points is 0 but the elevation has changed in intermediate points. One solution can be considering the elevation deference between the highest point and the lowest point of on the segment. But in this case also the elevation difference along the stream is not depicted. We need a better formula for slope calculation that depicts the change in elevation along the stream segment by taking into account elevation change at each point instead of considering the elevation difference between only two points*.

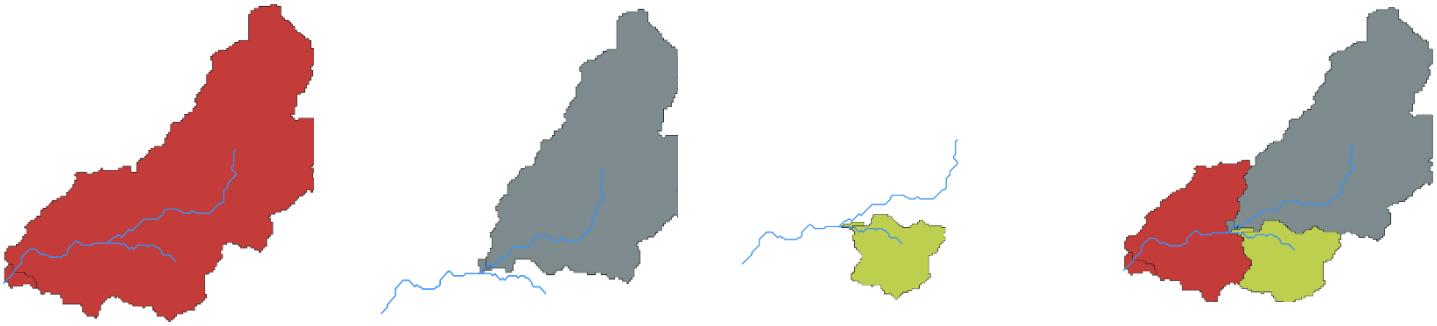
c) Finding Basin Area – The following steps are followed compute the basin area.

- The most challenging part here is to find a proper point on which the watershed area can be computed.
- Our initial assumption was that the closest point to the junction point i.e. 'lastPoint', mentioned earlier can serve as the required point.

- But we observer that it doesn't always cover the entire segment and its resources (which was expected). ***[*Why does it happen mentioned later]***
- First, we compute the watershed ("RBasin[*SEGMENT ID*].tif") w.r.t the 'lastPoint' using *'grass7:r.water.outlet'* function and check if it is covering the entire stream segment or not.
- For this, we check if the coordinate of each point along the segment belongs inside the watershed area computed earlier.
- In case some points are outside the watershed area, the first such point is considered to be the source of secondary watershed ("secRBasin-[*SEGMENT ID*].tif) i.e. w.r.t this point we compute the water shed again.
- Finally, these raster watershed areas are converted into vector layers ("VBasin[*SEGMENT ID*].shp" and "secVBasin[*SEGMENT ID*] .shp") using *'grass7:r.to.vect'* function.
- We take the union of these tow vector layer to compute the final watershed area (Basin[*SEGMENT ID*].shp) using *'native:union'* function. The area of this final vector layer is the watershed area value for a stream segment.

Find Sources and Differential Watershed Area:

- A stream segment is called source of another stream segment if its end point ("NODE_B") is the starting point ("NODE_A") of the other.
- The stream segment sources are saved in "sources" attribute of each stream_network object.



a) Watershed basin for order 2 stream segment

b) Watershed basin for (1st source) order1 stream segment

c) Watershed basin for (2nd source) order1 stream segment

d) Subtracting basin area of source streams from order 2 stream network

Figure: Subtracting basin areas of source streams (order 1) from basin area of higher order (order2) stream

- In the previous step we computed the basin area for each segment but in that basin area a stream segment with higher order (>1) also included the basin area of its sources. But to compute discharge of each segment separately it is required to subtract the basin areas of source segment from the its corresponding higher order child stream segment basin area.

- This is implemented simply by subtracting basin areas of source segments (from source list) from each segment (***Figure***).
- If a segment has a invalid source segment (length < pixelSize*√2) then it is neither added in the source list of its child segment nor considered while competing differential watershed area for the child segment, instead such parent child segment pairs are stored in a dictionary named "transferSources" where, the source segment ids are used as keys to save the child segment ids.
- This dictionary is used later to transfer sources from the source segment to the child segment.
- Moreover, after computing the differential watershed area if it comes negative, the segment is marked as invalid (valid=false) and not considered for further computation.
- It happens when the watershed area of the segment covers that segment but not its parent segments. In such cases the watershed area becomes < the sum of watershed areas of the parent segments. Hence, the differential watershed area become negative. ***[Why mentioned later]***
- We also need to remove these segments from the source list of its child segment (don't need to transfer its sources).
- We save such segments in a list named "removeSource".

