POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria per l'Ambiente e il Territorio

Specializzazione in Rischi naturali e protezione civile

Tesi di Laurea Magistrale

CONTROLLO E AUTOMAZIONE DELLA MODELLAZIONE IDRAULICA FLUVIALE BIDIMENSIONALE

Relatore: Daniele GANORA

Laureando: Luca AIROLA

A.A. 2020-2021

Abstract:

L’obiettivo della seguente tesi è quello di presentare una metodologia per l’automazione della modellazione idraulica fluviale mono e bidimensionale basata sul modello numerico HEC-RAS, uno dei software più usati nell’ambito della modellazione idraulica fluviale. Il processo di automazione viene gestito mediante dei codici redatti in Python che permettono il controllo delle diverse funzionalità di HEC-RAS.

Il seguente lavoro si suddivide idealmente in due parti. Nella prima verranno descritti i concetti base e presentati gli elementi indispensabili per poter controllare HEC-RAS; in questo contesto sarà introdotto l’HEC-RAS Controller ossia una raccolta di funzioni (apertura progetto, gestione finestre, avvio della simulazione, ecc.) che possono essere richiamate dal codice Python e quindi facilmente iterate o controllate automaticamente.

Nella seconda parte vi sarà un’applicazione ad un caso studio con lo scopo di analizzare determinati scenari di eventi alluvionali che potrebbero potenzialmente colpire il territorio comunale di Moncalieri, eseguendo simulazioni in serie ed in modo automatico e salvando per ognuna di esse opportuni dati che verranno utilizzati in seguito per l’elaborazione di mappe di pericolosità alluvionale.

Si tratta quindi di un metodo complementare al tradizionale approccio di singole simulazioni che può essere molto utile per velocizzare il processo quando si prevede di dover eseguire ripetitivamente una determinata operazione, come nel caso di calibrazione del modello o per la valutazione probabilistica di allagamento di una certa area.

Abstract (English version):

The aim of the following thesis is to present a methodology for the automation of one and two-dimensional river hydraulic modeling based on the HEC-RAS numerical model, one of the most used software in the field of river hydraulic modeling. The automation process is managed by means of codes written in Python that allows the control of the various functions of HEC-RAS.

The following work is divided into two main parts. The first describes the basic concepts and presents the essential elements of HEC-RAS control; in this part, the HEC-RAS Controller, which is a collection of functions (project opening, window management, simulation start, etc.), will be introduced. The Controller can be called up from a Python code and therefore easily iterated or controlled automatically.

In the second part, an application to a case study is presented with the aim of analyzing certain complex flood scenarios that could potentially affect the municipal area of Moncalieri. Simulations are automatically run in series and for each one, appropriate data are saved for the subsequent elaboration of flood hazard maps.

It is therefore a complementary method to the traditional approach of single simulations that can be very useful for speeding up the process when it is expected to repeat a certain operation, as in the case of model calibration or for the probabilistic evaluation of flooding of a certain area.

INDICE

1 – Introduzione

La modellazione idraulica fluviale è un campo dell'ingegneria idraulica che consente di studiare gli effetti che possibili eventi di piena hanno sui territori limitrofi. Ciò è fattibile mediante l'utilizzo di determinati software che, risolvendo le equazioni delle correnti a pelo libero, permettono di ricavare i tiranti idrici raggiunti in alveo ed al di fuori laddove vi siano punti sensibili che si intendono monitorare dal punto di vista del rischio idraulico. Nel seguente lavoro verrà utilizzato il software HEC-RAS che, a seguito di alcune operazioni preliminari come la creazione della geometria dell'alveo, la definizione della scabrezza e delle portate che sollecitano il modello restituisce, come detto in precedenza, i tiranti idrici nel dominio di studio. Si tratta di un programma in costante evoluzione che permette ad oggi di fare simulazioni monodimensionali (flusso idrico che si muove in una sola direzione perpendicolare a delle sezioni di riferimento), bidimensionali (flusso idrico che si muove nelle due direzioni piane) e volendo anche miste.

Spesso le simulazioni idrauliche vengono eseguite manualmente; tuttavia, quando è necessario far variare alcuni parametri del modello e ripetere le simulazioni molte volte è opportuno automatizzare il processo di modellazione per effettuare facilmente simulazioni in serie ed elaborarne in maniera sistematica i risultati. Tale procedura si rivela particolarmente utile per esempio quando si vuol far variare la scabrezza in fase di calibrazione oppure quando si fanno variare gli idrogrammi per effettuare analisi di sensitività. Tutto ciò è reso possibile dal HEC-RAS Controller ovvero una raccolta di funzioni che, implementate all'interno di un codice creato mediante un generico linguaggio di programmazione, ne consente il controllo e l'automazione. E' quindi possibile modificare i file di input, leggere ed analizzare i file di output; in questa tesi è stata pertanto implementata e testata un'interfaccia in linguaggio Python che permettesse di eseguire tutte le precedenti operazioni.

Il lavoro che segue vedrà una prima parte in cui, oltre ad una descrizione dei software utilizzati, si illustreranno i principi del HEC-RAS Controller e le funzioni principali che permettono di automatizzare il processo di modellazione. Nella

seconda parte invece si applicheranno questi concetti ad un caso studio, andando a descrivere la sequenza di azioni che devono essere messe in atto per poter avviare l'esecuzione automatica delle simulazioni e per il salvataggio di determinati dati. Proprio da questi dati verrà mostrato come poter realizzare una mappa statica ovvero una rappresentazione della distribuzione dell'altezza d'acqua raggiunta nel dominio di studio, individuando quindi aree potenzialmente allagabili. Infine si vedrà come estrarre la sequenza temporale dell'altezza d'acqua presente in alcuni punti del dominio che può risultare molto utile al fine di stimare il tempo di allagamento di una certa zona.

2 – Generalità su HEC-RAS

2.1 – Descrizione generale

L'acronimo HEC-RAS si riferisce a Hydrologic Engineering Center's - River Analysis System ed è un software di modellazione idraulica di pubblico dominio sviluppato in collaborazione con U.S Federal Government. Permette di fare simulazioni monodimensionali in moto permanente e moto vario, simulazioni bidimensionali in moto vario, modellazione di trasporto solido e modellazioni riguardanti la qualità dell'acqua (trasporto di nutrienti).
Al momento dell'apertura del software appare la finestra principale di HEC-RAS che, lavorando con la versione 5.0.7, corrisponde a quella riportata in Figura 2.1.

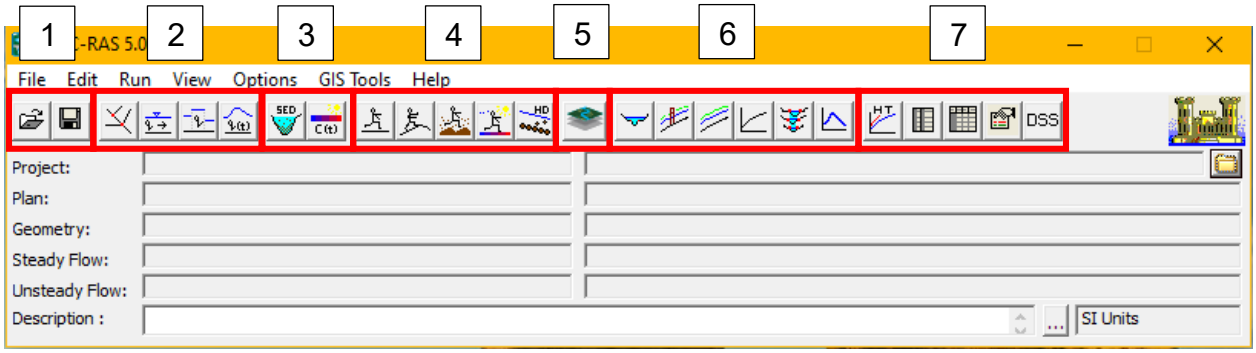


Figura 2.1: Finestra principale di HEC-RAS

Vi sono una serie di comandi suddivisi in blocchi che permettono di svolgere tutta una serie di operazioni quali:

1. Aprire un progetto già esistente e salvare le modifiche apportate;
2. Visualizzare e modificare la geometria, i dati di moto permanente, i dati di moto semi-permanente e i dati di moto vario;
3. Inserire dati sul trasporto di sedimenti e trasporto di sostanze disciolte;
4. Eseguire simulazioni in moto permanente, in moto vario, sul trasporto di sedimenti, trasporto di sostanze disciolte e valutazione dell'erosione al piede delle pile dei ponti;
5. Apertura del RAS-Mapper (sistema GIS interno al software);

6. Visualizzare output in forma grafica, sezioni, profili, idrogrammi e scala di deflusso delle portate;
7. Visualizzare output in forma tabellare

Le righe successive fanno riferimento ai file che devono essere predisposti al fine dell'esecuzione delle simulazioni, questi sono:

1. Project, file generale con estensione ".prj";
2. Plan, file riferito a ciascuna simulazione avente estensione ".p0X" (da .p01 a .p99);
3. Geometry, file riferito alla geometria avente estensione ".g0X" (da .g01 a .g99);
4. Steady Flow, file dei dati di moto permanente avente estensione ".p0X" (da .p01 a .p99);
5. Unsteady Flow, file dei dati di moto vario avente estensione ".u0X" (da .u01 a .u99).

Nell'ultima riga, invece, è presente uno spazio per l'inserimento di una descrizione del progetto e nell'angolo alla destra vi è indicato il sistema di riferimento che dev'essere settato inizialmente (andando su Option → Unit System) scegliendo tra "US Customary" e System International (Sistema di riferimento Internazionale o sistema di riferimento metrico).

Nei paragrafi successivi si analizzeranno i concetti base e le principali equazioni che stanno alla base della modellazione monodimensionale e bidimensionale

2.2 – Modellazione monodimensionale

Il modello monodimensionale è lo schema numerico più semplice. Dal punto di vista geometrico e computazionale si basa essenzialmente sulla generazione di sezioni distribuite omogeneamente lungo l'asse fluviale, posizionate in corrispondenza di punti notevoli e con una direzione il più possibile ortogonale alla direzione del flusso (Figura 2.2) [1].

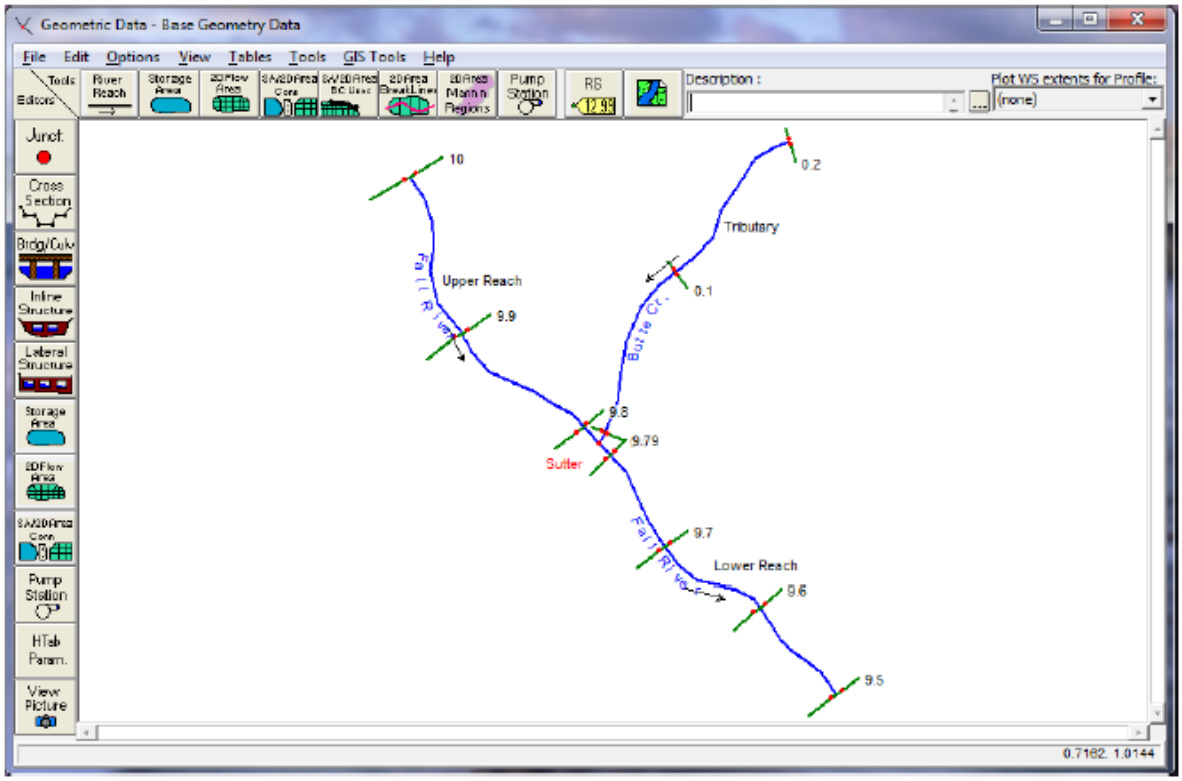


Figura 2.2: Editor geometria

Cliccando su Cross Section si apre l'Editor rappresentato in Figura 2.3 in cui sono riportati gli elementi necessari per definire una sezione.

