In-depth explanation the grid creation

This document explains in more detail how the function create_grid() and create_grid_one_profile functions.

The goal of theses two functions is to go from data in 1.5D in a profile form to a 2D grid. There are two main way of doing this: a) Through the usage of the triangle module b) through the definition of a middle profile used as a guide to create the grid . Method a) is implemented in create_grid() and method b) is implemented in create_grid_only_one_profile().

Walk-through for create_grid()

The number of extra profile is the number of profile added by the function between the original profile of the hydrological model. We need at least one.

We can use this function in two modes. First, this function can create a grid for the whole profile, regardless on the wet and dry area of the profiles. It can also create a grid only for the sections of the profile which are wet, using the data from the vh_pro variable. In this last case, we will need to cut the profile to extract only the wet area. In addition, we do not want to use only the points defining the form of the profile as guiding point for the grid. There are two reasons for this: a) HEC-RAS do not give the velocity only on the points defining the profile. b) In other 1D model, there are many cases where the velocity distribution should be composed of more points than only the ones defining the profile. See Figure 1.

Update coord_pro

This is done by the function update_coord_pro_with_vh_pro(). We have the distance along the profile of all point of interest, but we only have the coordinate of the points defining the profile. This function gets the coordinates of all points the following way, profile by profile:

First, in case that the coordinate system is not meter, we calculate the variable coord_change to pass from the distance in coordinate system to meter. We simply compare the distance between the first and last point of the profile with the total distance of the profile given in coord_pro_p[3][-1].

Then we take each point on which we know the distance. We find the point on which the profile is defined which is the closest before this this point. As the points on which is defined the profile are ordered, we can use

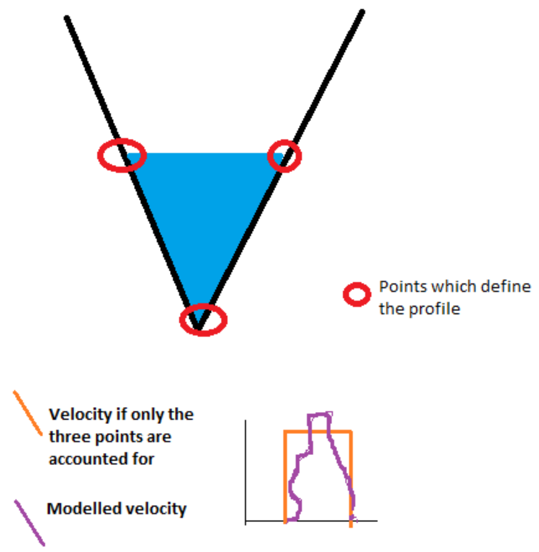
Points which define the profile
Velocity if only the three points are accounted for
Modelled velocity

Figure 1

bisect for this.

The coordinates of the under interest point is the vector sum of the point found with bisect and the difference of distance between the two point in the direction of the profile in the coordinate system of the (x,y) coordinate. The profiles are not always straight so the direction of the profile should be recalculated each time that the point found by bisect change. See Figure 2. It is not good for this function if you have two identical points in the profile so in this case it would one point of the point. However, the identical point should have been corrected before in the function related to the loading of the hydrological data.

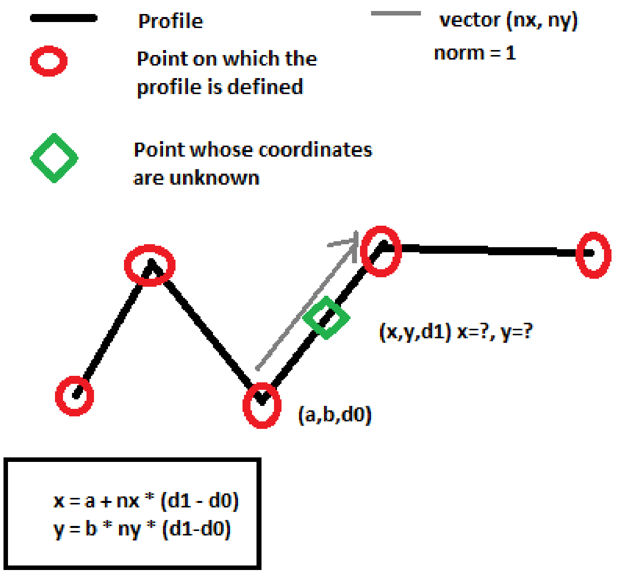

Figure 2

Create extra profiles

Now that we have the coordinates of the points, we create the extra_profile and all known point to vector (point_all) which will be used by triangle. These "extra" profile are not added between the reaches. So if the river has more than one reach, we do not add profile at the junction between the reaches. In general, a separate grid is created for each reach.