Transferring Sources from Invalid Segments to Its Child Segment:

- There are two types of invalid segments as mentioned earlier –
    - One with length < pixelSize*√2. Such segments and their child are stored in list named "transferSources". So, we can simply get the source list of the invalid segment and append them in the source list of the child segment and change the differential watershed area accordingly (subtract the watershed areas of newly added source segments).
    - The other one has negative differential watershed area. These are stores in list named "removeSource". We don't have ids of the child segments of these segments. So, we have to check the source list of each segment and remove these invalid segments from the source list. (No need to transfer sources) ***[Why mentioned later]***

Finding Peak Runoff from the input csv file:

The getPeakRunoff() is used to get the peak runoff from the input csv file. This function simply computes the sum of primary runoff and secondary runoff (column 9 and 12 in input csv file) and take the maximum value of it. In case there is some problem in peak runoff computation the execution ends.

Computing Discharge for each stream segment:

- The getQValue() is a simple recursive function that computes the discharge for each stream segment.
- To compute the discharge value for a segment first we need to convert the area into hectare and then use the following equation

$$Discharge = \frac{Peak\ Runoff * Basin\ Area * 10}{3600}$$

- Also, for each segment we need to add the discharge values of its source segments to compute the actual discharge.
- These values are saved into a list named 'q_valie' attribute of the stream_network object.

Computing Top Width and Bottom Width:

- The function computeFeatures() computes the bottom width and top width using the Newton Raphson method and the hydrology formula mentioned in the earlier report.
- If the bottom width coms negative, we use the following rule –

  Bottom Width = order+(order-1)
- Initially we used the *sympy* package to solve such equation but it used to take long time. Hence, we switched to Newton Raphson method.

Adding attribute values into the stream vector layer:

- Once all stream feature values are computed, we add these values into the feature table of the stream vector layer.
- The attribute values are stored into the table in the following format-
  - LENGTH (meter)
  - DEPTH (meter)
  - BASIN AREA (meter^2)
  - SLOPE (no unit)
  - B_WIDTH (meter)
  - T_WIDTH (meter)
  - Q (meter^3/second)
  - SOURCES (list of "SEGMENT_ID")

Creating Stream Features csv file:

- While adding the attribute values into the vector layer, we also save the feature values for each segment into an ordered dictionary and keep them in a list named 'streamFeaturesList'.

- The constant parameters taken as input from the user are also included into the dictionary.
- Finally, a csv file is created with the name "[*input filename*]_Stream Features.csv" in which the stream feature values are written from the list of ordered dictionary.
- Here the value of 'watershed_area' is in hectare unit and 'length' is in km. This done to make these values ready for next step where these are need to be used in these specific units.

Creating Drainage object:
The models.py file defines Drainage class structure representing a surface drainage-network. This module also contains a function 'run_stream_model_for_time_step()' to compute the dynamin features of the stream segments. It computes the following features for each stream segment in every time stamp - swat runoff, total volume stored, volume in, discharge, Transmission loss, bank in, return flow from bank, evaporation loss, total loss, Volume after loss, volume out, volume stored end timestep, cross section, Depth water level, width water level, wetted perimeter, hydraulic radius, velocity, travel time, fraction time step and storage coefficient.

In order to compute these dynamic features using the function we create an object of the Drainage class and assign the stream feature values properly. We use the values from 'streamFeaturesList' for this assignment.

Extracting the Runoff values from input file:
Next, we create a list containing runoff values at different time stamp from the input runoff files by taking the sum of primary and secondary runoffs (column 9 and 12). These values are used in the next step to compute the dynamic features of the stream segments for each time stamp.

Computing Dynamic stream features:
In this step we use the runoff values and stream feature values as input to the 'run_stream_model_for_time_step()' function that computes the dynamic features of the stream segments and store them in Transient class attributes.

Creating csv file with time stamp-wise dynamic features:
Finally, the stream dynamic features are saved into the csv file named "[*input filename*]_Stream Flow Data.csv". The features are written in segment id order i.e. first the features for segment id 1 are written for each time stamp and then segment id 2 and so on.

Output:
(considering input filename is test)
- test_Stream.shp (stream vector layer containing the stream characteristics)

- test_Stream Features.csv (constant stream characteristics in csv file)
- test_Stream Flow Data.csv (dynamic stream features for each time stamp in csv file)
- Also, a "_temp_" file is created that contains all layers used in this process.

