UNIVERSITY OF NAIROBI

DEPARTMENT OF GEOSPATIAL AND SPACE TECHNOLOGY

MASTER OF SCIENCE IN GEOGRAPHIC INFORMATION SYSTEM

GROUP 1: AUTOMATION OF DELIANATION OF WATERSHED ANALYSIS USING WHITEBOX TOOLS IN PYTHON PROGRAMMING LANGUAGE

GROUP 1: MEMBERS

NAME: PASSIANY MIKE	**REG. NO: F56/42147/2022**
NAME: PASCAL ADONGO	**REG. NO: F56/42451/2022**
NAME: FAITH KAGENI	**REG. NO: F56/42449/2022**
NAME: DAMARIS NDUNG’U	**REG. NO: F56/43133/2022**

FGS 6102: GIS PROGRAMMING

Dr Collins Mwange

16th May, 2023.

AUTOMATION OF DELINEATION OF WATERSHED ANALYSIS USING WHITEBOX TOOLS IN PYTHON PROGRAMMING LANGUAGE

Introduction

The purpose of this tutorial is to provide a step-by-step guide on automating the process of delineating watershed analysis using Whitebox Tools in the Python programming language. This tutorial utilizes various hydrology tools available in Whitebox Tools to analyze the hydrology of a given area. By following this tutorial, users will gain an understanding of the workflow involved in hydrological analysis and will be able to adapt it to their specific needs.

Prerequisites

This tutorial assumes that the user has a basic understanding of GIS software and is familiar with Python programming.

Data Acquisition

To begin the analysis, the only required dataset is a Digital Elevation Model (DEM) file of the area of interest. The focus of this tutorial is on the analysis of [provide specific focus or purpose of the analysis].

PROCEDURE

Setting up the Environment

The required libraries, including tkinter, subprocess, os, and shutil, are imported.
The path to the Whitebox Tools executable is set.
The root Tkinter window is initialized and hidden.

```python
from tkinter import filedialog
import tkinter as tk
import subprocess
import os
import shutil
os.environ["RUST_BACKTRACE"] = "1"
```

Selecting the Clipped DEM File

A file dialog is opened to allow the user to select the clipped DEM file for further analysis.

```python
# Select the clipped DEM file using a file dialog
clipped_dem_path = filedialog.askopenfilename(title="Select Clipped DEM File")
```

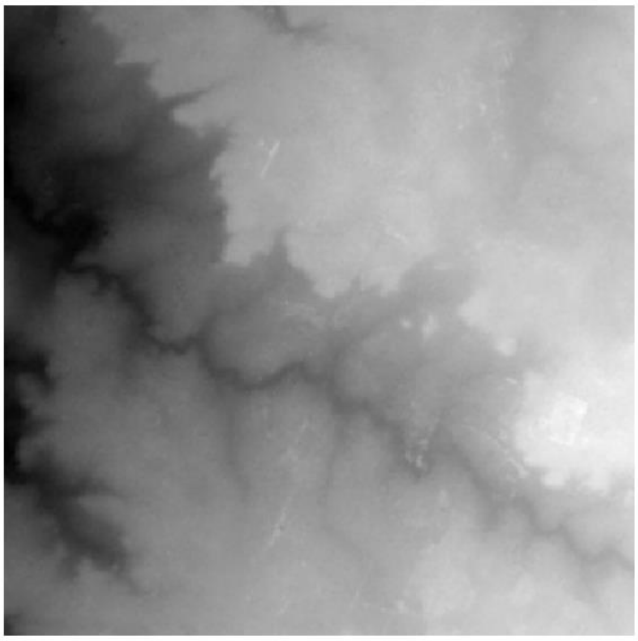


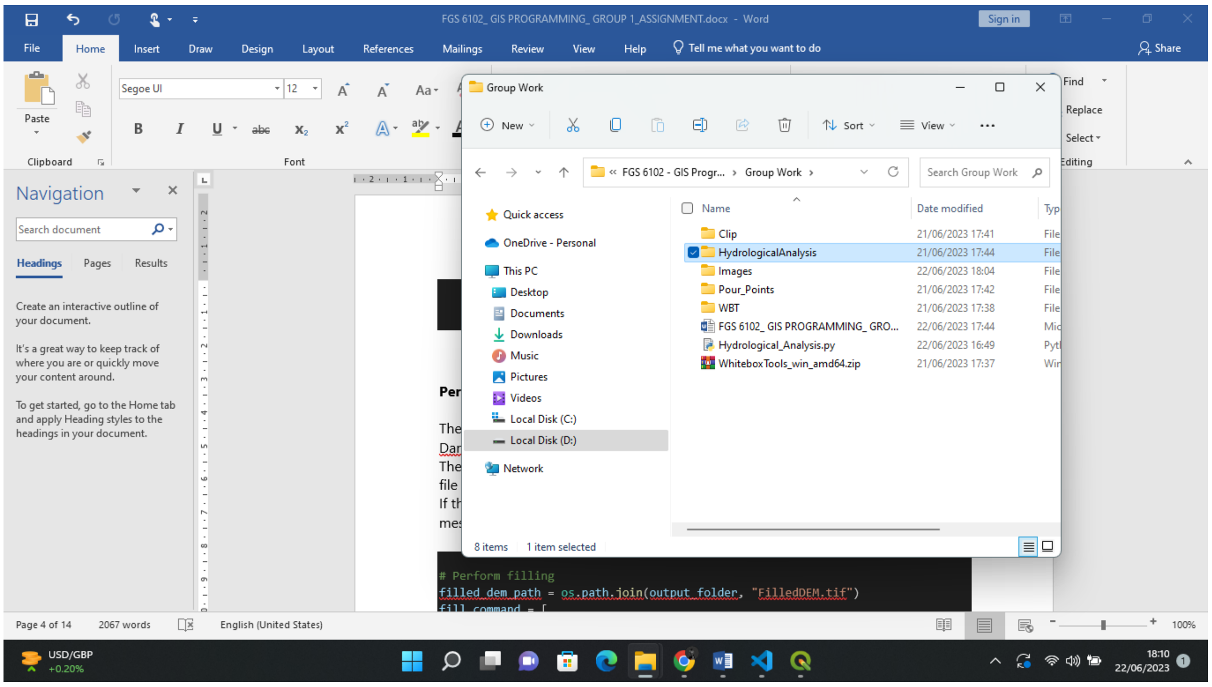
***Fig1**: Clipped Digital Elevation Model*

