DRAINAGE AREA ESTIMATION IN PRACTICE, HOW TO TACKLE ARTEFACTS IN REAL WORLD DATA

Abdulghani, HASAN[1], Petter, PILESJÖ[2], Andreas, PERSSON[2]

[1]Department of Earth and Ecosystem Sciences, Faculty of science, Lund University, Sölvegatan 12, 22362, Lund, Sweden

abdulghani.hasan@nateko.lu.se

[2]GIS Centre, Lund University, Sölvegatan 10, 22362, Lund, Sweden

Abstract

Large improvements in flow estimation have been made the last decades. However, most, if not all, of the proposed algorithms have been developed and tested on mathematical, or manipulated natural, surfaces. There is an urgent need to develop algorithms dealing with natural artefacts in the landscape, like flat areas and depressions (sinks), caused by man, generalisation (of data type), or by errors in e.g. interpolation. The aim of this study is to present and test practical solutions making it possible to estimate drainage area over natural surfaces with a special focus on sinks and flat areas.

Compared to other studies the main contributions of this research are that we have: adapted surface flow routing algorithms over flat areas to multiple flow algorithms, and developed algorithms letting the user decide which sinks to remove, either by beaching or filling. In both cases the user has the possibility to influence the result, by defining breaching points and deciding thresholds regarding area, volume or depth when filling sinks.

In this study algorithms making it possible for the user to fill sinks depending on area, depth or volume have been developed. This increases the possibility to separate natural sinks from ones coursed by data or analysis errors. Also an interactive semi-automatic way of breaching break lines in the terrain is presented. This is needed instead of filling sinks caused by man-made artefacts, like road banks and train lines. Existing culverts are then replaced by used defined breach lines.

Multiple flow functions to route overland flow over flat surfaces, caused by filled sinks or generalisation, are presented. The flat areas are classified as either ‘flow-out’ or ‘flow-in’. Flow-out occurs when one or more cells on the flat area border have an elevation value lower than the flat area cells. A flat area is classified as ‘flow-in’ when all cells on the border of flat area have elevations higher than the flat area cells. These functions, together with the ones for sink removal, are exemplified and tested on real-world data. Results clearly indicate the benefits of the presented algorithms, making it possible to model overland flow and estimate flow accumulation/drainage area continuously in digital elevation models, without imposing vector hydrology covers (streamlines, lakes etc.).

Keywords: drainage area, flow accumulation, DEM, multiple flow distribution, flat areas, sinks

INTRODUCTION

Catchment topography is critical for models of distributed hydrological processes (Quinn, et al. (1991), Seibert, et al. (1997)). Slope controls flow pathways for surface as well as near surface flow, and influences the sub surface flow pattern substantially. The key parameter derived from catchment topography is flow distribution, which tells us how overland flow is distributed over the catchment area.

The fact that flow distribution over a land surface is a crucial parameter in hydrological modelling, in combination with available digital data, has rapidly increased the use of Digital Elevation Models (DEM). Modelling has made it possible to estimate flow on each location over a surface, normally by the use of gridded raster DEMs (see e.g. Zhou, et al. (2011)). Based on the flow distribution estimation on each location represented by a DEM, the drainage pattern over an area, as well as various hydrological parameters such as catchment area and up-stream flow accumulation, can be modelled.

Generally, surface flow and flow accumulation are estimated by the use of two different types of raster algorithms; either approximating to a single, or multiple, flow directions (see e.g. Beven, et al. (1993), Wilson, et al. (2000), Zhou, et al. (2011)). If working with raster data, multiple flow algorithms assume transport to more than one adjacent cell, while a single flow algorithm only distributes water to one neighbour cell at a time in the raster. In many cases, single directional flow algorithms produce satisfying results over concave surfaces, while it is often more appropriate to divide the flow into two or more directions if the surface is flat or has a convex form (Seibert, et al. (2007), Zhou, et al. (2011)). Combinations of the two types are often to prefer when modelling e.g. water flow over natural surfaces (Pilesjö, et al. (1998)).

The single flow algorithm was described by O'Callaghan, et al.(1984). It assumes that flow follows only the steepest downhill slope. Using a raster DEM, implementation of this method resulted in that hydrological flow at a point only follows one of the eight possible directions corresponding to the eight neighbouring grid cells (Band (1986), ESRI (1991), Mark (1984), O'Callaghan, et al. (1984)). However, for the quantitative measurement of the flow distribution, this over-simplified assumption must be considered as illogical and would obviously create significant artefacts in the results, as stated by e.g. Freeman (1991), Holmgren (1994), Pilesjö, et al. (1996), Seibert, et al. (2007), Wolock, et al. (1995), Zhou, et al. (2011).

Attempts to solve the problem connected to the single flow algorithms have led to several proposed ‘multiple flow direction’ algorithms (Freeman (1991), Holmgren (1994), Pilesjö, et al. (1996), Pilesjö, et al. (1998), Quinn, et al. (1995), Seibert, et al. (2007), Zhou, et al. (2011)). Many of these algorithms estimate the flow distribution values proportionally to the slope gradient, or risen slope gradient, in each direction from the centre cell.

Even if large improvements in flow estimation have been made during the last decades, most, if not all, of the proposed algorithms have been developed and tested on mathematical, or manipulated natural, surfaces. There is an urgent need to develop algorithms (or practical solutions) dealing with natural artefacts in the landscape, like flat areas and depressions (sinks), caused by man, generalisation (of data type), or by errors in e.g. interpolation.

Most flow distribution algorithms available imply sink removal before estimating flow distribution from each cell to its surrounding cells. Normally this is done by ‘filling’ the depression. The result in the DEM is that all cells in the former sink get elevation values equal to the lowest cell value of the border cells. This of course creates a flat area. It can be discussed if, when and how sinks will be removed. Depending on origin (natural or a result of errors, generalisation and/or man) some sinks might not be removed. There might also be reasons not to fill a sink to remove it, but rather breach the barrier creating the sink. Additionally, it is not rare that flow routing algorithms are not able to estimate flow over flat areas (see e.g. Costa-Cabral, et al. (1994), Fairfield, et al. (1991), O'Callaghan, et al. (1984), Tarboton (1997), Pilesjö, et al. (1998)).

Sink removal and surface flow routing have been discussed by many researchers during the last decades (Beven, et al. (1979), Jenson, et al. (1988), Kenny, et al. (2008), Lindsay, et al. (2005), Oimoen (2000)). However, these are not adapted to multiple flow algorithms, and/or do not allow the user to interact in the decision if and how to remove a sink. In this research we have combined a well-functioning multiple flow algorithm (see Hasan, et al. (2011)) with modified and developed algorithms for surface flow routing over flat areas. We also discuss and propose interactive solutions for man-made artefact in DEM data.

COMMON ARTEFACTS IN REAL WORLD DIGITAL TOPOGRAPHIC DATA

Below we briefly present and discuss some common topographic artefacts, influencing flow estimation, in real world data.

Depressions/sinks

A sink is an isolated drainage basin, not directly linked to the general drainage pattern where all overland flow ends up the oceans. In a DEM it is characterized by one or more cells surrounded by cells with higher elevation values. Sinks are natural features that frequently occur in the terrain. The water in natural sinks is further transported as subsurface flow until it reaches the ground water table (in oceans, streams. lakes, or as ground water).

Non-natural sinks also often occur in a DEM. These can be a result of data sampling/incomplete data, missing depressions/channels transporting surface flow, and/or inappropriate interpolation algorithms chosen when creating gridded raster DEMs based on point or line data. Also scale/cell size can cause sinks. If the resolution in the DEM is relatively low, small streams and channels, transporting water, can be ‘hidden’. Kenny et al. (2008) highlight the presence of sinks in real world’s data, and also the need for well-functioning algorithms for sink removal.

The logical way of thinking when carrying out hydrological modelling is that sinks should not be removed (by filling or breaching) unless we are sure that they are non-natural artefacts. However, depending on the size of the sink there might be good reasons to assume that also natural sinks are directly linked to the down-hill drainage areas, through e.g. wetlands. If so, also some of the natural sinks should be removed.

Sinks need to be filled in order to estimate flow directions from all cells and connects them into flow distribution and flow accumulation. It should be noted that any unfilled sink will result in stopping flow at that cell. Sometimes this is not desirable.

Based on the discussion about natural and non-natural sinks above we can conclude that there often is a need for sink removal. It would be desirable if the cell removal, additional to other knowledge, could be based on size and form of the sinks. Area, volume as well as depth of a sink might help the user to decide if it should be removed or not (see Figure 1). Also the form of the sink can be a help for the user to decide if it should be removed by filling or breaching /see below).

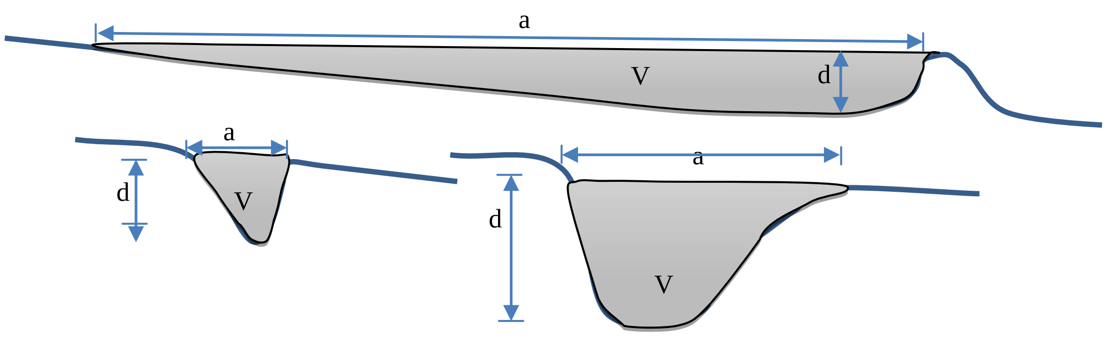


Figure 1. Examples of different forms and sizes of sinks. These parameters are a help when judging if the sink is natural or not, as well as if natural sinks should be removed. V = volume, d = depth, and a equals the area of the sink.