For each profile in a reach, we first add the points which are already known. We do not forget to take not of the indices of the start and end of each of the "known" profile. We have three lists of indices:

- ind_s: indices of the start of the known profile
- ind_e: indices of the end of the known profile
- ind_p the indices of the start of each profile (known and extra profiles)

To find the coordinates of the extra profile, we have two cases: a) we create straight extra profile b) we create extra profile in the similar form to the nearest profile. Both cases are implemented, but we use only the case b). To switch to case a), one should change the boolean all_straight to True. We will now explain case b) as this is the one used here. Case b) is implanted in the function find_profile_between().

Find the profile between the main known profiles

To find the profile between the different profiles, we create first a line which is a straight line between the first and the last point of the profile situated before the extra-profile. This line is useful because the profiles are not always straight, so it gives a general direction.

We then project all points on this line. This is to be able to order the points when needed.

We then take each point and create a line passing by this point and perpendicular to this point. The length of this line is given by the 'far" variable. It should be long enough but the variable "far" should be small enough to avoid problem with machine precision.

Now, we have a perpendicular line passing by one point. We find the intersection of this line with the other profileIf the point is on the profile before, the intersection will be on the profile after (in the direction of the river flow) and inversely. If there are more than one intersection possible, we take the first intersection. The method is similar is the two cases. We take each part of the profile on which we looks for an intersection (profile might not be straight). To see if there is an intersection, we use a classical algorithm based on the cross-product. It is explained for example in:

http://stackoverflow.com/questions/563198/how-do-you-detect-where-two-line-segments-intersect

Or any good book about computational geometry. An issue is about what to do if the segments[1] are collinear or if the intersection is at the exact end of a segment. The function has a switch to change this behavior if needed. Currently, the function gives an intersection for these two special cases.

If not intersection is found, we create a line as a start and the end of the profile, using the direction of the last and first segment to see if we could find an intersection there. If we do not find an intersection, we note that we could not find an intersection for this point.

To create each profile, we now take each point and its intersection with the profile and we find the point between. If we only have one profile, this is the middle point. Otherwise the position of this point is given by the parameter div and div2.

We could stop here as this list of newly created point forms the added extra profile. However, nothing says that the limit of the profile is entirely coherent with the length of the basic profiles. If we want insure this (which is important for the triangle module) , we need to trim the extra-profiles using the trim switch. For this, we create a line between the first point of both “known”profile and a second line between the last points of both ‘known” profile. The direction of the profile does not matter as long as it does not change in one hydrological model. Then, we check if there is an intersection between these two lines and the extra profile. If yes, we take out the point situated before or after these lines.

Manage islands

If we want to create a grid which reflects the wet/dry part of each profile, we need to account for the case where only part of the profile is dry. In other word, we need to account for the case where we have an island on the river. If we do a grid on the whole profile, this is not important.

As we only have info on the dry zone on the profile, we have to create a 2D zone representing the island based on the 1D information of the profile.

First we define on which extra-profile will end and start the island (indicated by the red circle in the next figure). As a summary, we take the extra-profile in the middle of both basic profiles, account for the fact that we can have an even or uneven number of profiles.

Then, for each profile, we check if there is an island, in other words, if there is a point where water height is zero, apart from the first and last water height data of the profile.

[1] Here I talk about line and segments without making a lot of difference. This is because “line” are not infinite in this function, their length is defined by the “far” variable. So, for the computer, they are all segments even if the length of some segments does not matter and could be conceptualized as line in our mind.