[1] Figura estratta dal manuale HEC-RAS 5.0, apribile dal HELP generale (Figura 2.1)

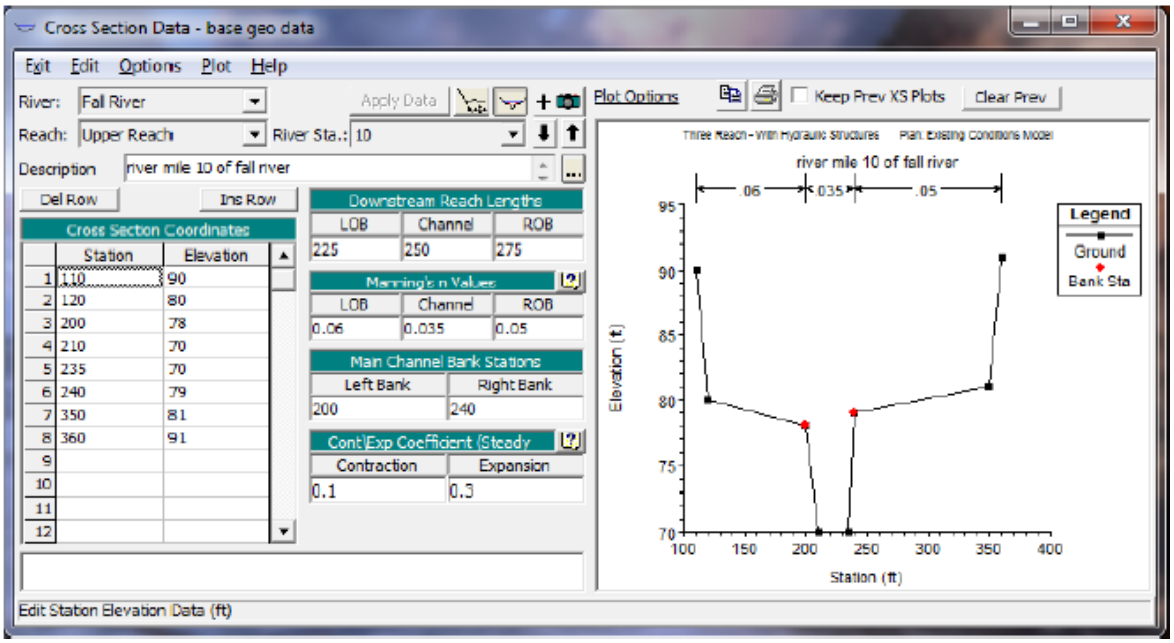


Figura 2.3: Editor cross section

Osservando una generica sezione di un alveo fluviale si possono distinguere tre zone caratterizzanti:

1. Main channel, porzione centrale dell'alveo
2. LOB (Left OverBank), golena sinistra
3. ROB (Right OverBank), golena destra

La loro disposizione e formazione può avvenire o a seguito di rilievo topografico, quindi con la determinazione delle coordinate X,Y,Z, oppure basandosi su un DTM (Digital Terrain Model) ovvero un file contenente le quote altimetriche di una certa superficie che ne permette l'automatica generazione.
Su di esse si basa poi la risoluzione delle equazioni che verranno mostrate nei successivi sottoparagrafi.

2.2.1 – Moto permanente

Si riportano di seguito le equazioni, implementate nel software, per la risoluzione del moto in condizioni permanenti[2]. La determinazione del profilo di corrente avviene mediante un processo iterativo chiamato “standard step”. Indicando con 1 la sezione di monte e 2 la sezione di valle, l’equazione può essere scritta nella seguente forma:

Equazione 2.1: equazione di conservazione dell’energia

$$Z_2 + Y_2 + \frac{\alpha_2 \cdot V_2^2}{2g} = Z_1 + Y_1 + \frac{\alpha_1 \cdot V_1^2}{2g} + h_e$$

Dove:

- Z_i = quote di fondo;
- Y_i = tiranti idrici;
- α_i = coefficienti di ragguaglio delle altezze cinetiche;
- V_i = velocità;
- g = accelerazione di gravità;
- h_e = perdita di carico totale nel tratto

[2] Maggiori informazioni sulle equazioni che verranno mostrate anche nei successivi paragrafi in merito alla modellazione bidimensionale possono essere ritrovate sul manuale di HEC-RAS presente online oppure accedendo al HELP presente nella facciata principale del software.

Si riporta in Figura 2.4 una rappresentazione schematica dell'equazione sopra indicata:

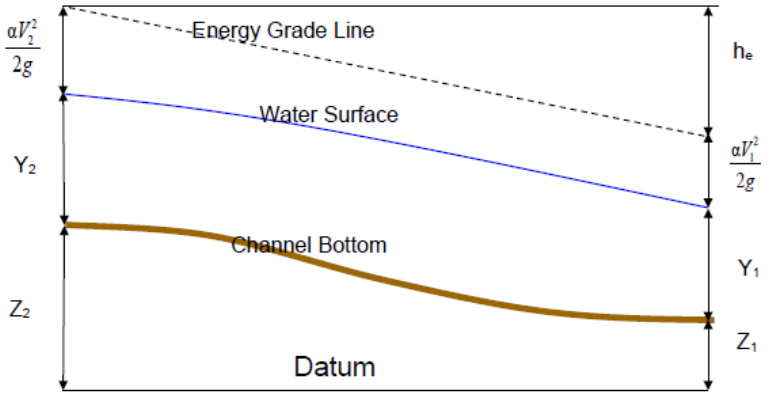


Figura 2.4: Schema esemplificativo dell'equazione 2.1

Ragionando sempre tra due sezioni, le perdite di carico sono attribuibili a due fattori principalmente, uno dovuto alle perdite distribuite lungo il tratto dovuto all'attrito, l'altro relativo alle contrazioni ed espansioni della vena liquida.
L'espressione per il calcolo viene di seguito riportata:

Formula 2.2: espressione per il calcolo delle perdite di carico

$$h_e = L \cdot S_f + C \cdot \left| \frac{\alpha_2 \cdot V_2^2}{2g} - \frac{\alpha_1 \cdot V_1^2}{2g} \right|$$

Dove:

- L = lunghezza del tratto mediata sulle portate;
- S_f = cadente della linea dei carichi totali;
- C = coefficiente di contrazione ed espansione che variano in base alla tipologia di corrente. Per correnti lente variano tra 0.3 e 0.5 mentre per correnti veloci tra 0.01 e 0.03;

La determinazione del parametro "L" avviene tenendo in considerazione le tre diverse distanze, tra una sezione e l'altra, riferite alle aree individuabili in una sezione (LOB, Ch, ROB) e mediate con le rispettive portate di competenza; la

considerazione di queste tre lunghezze consente di tener conto dell'andamento curvilineo dell'asse fluviale. L'espressione pertanto è:

Formula 2.2: espressione per il calcolo del parametro L

$$L = \frac{L_{LOB} \cdot Q_{LOB} + L_{Ch} \cdot Q_{Ch} + L_{ROB} \cdot Q_{ROB}}{Q_{LOB} + Q_{Ch} + Q_{ROB}}$$

Dove:

- L_{LOB} = lunghezza del tratto relativo alla golena di sinistra;
- L_{Ch} = lunghezza del tratto relativo al canale principale;
- L_{ROB} = lunghezza del tratto relativo alla golena di destra;
- Q_{LOB} = portata media transitabile sulla golena di sinistra;
- Q_{Ch} = portata media transitabile nel canale centrale;
- Q_{ROB} = portata media transitabile sulla golena di destra.

La determinazione delle altezze cinetiche richiede la suddivisione del flusso in porzioni all'interno delle quali si possa considerare la velocità uniformemente distribuita.
L'approccio standard utilizzato da HEC-RAS consiste nel discretizzare le sezioni rispetto al valore della scabrezza come mostrato in Figura 2.5.

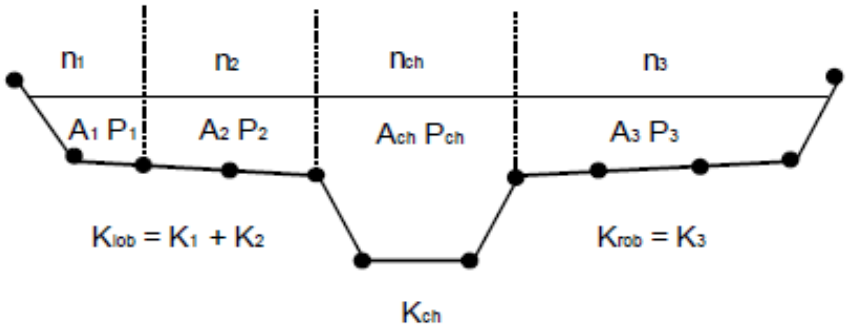


Figura 2.5: esempio di discretizzazione di una sezione eseguita da HEC-RAS

La discretizzazione in base alla scabrezza è dovuta al fatto che viene poi utilizzata la formula di Manning per determinare la conduttanza (conveyance), parametro che dipende dalla distribuzione di velocità all'interno della sezione:

Formula 2.3: espressione per il calcolo della portata e conveyance

$$Q = K \cdot \sqrt{S_f} \quad con \quad K = \frac{1.486}{n} \cdot A \cdot R^{2/3}$$

Dove:

- K = conduttanza (m^3/s);
- n = coefficiente di scabrezza di Manning ($m^{-1/3}/s$);
- A = area di flusso (m^2);
- R = raggio idraulico (m)
- S_f = pendenza della linea dei carichi totali

Determinati i "K" per le varie porzioni della sezione essi devono poi essere sommati in modo da ricavare i rispettivi valori per le aree laterali (K_{LOB} e K_{ROB}), K_{CH} viene generalmente calcolato come elemento unico senza suddivisioni. Il valore finale "Kt" sarà poi dato dalla somma dei tre contributi ($K_{LOB} + K_{CH} + K_{ROB}$).
Si può così determinare il valore di velocità per ciascun tratto (Formula 2.4) a partire dalla portata calcolata con la Formula 2.3 e l'area di flusso e di conseguenza determinare le rispettive altezze cinetiche.