Man-made structures (non-natural break lines)

Many man-made structures disturb or block estimated flow in a DEM. The main reason for this is that e.g. tunnels and culverts are not captured within the data collection, and thus not included in the terrain model. A DEM covering an area including man-made barriers may thus result in non-natural sinks close to these structures. The barriers are often roads and railway lines, where structures like culverts, siphons and tunnels have been ‘hidden’ when constructing the DEM. The resulting sinks should be removed since, in these cases, all cells are actually connected and the flow should continue through the man-made artefact. It is thus crucial to identify cells on both ‘sides’ of the artefact, and let water flow between these points e.g. by breaching the barrier. Such connecting flow structures cannot be detected in the data when creating DEMs.

Therefore, it’s crucial to find solution how to handle this before estimating flow, flow accumulation and drainage area from a DEM.

Flat areas

A flat area cell can be defined as a cell surrounded by one or more cells with the same elevation, and no cell with lower elevation (see Figure 2). Flat cells do not exist in the reality, but sometimes in DEMs depending on generalisation of data type (maybe only integer is used, or a limited number of decimals) as well as filled sinks. This implies that flow should be modelled over flat areas, also justified by the fact that water flows over flat areas due to the change in the energy when water is accumulated (depth increases). Water naturally always flows to places with lower elevation, even if the terrain is ‘terraced’.

The question is of course how flow should be modelled over these flat areas. The flow can be defined as either ‘flow-out’ or ‘flow-in’. Flow-out implies that the flow from the flat area cells is directed out of the flat area, while flow-in occurs when the flow is converging in the centre of the flat area. The cell in the centre of the flat area will then have no flow out, and is treated as a sink.

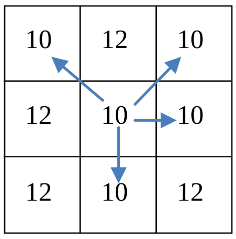


Figure 2. A 3 by 3 cell window exemplifying a flat area cell (the centre cell). The flat cell is surrounded by one or many cells with the same elevation, and no cells with lower elevation. Flow from a flat cell can be none, or outflow to one or more of the neighbouring cells with same elevation.

AIM

It is a fact that the presence of sinks, man-made structures, as well as flat areas in digital elevation models, does course great difficulties in hydrological modelling. No operational solutions how to tackle these problems in combination with a multiple flow routing algorithm have yet been presented.

The aim of this study is to develop, present and test practical solutions making it possible to estimate flow accumulation/drainage area over natural surfaces, represented as gridded raster DEMs, in an effective and usable way. The solutions will be applied on a real-world data set in order to exemplify their effects and usability.

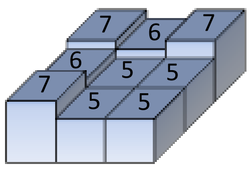
METHODOLOGY

Below follows a step-by-step description of how to tackle the above-mentioned artefacts in real world data. All software referred to is available through www.gis.lu.se/petter.

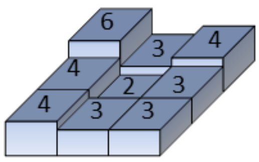
Cell classification

In order to separate and deal with the artefacts in the real-world DEMs we decided to label/classify all cells in the DEM. Three different classes were used, namely ‘Flat’, ‘Sink’, or ‘Undisturbed’ cells. Figure 3 is illustrating the different classes using a 3 by 3 cell window.

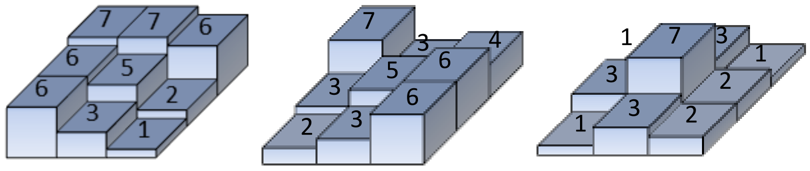
In a 3 by 3 window, the centre cell is classified as flat if at least one of the eight surrounding cells has the same elevation as the centre cell, while all other cells have a higher elevation value (see Fig. 3a). A flat area is a group of cells consisting of at least two flat cells neighbouring each other. A cell is classified as sink when all its neighbours have higher elevations (see Fig. 3b), while all remaining cells, having at least one cell with a lower elevation value are classified as undisturbed (see Fig 3c).



(a) Flat



(b) Sink



(c) Undisturbed

Figure 3. Illustrations of the different topographic forms classified by the use of a 3 by 3 cell window. A flat cell (a) is surrounded by cells with equal and higher elevations; a sink cell (b) is surrounded only by cells with higher elevations, while undisturbed cells (c) are neighboring at least one cell with lower elevation.

Filling sinks

As stated above some sinks in the DEM data might be a result of man-made artefacts, and need to be removed. In order to do this, and also include flexibility regarding area, depth and volume of the sink, a function was created in MATLAB (MathWorks (2008)). This function, attached in Appendix 1, makes it possible for the use to select threshold values (area, depth and volume) for sink removal through filling.

The idea behind the function is rather straight forward; Elevation values in flat areas (consisting of neighbouring cells classified as flat, see above) are compared to the elevation value of the lowest adjacent 'non-flat area' cell. The number of cells in the flat area multiplied by the cell size equals the area, the maximum difference in elevation equals the depth of the sink, and the sum of all differences in elevation multiplied by the cell size equals the volume.

The user is presented statistics (area, depth and volume) of the sinks, and then has the possibility to eliminate sinks of decided form and size by filling. The function also makes it possible to identify sinks to be removed by breaching (see Section Breaching break lines below).

Breaching break lines

If we know that man-mad artefacts, like roads, train lines or other walls are present in the data or suspected by visual interpretation of the DEM or analysis of the form and sizes of sinks (see above) there is a need for breaching of these artefacts. Thus a function (see Appendix 2) to breach the cells was added, in order to enable users to deal with e.g. man-made flow barriers like roads and railway lines or any other type of break lines. This function breaches the barrier by connecting two user-defined points/cells on the opposite sides of the obstruction. This is done in a semi-automatic way, where the user selects the approximate location of the two end points of the breach line, and the program searches and proposes suitable points/cells (to be confirmed or changed by the user). Rules used to propose these points are that, the starting point (higher elevation) should be the lowest point on the up-slope side of the barrier and higher than the end point, being the highest point on the other side of the barrier. When the points are selected the elevation of all cells in between the points will be changed according to a linear regression line between the points. The function as partly illustrated in Figure 4, where it can be seen how the elevation values are changed along the breach line to let the flow pass the barrier.

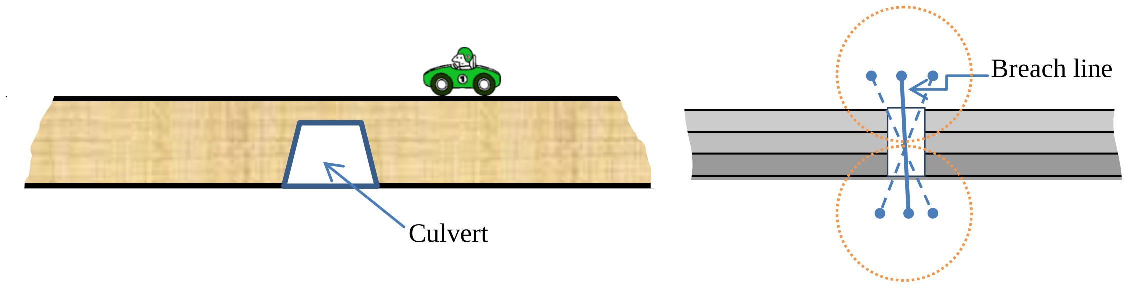


Figure 4. Illustration of how a man-made artefact like a road can be breached. To the left we see a culvert connecting the two sides of the barrier. This is however normally not represented in the DEM. To the right we see the result of the breaching function, where the user have identified the end point of a line (solid line) then the software search for the optimal ends within defined search radius (dashed lines). The breached line will enable water to pass through the barrier.

Flow distribution

The amount and direction of flow from each cell in a DEM to its surrounding cells depend on which flow distribution algorithm is used. However, in this study it is not our aim to explain and discuss estimated flow distribution using different flow algorithms, or to compare them. We are instead focusing on how to distribute flow and estimate drainage area/flow accumulation over flat surfaces. However, we want to stress that it is important to use a so called multiple flow algorithm (see e.g. Zhou, et al.(2011)). We have used a modification of the algorithm presented by Zhou, et al.(2011), based on facets with the cell centres as corner points (see below).

In Figure 2 we illustrated the main problem connected to flat cells; How to decide to which cells water should be distributed if they are all equal in elevation? In figure 5 we present a proposed solution of the scenario of Figure 2. Figure 5a illustrates the possible flow paths from the centre cell, not violating the rule that no water is allowed to flow up-hill. In Figure 5b we have extended the window to become 5 by 5 cells, and can logically explain why water is distributed to the upper right cell. The idea behind the algorithm, presented as a MATLAB function in Appendix 3, is that flow is directed in the direction of a lower cell if possible. In the case of Figure 5 the upper right cell in the 5 by 5 cell window has an elevation of 8, indicating that the flow is directed in this direction (blue arrows), in not in the other direction (red arrow).

Figure 5 also reveals the importance of only distributing flow to cells with lower elevation values in 'Undisturbed' terrain. This since cells with an elevation equal to the centre cell, if allocated water, might not represent a 'way out' for the water, but a 'dead end', 'trapping' the water in the sink.