Creating Output Folder

A folder named "HydrologicalAnalysis" is created to store the outputs.
If the folder already exists, it is skipped.
Defining the 'run_whitebox_command' Function:
This function is responsible for executing Whitebox Tools commands.
It takes a command and an output file as inputs.
It calls the subprocess module to execute the command.
It checks if the output file exists and returns True if it does, else prints an error message and returns False.

```python
# Create a folder for the outputs
output_folder = "HydrologicalAnalysis"
os.makedirs(output_folder, exist_ok=True)


def run_whitebox_command(command, output_file):
    try:
        subprocess.call(command)
        if os.path.exists(output_file):
            return True
        else:
            print(f"Error: Output file '{output_file}' not found.")
    except subprocess.CalledProcessError as e:
        print(f"Error executing command: {e}")
    return False
```



Name	Date modified	Typ
Clip	21/06/2023 17:41	File
HydrologicalAnalysis	21/06/2023 17:44	File
Images	22/06/2023 18:04	File
Pour_Points	21/06/2023 17:42	File
WBT	21/06/2023 17:38	File
FGS 6102_ GIS PROGRAMMING_ GRO...	22/06/2023 17:44	Mic
Hydrological_Analysis.py	22/06/2023 16:49	Pyt
WhiteboxTools_win_amd64.zip	21/06/2023 17:37	Win

```
# Perform filling
filled_dem_path = os.path.join(output_folder, "FilledDEM.tif")
fill_command = [
```

Fig 2: *A screenshot of the created folder to contain the output files*

Performing Filling

The Whitebox Tools command for filling depressions using the Planchon and Darboux algorithm is defined.
The 'run_whitebox_command' function is called with the fill command and the output file path.
If the command execution is successful and the output file is generated, a success message is printed; otherwise, an error message is displayed.

```
# Perform filling
filled_dem_path = os.path.join(output_folder, "FilledDEM.tif")
fill_command = [
    whitebox_path,
    "--run=fill_depressions_planchon_and_darboux",
    f"--input='{clipped_dem_path}'",
    f"--fix_flats=true",
    # f"--flat_increment=none",
    f"--output='{filled_dem_path}'"
]
if run_whitebox_command(fill_command, filled_dem_path):
    print("Filling completed.")
else:
    print("Error occurred during filling.")
```

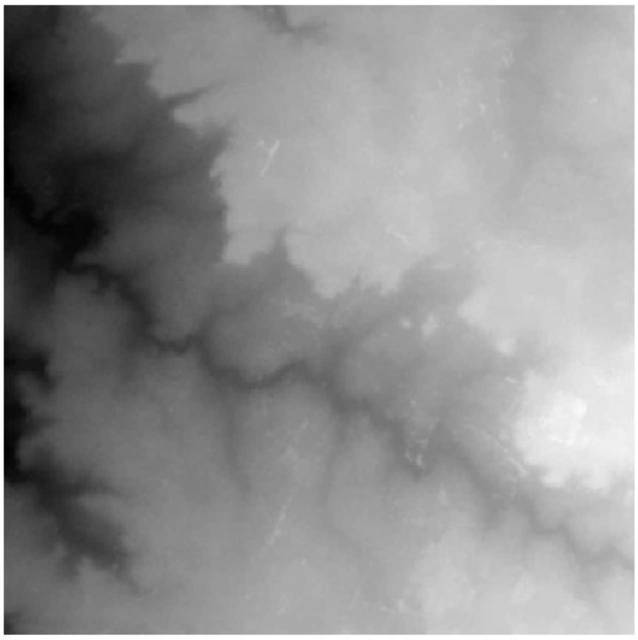


Fig 3: *A Filled Digital Elevation Model Image*

Performing Flow Direction

The Whitebox Tools command for calculating flow direction using the D8 algorithm is defined.
The 'run_whitebox_command' function is called with the flow direction command and the output file path.
Similar to the previous step, success or error messages are printed based on the command execution.

```
# Perform flow direction
flow_direction_path = os.path.join(output_folder, "FlowDirection.tif")
flow_direction_command = [
    whitebox_path,
    "--run=D8Pointer",
    f"--dem='{filled_dem_path}'",
    f"--output='{flow_direction_path}'",
    f"--esri_pntr=true"
]
if run_whitebox_command(flow_direction_command, flow_direction_path):
    print("Flow direction calculation completed.")
else:
    print("Error occurred during flow direction calculation.")
```