Formula 2.4: espressione per il calcolo velocità

$$V_i = Q_i / A_i$$

A partire da tali altezze, come riportato in Figura 2.6 dev'essere poi stimata un'altezza cinetica media che, analiticamente, può essere calcolata con la formula

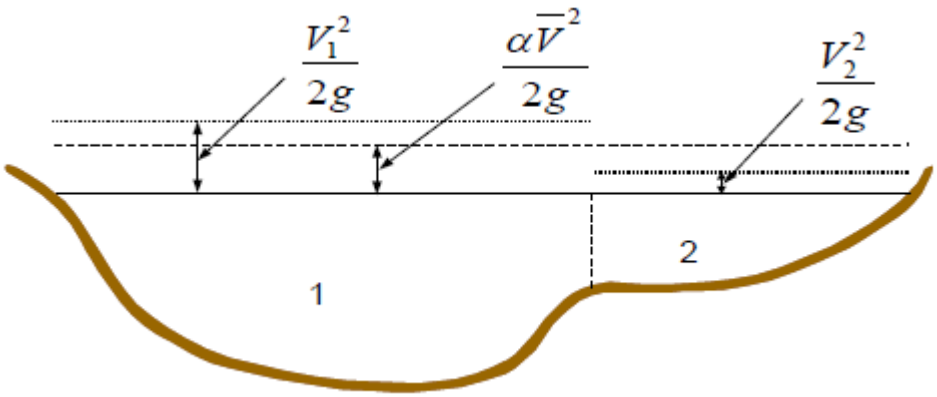


Figura 2.6: Esempio riguardo la determinazione dell'altezza cinetica media

2.5.

Formula 2.5: espressione per il calcolo dell'altezza cinetica media

$$\alpha \frac{V_{med}^2}{2g} = \frac{\sum Q_i \frac{V_i^2}{2g}}{\sum Q_i}$$

Dove:

- V_{med} = velocità media di una data sezione;
- Qi = portata che transita nella i-esima porzione della sezione;
- Vi = velocità del flusso che transita nella i-esima sezione;
- g = accelerazione di gravità;
- α = coefficiente di ragguaglio delle velociità calcolabile con la formula 2.6

Formula 2.6: espressione per il calcolo del coefficiente di ragguaglio

$$\alpha = \frac{(A_t)^2 \left[\frac{K_{LOB}^3}{A_{LOB}^2} + \frac{K_{Ch}^3}{A_{Ch}^2} + \frac{K_{ROB}^3}{A_{ROB}^2}\right]}{K_t^3}$$

Dove:

- At = sezione idrica totale;
- K_{LOB}, K_{Ch}, K_{ROB} = conduttanze riferite alle tre porzioni (golena sx, canale principale, golena dx);
- A_{LOB}, A_{Ch}, A_{ROB} = sezioni idriche relative alle tre porzioni (golena sx, canale principale, golena dx);
- Kt = conduttanza complessiva.

Infine, la pendenza della linea dei carichi totali può essere determinata secondo una delle formule sotto riportate:

Formula 2.6: espressioni per il calcolo del parametro “Sf”

$$S_f = \left(\frac{Q_1 + Q_2}{K_1 + K_2}\right)^2$$

$$S_f = \frac{S_{f1} + S_{f2}}{2}$$

$$S_f = \sqrt{S_{f1} \cdot S_{f2}}$$

$$S_f = \frac{2\left(S_{f1} \cdot S_{f2}\right)}{S_{f1} + S_{f2}}$$

L’equazione 2.1 normalmente utilizzata per la risoluzione del problema non è valida nel caso in cui vi sia variazione di regime di moto, ovvero quando si verifica il passaggio attraverso l’altezza critica. Questi casi che possono verificarsi in presenza di confluenze, restrizioni della sezione idrica o variazioni repentine di pendenza vengono affrontate facendo ricorso all’equazione globale di equilibrio dinamico riportata di seguito:

Formula 2.5: equazione globale di equilibrio dinamico

$$P_2 - P_1 + W_x - F_f = Q \cdot \rho \cdot \Delta V_x$$

Dove:

- $P_{1,2}$ = spinta idrostatica in corrispondenza delle sezioni 1 e 2;
- W_X = componente della forza peso nella direzione del moto;
- F_f = forza resistente dovuta all'attrito;
- Q = portata;
- P = densità dell'acqua;
- ΔV_X = variazione di velocità fra le sezioni 1 e 2 nella direzione del moto;

Esplicando i termini si ottiene l'equazione effettivamente implementata ed utilizzata dal software:

Formula 2.6: equazione globale esplicitata

$$\frac{\beta_2 \cdot Q_2^2}{gA_2} + A_2 Y_2 + \left(\frac{A_1 + A_2}{2}\right) \cdot L \cdot S_0 - \left(\frac{A_1 + A_2}{2}\right) \cdot L \cdot S_f = \frac{\beta_1 \cdot Q_1^2}{gA_1} + A_1 Y_1$$

Dove:

- $A_{1,2}$ = area bagnata relativa alle sezioni 1 e 2;
- L= distanza fra le sezioni 1 e 2 misurate lungo la direzione x;
- $\beta_{1,2}$ = coefficiente di ragguaglio che tiene conto delle variazioni nella distibuzione della velocità in canali irregolari;
- S_0 = pendenza del canale;
- S_f = cadente della linea dei carichi totali;
- Y_i = tiranti idrici

2.2.1 – Moto vario

Le leggi fisiche che regolano il moto vario di una corrente a pelo libero sono un'equazione di conservazione della massa (Formula 2.7) e un'equazione di conservazione della quantità di moto (Formula 2.8) nell'intervallo compreso fra due sezioni consecutive.

Formula 2.7: equazione di conservazione della massa

$$\frac{\partial Q}{\partial A} + \frac{\partial A}{\partial t} = 0$$

Formula 2.8: equazione di conservazione della quantità di moto

$$\frac{1}{A}\frac{\partial Q}{\partial t} + \frac{1}{A}\frac{\partial\left(\frac{Q^2}{A}\right)}{\partial x} + g\frac{\partial h}{\partial x} - g\left(S_0 - S_f\right) = 0$$

Dove:

- Q = portata;
- t = tempo;
- h = tirante idrico;
- g = accelerazione gravitazionale;
- Sf = pendenza della linea dei carichi totali;
- S_0 = pendenza del canale;
- x = direzione del flusso (moto monodimensionale).

Tali equazioni non presentano una soluzione analitica, ma possono essere risolte numericamente ricavando i valori di portata e tirante per ciascuna sezione e ad ogni passo temporale. Inoltre sono applicabili sotto le seguenti ipotesi:

- Distribuzione idrostatica della pressione;
- Alveo poco pendente (<10%);
- Alveo prismatico, ovvero non vengono considerate le variazioni di sezione e pendenza tra due sezioni trasversali;

- Utilizzo dell'equazione di Manning per la definizione delle perdite di carico;
- Velocità e accelerazioni verticali trascurabili rispetto alle stesse nella direzione del moto;
- Incomprimibilità del fluido.

Le equazioni utilizzate da HEC-RAS sono le medesime riportate in precedenza con in aggiunta il contributo dato dal deflusso laterale:

Formula 2.9: equazione di conservazione della massa

$$\frac{\partial Q}{\partial A} + \frac{\partial At}{\partial t} - q_1 = 0$$

Formula 2.10: equazione di conservazione della quantità di moto

$$\frac{\partial Q}{\partial t} + \frac{\partial VQ}{\partial x} + gA\left(\frac{\partial z}{\partial x} + S_f\right) = 0$$

Dove:

- At = area bagnata;
- q_1 = flusso laterale per unità di lunghezza
- $\frac{\partial z}{\partial x}$ = pendenza del fondo
- Pendenza piezometrica esprimibile tramite la formula di Manning (Formula 2.11)

Formula 2.11: equazione di Manning

$$S_f = \frac{Q|Q|n^2}{2.208R^{4/3}A^2}$$

Dove:

- n = coefficiente di scabrezza di Manning
- R = raggio idraulico
- A = area bagnata

2.3 – Modellazione bidimensionale

La modellazione bidimensionale, che sarà quella utilizzata nel seguente lavoro, presenta una serie di vantaggi nei confronti della modellazione 1D, quali:

- Il maggior dettaglio del deflusso nelle due direzioni piane, specialmente in quelle zone ricche di fabbricati dove con la semplice modellazione 1D non si riesce a simulare con precisione la direzione di deflusso della corrente;
- Necessità di minor tempo per la stesura della geometria, si tratta infatti di tracciare il perimetro del dominio di studio e all'interno di esso effettuare una discretizzazione spaziale (Mesh) in sotto-aree elementari. Ciò avviene basandosi su un DTM dell'area che, grazie alle tecnologie di rilevazione presenti al giorno d'oggi, hanno un'elevata risoluzione spaziale garantendo un output altrettanto dettagliato. Lo stesso aggiornamento del software ha permesso di passare da una maglia di calcolo strutturata e non flessibile ad una non strutturata e flessibile permettendo di essere maggiormente dettagliati laddove necessario, riducendo la dimensione delle celle e di impostare una risoluzione più grossolana laddove non vi è necessità di dettaglio.

Tutto ciò comporta però tempi computazionali più lunghi in funzione del numero di celle e della discretizzazione temporale voluta.
L'unica tipologia di moto sfruttabile in campo bidimensionale è il moto vario (unsteady flow), ciò implica la necessità di avere idrogrammi da inserire come input.
Nel sotto-paragrafo successivo si riportano le equazioni alla base del moto bidimensionale implementate in HEC-RAS.

2.3.1 – Moto vario

Le equazioni che si riportano di seguito sono conosciute come equazioni 2D di De Saint Venant dette anche Shallow Water Equations ottenibili mediando le equazioni del moto più complete di Navier Stokes rispetto alla direzione verticale e imponendo la conservazione della massa e della quantità di moto all'interno di un volume di controllo.

Formula 2.11: equazione di conservazione della massa

$$\frac{\partial H}{\partial t} + \frac{\partial (hu)}{\partial x} + \frac{\partial (hv)}{\partial y} = Q_{in} - Q_{out}$$

Dove:

- $\frac{\partial H}{\partial t}$ = variazione del livello idrico nel volume di controllo;
- $\frac{\partial (hu)}{\partial x} + \frac{\partial (hv)}{\partial y}$ = variazione spaziale della portata in ingresso e uscita dove "u" e "v" sono i vettori velocità nelle due direzioni principali x e y;
- $Q_{in} e\ Q_{out}$ = portate in ingresso e uscita dal volume di controllo.

Formula 2.12: equazioni di conservazione della quantità di moto

$$\frac{\partial u}{\partial t} + \left(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}\right) = -g\frac{\partial H}{\partial x} + v_t\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) - c_f u + fv$$

$$\frac{\partial v}{\partial t} + \left(u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y}\right) = -g\frac{\partial H}{\partial y} + v_t\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) - c_f v + fu$$

Dove:

- $\frac{\partial u}{\partial t}$ = accelerazione locale;
- $\left(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}\right)$ = accelerazione convettiva;
- $g\frac{\partial H}{\partial x}$ = gradiente della pressione idrostatica;
- $v_t\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$ = termine legato alla viscosità;

- $c_f u$ = resistenza dell'alveo
- $f v$ = parametro di Coriolis

Le equazioni riportate sono applicate ad ogni cella della mesh mediante metodi numerici alle differenze o volumi finiti.

3 – Generalità su Python

3.1 – Tipologia di linguaggio

Nel corso degli anni ed in particolare a partire dalla prima metà del secolo scorso si svilupparono, in ambito informatico, diversi sistemi per l'analisi e l'elaborazione di dati.
I primi microcalcolatori apparsi nella seconda metà del secolo, permettevano l'inserimento di istruzioni e dati esclusivamente sotto forma numerica attraverso sequenze di zero ed uno (bit) e sfruttando quindi un linguaggio detto di "basso livello". Col passare degli anni vennero sviluppati linguaggi di programmazione con una sintassi più simile al linguaggio umano che li resero di più semplice approccio e a cui venne dato il nome di linguaggi di "alto livello". Anche in questo caso il programma veniva poi "compilato" per ottenere la traduzione nel linguaggio macchina, ovvero di basso livello. Tra i molti linguaggi di programmazione di alto livello oggi presenti vi è Python che è stato utilizzato in questo lavoro come interprete per l'automazione di HEC-RAS e per altre analisi di dati.