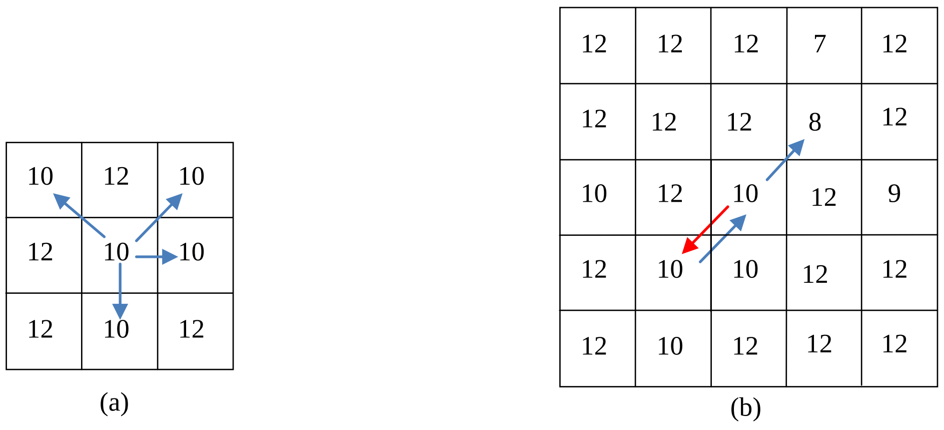


Figure 5. An example of flow distribution from a flat cell. In (a) all possible flow directions are identified, and in (b) the optimal flow path, directed (via another flat cell) to a cell with a lower elevation value is identified.

Two different functions has been developed and added to the main flow distribution program in order to handle flow over flat areas. The functions are attached in Appendix 3 and 4.

FLAT_Flow_out function (see Appendix 3) is a function directing the flow from flat cells where a way out of the flat area can be found. Examples are illustrated in Figure 5 above and Figure 6 below. Estimating the flow directions is done according to the following steps:

1. Identify outlets, i.e. cells just outside the border of the flat area with elevation values less than the flat area.
2. Assign neighbouring flat cells flow directions pointing at the outlet cells (by vector addition and splitting, see below).
3. Assign neighbouring flat cells flow directions pointing at the cells assigned in step 2.
4. Continue like above until all flat cells have been assigned flow directions.

So, the procedure starts by giving neighbouring cells to the ‘out flow’ cell flow directions ‘pointing’ to that cell, and then stepwise move further and further away from the out flow cell assigning flat cells flow directions to neighbours with a defined direction. Cells with the highest number of neighbours with defined flow directions are processed first, followed by cells with lower numbers of “defined neighbours”.

The flow routing from a flat cell is done by vector addition. Since we are using a multiple flow algorithm a non-flat cell normally distributes water to more than one neighbouring cell. The flow distribution can be described as vectors in different directions (maximum eight, equivalent to the directions to the eight neighbour cells), where each vector has a length corresponding to the amount of water directed in that direction (0 – 100%). By adding all these vectors for all non-flat neighbour cells (maximum eight neighbor cells with maximum eight vectors each) a new vector is created (see Pilesjö, et al. (1998)). This vector is supposed to represent the flow from the flat cell. Normally the direction of this vector falls in between to neighbouring cells. If so it is split proportionally (see Pilesjö, et al. (1998)), resulting in multiple flow also over flat areas.

The FLAT_Flow_in function (see Appendix 4) is used when there is no way out from the flat area. An example is presented in Figure 6b below. All cells just outside the border of the flat area have elevations higher than the flat area cells, and this result in a converging flow in the centre. This centre cell will have no defined flow and will be treated as a sink (see above). The flow directions of the surrounding cells will be estimated (by vector addition) starting from the flat cells that have the maximum number of known distributed flow directions cells (i.e. the border cells of the flat area).

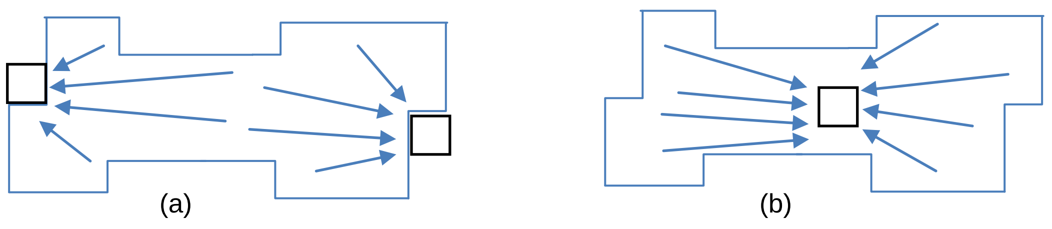


Figure 6. Illustrations how flow distribution is assigned to cells in two different types of flat areas. In example (a) the flat area has two outflow cells, resulting in a continuous flow pattern from the centre of the flat area, while in (b) the flat area has no outflow, resulting in a sink (or actually a cell treated as a sink) in the middle of the flat area.

As mentioned above, estimated flow directions from undisturbed cells will be dependent on the used algorithm. In our study, we have used a development of a facet-based algorithm presented by (Zhou, et al. (2011)).

Above we have discussed flow distribution over flat areas. If an area is undisturbed, there are a large number of flow algorithms to choose between (see e.g. O'Callaghan, et al. (1984), Holmgren (1994), Mark (1984), Freeman (1991), Pilesjö, et al. (1998)), all capable of estimating water flow. We have used an improved version of the facet-based presented in (Zhou, et al. (2011)). This method starts by creating triangular facets around centre cells in 3 by 3 cell windows in the regular gridded DEM (see Figure 7).On these eight facets water flow is modelled by using slope and slope direction of each facet. Incoming (from other cells/facets) as well as produced (precipitation) water is tracked over a facet, all the way to the facet border where it enters another facet.

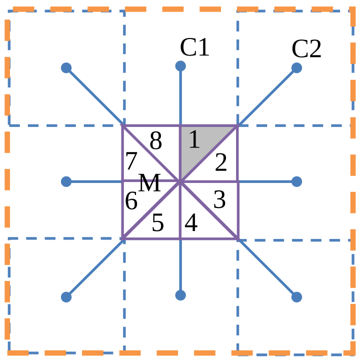


Figure 7. In the used flow distribution algorithm facets are created around each centre cell. The slopes and slope directions of these facets are used to track the theoretical water flow, and estimate to which neighboring cell/cells water will be transported.

It should be noted that, independently of flow algorithm, estimated flow from sink cells is estimated to zero. This means that an unfilled sink interrupts the flow pattern within the DEM, creating an isolated drainage area

Flow accumulation/Drainage area

In order to estimate flow accumulation (often referred to as drainage area) the drainage paths over the surface have to be traced, and the number of up-slope cells (area) transporting water to each and every cell has to be calculated. This is normally (see Oimoen (2000), Lindsay, et al. (2005), Band (1986), Beven, et al. (1979), Jenson, et al. (1988)) done by starting a ‘search function’ from peak cells, only transporting water to surrounding cells but not receiving water from neighbour cells. The drainage area for these peak cells will be one cell, and the drainage areas for the down-slope neighbour cells that they are transporting water to will be proportionally updated. For example, if water is transported to two neighbour cells on an equal basis both these cells will get an updated drainage area of +0.5. Then these peak cells are flagged as visited, and be neglected in further processing. After this the software will again search for cells with only out flow. Since the original peaks are excluded this will be cells neighbouring the peaks. Flow to down-slope neighbouring cells will be estimated, and then next iteration will start one ‘level’ down. This continues until all cells have been examined. A detailed description of the estimation of flow accumulation can be found in (Pilesjö, et al. (1998)). The method is equivalent to counting the flow paths/flow packages in the parts of the eight facets covering the centre cell (see Figure 7).

EXPERIMENT

The proposed methods have been tested using real-world data. The newly developed drainage area algorithms focusing on sink removal and flow estimation on flat areas have been applied to a real-world DEM, and the effects have been studied and compared. The spatial pattern of derived flow distribution is visually examined to detect significant artefacts.

Real-World DEM

The real-world earth surface elevation data used are from the Stordalen mire and its catchment area. Stordalen is a peatland area in the Arctic region, 10 km west of Abisko (68º 20' N, 19º 03' E) in northern Sweden. The Stordalen mire is a very flat area with many ponds and water bodies, resulting in a large number of flat areas and sinks. Moreover, a road and a railway line are constructed in this area, with many culverts to connect the flow of water between the two sides of the road or the two sides of the railway line. This site was thus judged as suitable for testing of the new methods dealing with flat areas, sinks and culverts.

An airborne LiDAR device has been used to measure the surface elevation. LiDAR is an acronym for "Light Detection And Ranging", and is a laser-based, remote sensing system used to collect various kinds of environmental data, including topographic data (Fowler (2001)). Over the area defined above the total number of measured elevation data points (the raw data) is 76 940 341. This results in a high resolution data set with an average spatial distribution of approximately 13 points/m2. The LiDAR data in the present study were retrieved with a TOPEYE S/N 425 system mounted on Helicopter SE-HJC. The altitude when sampling was 500 m. The LiDAR data have been post processed and adjusted against 54 known points connected to the national geodetic network. The mean vertical error after post-processing corrections is +0.004 m and the average magnitude of errors is 0.022 m. The RMSE is 0.031 m and the standard deviation is 0.031 m. For more details see (Hasan, et al. (2011)).

The raw LiDAR data were interpolated into a raster DEM by using inverse distance interpolation (Shepard (1968)). An accuracy assessment has been conducted by Hasan et al. (2011) on the created DEMs to find the best combination of Search radius and cell size. The selected optimum search radius was set to 1 m and the cell size set to 1 m also (see Hasan et al. (2011)). The DEM is presented in Figure 8 below.