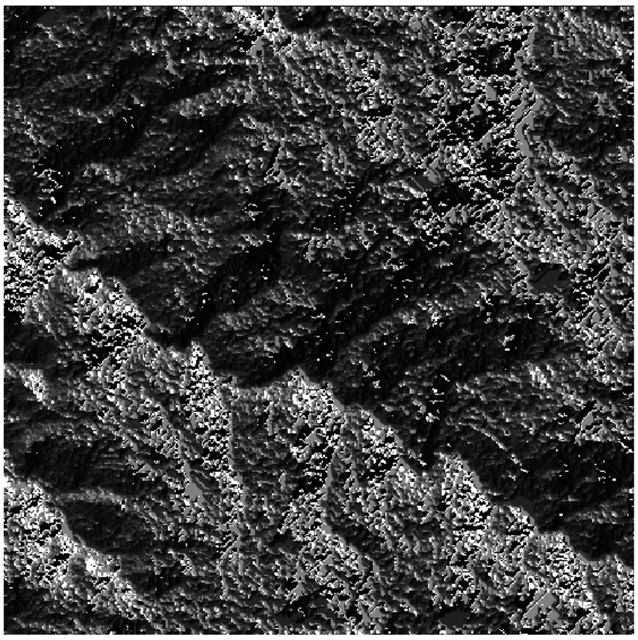


Fig 4: *A Flow Direction Image from the Filled Digital Elevation Model*

Performing Flow Accumulation

The Whitebox Tools command for calculating flow accumulation is defined.
The 'run_whitebox_command' function is called with the flow accumulation command and the output file path.
Success or error messages are printed accordingly.

```
# Perform flow accumulation

flow_accumulation_path = os.path.join(output_folder, "FlowAccumulation.tif")
flow_accumulation_command = [
    whitebox_path,
    "--run=d8_flow_accumulation",
    f"--dem='{flow_direction_path}'",
    f"--out_type='Specific Contributing Area'",
    f"--output='{flow_accumulation_path}'",
    f"--outputtype='cells'",
    f"--esri-pntr=true",
    f"--pntr=true"
]
if run_whitebox_command(flow_accumulation_command, flow_accumulation_path):
    print("Flow accumulation calculation completed.")
else:
    print("Error occurred during flow accumulation calculation.")
```

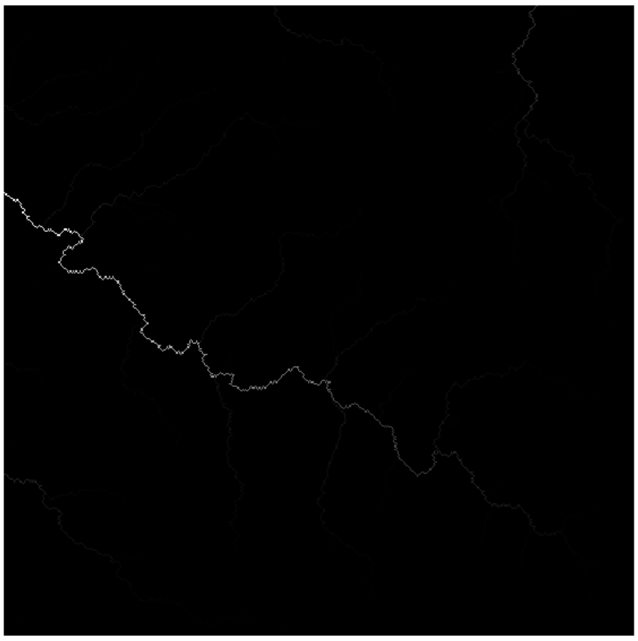


Fig 5: *A Flow Accumulation raster output from the Flow Direction raster file*

Performing Raster Calculator

The Whitebox Tools command for performing raster calculations is defined.
The 'run_whitebox_command' function is called with the raster calculator command and the output file path.
Success or error messages are printed accordingly.

```python
# Perform raster calculator
raster_calculator_path = os.path.join(output_folder, "RasterCalculator.tif")
raster_calculator_command = [
    whitebox_path,
    "--run=RasterCalculator",
    f"--statement='{flow_accumulation_path}' <= 0.11)",
    f"--output='{raster_calculator_path}'"
]
if run_whitebox_command(raster_calculator_command, raster_calculator_path):
    print("Raster calculator completed.")
else:
    print("Error occurred during raster calculator.")
```

Performing Extract Streams

The Whitebox Tools command for extracting streams is defined.
The command is executed using the subprocess module.

```
# Perform extract streams
extracted_streams_path = os.path.join(output_folder, "ExtractedStreams.tif")
subprocess.call([
    whitebox_path,
    "--run=extract_streams",
    f"--flow_accum='{flow_accumulation_path}'",
    f"--threshold=0.090",
    # "--zero_background",
    f"--output='{extracted_streams_path}'"
])
```