3.2 – Vantaggi

Python, a differenza di altri linguaggi di programmazione, presenta una serie di vantaggi. Il primo sicuramente è quello di essere gratuito e scaricabile liberamente[3] per tutta una serie di piattaforme quali Windows, MacOS o Linux. Permette di assegnare dinamicamente i valori alle variabili senza la necessità di dover effettuare un'inizializzazione; questo è indubbiamente un vantaggio che permette di semplificare e ridurre la lunghezza del codice rendendolo più compatto e leggibile. Anche l'utilizzo delle liste (vettori) e le liste di liste (matrici) è diverso rispetto ad altri linguaggi, infatti sebbene sia necessario specificare attraverso un'inizializzazione la

[3] Il download di Python può essere eseguito accedendo al sito ufficiale: https://www.python.org/

volontà di creare un vettore o una matrice non è richiesta né la specificazione della dimensione né della tipologia di dato che può essere intero (int), decimale (float) o letterale (stringa); anche qui si può parlare di allocazione dinamica (questo . Oltre a questi primi aspetti è importante sottolineare la modularità di Python; esistono infatti tutta una serie di moduli che possono essere installati ed importati nell'ambiente di programmazione e consentono di svolgere tutta una serie di operazioni che non sarebbe possibile svolgere con la versione base. Dell'installazione e dei tipi di moduli che sono stati utilizzati in questo lavoro si tratterà nel paragrafo successivo ma è importante anticipare come, con l'utilizzo di appositi moduli, sia possibile analizzare particolari tipi di file (anche questi verranno trattati nei paragrafi successivi) che con altri linguaggi di programmazione non sarebbe possibile analizzare; questo rappresenta un ulteriore ed importante vantaggio a favore di Python.

3.3 – Avviare Python e IDLE in Windows

Dopo aver eseguito l'installazione di Python è possibile avviarlo ricercando, nella barra di ricerca a fianco del pulsante start, l'espressione IDLE che è l'acronimo di "Integrated Development Learning Environment" ovvero ambiente integrato per lo sviluppo e l'apprendimento. Come suggerisce il nome, si tratta di un software che mette a disposizione del programmatore un ambiente di lavoro nel quale sono presenti funzioni utili come la colorazione del codice, l'evidenziazione di "errori di sintassi" in tempo reale ecc. La prima finestra che appare è la "Shell di IDLE" ovvero l'ambiente in cui si possono impartire direttamente i comandi oppure osservare il risultato di programmi eseguiti da file e aperti nell'Editor (Figura 3.1).

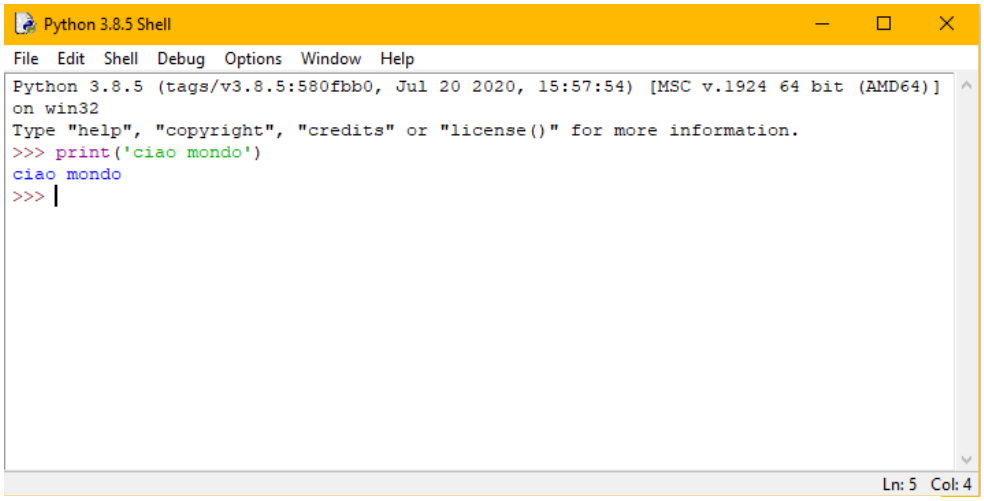


Figura 3.1: Finestra “Shell” di Python

Lo svantaggio di lavorare solamente nella Shell è che non è possibile salvare i risultati e quindi andrebbero persi nel momento in cui si chiude Python. Pertanto quando si deve sviluppare un codice che si intende poi salvare per essere riutilizzato quello che si fa è di aprire un nuovo file cliccando su File → New File. (Figura 3.2)

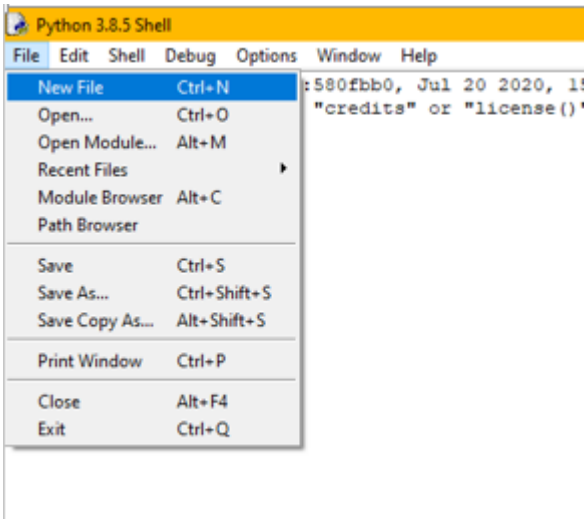


Figura 3.2: Apertura di un nuovo file dalla Shell di Python

Si aprirà così una nuova finestra dell'IDLE corredata da un menu con alcune voci diverse rispetto a quelle della Shell. (Figura 3.3)

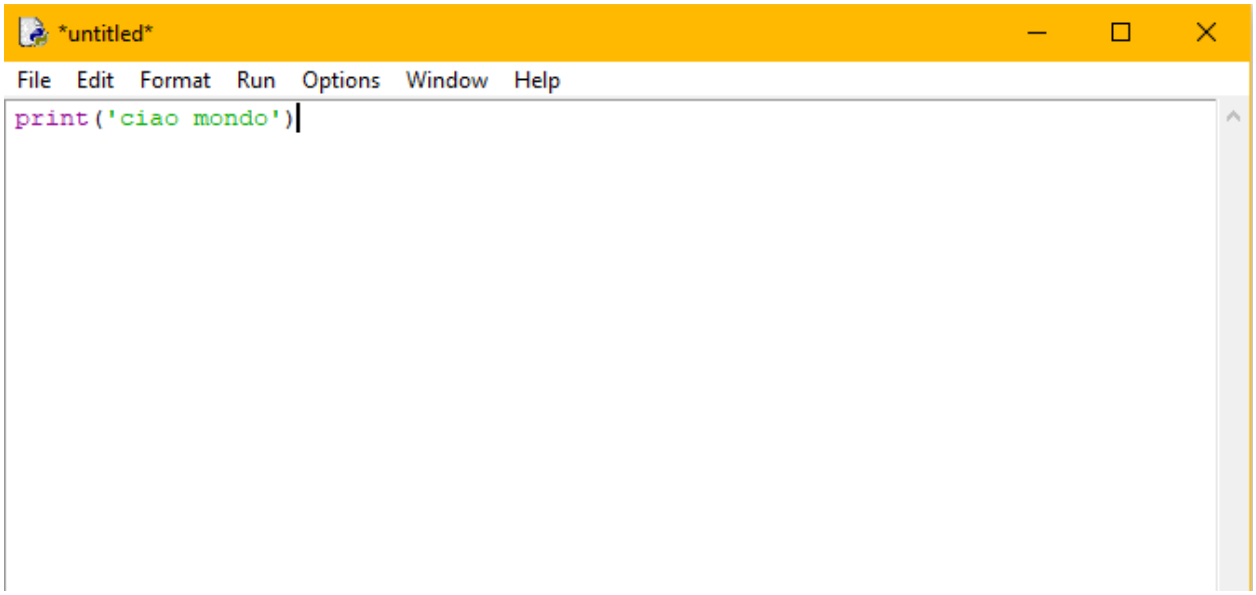


Figura 3.3: Apertura del File in cui sviluppare il codice

Una volta aperto è quindi possibile iniziare a sviluppare il codice, salvarlo ed eventualmente riprenderlo in secondo momento andando su File → Open. Quando si ritiene di aver concluso o semplicemente si vuole osservare la bontà del funzionamento di una parte del codice stesso è necessario eseguire il programma cliccando su Run → Run Module. (Figura 3.4)

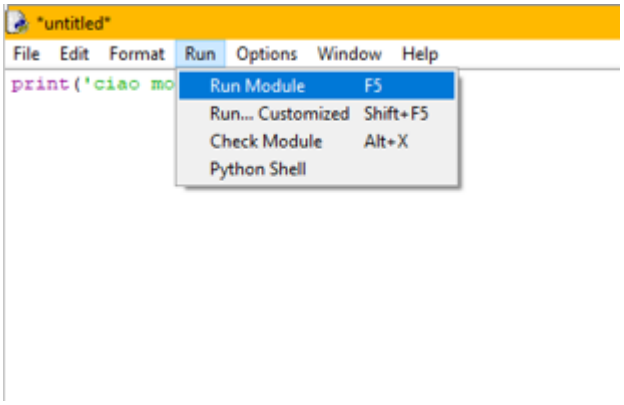


Figura 3.4: Modalità di avvio dell'esecuzione del codice

Verrà richiesto di salvare le ultime modifiche e successivamente, in assenza di errori di sintassi o di esecuzione, verrà riaperta la finestra di Shell con l'output. (Figura 3.5)

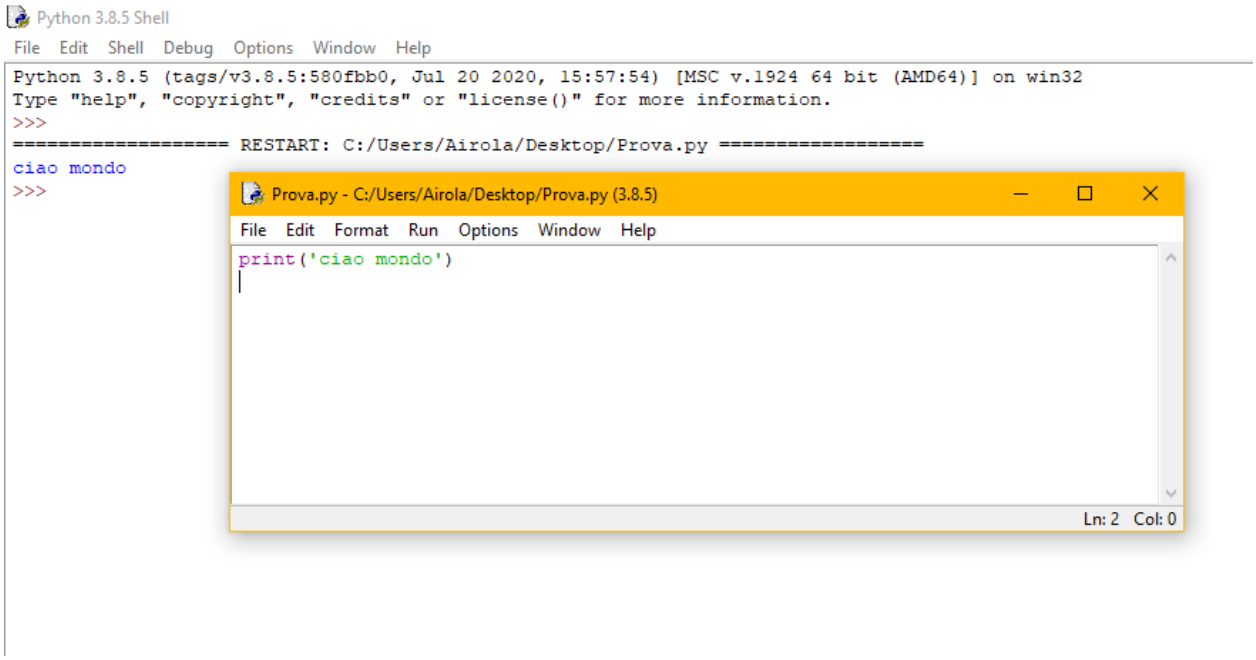


Figura 3.5: Visualizzazione delle due finestre principali con l'output nella Shell

3.4 - Installazione dei moduli e loro funzionalità

L'utilizzo dei moduli in Python sono di fondamentale importanza. Si tratta di raccolte di codici costituiti da funzioni, variabili e costanti che, una volta importati all'interno del programma, consentono lo svolgimento di svariati compiti.