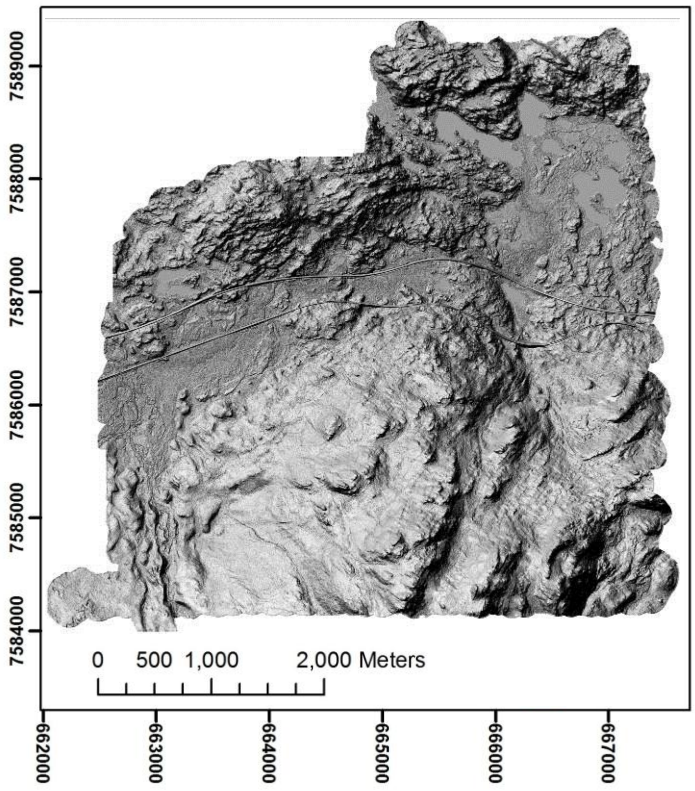


Figure 8. Hill-shade of the used DEM covering the Stordalen area in northern Sweden.

RESULTS

Figure 9a shows the sinks in the real-world DEM while Figure 9b shows the remain sinks after breaching all man-made structures using the culvert breaching approach presented in this study. Large sinks are removed by breaching few cells. Referring to Figure 9 one can clearly identify the problem with a large number of sinks, which would heavily influence estimated drainage area if not removed.

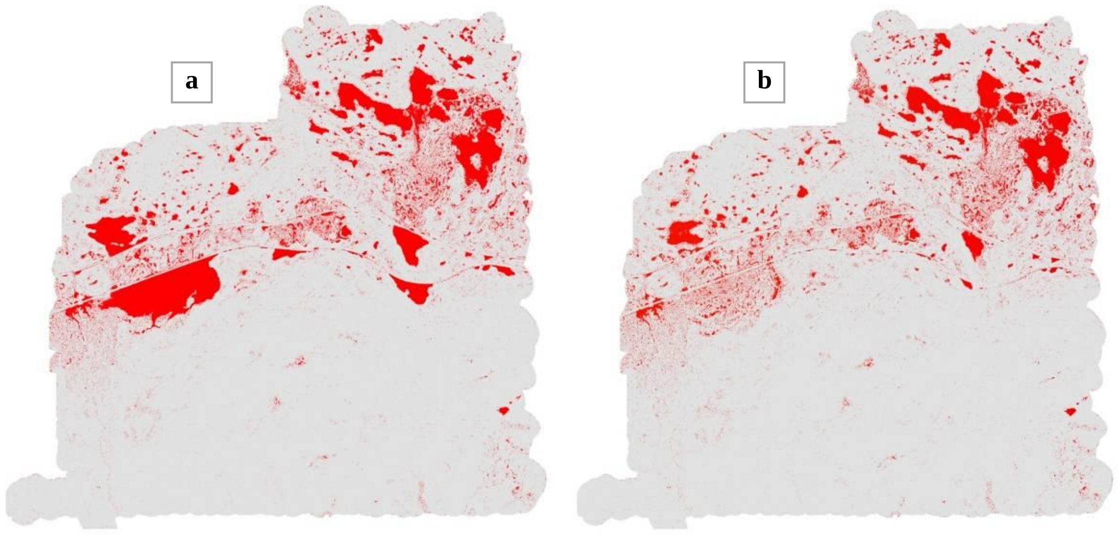


Figure 9a and 9b. Figure 9a illustrates the sinks (coloured red) in the Stordalen area. Figure 9b illustrates the remain sinks after breaching all man-made structures with the culvert approached presented in this study.

Figure 10 a-d shows show estimated drainage area, using two different algorithms and two different zooming views. In Figure 10a and figure 10b the single flow D8 algorithm (Jenson, et al. (1988)) has been applied, while the result of the newly developed algorithms presented in this paper is illustrated in Figure 10c and 10d.

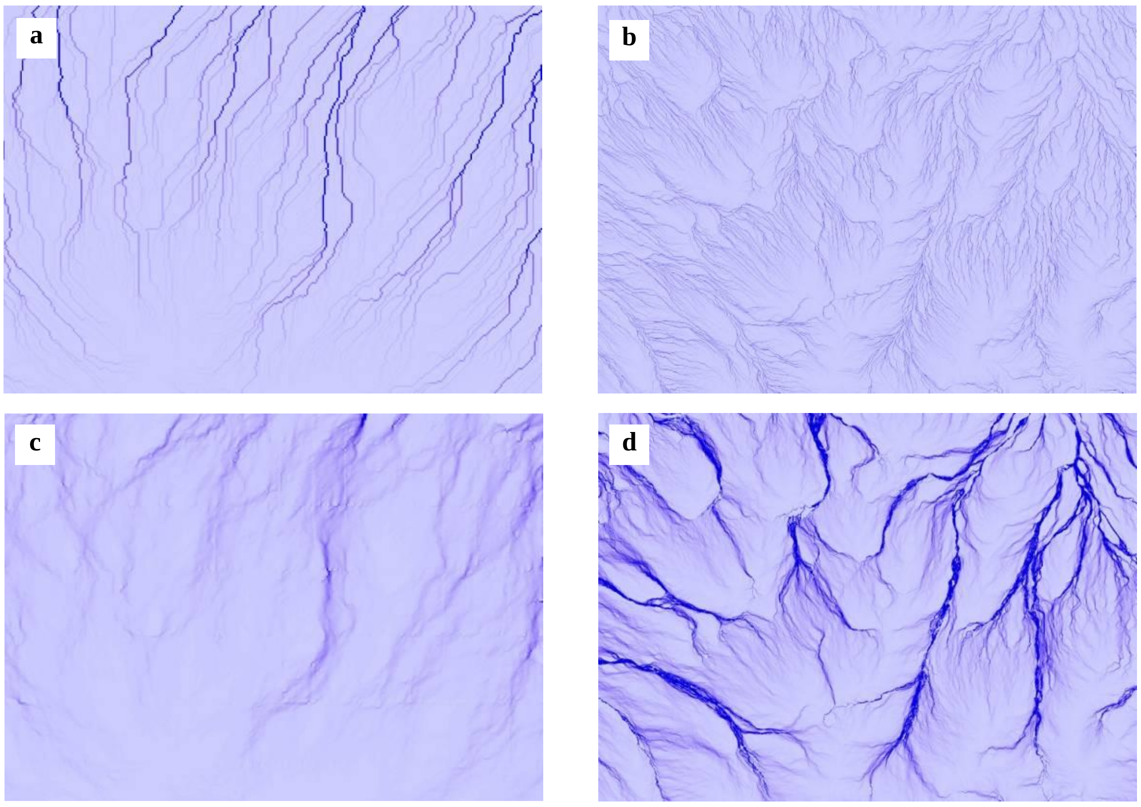


Figure 10a-d. Shows estimated drainage area, using two different algorithms and two different zooming views. In Figure 10a and Figure 10b the single flow D8 algorithm (Jenson, et al. (1988)) has been applied, while the result of the newly developed algorithms presented in this paper is illustrated in Figure 10c and 10d.

In Figure 11 we see the effect of breaching a barrier. A sink created by a train line is filled, and the resulting drainage area estimation is presented in Figure 11a. If, instead of filling the sink, the barrier (in this case the train line) is breached a totally different flow and drainage area pattern is created (Figure 11b). This highlights the importance of flexibility and choice in sink removal, made possible by the introduction of the above-mentioned algorithms.

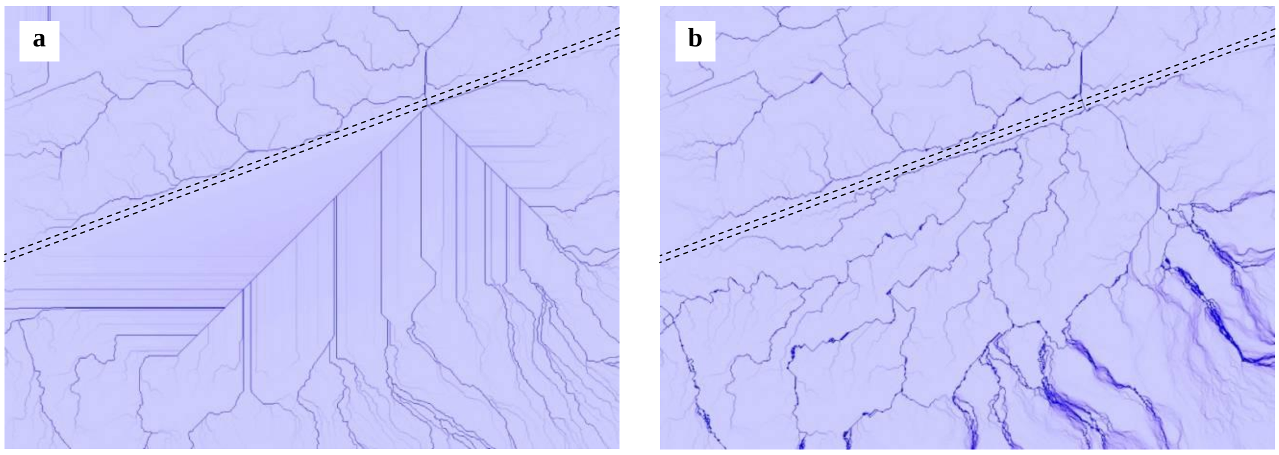


Figure 11. Illustrations of the result of using and not using the function to breach barriers and connect the flow between two sides of a barrier, in this case a railway line. In figure 11a (left) the sink has been filled, and the flow accumulation/drainage area has been estimated. This results in a large area of 'regular' flow. In Figure 11b (right) a few cells in the barrier have been breached, resulting in a more 'natural' estimated flow.