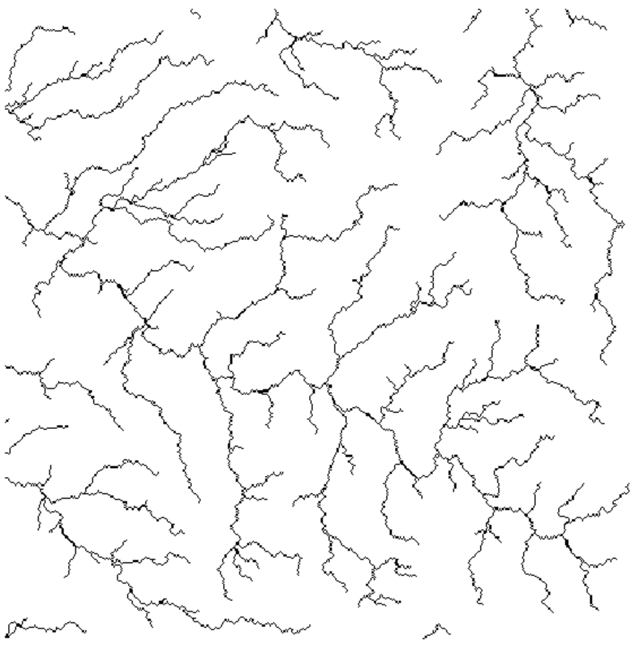


Fig 6: *An Extracted Streams Output from the Flow Accumulation Raster*

Performing Strahler Stream Order

The Whitebox Tools command for calculating stream order using the Strahler algorithm is defined.
The 'run_whitebox_command' function is called with the stream order command and the output file path.
Success or error messages are printed accordingly.

```
# Perform stream order
stream_order_path = os.path.join(output_folder, "StreamOrder.tif")
```

```
stream_order_command = [
    whitebox_path,
    "--run=strahler_stream_order",
    f"--streams='{extracted_streams_path}'",
    f"--d8_pntr='{flow_direction_path}'",
    f"--output='{stream_order_path}'",
    f"--esri_pntr=true"
]
if run_whitebox_command(stream_order_command, stream_order_path):
    print("Stream order calculation completed.")
else:
    print("Error occurred during stream order calculation.")
```

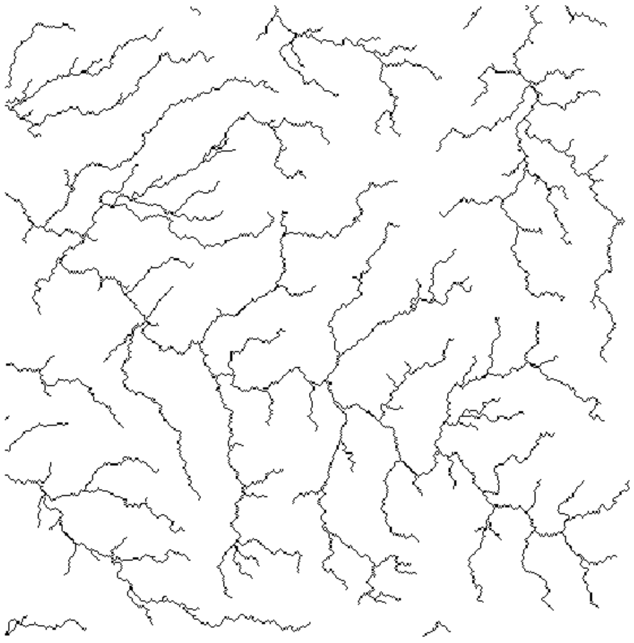


Fig 7: *A Strahler Stream Order Output File from the Extracted Streams File*

Performing Hack Stream Order

The Whitebox Tools command for calculating stream order using the Hack algorithm is defined.
The 'run_whitebox_command' function is called with the hack stream order command and the output file path.

Success or error messages are printed accordingly.

```
# Perform hack stream order
hack_stream_order_path = os.path.join(output_folder, "HackStreamOrder.tif")
hack_stream_order_command = [
    whitebox_path,
    "--run=hack_stream_order",
    f"--streams='{extracted_streams_path}'",
    f"--d8_pntr='{flow_direction_path}'",
    f"--output='{hack_stream_order_path}'",
    f"--esri_pntr=true"
]
if run_whitebox_command(hack_stream_order_command, hack_stream_order_path):
    print("Stream order calculation completed.")
else:
    print("Error occurred during stream order calculation.")
```

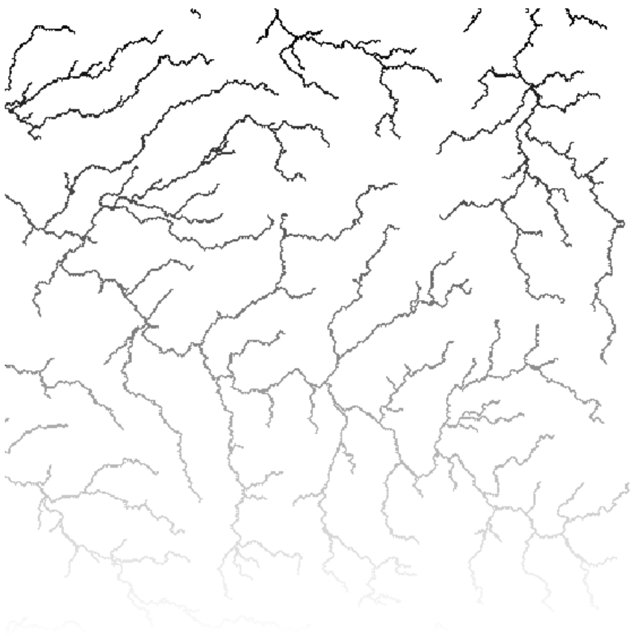


Fig 8: *A Hack Stream Order Output File*

Topological Stream Order

The `topological_stream_order_path` variable specifies the output path for the topological stream order raster.
The `topological_stream_order_command` list contains the command-line arguments to be passed to Whitebox Tools.
The `run_whitebox_command` function is called to execute the Whitebox Tools command and store the result in the specified output file.
If the command executes successfully, the message "Stream order calculation completed." is printed; otherwise, an error message is displayed.

```
# Perform topological stream order
topological_stream_order_path = os.path.join(
    output_folder, "TopologicalStreamOrder.tif")
topological_stream_order_command = [
    whitebox_path,
    "--run=topological_stream_order",
    f"--streams='{extracted_streams_path}'",
    f"--d8_pntr='{flow_direction_path}'",
    f"--output='{topological_stream_order_path}'",
    "--esri_pntr=true"
]
if run_whitebox_command(topological_stream_order_command,
topological_stream_order_path):
    print("Stream order calculation completed.")
else:
    print("Error occurred during stream order calculation.")
```