Al momento dell'installazione di Python vengono automaticamente scaricati una serie di moduli che sono salvati all'interno della "libreria standard" e possono essere immediatamente utilizzati a seguito della semplice importazione nel codice.

Esempio di importazione di un modulo:

```
import math
```

Tra di essi vi sono quelli per la gestione del testo, gestione di file e cartelle, reti e internet, matematica, algoritmi, database, sviluppo, interfacce grafiche, multimedia, gestione di data e ora e molti altri. Altri, invece, possono derivare da librerie esterne e pertanto richiedono l'installazione prima dell'utilizzo, come riportato nella Figura 3.6.

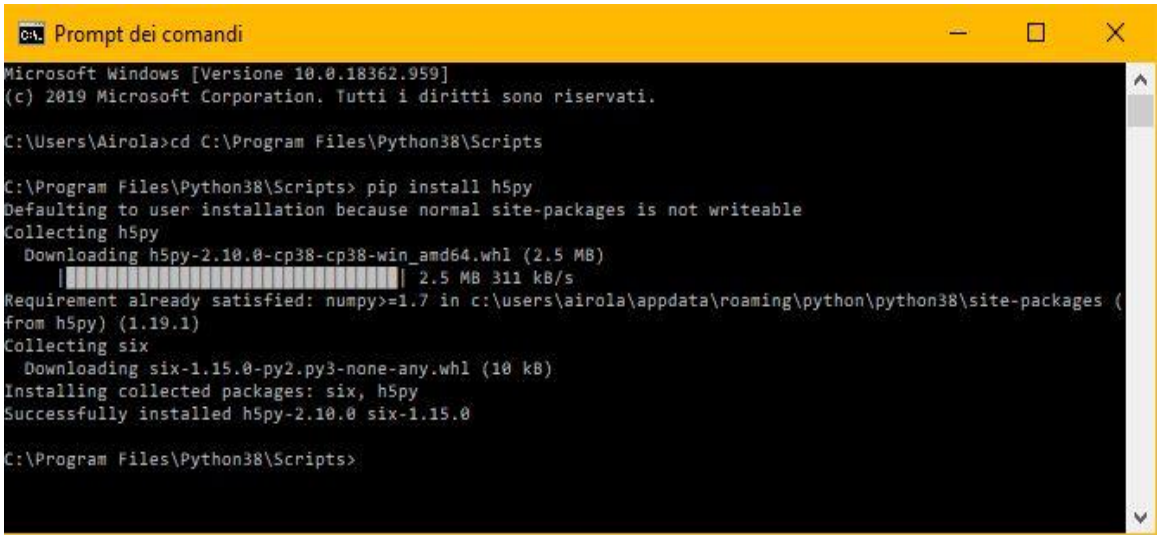


Figura 3.6: Installazione del modulo h5py

Con riferimento alla terza e quarta riga di Figura 3.6 il processo di installazione prevede:

1. Apertura del terminale di sistema → in Windows ricercare CMD o Prompt dei comandi

2. Cambio della directory → scrivendo "cd" seguito dal percorso che porta all'interno della cartella Scripts contenuta nella cartella principale di Python generata al momento dell'installazione del software, come mostrato in Figura 3.7;

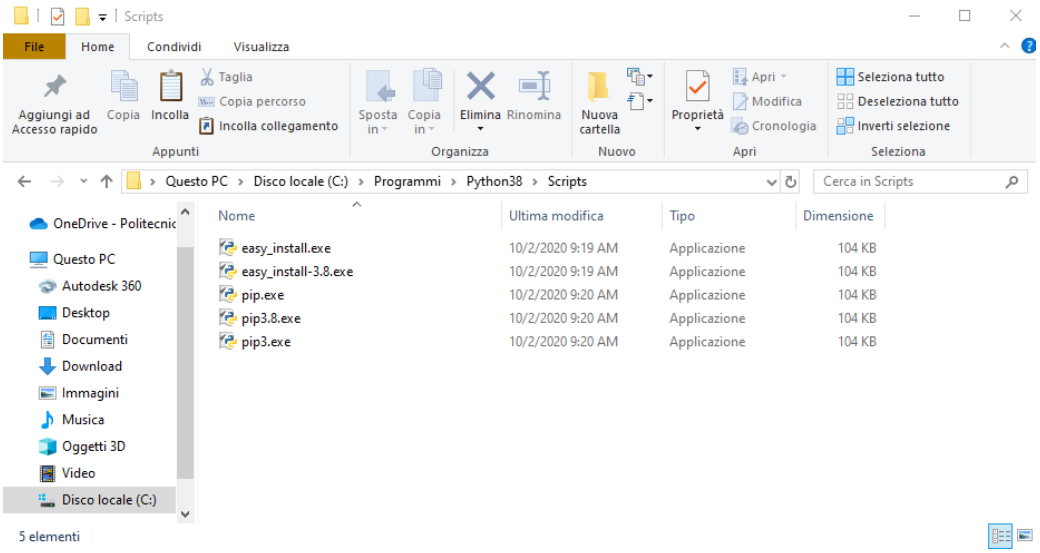


Figura 3.7: Percorso di salvataggio della cartella Scripts

3. Utilizzo del comando “pip install” seguito dal nome del modulo;

4. Premere invio per iniziare l’installazione.

I moduli che verranno utilizzati in questo lavoro sono di seguito elencati e decritti:

- ✓ **win32com.client**, permette di interfacciarsi con il software HEC-RAS mediante l’HEC-RAS Controller ovvero una delle tre classi dell’HEC-RAS API (Application Programming Interface), contenente una serie di “blocchi di codici” o “funzioni” create appositamente per facilitare e controllare le operazioni ed il comportamento di HEC-RAS nella modifica dei file di input e nella lettura dei file di input e output;
- ✓ **openpyxl**, modulo installato per mezzo del comando “pip install” che permette di interfacciarsi con documenti Excel. In particolare la funzione “load_workbook” permette di leggere dati da un file Excel già esistente mentre, “Workbook”, può essere utilizzata per creare e salvare in un apposito file Excel determinati dati;

- ✓ **h5py**, modulo installato per mezzo del comando “pip install” che permette di leggere i file “.hdf” contenenti gli output generati da HEC-RAS e, di conseguenza, estrarre quelli voluti;
- ✓ **numpy,** modulo installato per mezzo del comando “pip install” che permette di lavorare con vettori e matrici;
- ✓ **os**, modulo presente nella libreria standard che permette di gestire file e cartelle;
- ✓ **math,** modulo presente nella libreria standard utilizzato, in questo contesto, nell’arrotondamento di numeri in virgola mobile.

Si riporta la parte di codice inerente all’importazione dei moduli:

```
1. import win32com.client
2. from openpyxl import load_workbook
3. from openpyxl import Workbook
4. import h5py
5. import numpy as np
6. import os
7. import math
```

Come si può notare le sintassi per importare i moduli possono essere varie, si può semplicemente indicare il nome del modulo (1,4,6,7), indicare un’abbreviazione da utilizzare per richiamarlo (5) oppure specificare la funzione da estrarre (2,3).

4 – L'HEC-RAS Controller & Python

4.1 – HEC-RAS API & HEC-RAS Controller

Ai fini del controllo e dell'automatizzazione di HEC-RAS è importante capire quali sono le principali parti che lo compongono a livello base di costruzione del software. L'HEC-RAS API (Application Programming Interface) costituisce la libreria di riferimento che contiene tre classi, ciascuna con il proprio set di procedute o meglio "blocchi di codice di programmazione" che eseguono determinate azioni, progettate specificamente per facilitare e controllare il funzionamento e il comportamento di HEC-RAS.
Le tre classi sono:

- HEC-RAS Controller
- HEC-RAS Flow
- HEC-RAS Geometry

La classe HEC-RAS Flow come anche HEC-RAS Geometry non sono state più aggiornate dagli sviluppatori in quanto le loro funzioni sono state integrate progressivamente nell'HEC-RAS Controller che, quindi, contiene la stragrande maggioranza delle procedure utili per controllare HEC-RAS. Più di cento sono le subroutines e le funzioni disponibili Goodell (2014).

4.1.1 – Subroutines e Funzioni

Le subroutines sono dei blocchi di codice che eseguono determinati compiti. Questi codici non sono visibili, sono interni all'HEC-RAS API e richiamabili mediante il loro nome nel programma. Alcune di queste subroutines richiedono alcune informazioni, chiamate argomenti, altre no. Ad esempio, una delle più semplici nell'HEC-RAS Controller è **Project_Save**, che salverà il progetto aperto in quel momento e non richiederà il passaggio di alcun argomento. Diversamente da quanto appena

descritto, invece, **Project_Open** è una subroutine che richiede il passaggio di un argomento che corrisponde al nome del progetto incluso al percorso di salvataggio. Le funzioni sono invece procedure che restituiscono informazioni, che possono essere utilizzate successivamente nel programma. In particolare, una funzione prende alcuni dati di input, li elabora e restituisce un risultato. Come le subroutines, il codice che sta dietro una funzione non è visibile e soltanto il suo nome deve essere specificato; inoltre alcune richiedono il passaggio di argomenti mentre altre no.

La prima differenza tra subroutines e funzione è che una funzione restituisce un valore, che sia sotto forma di valore numerico o stringa; mentre una subroutines esegue un'operazione senza restituire un valore preciso. Infatti molte delle funzioni più utilizzate nell'HEC-RAS Controller sono "funzioni d'informazione" che restituiscono informazioni sul progetto. Un esempio di funzione è **CurrentProjectTitle** che restituisce il titolo del progetto HEC-RAS attivo.

Vi sono poi casi particolari in cui alcune funzioni lavorano in modo simile ad una subroutine, è il caso delle funzioni Booleane che non restituiscono un vero e proprio output numerico ma restituiscono 'true' o 'false' indicando se l'esecuzione delle funzione è andata a buon fine oppure no. Infine vi è anche il caso opposto ovvero subroutine che sono assimilabili a funzioni e quindi, alla fine dell'esecuzione del blocco di codice, restituiscono un valore.

Si riportano di seguito alcuni esempi delle subroutines e funzioni più comunemente usate.