DISCUSSION AND CONCLUSIONS

In this research paper we have presented new and modified solutions for surface water routing based on a digital elevation model (DEM). These include different ways of sink removal, as well as water routing over flat surfaces. The related problems are not at all new, but have been discussed by several authors during the last decades (see e.g. Oimoen (2000), Lindsay, et al. (2005), Beven, et al. (1979), Jenson, et al. (1988) and Kenny, et al. (2008)). Our contributions to the discussions and solutions are mainly that:

- Our solutions are designed to be applied in combination with multiple flow algorithms, proved to be superior to the "normally used" single flow approach (see e.g. Zhou, et al. (2011)).
- We allow the user to interactively choose which sinks that should be filled or breached.
- We allow the user to interactively set thresholds for sinks to be filled, referring to sink area, sink volume, and sink depth.
- We allow the user to interactively define points/cells to breach barriers.
- We are only using the DEM for the flow routing, and no vector layers, like streams and lakes are needed.

It should also be noted that elevation (cell) values are not automatically updated, but the user is given the option to keep the original values, in order not to disturb e.g. estimations of slope and aspect.

It is well known that multiple flow algorithms are more realistic than e.g. the D8 solution (O'Callaghan, et al. (1984)). Thus, multiple flows over flat areas are logical. By summarizing the flow vectors (in different directions) of neighbouring cells, with defined flows, around a flat cell, and then split the summarized vector into two adjacent cells, a more realistic flow is modelled.

The question of which sinks that is results of errors and/or generalization is almost impossible to reply. However, by letting the user decide this, based on expert knowledge and/or visual interpretation of the DEM, and define thresholds for the area, volume or depth of these, we might get a more realistic representation of the reality after a sink removal.

The proposed culvert function can be cost effective. Breaching a few cells representing e.g. a culvert costs less than filling a large area around the sink. Moreover the culvert function can be used by civil engineers to help select the best locations for constructing culverts, tunnels or siphons.

We suggest that the methods presented in this paper are used together with other solutions, e.g. breaching according to (Garbrecht, et al. (1999)), and sink flow routing according to Seibert et al. (2007). Seibert et al. (2007) present an "elegant solution", where flow from a sink is transported as subsurface flow until it reaches the closest cell with a lower elevation than the sink bottom. From there the flow continues on the surface. Of course the results will also improve if methods like the one presented by Kenny, et al. (2008), including vector features if streams, lakes etc., will be used. However, integration of these methods can be time consuming, and they should preferably be adapted to multiple flow solutions of surface flow routing.

When estimating flow accumulation/drainage area and wetness index in order to increase accuracy in e.g. carbon modelling, data are often relatively course and generalized (see e.g. Hasan, et al. (2011)). Detailed stream networks are seldom available and the users have to rely on the DEM only. Under such circumstances the methods presented above are very useful.

REFERENCES

Quinn, P., K. Beven, P. Chevallier and O. Planchon. (1991) The prediction of hillslope flow paths for distributed hydrological modelling using digital terrain models. Hydrological Processes, 5, 59-79.

Seibert, J., K. H. Bishop and L. Nyberg. (1997) A test of TOPMODEL's ability to predict spatially distributed groundwater levels. Hydrological Processes, 11, 1131-1144.

Zhou, Q., P. Pilesjö and Y. Chen. (2011) Estimating surface flow paths on a digital elevation model using a triangular facet network. Water Resour. Res., 47, W07522.

Beven, K. J. and I. D. Moore. (1993) Terrain analysis and distributed modelling in hydrology / edited by K.J. Beven and I.D. Moore. Wiley & Sons, Chichester, England ; New York.

Wilson, J. P. and J. C. Gallant. (2000) Terrain Analysis: Principles and Applications. Wiley,

Seibert, J. and B. L. McGlynn. (2007) A new triangular multiple flow direction algorithm for computing upslope areas from gridded digital elevation models. Water Resour. Res., 43, W04501.

Pilesjö, P., Q. Zhou and L. Harrie. (1998) Estimating flow distribution over digital elevation models using a form-based algorithm. Geographic Information Sciences, 4, 44-51.

O'Callaghan, J. F. and D. M. Mark. (1984) The extraction of drainage networks from digital elevation data. Computer Vision, Graphics, and Image Processing, 28, 323-344.

Band, L. E. (1986) Topographic partition of watersheds with digital elevation models. Water Resources Research, 22, 15-24.

ESRI. (1991) Cell-based Modelling with GRID, Environmental System Research Institute.

Mark, D. M. (1984) Automated detection of drainage networks from digital elevation models. Cartographica, 21, 168-178.

Freeman, T. G. (1991) Calculating catchment area with divergent flow based on a regular grid. Computers and Geosciences, 17, 413-422.

Holmgren, P. (1994) Multiple flow direction algorithms for runoff modelling in grid based elevation models: an empirical evaluation. Hydrological Processes, 8, 327-334.

Pilesjö, P. and Q. Zhous, (1996), A multiple flow direction algorithm and its use for hydrological modelling, Geoinformatics´96, West Palm Beach, FL, April 26-28, 366-376.

Wolock, D. M. and G. J. McCabe. (1995) Comparison of single and multiple flow direction algorithms for computing topographic parameters in TOPMODEL. Water Resources Research, 31, 1315-1324.

Quinn, P. F., K. J. Beven and R. Lamb. (1995) The ln(a/tan-beta) Index - how to calculate it and how to use it within the TOPMODEL framework. Hydrological Processes, 9, 161-182.

Costa-Cabral, M. C. and S. J. Burges. (1994) Digital elevation model networks (DEMON): A model of flow over hillslopes for computation of contributing and dispersal areas. Water Resources Research, 30, 1681-1692.

Fairfield, J. and P. Leymarie. (1991) DRAINAGE NETWORKS FROM GRID DIGITAL ELEVATION MODELS. Water Resources Research, 27, 709-717.

Tarboton, D. G. (1997) A new method for the determination of flow directions and upslope areas in grid DEMs. Water Resources Research, 33, 309-319.

Beven, K. J. and M. J. Kirkby. (1979) A physically based, variable contributing area model of basin hydrology. Hydrological Sciences Journal, 24, 43-69.

Jenson, S. K. and J. O. Domingue. (1988) Extracting topographic structure from digital elevation data for geographic information system analysis. Photogrammetric Engineering and Remote Sensing, 54, 1593-1600.

Kenny, F., B. Matthews and K. Todd. (2008) Routing overland flow through sinks and flats in interpolated raster terrain surfaces. Computers & Geosciences, 34, 1417-1430.

Lindsay, J. B. and I. F. Creed. (2005) Sensitivity of digital landscapes to artifact depressions in remotely-sensed DEMs. Photogrammetric Engineering and Remote Sensing, 71, 1029-1036.

Oimoen, M. J.s, (2000), An effective filter for removal of production artifacts in US Geological Survey 7.5-minute digital elevation models, Proceedings of the 14th International Conference on Applied Geologic Remote Sensing, Las Vegas, NV, USA. Veridian ERIM International, Ann Arbor, MI, pp. 311-319.

Hasan, A., P. Pilesjö and A. Persson. (2011) The use of LiDAR as a data source for digital elevation models - DEM resolution versus accuracy and estimated slope, drainage area and wetness in northern peatlands. Hydrology and Earth System Sciences, discussion paper.

MathWorks. (2008) MATLAB R2008B. R2008B.

Fowler, R. (2001) Topographic Lidar Maune, Digital Elevation Model Technologies And Applications, American Society for Photogrammetry and Remote Sensing, Maryland.

Shepard, D. (1968) A two-dimensional interpolation function for irregularly-spaced data. Proceedings of the 1968 23rd ACM national conference.

Garbrecht, J. and L. W. Martz. (1999) TOPAZ: an automated digital landscape analysis tool for topographic evaluation, drainage identification, watershed segmentation and subcatchment parameterization; TOPAZ overview. United States Department of Agricultural, Agricultural Research Service (USDA-ARS) Publication No. GRL 99-1, Washington, DC, USA, 26pp.

Appendix 1

```
function [FilledDEM,MaxFilleddepth] = fillsinksAH(DEM,Maxdepth)

% this function is for filling pits or sinks or topographic depressions
%
% Date: 11 October, 2010
% Check inputs
error(nargchk(1, 2, nargin))
if nargin == 1;

else
    if ~isscalar(Maxdepth);
        error('Maxdepth must be a scalar')
    end
    if Maxdepth<=0;
        error('Maxdepth must not be zero or negative')
    end
end

% identify nans
INAN      = isnan(DEM);
% set nans to -inf
DEM(INAN) = -inf;

if nargin == 1;
    % fill sinks using imfill
    FilledDEM      = imfill(DEM,8,'holes'); % 8-neighborhood
 MaxFilleddepth=max(max(FilledDEM-DEM));%Max filled depth
 elseif nargin==2;
   MaxFilleddepth=Maxdepth;
    % imcomplement will make all sinks minima to be maxima
    DEM  = imcomplement(DEM);

    % Identify regional maxima
    Imax = imregionalmax(DEM,8);
    Imax = imclearborder(Imax);

    % create marker
    marker = -inf(size(DEM));
    marker(Imax) = DEM(Imax)-Maxdepth;
    FilledDEM  = imreconstruct(marker,DEM);

    % To check if there any regional maxima that have been filled
    % to exactly Maxdepth
    IImax  = FilledDEM(Imax) <= DEM(Imax)-Maxdepth;

    % if yes, repeat and create new marker
    if any(IImax)
        Imax = find(Imax);
        marker(Imax(IImax)) = DEM(Imax(IImax));
        FilledDEM  = imreconstruct(marker,DEM);
    end

    % complement DEM again so all maxima will be minima again (original)
    FilledDEM = imcomplement(FilledDEM);

end

% set the NaN in the DEM to be NaN
FilledDEM(INAN) = nan;
```