Stream Link

The `stream_link_path` variable specifies the output path for the stream link raster.
The `stream_link_command` list contains the command-line arguments for the stream link operation.
The process is similar to the previous step, where the `run_whitebox_command` function is used to execute the command and handle success or failure messages.

```
# Perform stream link
stream_link_path = os.path.join(output_folder, "StreamLink.tif")
stream_link_command = [
    whitebox_path,

    "--run=stream_link_class",
    f"--d8_pntr='{flow_direction_path}'",
    f"--streams='{stream_order_path}'",
    f"--output='{stream_link_path}'",
    f"--esri_pntr=true",
]
if run_whitebox_command(stream_link_command, stream_link_path):
    print("Stream Link conversion completed.")
else:
    print("Error occurred during stream link conversion.")
```

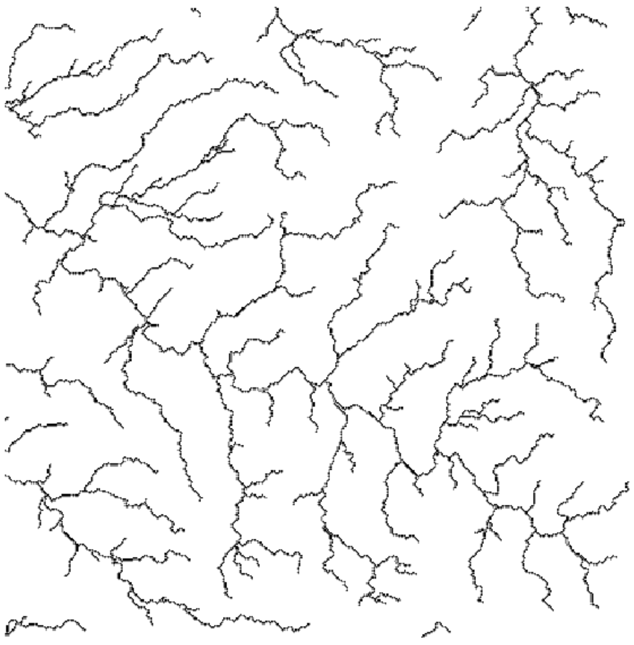


Fig 9: *A Stream Link Output File from the input Stream Order Output*

Remove Short Streams

The `remove_short_streams_path` variable specifies the output path for the raster after removing short streams.
The `remove_short_streams_command` list contains the command-line arguments for removing short streams.
Again, the `run_whitebox_command` function is used to execute the command and handle success or failure messages.

```
# Perform remove short streams
remove_short_streams_path = os.path.join(
    output_folder, "RemoveShortStreams.tif")
remove_short_streams_command = [
    whitebox_path,
    "--run=remove_short_streams",
    f"--d8_pntr='{flow_direction_path}'",
    f"--streams='{stream_order_path}'",
    f"--output='{remove_short_streams_path}'",
    f"--esri_pntr=true"
]
if run_whitebox_command(remove_short_streams_command,
remove_short_streams_path):
    print("Stream to feature conversion completed.")
else:
    print("Error occurred during stream to feature conversion.")
```

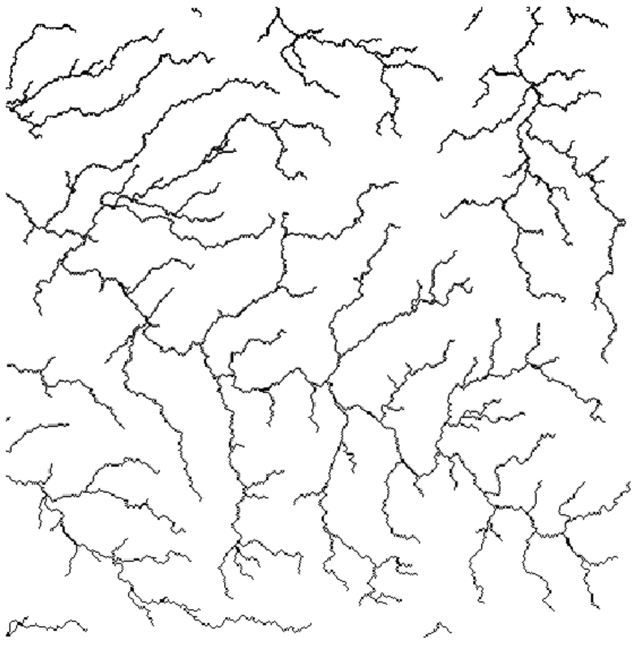


Fig 10: *A Remove Short Streams Output File*

Find Main Stem

The `find_main_stem_path` variable specifies the output path for the raster representing the main stem feature.
The `find_main_stem_command` list contains the command-line arguments for finding the main stem.
The `run_whitebox_command` function is used to execute the command and handle success or failure messages.

```
# Perform find main stem
find_main_stem_path = os.path.join(output_folder,
"Find_Main_Stem_Feature.tif")
find_main_stem_command = [
    whitebox_path,
    "--run=find_main_stem",
    f"--d8_pntr='{flow_direction_path}'",
    f"--streams='{hack_stream_order_path}'",
    f"--output='{find_main_stem_path}'",
    f"--esri_pntr=true"
]
if run_whitebox_command(find_main_stem_command, find_main_stem_path):
    print("Stream to feature conversion completed.")
else:
    print("Error occurred during stream to feature conversion.")
```