Subroutines:

- **Compute_ShowComputationWindow:** rende visibile la finestra di Computation durante la simulazione;
- **Compute_HideComputationWindow:** chiude la finestra di Computation durante la simulazione;
- **Edit_AddBC:** per aggiungere un ponte, richiede il passaggio di quattro argomenti quali: river name, reach name, river station del nuovo ponte e una stringa per la restituzione di eventuali errori;
- **Edit_AddIW:** per aggiungere una inline structure, gli argomenti richiesti sono gli stessi di Edit_AddBC;

- **Edit_AddLW:** per aggiungere una lateral structure, gli argomenti richiesti sono gli stessi di Edit_AddBC;
- **Edit_AddXS:** per aggiungere una cross section, gli argomenti richiesti sono gli stessi di Edit_AddBC;

Funzioni:

- **Compute_CurrentPlan:** per eseguire la simulazione in base al 'Plan' preimpostato, richiede il passaggio di tre argomenti: una variabile numerica per la restituzione del numero di messaggi elaborati durante la simulazione, una variabile stringa per restituire i messaggi testuali e una variabile booleana settata su 'true' o 'false' in base a se si vuole che il programma si blocchi durante la simulazione oppure continui;
- **CurrentProjectFile:** restituisce il nome ed il percorso del progetto HEC-RAS**;**
- **CurrentProjectTitle:** restituisce il titolo del progetto;
- **CurrentSteadyFile:** restituisce il nome ed il percorso dello 'steady flow file';
- **CurrentUnsteadyFile:** restituisce il nome ed il percorso dello 'unsteady flow file';

4.2 – Richiamare una funzione o una subroutine

Definito quindi cosa si intende per funzione e subroutine, in questo paragrafo si vuol descrivere come possano essere richiamate all'interno di un codice scritto in Python.
Richiamando quanto detto nel capitolo 3 ed in particolare nel paragrafo 3.4, per poter accedere all' HEC-RAS Controller e quindi a tutte le funzioni e subroutines in esse contenute è necessario utilizzare il modulo "Pywin32" ed in particolare il sotto-modulo "win32com.client".

4.2.1 - Inizializzazione dell'HEC-RAS Controller

Prima di poter utilizzare il contenuto dell'HEC-RAS Controller è necessario inizializzarlo come specificato di seguito:

```
import win32com.client
hec = win32com.client.Dispatch("RAS507.HECRASController")
```

La prima linea è stata riportata per indicare e ricordare il modulo di riferimento, mentre nella seconda vengono assegnate alla variabile "hec" le funzioni dell'HEC-RAS Controller. Questa assegnazione prevede l'utilizzo di una funzione interna al modulo win32com.client denominata "Dispatch" e prevede l'indicazione della versione di HEC-RAS installata sul proprio PC (qui si è utilizzata la versione HEC-RAS 5.0.7).
Dal momento in cui è stata effettuata l'inizializzazione è possibile richiamare una funzione o una subroutine attraverso la sintassi di seguito riportata:

```
hec . nome della funzione o subroutine (eventuali argomenti da passare)
```

Si riporta di seguito un semplice esempio inerente l'apertura della finestra principale di HEC-RAS:

```
hec.ShowRas()
```

In Figura 4.1 si riportano le tre righe di codice sopra descritte con l'output ottenuto eseguendo il programma.

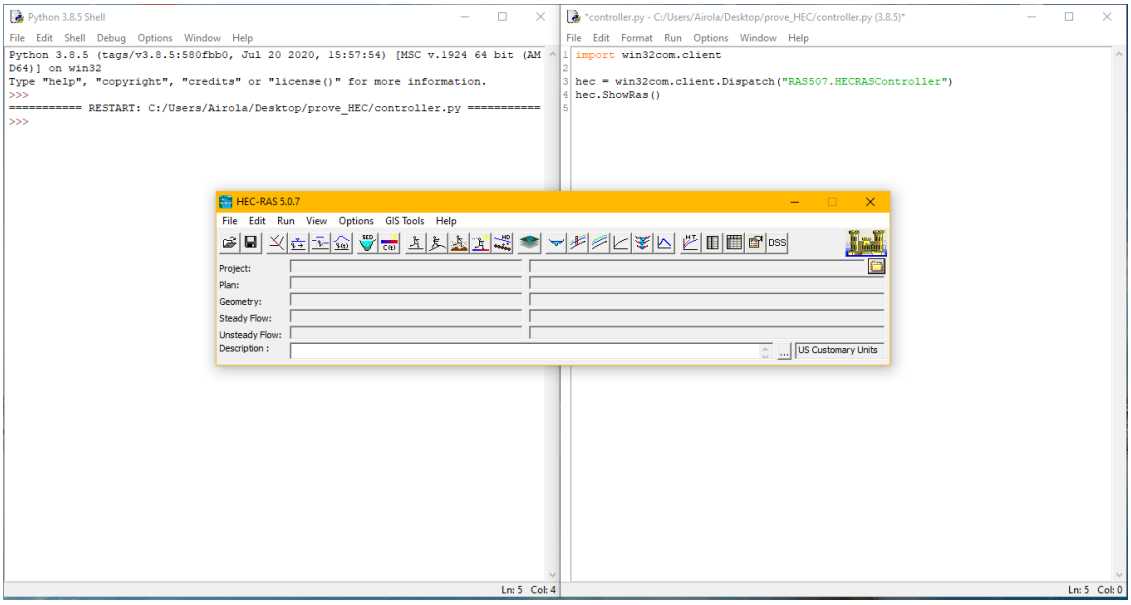


Figura 4.1: Apertura della finestra principale di HEC-RAS

4.3 - Apertura di un progetto HEC-RAS

Una volta appreso come richiamare una funzione o una subroutine, una delle prime operazioni che ci si trova ad affrontare è l'apertura di un progetto.
Vi possono essere due casi:

1. Creazione di un nuovo progetto;
2. Riapertura di un progetto già esistente.

Nel primo caso sarà necessario utilizzare la subroutine "**Project_New**" che prevede il passaggio di due argomenti quali: il titolo del nuovo progetto HEC-RAS e il percorso di salvataggio:

```
hec.Project_New('New_project',r'C:\Users\Airola\Desktop\prove_HEC\Controller.prj')
```

In Figura 4.2 si riporta un esempio con il rispettivo output generato mettendo come nome del progetto "New Project":

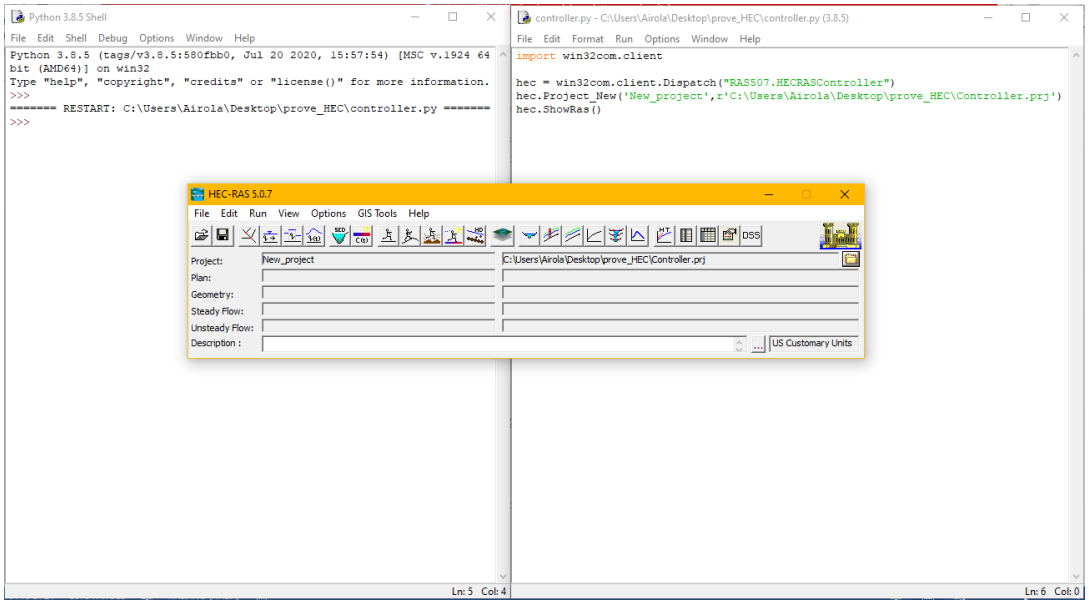


Figura 4.2: Apertura di un nuovo progetto HEC-RAS

Nel secondo caso bisogna utilizzare la subroutine “**Project_Open**” in cui si deve passare una variabile, qui chiamata RASProject, a cui è stato precedentemente associato il percorso di salvataggio del progetto da aprire. Tale assegnazione avviene utilizzando la sintassi di seguito riportata:

```
RASProject=os.path.join(os.getcwd(),r'C:\Users\Airola\Desktop\prove_HEC\Controller.prj')
```

Successivamente si può procedere all’apertura con:

```
hec.Project_Open(RASProject)
```

Esempio con il rispettivo output generato dopo aver eseguito il programma:

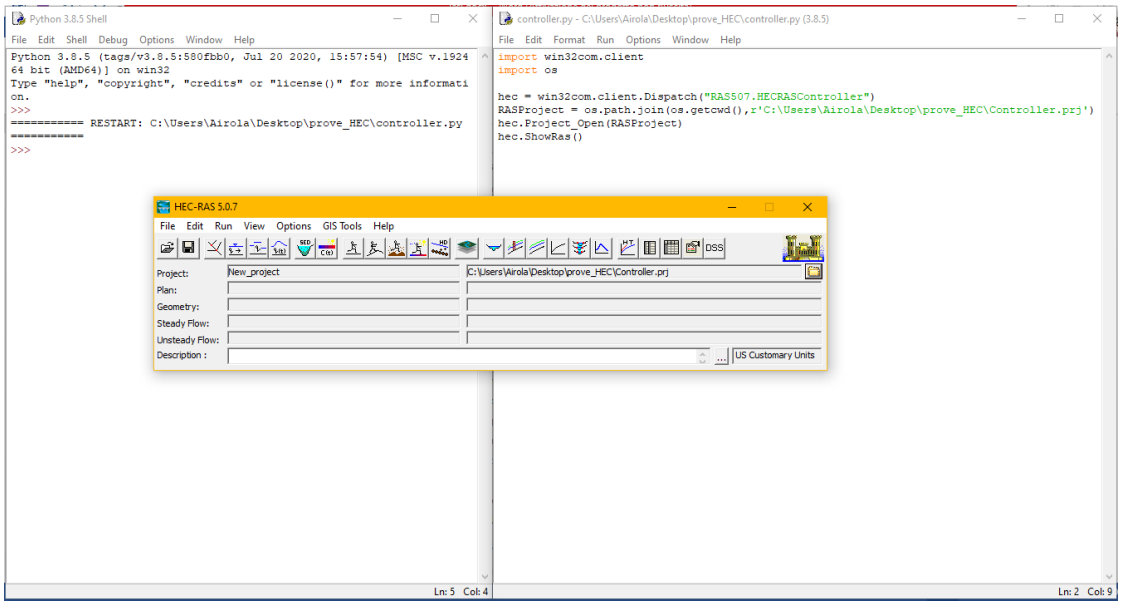


Figura 4.3: Riapertura di un progetto HEC-RAS già esistente

4.4 – Apertura del Editor della geometria

Un'altra operazione utile nel momento in cui si lavora in HEC-RAS è l'apertura della finestra della geometria. La subroutine di riferimento è "**Edit_GeometricData**" che non prevede il passaggio di alcun argomento.
L'inserimento nel codice avviene mediante la seguente sintassi:

```
hec.Edit_GeometricData()
```

Esempio di apertura del Editor della Geometria:

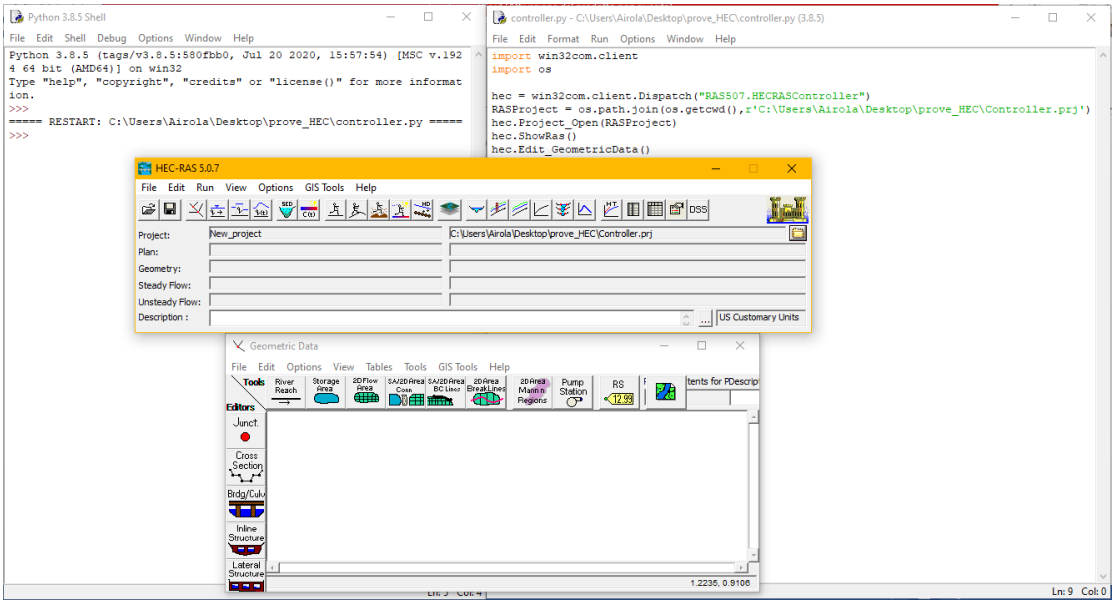


Figura 4.4: Apertura del Editor della geometria

L'apertura della finestra della geometria, se richiamata nel mezzo del codice, provoca l'arresto temporaneo del programma. Ciò significa che si possono effettuare le operazioni volute e soltanto al momento del salvataggio e della chiusura della finestra il programma riprenderà ed eseguirà le restanti righe di codice. Questa tipologia di funzionamento avviene anche per altre subroutine quali "Edit_BC", "Edit_IW", Edit_LW e "Edit_XS" rispettivamente per la modifica di un ponte, di un struttura in linea, di una struttura in derivazione e di una sezione.