Appendix 2

```
%           Culvert Function
%
%   This function is for connecting the beginning point of a culvert with the end point and
%   linearly interpolates the elevation for all cells in-between (breach line).
%
%   Author: Abdulghani Hasan, Date: 30 / 03 / 2010
%% ************************************************************
function [ZT]=culverts(ZT,nrows,ncols,cellsize,xllcorner,yllcorner)
%Each culvert is of two cells
%Name of culverts pints file is Culvertpoints.txt
%x1 y1    start point culvert 1
%x2 y2    end point culvert 1
%x3 y3    start point culvert 2
%x4 y4    end point culvert 2
%
% load text file for culverts points coordinates
load CulvertsrealmyDEM.txt
INfile=CulvertsrealmyDEM;
%
xcol=INfile(:,1);%column x
ycol=INfile(:,2);%column y
%
%to find the matrix size
[np trash]=size(INfile); %INfile=total number of coordinates
%numberofculverts=np/2;
YesNo=input('Input a search radius for better culverts ends,Yes=1 NO=2 = ');
%
if YesNo ==1
SR=input('Input the search radius for searching around each cell/point in meter = ');
end
for j=1:2:np %since each culvert is from two cells (go for odd numbers
%
pointsdistance=((xcol(j)- xcol(j+1))^2 + (ycol(j)- ycol(j+1))^2)^0.5;
%
if YesNo ~=1 %&& pointsdistance >= (2*cellsize)
SR=pointsdistance/2;
end
if pointsdistance >= (2*cellsize)
SRcells = fix((SR/cellsize));%number of search radius cells
%
if SR < cellsize
  SRcells=0;
end
   PL=0;
    r1=(nrows - fix((ycol(j)-yllcorner)/cellsize));
    c1=fix((xcol(j)-xllcorner)/cellsize)+1;
    r2=(nrows - fix((ycol(j+1)-yllcorner)/cellsize));
    c2=fix((xcol(j+1)-xllcorner)/cellsize)+1;
%searching for lowest point around cells
if SRcells > 0 %if the search radius is larger than cell size searching around point 1
    startr=r1-SRcells;
    endr=r1+SRcells;
    startc=c1-SRcells;
    endc=c1+SRcells;
    for cc1=startc:endc
       for rr1=startr:endr
           if  cc1 > 0 && rr1 > 0 && cc1 < ncols && rr1 < nrows %check not out range
           if  ZT(r1,c1) > ZT(rr1,cc1)
               r1=rr1;
               c1=cc1;
           end
           end
       end

    end
  %searching around point 2
    startr=r2-SRcells; endr=r2+SRcells;
    startc=c2-SRcells; endc=c2+SRcells;
    for cc2=startc:endc
       for rr2=startr:endr
           if  cc2 > 0 && rr2 > 0 && cc2 < ncols && rr2 < nrows %check not out range
           if ZT(rr2,cc2) > ZT(r2,c2) && ZT(rr2,cc2) < ZT(r1,c1)
              % ZT(r2,c2) = ZT(r,c);
              r2=rr2;
               c2=cc2;
           end
           end
       end

    end
end
    X1=(c1*cellsize) + xllcorner - 0.5*cellsize;
    X2=(c2*cellsize) + xllcorner - 0.5*cellsize;
    Y1=((nrows-r1)*cellsize) + yllcorner + 0.5*cellsize;
    Y2=((nrows-r2)*cellsize) + yllcorner + 0.5*cellsize;
%
    Q1=[X1,Y1]; Q2=[X2,Y2];
    startC=c1;  endC=c2;
    if c2<c1
      startC=c2; endC=c1;
    end

    startR=r1;   endR=r2;
    if r2<r1
      startR=r2; endR=r1;
    end

    for cc=startC:endC
       for rr=startR:endR

           xcell= ((cc)*cellsize)+ xllcorner -(0.5*cellsize);
          ycell= (nrows-rr)*cellsize + yllcorner +(0.5*cellsize);
          P=[xcell,ycell];

%if the distance between a point and line is less than or equal 1/2 cell then the cell is in culvert
          d = abs(det([Q2-Q1;P-Q1]))/norm(Q2-Q1); % for row vectors.
%
          if d<=(0.5*cellsize)
           PL=PL+1; rdx(PL)=rr; cdx(PL)=cc;
         end
       end
    end
    %calculate the constant difference change in elevations
    constElvDiff=(abs(ZT(r1,c1) - ZT(r2,c2)))/(PL-1);
    %calculate all distances from the first point to all others that assigned r and c
      for i=1:PL
      Wd(i)=((r1-rdx(i))^2 + (c1-cdx(i))^2)^0.5;
      end
      pointmat=[rdx;cdx;Wd]; pointmat=pointmat';
      sorrtcells=sortrows(pointmat,3);
%
    if ZT(r1,c1)>ZT(r2,c2)
      for i=2:(PL-1)
           T=i-1;
       ZT(sorrtcells(i,1),sorrtcells(i,2))=ZT(r1,c1) - (T*constElvDiff);
      end
    end
         %if the second point is higher than the first
         if ZT(r2,c2)>ZT(r1,c1)
       for i=2:(PL-1)
       T=i-1;
       ZT(sorrtcells(i,1),sorrtcells(i,2))=ZT(r1,c1) + (T*constElvDiff);
       end
         end
         clear cdx; clear rdx: clear Wd
end
end
```

Appendix 3

```
% FLAT_Flow_out Function
% use this function if the cell is Flat to check if there is a flow out
% from Flat region
% Author: Abdulghani Hasan,  Date: 10/2010

function[FP1,FP2,FP3,FP4,FP5,FP6,FP7,FP8,Check_Matrix]=FLAT_Flow_out(FP1,FP2,FP3,FP4,FP5,FP6,FP7,FP8
,Classification_matrix,Check_Matrix,ZT,nrows,ncols)
%coordinates and numbering a 3*3 window cells
    %-------------------------
    %| 8  |   1   |   2   |
    %| 7  |   10  |   3   |
    %| 6  |   5   |   4   |
    %-------------------------
 Check_MatrixF=zeros(nrows,ncols); %check matrix for flat cells
 Check_MatrixF(Classification_matrix==3)=1;
 FL = bwlabel(Check_MatrixF);
MAXFLAT=max(max(FL));
% matrix the numbers in this matrix will be change according to the number of cell with defined flow
path around each cell.
for kk=1:MAXFLAT %Go through all flat cells
    [tr,tc]=find(FL ==kk);
    [n,trash]=size(tr); %find n the size of that matrix
    remain_flat_areas=MAXFLAT-kk
    nn=0;
    Remain_cell=n;
%put all cells around this cell in a matrix and do the same for all 8 %directions
Label=1;
while Remain_cell>0

for k=1:n
if tr(k)>1 && tc(k)>1 && tr(k)<nrows && tc(k)< ncols && Check_Matrix(tr(k),tc(k))==0 %excluding the
boundary cells
% to find the boundary of current cell
  Cell8=Check_Matrix(tr(k)-1,tc(k)-1); Cell1=Check_Matrix(tr(k)-1,tc(k)); Cell2=Check_Matrix(tr(k)-
1,tc(k)+1);...
            Cell3=Check_Matrix(tr(k),tc(k)+1);Cell4=Check_Matrix(tr(k)+1,tc(k)+1);...
            Cell5=Check_Matrix(tr(k)+1,tc(k)); Cell6=Check_Matrix(tr(k)+1,tc(k)-1);
Cell7=Check_Matrix(tr(k),tc(k)-1);
%putting all 8 cells in a vector matrix with correct coordinates according to the new 3 * 3 window
cell sequence
    Check_boundary=[ Cell1; Cell2; Cell3; Cell4; Cell5; Cell6; Cell7; Cell8 ];
    %for elevation matrix
    Cell8=ZT(tr(k)-1,tc(k)-1); Cell1=ZT(tr(k)-1,tc(k)); Cell2=ZT(tr(k)-1,tc(k)+1);...
            Cell3=ZT(tr(k),tc(k)+1);Cell4=ZT(tr(k)+1,tc(k)+1);...
            Cell5=ZT(tr(k)+1,tc(k)); Cell6=ZT(tr(k)+1,tc(k)-1); Cell7=ZT(tr(k),tc(k)-1);
    %Putting all 8 cells in a vector matrix
    ZT_boundary=[Cell1; Cell2; Cell3; Cell4; Cell5; Cell6; Cell7; Cell8];
    idxfLB=find(Check_boundary==Label); %find indexes of done cellss
     lng=length(idxfLB); %find the number of done cells around current cell
    %search for the way out between already checked cells
     FB=0;
     while FB<lng
         FB=FB+1;
        s=idxfLB(FB); %find the direction of that index, go through done cells
     if ZT(tr(k),tc(k)) >= ZT_boundary(s)%
      if  s==8
            rn=-1; cn=-1;
   elseif s==1
            rn=-1; cn=0;
   elseif s==2
            rn=-1; cn=+1;
   elseif s==3
             rn=0; cn=+1;
   elseif s==4
             rn=+1; cn=+1;
   elseif s==5
             rn=+1; cn=0;
   elseif s==6
             rn=+1; cn=-1;
   elseif s==7
             rn=0; cn=-1;
      end

  Cell82=FP1(tr(k)+rn,tc(k)+cn); Cell12=FP2(tr(k)+rn,tc(k)+cn); Cell22=FP3(tr(k)+rn,tc(k)+cn);...
            Cell32=FP4(tr(k)+rn,tc(k)+cn);Cell42=FP5(tr(k)+rn,tc(k)+cn);...
            Cell52=FP6(tr(k)+rn,tc(k)+cn); Cell62=FP7(tr(k)+rn,tc(k)+cn);
Cell72=FP8(tr(k)+rn,tc(k)+cn);
    %putting all 8 cells in a vector matrix
    FPCellsmat2=[Cell12; Cell22; Cell32; Cell42; Cell52; Cell62; Cell72; Cell82 ];
    Cell8=Check_Matrix(tr(k)+(rn-1),tc(k)+(cn-1)); Cell1=Check_Matrix(tr(k)+(rn-1),tc(k)+(cn));
Cell2=Check_Matrix(tr(k)+(rn-1),tc(k)+(cn+1));...
            Cell3=Check_Matrix(tr(k)+(rn),tc(k)+(cn+1));
Cell4=Check_Matrix(tr(k)+(rn+1),tc(k)+(cn+1));...
            Cell5=Check_Matrix(tr(k)+(rn+1),tc(k)+(cn)); Cell6=Check_Matrix(tr(k)+(rn+1),tc(k)+(cn-
1)); Cell7=Check_Matrix(tr(k)+(rn),tc(k)+(cn-1));
    %putting all 8 cells in a vector matrix
    Cellsmat2=[ Cell1; Cell2; Cell3; Cell4; Cell5; Cell6; Cell7; Cell8];
    Cell8=FL(tr(k)+(rn-1),tc(k)+(cn-1)); Cell1=FL(tr(k)+(rn-1),tc(k)+(cn)); Cell2=FL(tr(k)+(rn-
1),tc(k)+(cn+1));...
            Cell3=FL(tr(k)+(rn),tc(k)+(cn+1)); Cell4=FL(tr(k)+(rn+1),tc(k)+(cn+1));...
            Cell5=FL(tr(k)+(rn+1),tc(k)+(cn)); Cell6=FL(tr(k)+(rn+1),tc(k)+(cn-1));
Cell7=FL(tr(k)+(rn),tc(k)+(cn-1));
    %putting all 8 cells in a vector matrix
    CellsmatFL2=[ Cell1; Cell2; Cell3; Cell4; Cell5; Cell6; Cell7; Cell8 ];
    skip=0;
        FPtemp=zeros(8,1);
       idxfp=find(FPCellsmat2>0);
    leng=length(idxfp);
    for fp=1:leng
        if Cellsmat2(idxfp(fp))==0;%%flow to another flat cell
         skip=1;
        if CellsmatFL2(idxfp(fp))~=kk;%flow to another flat cell but in different flat region
         skip=0;
        end
        end
    end

      if skip==0
          FPtemp(s)=100;
    %give the calculated flow path values in FPtemp to the correct matrix
    FP1(tr(k),tc(k))=FPtemp(1);
    FP2(tr(k),tc(k))=FPtemp(2);
    FP3(tr(k),tc(k))=FPtemp(3);
    FP4(tr(k),tc(k))=FPtemp(4);
    FP5(tr(k),tc(k))=FPtemp(5);
    FP6(tr(k),tc(k))=FPtemp(6);
    FP7(tr(k),tc(k))=FPtemp(7);
    FP8(tr(k),tc(k))=FPtemp(8);
      %check the matrix as done
    Check_Matrix(tr(k),tc(k))=Label+1;
    Check_MatrixF(tr(k),tc(k))=0;
    Remain_cell=Remain_cell - 1;
    FB=lng;
     end
     end
     end % end of while

 end
end
nnn=nn-Remain_cell;
    if nnn == 0
       Remain_cell=0;
    end
Label=Label+1;
nn=Remain_cell;
end %end while remain

end

Check_Matrix(Check_Matrix>1)=1;
```