Shreve Stream Magnitude

The `shreve_stream_magnitude_path` variable specifies the output path for the Shreve stream magnitude raster.
The `shreve_stream_magnitude_command` list contains the command-line arguments for calculating the Shreve stream magnitude.
The `run_whitebox_command` function is used to execute the command and handle success or failure messages.

```
# Perform shreve stream magnitude

shreve_stream_magnitude_path = os.path.join(
    output_folder, "ShreveStreamMagnitude.tif")
shreve_stream_magnitude_command = [
    whitebox_path,
    "--run=shreve_stream_magnitude",
    f"--d8_pntr='{flow_direction_path}'",
    f"--streams='{extracted_streams_path}'",
    f"--output='{shreve_stream_magnitude_path}'",
    f"--esri_pntr=true"

]
if run_whitebox_command(shreve_stream_magnitude_command,
shreve_stream_magnitude_path):
    print("Stream to feature conversion completed.")
else:
    print("Error occurred during stream to feature conversion.")
```

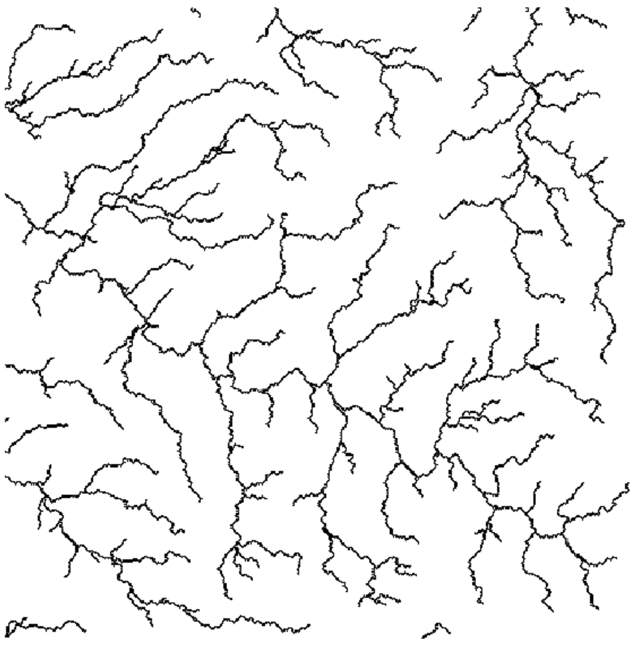


Fig 11: *A Shreve Stream Output File from the input Extracted Streams Output*

Basins

The `basins_path` variable specifies the output path for the basins raster. The subprocess.call function is used to execute the command for generating basins.

```
# Perform basins
basins_path = os.path.join(output_folder, "Basins.tif")
subprocess.call([
    whitebox_path,
    "--run=basins",
    f"--d8_pntr='{flow_direction_path}'",
    f"--esri_pntr=true",
    f"--output='{basins_path}'"
])
```

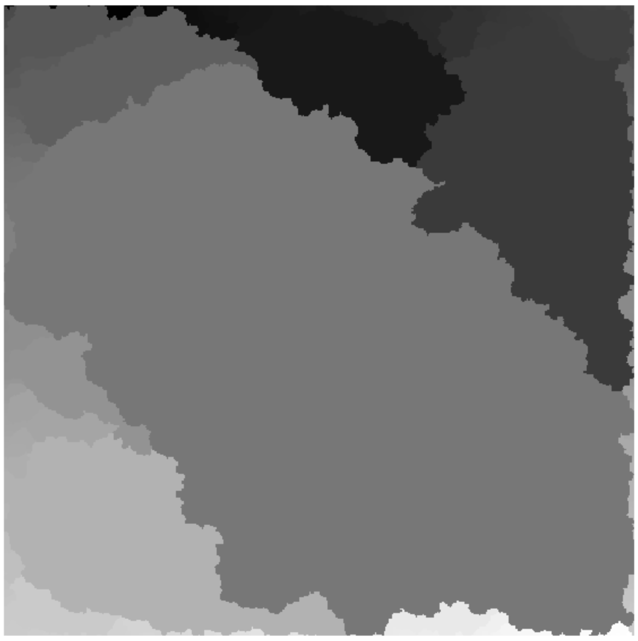


Fig 12: *A Basin Raster File Output*

Sub Basins

The `sub_basins_path` variable specifies the output path for the sub-basins raster. The subprocess.call function is used to execute the command for generating sub-basins.

```
# Perform sub basins
sub_basins_path = os.path.join(output_folder, "SubBasins.tif")
subprocess.call([
    whitebox_path,
    "--run=subbasins",
    f"--d8_pntr='{flow_direction_path}'",
    f"--streams='{extracted_streams_path}'",
    f"--output='{sub_basins_path}'",
    f"--esri_pntr=true"
])
```