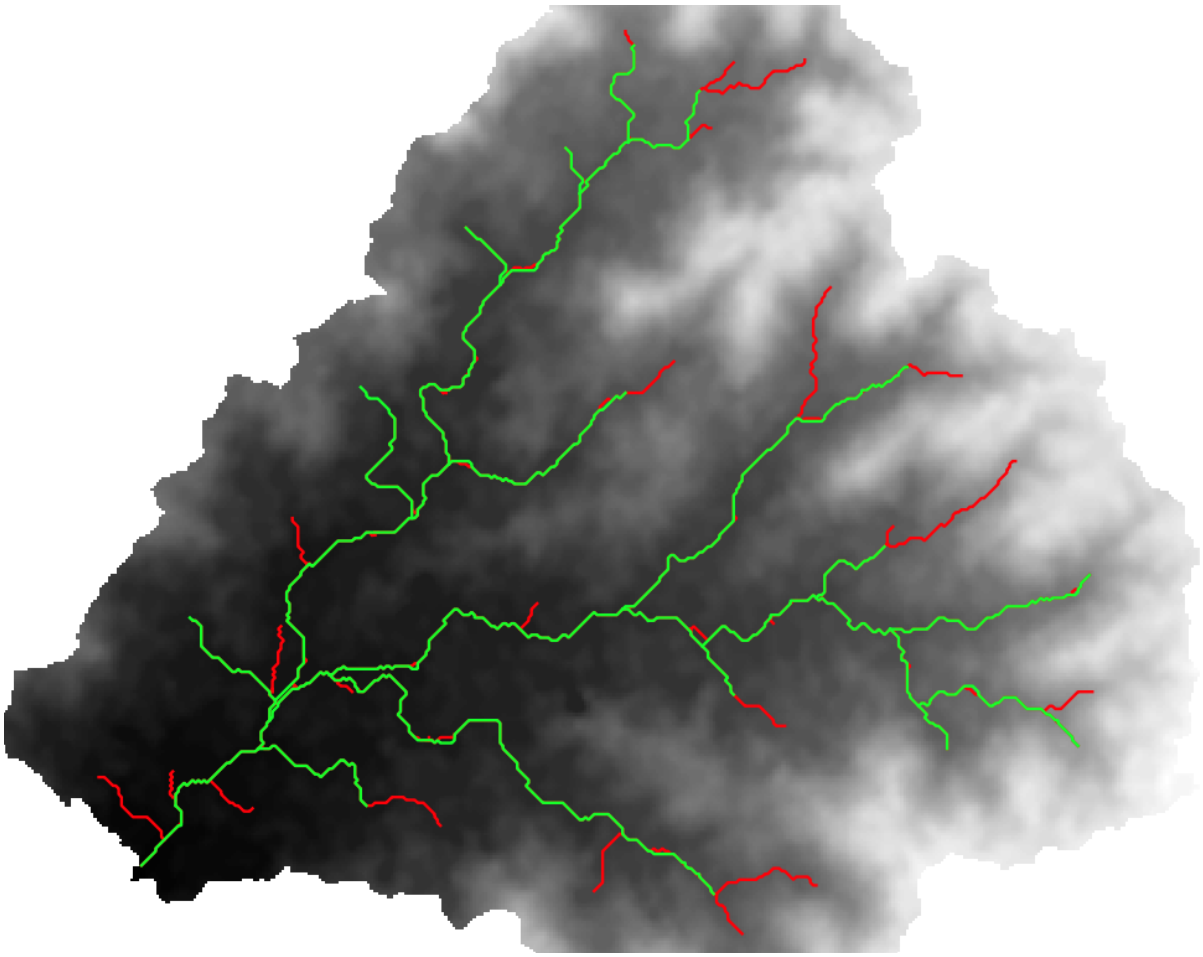
Why Primary watershed area sometimes fails to cover the entire segment or the parent segments?
Why sometimes differential watershed area is coming negative?