4.5 – Apertura del Editor dello Steady ed Unsteady Flow Data

In modo del tutto similare a quanto mostrato nel precedente paragrafo vi è l'apertura dell'Editor dello Steady ed Unsteady Flow Data. Le subroutine da richiamare sono rispettivamente "**Edit_SteadyFlowData**" ed "**Edit_UnsteadyFlowData**"; entrambe non richiedono il passaggio di argomenti.
La sintassi, sempre simile, è la seguente:

```
hec.Edit_SteadyFlowData()
```

```
hec.Edit_UnsteadyFlowData()
```

Esempio di apertura dello Unsteady Flow Data Editor:

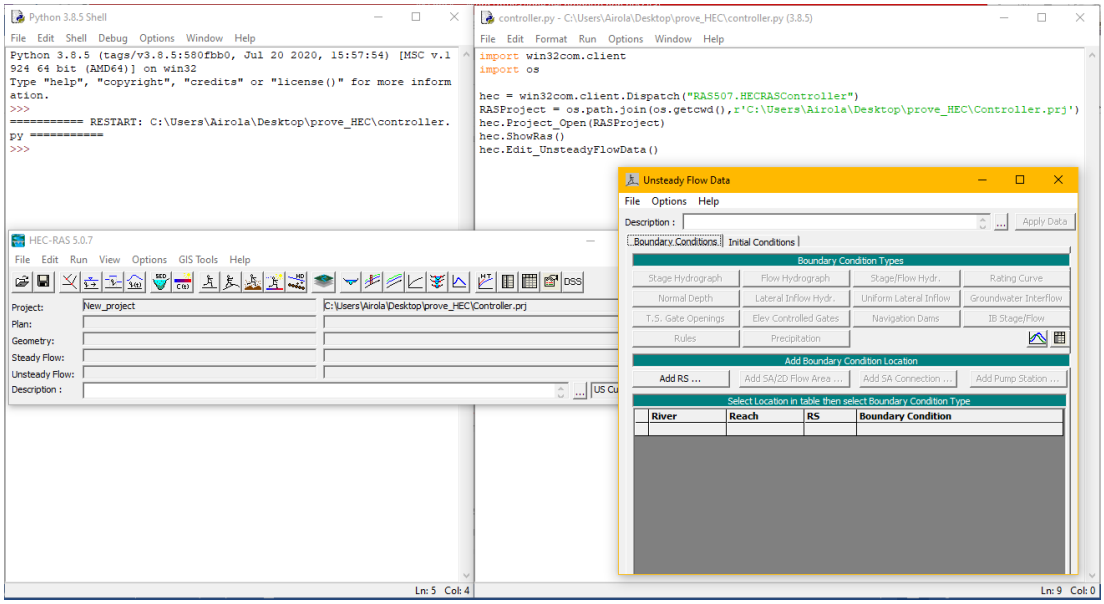


Figura 4.5: Apertura del Editor dello Unsteady Flow Data

Differentemente dalle subroutine viste nel precedente paragrafo queste non bloccano l'esecuzione del programma quindi, qualora si vogliano richiamare è necessario forzare la bloccatura del codice. Per far ciò, un'opportunità è quella di inserire subito dopo la subroutine una funzione "input" in cui si chiede conferma

all’utente riguardo la possibilità di procedere e quindi solamente in seguito all’immissione di una determinata espressione il codice proseguirà.

Esempio:

```
risp = "Si"
print("Digitare 'Si' se si può procedere con il codice")
risposta = input("Conferma: ")

flag = 1
while flag != 0:
    if risposta == risp:
        flag = 0
    else:
        print("Errore, ridigitare la risposta")
        risposta = input("Conferma: ")
```

Oltre all’inserimento della risposta che viene richiesta nella terza riga, vi è il controllo della correttezza della risposta inserita. In particolare è stata posta la condizione di confronto, tra la risposta digitata e la risposta voluta (opportunamente salvata in una variabile), all’interno di un ciclo in cui, nel caso di inserimento errato della conferma, ne richiede la digitalizzazione. Differentemente, se la conferma coincide, si esce dal ciclo e il programma continua analizzando le successive righe di codice.

4.6 – Apertura del Editor dello Steady o Unstaedy Flow Analysis

Altra importante finestra che viene sempre utilizzata per poter avviare una simulazione in HEC-RAS è quella dello Steady o Unsteady Flow Analysis. La subroutine che permette di aprirla mediante linguaggio di programmazione è "**Edit_PlanData**". In questo caso non è necessario specificare se si tratta di simulazione in moto permanente (Steay) o in moto vario (Unsteady) in quanto viene automaticamente percepito in base all'istruzione mostrata nel paragrafo precedentemente, ovvero lo Steady o Unsteady Flow Data, che dev'essere necessariamente inserita prima.
La sintassi è:

```
hec.Edit_PlanData()
```

Esempio di apertura dello Steady Flow Analysis Editor:

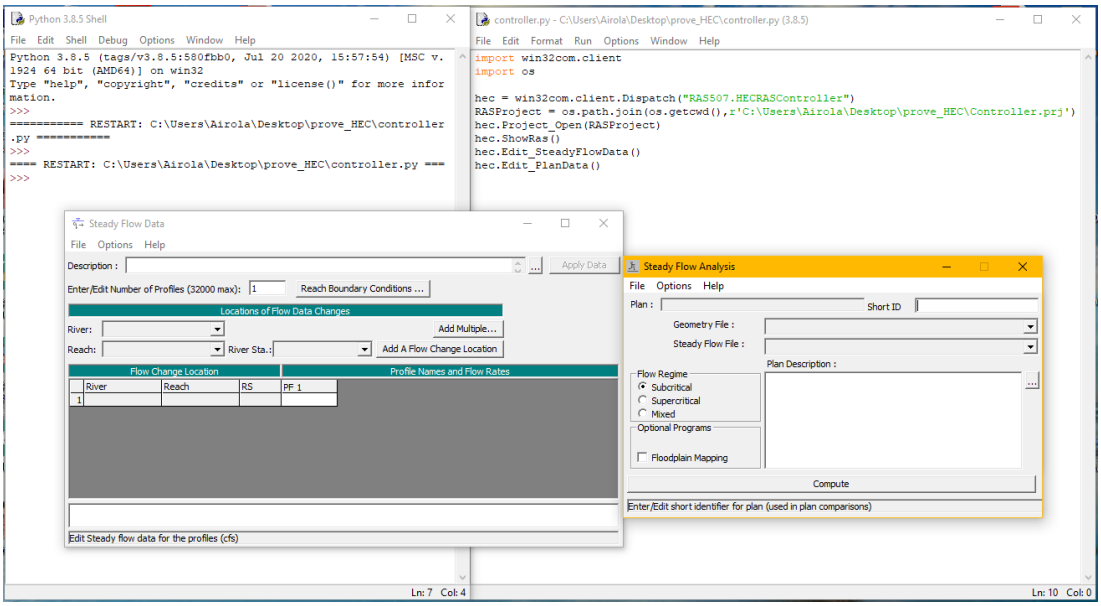


Figura 4.6: Apertura del Editor dello Steady Flow Analysis

Anche in questo caso, come per lo Steady e Unsteady Flow Data, l'apertura non implica l'arresto automatico del codice che dev'essere imposto appositamente, per farlo può essere richiamato il codice riportato alla fine del precedente paragrafo.

4.7 – Controllo esecuzione simulazioni

Una volta aperta ed impostata la finestra dello Steady o Unsteady Flow Analysis il passo successivo è avviare la simulazione. In questo caso la funzione di riferimento è "**Compute_CurrentPlan**" che necessita di tre argomenti:

1. nmsg, variabile numerica che restituisce il numero di messaggi prodotti;
2. msg(), variabile stringa che restituisce i messaggi prodotti dall'HEC-RAS Controller durante la simulazione;
3. blockingmode, variabile booleana che può essere settata su "True" o "False". Se settata su "False" il codice prosegue nel frattempo che viene eseguita la simulazione, invece se settata su "True" l'esecuzione del codice si blocca e riprenderà a simulazione terminata.

La sintassi per l'implementazione in Python prevede:

```
#inizializzazione variabili
NMsg,TabMsg,block = None,None,True
#esecuzione della simulazione
NMsg,TabMsg = hec.Compute_CurrentPlan(NMsg,TabMsg,block)
```

Un'ulteriore subroutine che può essere utile richiamare durante l'esecuzione delle simulazioni è "**Compute_ShowComputationWindow**". Questa permette di aprire la finestra in cui si vede l'avanzamento della simulazione e nella quale vengono anche indicati eventuali errori che ne pregiudicano la corretta esecuzione. Opposta alla subroutine appena indicata vi è "**Compute_HideComputationWindow**" che chiude la finestra nel caso non si sia interessati a visualizzare l'avanzamento. Entrambe non richiedono il passaggio di alcun argomento pertanto possono essere inserite nel codice semplicemente con la seguente sintassi:

```
hec.Compute_ShowComputationWindow()
```

```
hec.Compute_HideComputationWindow()
```

4.8 – Salvataggio e chiusura del progetto HEC-RAS

Per quanto riguarda il salvataggio vi sono due opportunità:

1. Salvataggio di un progetto, già precedentemente salvato e quindi dotato di un nome e di un percorso di salvataggio, con la subroutine "**Project_Save**". Non richiede argomenti pertanto la sintassi è:

```
hec.Project_Save()
```

2. Salvataggio di un progetto non ancora dotato di un nome e di un percorso di salvataggio con la subroutine "**Project_SaveAs**". Richiede l'indicazione del percorso di salvataggio e del nome, similmente a quanto riportato di seguito:

```
hec.Project_SaveAs(r'C:\Users\Airola\Desktop\prove_HEC\Controller.prj')
```

La chiusura di HEC-RAS avviene con la subroutine "**QuitRAS**", non sono richiesti argomenti:

```
hec.QuitRAS()
```

5 – Automazione simulazioni

Nel precedente capitolo si è visto come sia possibile controllare HEC-RAS richiamando alcune funzioni e subroutin in Python. Il controllo è sicuramente un primo aspetto fondamentale che fa da spartiacque ad alcune più importanti applicazioni dove il concetto di automazione può rivelarsi molto utile. Si tratta di far eseguire una serie di simulazioni in modo automatico andando, per ognuna di essa, ad immettere nuovi input e salvare degli output.
In questo capitolo si vuol, in un primo momento, descrivere quali siano le più importanti applicazioni per cui risulta utile il controllo e l'automazione di HEC-RAS e in un secondo momento mostrare effettivamente come questa automazione possa essere utilizzata sfruttando la programmazione; mostrando come effettuare la variazione di alcuni dati in input, come eseguire le simulazioni e come salvare gli output.