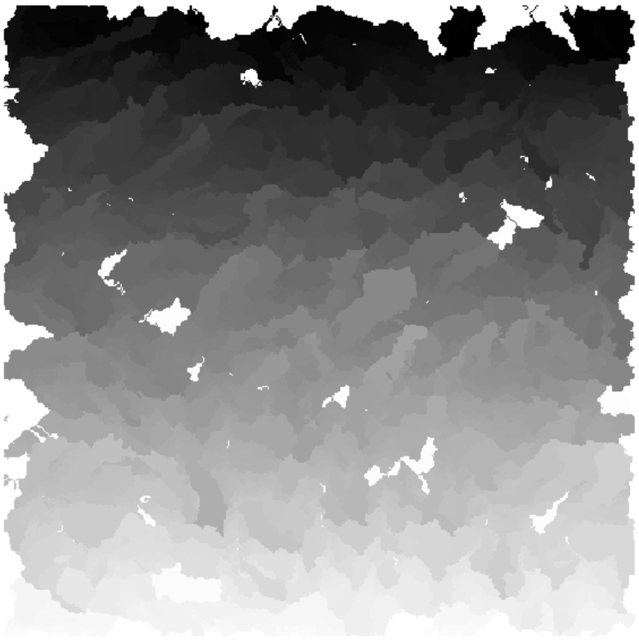


Fig 13: *A Sub Basin Raster Output TIF File Image*

Stream to Feature

The `stream_to_feature_path` variable specifies the output path for the vector representation of streams.
The `stream_to_feature_command` list contains the command-line arguments for converting raster streams to a vector feature.
The `run_whitebox_command` function is used to execute the command and handle success or failure messages.

```
# Perform stream to feature
stream_to_feature_path = os.path.join(output_folder, "StreamToFeature.shp")
stream_to_feature_command = [
    whitebox_path,
    "--run=raster_streams_to_vector",
    f"--d8_pntr='{flow_direction_path}'",
    f"--streams='{stream_order_path}'",
    f"--output='{stream_to_feature_path}'",
    f"--esri_pntr=true"
]
if run_whitebox_command(stream_to_feature_command, stream_to_feature_path):
    print("Stream to feature conversion completed.")
else:
    print("Error occurred during stream to feature conversion.")
```

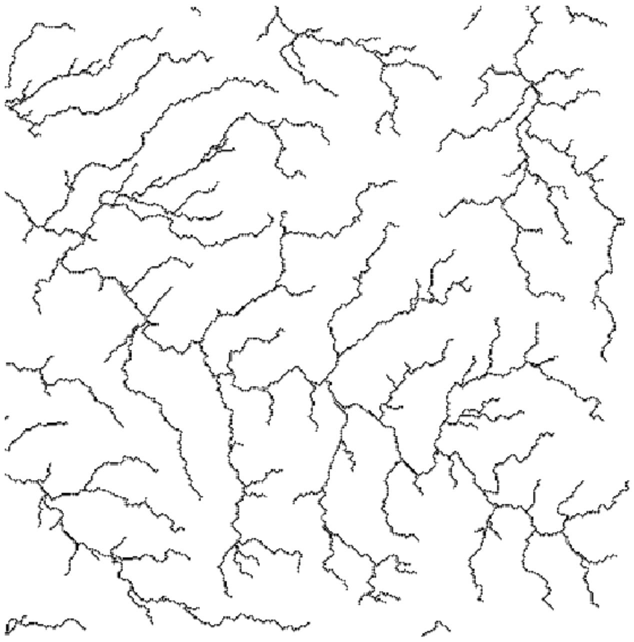


Fig 14: *A Stream Feature Vector File Output from the Input Strahler Stream Order Output*

Extract Nodes

The `extract_nodes_path` variable specifies the output path for the extracted pour points as a shapefile.
The `extract_nodes_command` list contains the command-line arguments for extracting nodes from the stream-to-feature conversion.
The `run_whitebox_command` function is used to execute the command and handle success or failure messages.

```
# Perform extract nodes
extract_nodes_path = os.path.join(output_folder, "PourPoints.shp")
extract_nodes_command = [
    whitebox_path,
    "--run=extract_nodes",
    f"--input='{stream_to_feature_path}'",
    f"--output='{extract_nodes_path}'"
]
if run_whitebox_command(extract_nodes_command, extract_nodes_path):
    print("Stream to feature conversion completed.")
else:
    print("Error occurred during stream to feature conversion.")
```

Snap Pour Points

The `pour_points_path` variable specifies the output path for the snapped pour points as a shapefile.
The `jenson_snap_pour_points_command` list contains the command-line arguments for snapping the pour points to the extracted streams.
The `run_whitebox_command` function is used to execute the command and handle success or failure messages.

```
# Perform snap pour points
pour_points_path = os.path.join(output_folder, "SnapPourPoints.shp")
jenson_snap_pour_points_command = [
    whitebox_path,
    "--run=jenson_snap_pour_points",
    f"--pour_pts='{pour_pnts_path}'",
    f"--streams='{extracted_streams_path}'",
    f"--output='{pour_points_path}'",
    f"--snap_dist=5"
]
if run_whitebox_command(jenson_snap_pour_points_command, pour_points_path):
    print("Snap pour points completed.")
else:
    print("Error occurred during snap pour points.")
```

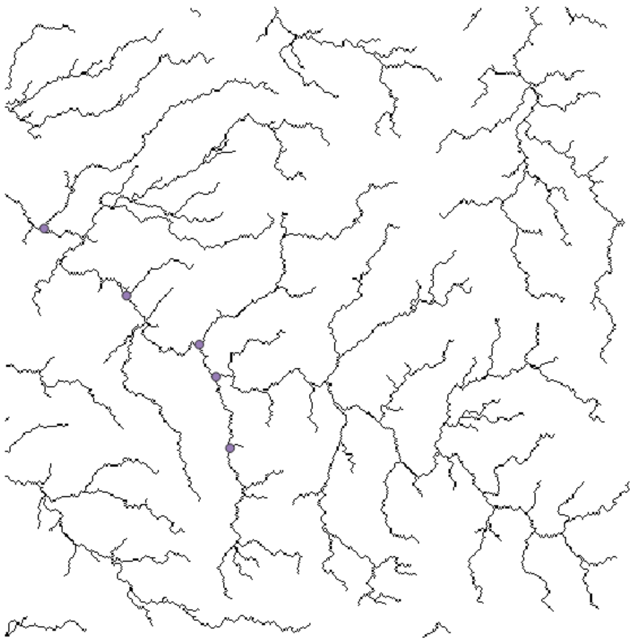


Fig 15: *A Snap Pour Points snapped by 5m Output Vector File Image*

User Input

The code assumes that the user will provide the path to the pour points file using a file dialog. The selected file path is stored in the `pour_pnts_path` variable.

```
# Select the Pour Points file using a file dialog
pour_pnts_path = filedialog.askopenfilename(title="Select Pour Points File")
```