Appendix 4

```
% FLAT_Flow_in Function
% use this function if the cell is Flat and there is no flow out from Flat region
% Author: Abdulghani Hasan,  Date: 10/2010
function[FP1,FP2,FP3,FP4,FP5,FP6,FP7,FP8,Check_Matrix]=FLAT_Flow_in(FP1,FP2,FP3,FP4,FP5,FP6,FP7,FP8,
Check_Matrix,ZT,Classification_matrix,nrows,ncols)
%coordinates and numbering a 3*3 window cells
    %-------------------------
    %| 8  |   1   |   2   |
    %| 7  |   10  |   3   |
    %| 6  |   5   |   4   |
    %-------------------------

Contradictive=zeros(nrows,ncols);
contradictivecells=0;
          contradictivecells2=0;

    cross=0; %temp to calculate how many cross flow we had

 [tr,tc]=find(Check_Matrix ==0);
[n,trash]=size(tr); %find n the size of that matrix
flat_cells_vector=zeros(nrows,ncols);
counter=n;
kk=8;

while kk > 0

for k=1:n %Go through all flat cells

     if tr(k)>1 && tc(k)>1 && tr(k)<nrows && tc(k)< ncols %excluding the boundary cells
        %put the cells around each no data cell in a searching window
        if Check_Matrix(tr(k),tc(k)) == 0 && Classification_matrix(tr(k),tc(k)) == 3
          Cell8=Check_Matrix(tr(k)-1,tc(k)-1); Cell1=Check_Matrix(tr(k)-1,tc(k));
Cell2=Check_Matrix(tr(k)-1,tc(k)+1);...
            Cell3=Check_Matrix(tr(k),tc(k)+1);Cell4=Check_Matrix(tr(k)+1,tc(k)+1);...
            Cell5=Check_Matrix(tr(k)+1,tc(k)); Cell6=Check_Matrix(tr(k)+1,tc(k)-1);
Cell7=Check_Matrix(tr(k),tc(k)-1);
    %Putting all 8 cells in a vector matrix
    Cellsmat=[Cell1; Cell2; Cell3; Cell4; Cell5; Cell6; Cell7; Cell8];
      % give the sum of the around cells to the centre cell
    flat_cells_vector(tr(k),tc(k))= nansum(Cellsmat);
        end
    end
end

kk=max(max(flat_cells_vector))

    %end of number of neighbours calculations

for k=1:n %Go through all the flat cells

% go through all cells, search for cells with the highest number of defined
%flow path search for the value larger of equal k
%put all cells around this cell in a matrix and do the same for all 8 directions
if flat_cells_vector(tr(k),tc(k)) >= kk && Check_Matrix(tr(k),tc(k)) == 0 ...
  && tr(k)>1 && tc(k)>1 && tr(k)<nrows && tc(k)< ncols && Classification_matrix(tr(k),tc(k)) == 3

       Cell8=FP1(tr(k)-1,tc(k)-1); Cell1=FP1(tr(k)-1,tc(k)); Cell2=FP1(tr(k)-1,tc(k)+1);...
            Cell3=FP1(tr(k),tc(k)+1);Cell4=FP1(tr(k)+1,tc(k)+1);...
            Cell5=FP1(tr(k)+1,tc(k)); Cell6=FP1(tr(k)+1,tc(k)-1); Cell7=FP1(tr(k),tc(k)-1);
    %Putting all 8 cells in a vector matrix
    FP1Cellsmat=[Cell1; Cell2; Cell3; Cell4; Cell5; Cell6; Cell7; Cell8];
    %
     %FP2
       Cell8=FP2(tr(k)-1,tc(k)-1); Cell1=FP2(tr(k)-1,tc(k)); Cell2=FP2(tr(k)-1,tc(k)+1);...
            Cell3=FP2(tr(k),tc(k)+1);Cell4=FP2(tr(k)+1,tc(k)+1);...
            Cell5=FP2(tr(k)+1,tc(k)); Cell6=FP2(tr(k)+1,tc(k)-1); Cell7=FP2(tr(k),tc(k)-1);
    %Putting all 8 cells in a vector matrix
    FP2Cellsmat=[Cell1; Cell2; Cell3; Cell4; Cell5; Cell6; Cell7; Cell8];
    %
    %FP3
       Cell8=FP3(tr(k)-1,tc(k)-1); Cell1=FP3(tr(k)-1,tc(k)); Cell2=FP3(tr(k)-1,tc(k)+1);...
            Cell3=FP3(tr(k),tc(k)+1);Cell4=FP3(tr(k)+1,tc(k)+1);...
            Cell5=FP3(tr(k)+1,tc(k)); Cell6=FP3(tr(k)+1,tc(k)-1); Cell7=FP3(tr(k),tc(k)-1);
    %Putting all 8 cells in a vector matrix
    FP3Cellsmat=[Cell1; Cell2; Cell3; Cell4; Cell5; Cell6; Cell7; Cell8];
    %FP4
       Cell8=FP4(tr(k)-1,tc(k)-1); Cell1=FP4(tr(k)-1,tc(k)); Cell2=FP4(tr(k)-1,tc(k)+1);...
            Cell3=FP4(tr(k),tc(k)+1);Cell4=FP4(tr(k)+1,tc(k)+1);...
            Cell5=FP4(tr(k)+1,tc(k)); Cell6=FP4(tr(k)+1,tc(k)-1); Cell7=FP4(tr(k),tc(k)-1);
    %Putting all 8 cells in a vector matrix
    FP4Cellsmat=[Cell1; Cell2; Cell3; Cell4; Cell5; Cell6; Cell7; Cell8];
    %
    %FP5
       Cell8=FP5(tr(k)-1,tc(k)-1); Cell1=FP5(tr(k)-1,tc(k)); Cell2=FP5(tr(k)-1,tc(k)+1);...
            Cell3=FP5(tr(k),tc(k)+1);Cell4=FP5(tr(k)+1,tc(k)+1);...
            Cell5=FP5(tr(k)+1,tc(k)); Cell6=FP5(tr(k)+1,tc(k)-1); Cell7=FP5(tr(k),tc(k)-1);
    %Putting all 8 cells in a vector matrix
    FP5Cellsmat=[Cell1; Cell2; Cell3; Cell4; Cell5; Cell6; Cell7; Cell8];
    %
    %FP6
       Cell8=FP6(tr(k)-1,tc(k)-1); Cell1=FP6(tr(k)-1,tc(k)); Cell2=FP6(tr(k)-1,tc(k)+1);...
            Cell3=FP6(tr(k),tc(k)+1);Cell4=FP6(tr(k)+1,tc(k)+1);...
            Cell5=FP6(tr(k)+1,tc(k)); Cell6=FP6(tr(k)+1,tc(k)-1); Cell7=FP6(tr(k),tc(k)-1);
    %Putting all 8 cells in a vector matrix
    FP6Cellsmat=[Cell1; Cell2; Cell3; Cell4; Cell5; Cell6; Cell7; Cell8];
    %
    %FP7
      Cell8=FP7(tr(k)-1,tc(k)-1); Cell1=FP7(tr(k)-1,tc(k)); Cell2=FP7(tr(k)-1,tc(k)+1);...
            Cell3=FP7(tr(k),tc(k)+1);Cell4=FP7(tr(k)+1,tc(k)+1);...
            Cell5=FP7(tr(k)+1,tc(k)); Cell6=FP7(tr(k)+1,tc(k)-1); Cell7=FP7(tr(k),tc(k)-1);
    %Putting all 8 cells in a vector matrix
    FP7Cellsmat=[Cell1; Cell2; Cell3; Cell4; Cell5; Cell6; Cell7; Cell8];
    %
    %FP8
      Cell8=FP8(tr(k)-1,tc(k)-1); Cell1=FP8(tr(k)-1,tc(k)); Cell2=FP8(tr(k)-1,tc(k)+1);...
            Cell3=FP8(tr(k),tc(k)+1);Cell4=FP8(tr(k)+1,tc(k)+1);...
            Cell5=FP8(tr(k)+1,tc(k)); Cell6=FP8(tr(k)+1,tc(k)-1); Cell7=FP8(tr(k),tc(k)-1);
    %Putting all 8 cells in a vector matrix
    FP8Cellsmat=[Cell1; Cell2; Cell3; Cell4; Cell5; Cell6; Cell7; Cell8];

    %put all 8 1D vector matrices in one 2D matrix (path8matrix)

    path8matrix=[FP1Cellsmat FP2Cellsmat FP3Cellsmat FP4Cellsmat FP5Cellsmat...
        FP6Cellsmat FP7Cellsmat FP8Cellsmat];sum this matrix to calculate the sum of the flow in
specific direction. this will result 8 vectors
    Vectors_Mag=nansum(path8matrix);
%the angle is going counter clockwise starting from the positive x-axis
%those vectors will have the following angles respectively FP1 = 90 and FP2 =45 and 3 on x axis
    Vectors_Ang=[90 45 0 315 270 225 180 135];
    %to calculate the resultant, decompose all vectors to x and y components (xcomp ycomp)
    xcomp = Vectors_Mag .* cos(Vectors_Ang * pi/180);
    ycomp = Vectors_Mag .* sin(Vectors_Ang * pi/180);

    %sum all the components in x direction and y direction
xsum = nansum(xcomp);
ysum = nansum(ycomp);
%calculate the resultant although it not needed here
resultant = sqrt(xsum^2 + ysum^2);%resultant magnitude is not needed

resang = (atan (ysum/xsum))*180/pi; %calculate the angle of the resultant

%if the sign of one component is - then the resultant should be in quarter 2 or 4
if sign(xsum) == -1;
        resultant_angle = 180 + resang;
    elseif sign(ysum)== -1 ;
        resultant_angle = 360 + resang;
    else
        resultant_angle = resang;
end;
%
FPtemp=zeros(8,1);

%check if it is pointing to on cell if yes give 100% flow to that direction
mm=0;
one_drainage_direction=0;

Cell_count2=0;