If yes, we check if we have consecutive points in the dry part of the profile. We are only interested by the first and the last point of each dry section of the profile. All the points in the middle of the dry section can be discarded. If there is a dry part in a profile, we now construct an island. For this we calculate the closest points on the selected extra_profile of the first and last point of the island (the greens point on **Erreur ! Source du renvoi introuvable.**). We usually have closest points, 2 on each profile. However, it is possible that the same point is the closest point of the first and the last points of the dry part of the profile. In this case, the island finishes on a peak, like a triangle.

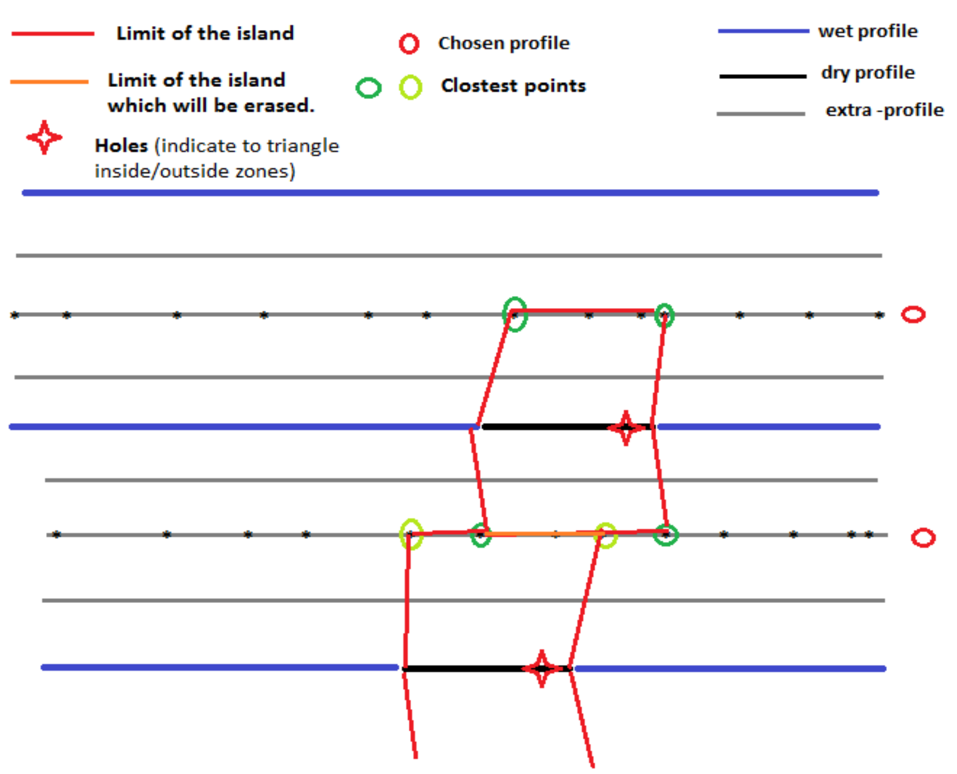
Limit of the island
Limit of the island which will be erased.
Holes (indicate to triangle inside/outside zones)
Chosen profile
Clostest points
wet profile
dry profile
extra -profile

Figure 3

To describe the island we use the list called seg_island. Each segment is described by two items of the list, like the first segment starts on seg_island[0] and ends on seg_island[1]. So this list should be read two by two. They are four columns in this list: the first column is the index of the point. The coordinate of this point is in point_all. So, to get the coordinates of the start point of the first segment, one should write: point_all[seg_island[0][0]]. The second column is on which reach is the island in case we have more than one reach. The third column is an indication if the segment is along the profile or perpendicular to the profile. If it perpendicular to the profile, it is indicated by -99, otherwise, we give the profile number. The last column is the island number, so that we can select all segments of each island. Careful with this column because we fuse together some of the islands afterwards, so this is the island number only when the ‘collinear” switch is False (see below).