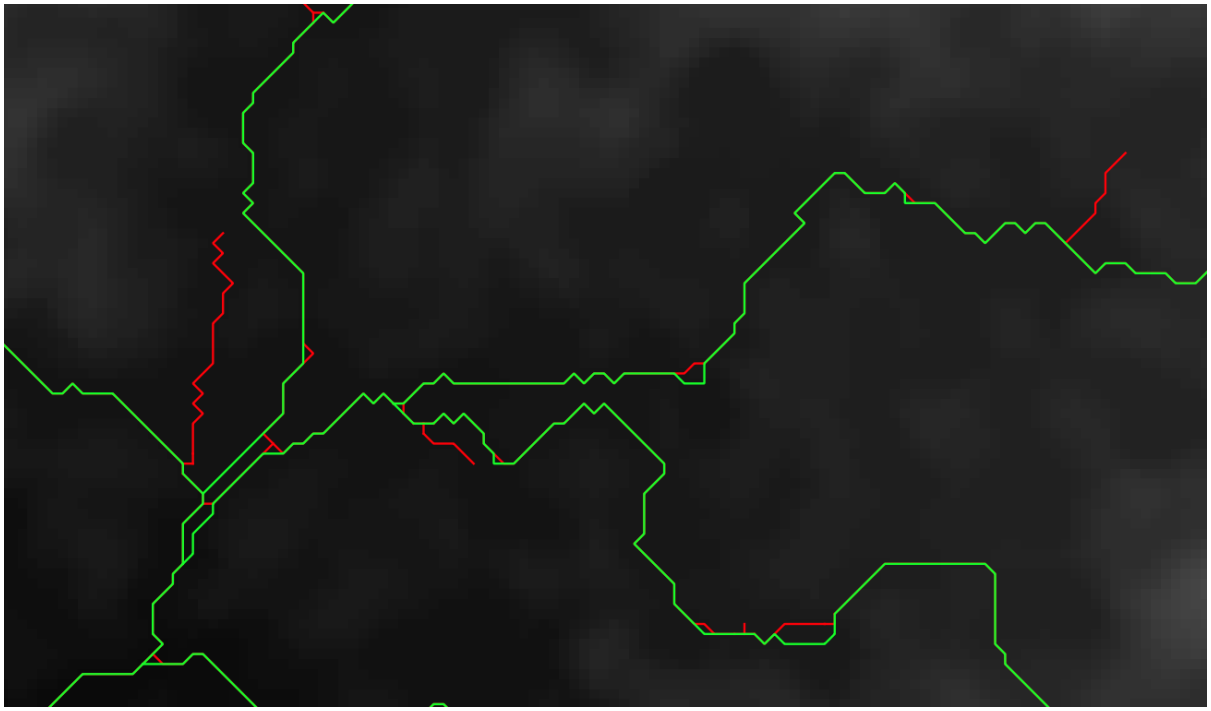
There are two different types of tool that have been used in this plugin –

- GRASS tool – to compute watershed raster
- SAGA tool – to generate main stream network.

Both these tools use different algorithms to compute the stream network. So, it is quite possible that the stream generated by these two methods are not exactly same. This is the actual reason why these errors occur.

Now the stream network we see as output is generated by SAGA. But the Drainage Direction layer used to generate the watershed raster layers are generated by GRASS and it visualizes the steam network in a different way. To observe this difference in visualizing the stream network I have computed the stream network using both these methods. These streams are shown in the image below. The red stream in generated by GRASS and the green is by SAGA.





The above image shows how the streams can differ in some regions. We compute the watershed raster layer according to the GRASS (red) stream but we want the watershed raster layer to be according to the SAGA (green) stream. This is the reason why the input coordinates are replace by the closest point on GRASS stream network as the watersheds are generated according to this stream not SAGA generated stream.

Now the question comes why two different tools has been used?

I have tried 3 different GRASS methods and 2 different SAGA (*Channel Network and Drainage Basins*) methods to generate the streams but only one method were able to give what we required – stream order and more importantly a way for parent-child linking in terms of "NOAD_A" and "NODE_B" which is required for differential watershed computation. No grass tool that I have explored were able to provide this.

Also, I have tried to perform the watershed computation using SAGA tool but could find any method the meet our requirement. There is a SAGA method (*Channel Network)* that can generate differential watershed fir all the stream segments. But the stream network it generates is different from the stream network we are currently using also the parent-child linking is not possible there.

Hence, I have to use two different methods. It is quite possible that there are some methods (in SAGA or GRASS) that meets our all requirements and give a consistence result. I might haven't been able to explore such method.

As future work this can be a good challenge to resolve this issue.

Future Work:

- Find a method that meets all requirements and resolves the watershed computation problem
- Merging of stream segments after deleting 1st order streams with length < Min Length.
- Finding a better slope computing method.
- Improving depth computing algorithm.
- If multiple coordinates are given as input and one coordinate belongs to a
- Currently, for multiple input coordinates the entire process in repeated. Some computation time can be saved if the common segments in different output stream network can be recognized and computed only once.
- Splitting a stream on a given point (where well or other water resource might exist) and computing features w.r.t these points.