Creating Pour Points

Before running the Watershed tool, it is necessary to create pour points, which define the watersheds for the output.
Pour points are typically located in areas of high accumulated flow, such as points along a stream network.
In Whitebox Tools, there is no automatic method to create a pour point file, so it must be done manually.
The first step in creating a pour point file is to make a blank file using the "Create Blank Outlet Raster" tool.
This tool generates a file that is identical to the input file, except that every cell contains "NoData" values.

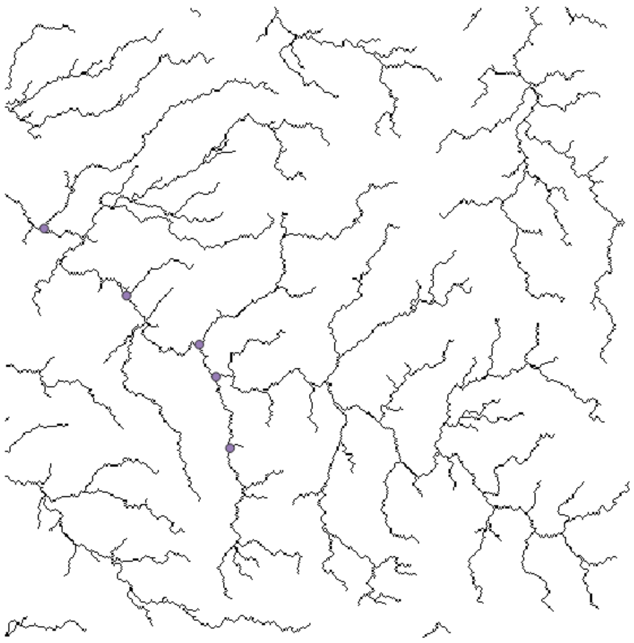


Fig 16: *The Input Pour Points Vector Point File Image manually created from the Stream Features generated in the previous steps*

Watershed Analysis

Once the pour point file is created, the watershed analysis can be performed using Whitebox Tools.
The previously explained steps can be followed to delineate watersheds based on the created pour points.
The Whitebox Tools command for delineating watersheds is defined.
The command is executed using the subprocess module.

```
# Perform watershed
watershed_path = os.path.join(output_folder, "Watershed.tif")
watershed_command = [
    whitebox_path,
    "--run=Watershed",
    f"--d8_pntr='{flow_direction_path}'",
    f"--pour_pts='{extract_nodes_path}'",
    f"--output='{watershed_path}'",
    f"--esri_pntr=true"
]
if run_whitebox_command(watershed_command, watershed_path):
    print("Watershed calculation completed.")
else:
    print("Error occurred during watershed calculation.")
```

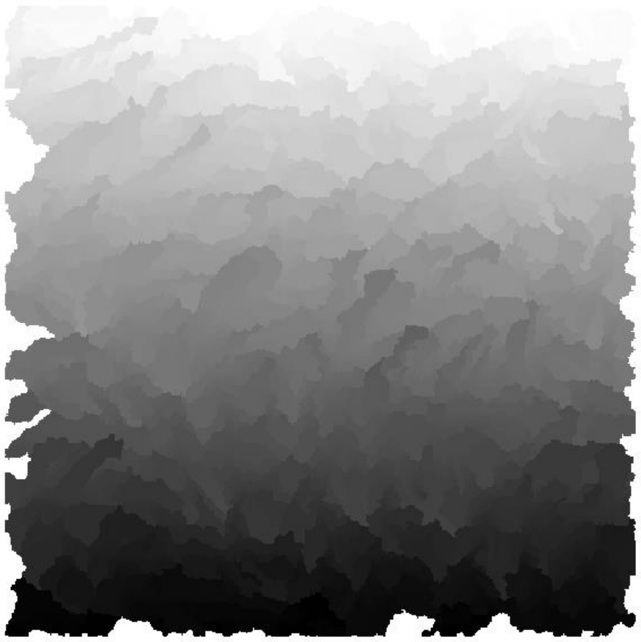


Fig 17: *A Watershed Output Raster File Image*

Displaying Results

The output files from the watershed delineation process are displayed or visualized using appropriate GIS software or libraries.
Example code snippets or instructions can be provided to visualize the results using popular libraries such as Matplotlib or GIS software like QGIS.

Conclusion

This tutorial has provided a comprehensive guide to automate the delineation of watershed analysis using Whitebox Tools in the Python programming language. By following the steps outlined in this tutorial, users can streamline the process of hydrological analysis and efficiently analyze the hydrology of a given area. This automation workflow saves time and effort by leveraging the power of Whitebox Tools and Python programming. Users can further customize and expand upon this tutorial to suit their specific analysis requirements and explore additional features offered by Whitebox Tools.