The triangle module needs to know where is the inside and outside of the island. For this, it needs a point called hole indicating the island inside. The position of the hole is on the dry part of the file between the first and the last point of the profile. The exact position does not matter, but it is better to put close to one end in case the geometry of the profile is special and island have strange forms. The next step is to control that the hole is really in the island. For this, we use the inside_polygon function. This function is an implementation of the “ray-casting” algorithm (which can be found on internet) which allows finding if a point is in a polygon. If yes, the island is kept. Otherwise, it is discarded. A warning is send as it result in a small error (the wet area is bigger than it should).

Various additional check

We could end here with the islands but we would have a lot of small islands which in reality would form one bigger island. To simplify the form of the grid we will now erase all 'collinear" segment. This is not a very good term because these segments are only collinear if the profiles are straight (see the orange segment on Figure 3). Otherwise, they might cross each other. Crossing segments can be a problem for the triangle module. Triangle can handle segments crossing each other but only in simple, clear cases. It is hard to estimate in advance which kind of crossed segment will results in a failed triangle run. Hence, it is good to get of rid of as many as possible of crossed segments. This is what does this part of the code.

Practically, we select each segment "along" a particular profile. In other words, we select all segments which have the same index of the third column of the variable seg_island. We then sort these segments so that they do not cross each other based on the fact that the points along a profile are ordered (so if you sort them, you get a list of segment on after the other).

We then have snippet of code to check if segment crosses exactly at the middle of another segments. It was useful before but is not use in the current version of the code.

We then have a script to check if segments cross as triangle can have problems with them in some cases. However, it tends to erase too different islands and to send warning which are not necessary. Moreover, it is very slow for complex cases. Hence, it is not used currently.

We then check if there is two identical points. This is somethings which will make triangle to fail in most cases so it should be avoided. If it finds identical points, it sends a warning. To keep this test quick, we use a special routing based on lexsort. Lexsort is a special way of sorting number where , if there are two time the same number, the array is sorted based on secondary info. For example, let's say we want to sort a= [2,3,2,4] where b = [5,2,1,0] is the secondary info. The resulting sort is ind = [2,0,1,3].

Order the information as needed

The next part is to join all the segments (the one which gives the limits of the reaches and the ones giving the limit of the island). To give the segment which gives the limit of the reach, we use the indices collected during the creation of the profile (ind_s and ind_e). As a summary we define the limit of the river by taking the first and last point of each "known" profile. We also add to the list of segment representing the limits two segments along the first and the last profile to close the polygon which represent the reach. Then the segments representing the island of this reach are added. We do the same of each reach. So the limits for each reach is a polygon giving the limit of the reach and a list of polygon representing the islands

Then we have part of the code which is not used. It was written to be able to directly incorporate the limits of the substrate to the grid creation. It avoids to having to cut the substrate in a second step, which is easier and quicker. However, the resulting grid was not so smooth because of the number of constraint on it. So it is not used.

Then, we finally send the information to triangle to create the grid. We do this separately for each reach. There are three lists to be sent as information: a) the coordinate of all points. It can seem quicker to send only the points related to the analyzed reach. However, it would really slow down the search for the point which overlaps between the reach, so it is easier to send all point for all reach. Triangle will stop the grid at the limit regardless of the position of the points. We then send a list of segment representing the limits (reach limit and island). Finally, we send a list of point which define the inside and outside of the reach (the “holes”). There is a long list of option for triangle. In case, triangle fails, it can be useful to try to run it without the ‘p’ switch so without option. It will ignore the islands in this case and create a bigger grid than needed, but it can be useful to see where the problems are.

We then find the area which overlaps between the reaches. It might be important afterwards when we calculate the surface of the river.

Next, we have a part of the code to find the centroid of the grid elements, which can be useful.

The last part is a special form of “return” which is necessary when we use an external process to send create_grid(). It is often done so because triangle can fail in difficult cases.

Create_grid_only_1_profile

This is a similar function than create_grid(), but it uses only one extra profile and directly create the grid instead of calling triangle. Hence, it is more stable, but there is little control on the quality of the grid, For example, if two cells overlaps because of a bug or a complicated geometry, this function will not send a warning which this would not happen with triangle output.

The first part of the function is the same than create grid (see section above about “Update coord_pro). The aim here is to reformat the profile to get the coordinate of all interesting points.