%if one_drainage_direction == 0;
%%this is to correct case where all cells are higher than the centre cell except one is same
elevation then the flow will be 100% to than direction
Cellz8=ZT(tr(k)-1,tc(k)-1); Cellz1=ZT(tr(k)-1,tc(k)); Cellz2=ZT(tr(k)-1,tc(k)+1);...
            Cellz3=ZT(tr(k),tc(k)+1);Cellz4=ZT(tr(k)+1,tc(k)+1);...
            Cellz5=ZT(tr(k)+1,tc(k)); Cellz6=ZT(tr(k)+1,tc(k)-1); Cellz7=ZT(tr(k),tc(k)-1);
        Cellz10=ZT(tr(k),tc(k));
    %putting all 8 cells in a vector matrix
    Cellsmatelv=[Cellz1; Cellz2; Cellz3; Cellz4; Cellz5; Cellz6; Cellz7; Cellz8];
    %
   %start from angle -90 to 135 to go with our cell numbering

  for theta=1:8
     if resultant_angle == Vectors_Ang(theta)
                 one_drainage_direction=theta;
         FPtemp(theta)=100; % give 100 present to that direction
         if Cellsmatelv(theta) > Cellz10 % check the elevation of that direct
           FPtemp(theta)=0;
         end
     end
  end
   %%%%%
   for v=1:4
%for v=1 then (v+1)is (2)refer to cell 1
%check if the current cell has a flow path to cell 1 then check if cell one has a flow path to cell
5 our cell
         if FPtemp(v) > 0 %the cell number that the flow is directed to it
            if path8matrix(v,v+4) > 0 %Contradictive
               Contradictive(tr(k),tc(k))=1;%lable 1 to Contradictive matrix
               contradictivecells=contradictivecells+1;
            FPtemp(v)=0; %set the flow to 0 in current cell

            end
         end
   end
     %the calculation is split to 2 portions to handle out of indexing
         for v=8:-1:5 %go from 8 to 5 do same as above
             if FPtemp(v) > 0
                if path8matrix(v,v-4) > 0 %Contradictive

         Contradictive(tr(k),tc(k))=1;%Contradictive
       contradictivecells=contradictivecells+1;
            FPtemp(v)=0; %set the flow to 0 in current cell

                end
            end
         end

    m2=0;

    for nn=1:8
        if Cellz10 == Cellsmatelv(nn) && isnan(Cellsmatelv(nn))==0
            Cell_count2=Cell_count2+1;
            m2=nn; %if the valley is one cell wide then m is the cell index
        end
    end
    %end
    if Cell_count2 ==1
        FPtemp=zeros(8,1);%this is to correct case 7 where I get 100 flow in two directions
        %100 to one direction because full angle 45 90 135... and 100 to other because just one cell
is the same elevation
        FPtemp(m2)=100;

     end %end of cell count2=0%end of case 5 two neighbour flat cells
  %if the resultant is directed to two cells. Split the flow to two portions
if one_drainage_direction == 0 && Cell_count2 ~= 1;
% sectors depending on the angle are going counter clockwise starting from
% x-axis, s= 1 between cells 2&3, s=2 between cells 2&1
%p=5
       s=fix(resultant_angle/45)+1;
        if s==1 || s==3 || s==5 || s==7
            A=(resultant_angle -((s-1)*45))*pi/180;
        end
        if s==2 || s==4 || s==6 || s==8
          A=((s*45)- resultant_angle)*pi/180;
        end
       %
        vector_a = (sin(A))/(sin(45*pi/180-A));
        Wa=vector_a/(vector_a +1);
        Wa=(fix(Wa*10000))/100;
        Wc=100-Wa;
   %the following are added to make the flow 100 present directed to one cell
   %if the pecentage to the other cell is less than 1 percentage
        if (Wa -Wc)>= 99
            Wa =100;
            Wc =0;
        end
        %oposit
        if (Wc -Wa)>= 99
            Wc =100;
            Wa =0;
        end
%subtract one (1) from the index to find the cell number
      if  s==1
           FPtemp(2)=Wa;
           FPtemp(3)=Wc;
   elseif s==2
           FPtemp(1)=Wc;
           FPtemp(2)=Wa;
   elseif s==3
           FPtemp(8)=Wa;
           FPtemp(1)=Wc;
   elseif s==4
           FPtemp(7)=Wc;
           FPtemp(8)=Wa;
   elseif s==5
           FPtemp(6)=Wa;
           FPtemp(7)=Wc;
   elseif s==6
           FPtemp(5)=Wc;
           FPtemp(6)=Wa;
   elseif s==7
           FPtemp(4)=Wa;
           FPtemp(5)=Wc;
        elseif s==8
            FPtemp(3)=Wc;
           FPtemp(4)=Wa;
      end
      %Check for Contradictive flow
     for v=1:4 %cells number 1-8 =2-9
 %check if the current cell has a flow path to cell 1 then check if
         %cell one has a flow path to cell 5 our cell
         if FPtemp(v) > 0 %the cell number that the flow is directed to it
            if path8matrix(v,v+4) > 0 %Contradictive
               Contradictive(tr(k),tc(k))=1;%lable 1 to Contradictive matrix
               contradictivecells=contradictivecells+1;
            FPtemp(v)=0; %set the flow to 0 in current cell
            %%%%

            for fff=1:8
                 if FPtemp(fff)>0
                    FPtemp(fff)=100;
                 end
             end

            end
         end
     end
     %the calculation is split to 2 portions to handle out of indexing
    for v=8:-1:5 %go from 8 to 5 do same as above
         if FPtemp(v) > 0
            if path8matrix(v,v-4) > 0 %Contradictive
               FPtemp(v)=0;
         Contradictive(tr(k),tc(k))=1;%Contradictive
       contradictivecells=contradictivecells+1;

             for fff=1:8
                 if FPtemp(fff)>0
                    FPtemp(fff)=100;
                 end
             end

           end
         end
    end
     %check again if the flow from the one cell is also contradictive give drainage path 0
     for v=1:4
         if FPtemp(v) > 0
            if path8matrix(v,v+4) > 0 %Contradictive
            FPtemp(v)=0;
                           Contradictive(tr(k),tc(k))=2;%Contradictive
                    contradictivecells2=contradictivecells2+1;
            end
         end
     end
         for v=8:-1:5
         if FPtemp(v) > 0
            if path8matrix(v,v-4) > 0 %Contradictive
               FPtemp(v)=0;
                              Contradictive(tr(k),tc(k))=2;%Contradictive
                    contradictivecells2=contradictivecells2+1;
           end
         end
         end
end
%check if there is flow to a higher cell
for ff=1:8
         if Cellsmatelv(ff) > Cellz10 && FPtemp(ff)>0

             FPtemp(ff)=0;
             for fff=1:8
                 if FPtemp(fff)>0
                    FPtemp(fff)=100;
                 end
             end

         end
end
%give the calculated flowpath values in FPtemp to the correct matrix
    FP1(tr(k),tc(k))=FPtemp(1);
    FP2(tr(k),tc(k))=FPtemp(2);
    FP3(tr(k),tc(k))=FPtemp(3);
    FP4(tr(k),tc(k))=FPtemp(4);
    FP5(tr(k),tc(k))=FPtemp(5);
    FP6(tr(k),tc(k))=FPtemp(6);
    FP7(tr(k),tc(k))=FPtemp(7);
    FP8(tr(k),tc(k))=FPtemp(8);
%check the matrix as done
    Check_Matrix(tr(k),tc(k))=1;
    flat_cells_vector(tr(k),tc(k))=0;
    counter=counter - 1

end
end
end
```