As in create_grid, we create one grid by reach, so there is a “for” loop passing through each reach. Also, in a similar way, we can create a grid for the whole porfile or only for the wet area of the profile. If the wet area of the profile is analyzed, the variable vh_pro is needed.

We then prepare the water height and velocity data. More precisely, we take out all the data where the profile is dry. This will be useful afterwards, when the island will be constructed. In create_grid_only_1_profile(), we do not interpolate velocity and height data, contrarily to create_grid(). So in the former case, we need to update the data to account for this fact.

Afterwards, we add one extra profile in the middle of each couple of profile. We assume that the profile does not cross. We do that using the same function than the function which add the extra profile in create_grid(). See the section about “Find the profile between the main known profiles“.

We then create the grid using the extra profile created before. The general idea is to create two rows of triangle, one before the added profile and one after. The position of the triangle is determined by the position of the velocity and height data. Each triangle has one velocity and height data linked with it. Because of that, there is no need for interpolation of velocity and height data as in create_grid().

To create the grid, we take a vector perpendicular to the profile. If the profile is not straight, we take the vector perpendicular to the first segment of the profile. It could look more logical to take the vector perpendicular to each segment of the profile, but the grid cells might cross on each other in this case. Hence, it is better to use only one direction by profile.

Next, we find the intersections between the line parallel to this perpendicular vector and passing by each point of the profile. The method to find the intersection is similar to the one described in about "Find the profile between the main known profiles". If do not find an intersection for a point, the intersection is found arbitrarily.

Now, we have the point forming the grid and they are in order. So we can create the connectivity table (called ikle here). For each point, we form two triangular cells (Figure 4). The creation on the connectivity table is based on the order of the variable point_all which is: one point of the profile, the point which intersect in the middle profile, the next profile point, … So for the first cell we have the profile point (length of point_all -1), the point of the profile before (length – 3) and the intersection point on the middle profile related to the point before (length -2). For the second cell, we have the intersection point related to this profile (length of point_all), the intersection point related to the point before and the point of the profile.

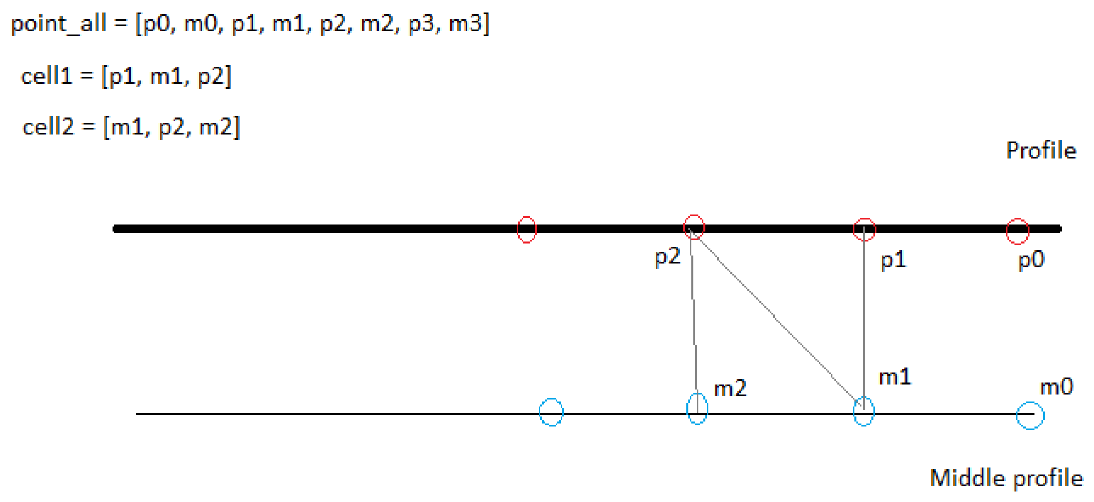

Figure 4

When the grid is created, we fill it with the data. As there is one data point per cell, this can be done directly as long as we are in a part of the profile with water.

The next part of the code is commented and not used, but it can be used to show the profile created (without the full grid) to check if this part was done successfully by HABBY.