5.1 – Possibili applicazioni

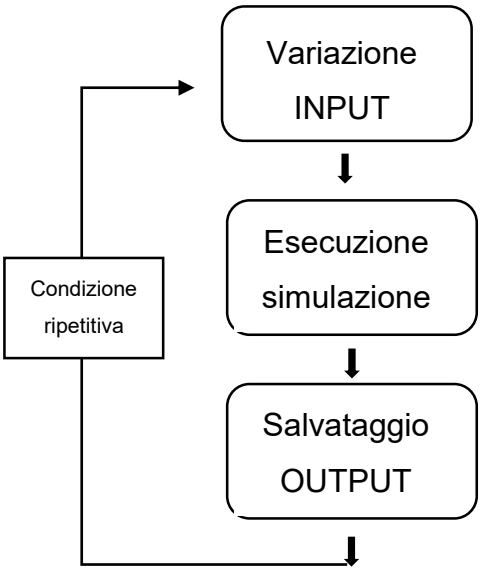
Il controllo e l'automazione delle simulazioni eseguite con HEC-RAS può essere utile per vari scopi ed applicabile in alcuni campi come:

- La valutazione del rischio alluvionale, attraverso la stesura di mappe di pericolosità idraulica. Si tratta di elaborati creati su un determinato dominio di studio, eseguendo molte simulazioni con diversi idrogrammi in input che a loro volta sono caratterizzati da portate di picco e volumi differenti che ne identificano la magnitudo dell'evento. Come si vedrà più nel dettaglio successivamente, questa serie di simulazioni possono essere eseguite in serie, "automaticamente", sfruttando la potenza della programmazione.
- La valutazione di alcuni aspetti legati alla modellazione idraulica fluviale come la calibrazione del modello. Si tratta di un'operazione ripetitiva che prevede l'esecuzione di numerose simulazioni in cui andare a variare, ad

ognuna di essa, il valore di scabrezza dell'alveo al fine di determinare quello più rappresentativo; ciò avviene basandosi su valori di portata e livelli idrici raggiunti da un evento realmente accaduto. In definitiva, quindi, quando il livello idrico ottenuto con le simulazioni tende a coincidere con quello realmente osservato allora si può assumere che il valore di scabrezza sia rappresentativo della porzione di alveo in questione. Anche in questo caso si fa riferimento ad una serie di simulazioni che devono essere eseguite variando un input, di conseguenza l'automazione del processo può risultare molto utile.

Generalizzando, quindi, si può pensare di applicare l'automazione in tutti quei casi in cui la ripetitività delle azioni, che si prevede di dover eseguire, possano essere semplificate, schematizzate e tradotte in codice mediante un linguaggio di programmazione.

Nella prosecuzione del seguente lavoro verrà analizzato un caso studio che si può considerare facente parte del primo punto tra i campi di applicazione sopra elencati. Come si può comprendere da quanto precedentemente descritto, quando si parla di controllo e automazione della modellazione in HEC-RAS si fa riferimento ad un processo iterativo che può essere così rappresentato:



Nei successivi paragrafi verranno approfonditi tali aspetti.

5.2 – Variazione INPUT

La variazione dei dati di input riveste un ruolo fondamentale nel processo di modellazione idraulica. Infatti, generalmente ciò che si fa quando si vuol effettuare una simulazione è impostare tutta una serie di dati iniziali quali ad esempio, la scabrezza dell'alveo, la portata in ingresso (steady simulation) o l'idrogramma (unsteady simulation) e si variano manualmente per ottenere scenari differenti che si intendono analizzare; questo concetto, supponendo di doverlo applicare ad un numero elevato di scenari, può essere automatizzato.
Per comprendere come sia possibile cambiare un determinato valore di input, non manualmente, ma mediante la programmazione bisogna innanzitutto capire quali sono i file creati da HEC-RAS che possono contenere parametri soggetti a variazione, questi sono:

- File relativo alla geometria (Geometry File);
- File relativo all'immissione dei valori di portata (Steady Flow File o Unsteady Flow File);

5.2.1 – INPUT riferiti alla geometria

Come già anticipato al secondo capitolo, in HEC-RAS è possibile sviluppare simulazioni monodimensionali e bidimensionali, il che inevitabilmente implica metodologie differenti per la stesura della geometria. Ciò che invece rimane comune è il metodo di salvataggio; infatti, in entrambi i casi, viene creato un file con estensione ".g" posto all'interno della cartella in cui vi è salvato il progetto generale di HEC-RAS. Si tratta di un'estensione di salvataggio particolare ma che non deve spaventare, tant'è che è possibile visualizzarne il contenuto aprendolo con un semplice editor di testo come "Blocco Note". Di seguito si riporta a titolo esemplificativo una parte di file derivante dal salvataggio della geometria bidimensionale creata per il caso studio che verrà trattato nel successivo capitolo.

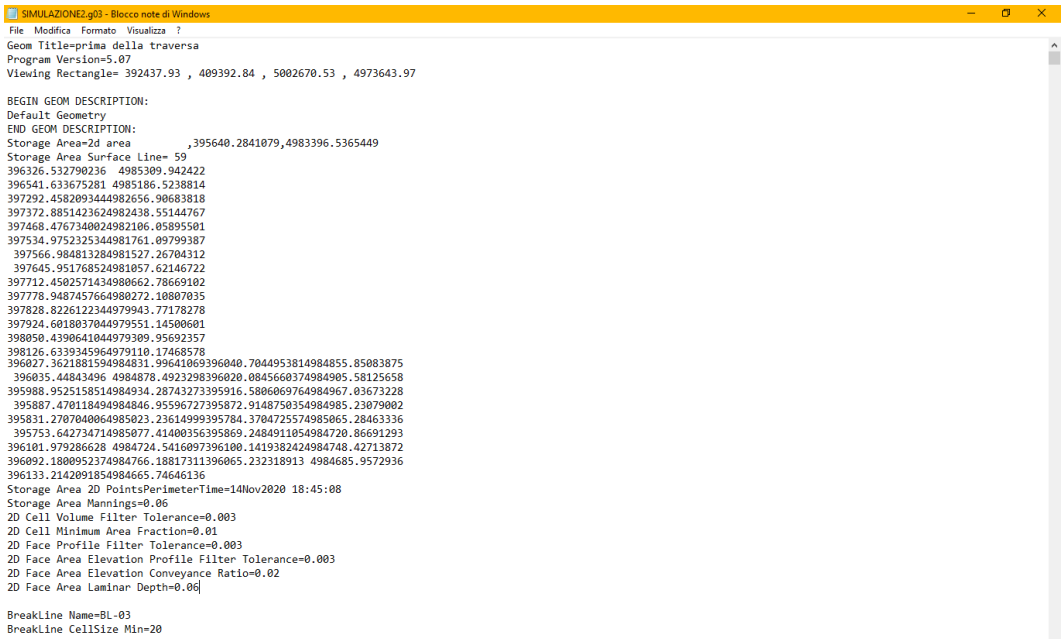
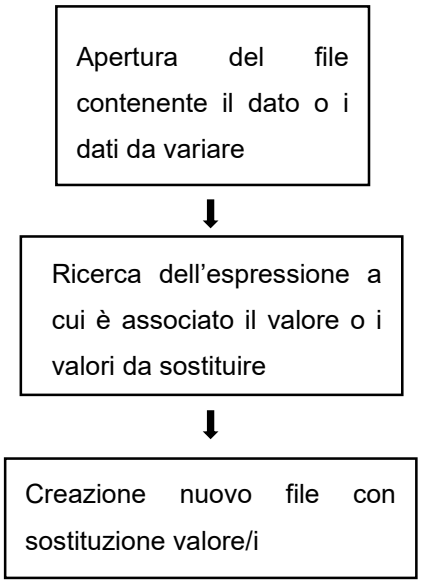


Figura 5.1: Estratto del file derivante dal salvataggio della geometria bidimensionale

Si tratta di un file strutturato in cui si riportano tutte le informazioni inerenti alla geometria creata. Trattandosi di una geometria bidimensionale vi sono tutta una serie informazioni sulle coordinate dei vertici delle linee che costituiscono il dominio, sulle coordinate dei centri dei poligoni costituenti la mesh e poi altre informazioni tra cui spicca sicuramente il coefficiente di attrito o “Storage Area Mannings”. Il processo di calibrazione, descritto all’inizio del capitolo, si basa sulla variazione ripetuta del seguente valore che pertanto può essere automatizzato.

Il fatto che tale file possa essere aperto con un semplice editor di testo gioca un ruolo altrettanto importante in quanto tutti i linguaggi di programmazione sono in grado di accedervi.

Ipotizzando di voler variare, ad ogni iterazione, il numero rappresentativo del coefficiente di scabrezza si può seguire un approccio generale del tipo:



L'operazione di ricerca di una data espressione per individuare la riga in cui andare ad effettuare la sostituzione può essere effettuata mediante un'apposita funzione realizzata in Python e di seguito riportata:

```
def ricerca_elemento(file,elem):
    riga=" "
    count=0
    while(riga):
        riga=file.readline()
        if elem in riga:
            return count
        count+=1
```

Tale funzione denominata "ricerca_elemento" richiede il passaggio di due argomenti: il primo è la variabile a cui è stato assegnato l'intero contenuto del file mentre il secondo è una variabile stringa che contiene l'espressione da ricercare. Viene inizializzata una variabile contatore "count" che aumenta il proprio valore di una unità man mano che le righe del file vengono scansionate e qualora l'espressione ricercata venisse trovata, attraverso la il comando "return" viene assegnato definitivamente alla variabile "count" il numero corrispondente alla riga del file in cui è stata trovata l'uguaglianza.

Questa, essendo una funzione generale, dev'essere definita inizialmente ma non entra a far parte del corpo principale del codice e pertanto dev'essere richiamata laddove necessario. Si riporta nelle righe successive un esempio di richiamo della sopracitata funzione, che è stata effettivamente utilizzata nell'analisi del caso studio di cui si parlerà nei successivi capitoli:

```
file_in= open (r"C:\Users\Airola\Desktop\caso studio\prova.txt","r")
elem = "Storage Area Mannings="
count = ricerca_elemento(file_in, elem)
file_in.close()
```

Prima operazione consiste nell'apertura del file in cui effettuare la ricerca, ciò avviene sfruttando il costrutto "open" e salvando il contenuto nella variabile "file_in" che rappresenta anche il primo argomento della funzione, successivamente dev'essere specificata l'espressione da ricercare e salvarla nella seconda variabile "elem". Inizializzati gli argomenti l'invocazione della funzione si effettua semplicemente richiamandola con il nome che gli era stato assegnato, preceduta dal nome della variabile in cui salvare il risultato prodotto.
L'altra operazione, consistente nella variazione del valore associato all'espressione precedentemente ricercata, può essere effettuata sfruttando due modalità:

1. Generando un numero "casuale": in Python è presente un modulo, installabile con la procedura mostrata in capitolo 3, denominato "random"; richiamando alcune specifiche funzioni, come ad esempio "randrange (start,stop,step)", è possibile generare numeri casuali partendo da un determinato intervallo; oppure facendo riferimento alla funzione "gauss (mu,sigma)" si generano numeri casuali facenti parte di una certa distribuzione avente un determinato valore medio e deviazione standard. Nell'esempio precedente è stata menzionata la distribuzione di Gauss, ma vi sono varie funzioni utilizzabili che fanno riferimento anche ad altre distribuzioni come ad esempio "betavariate (alpha, beta)" per la distribuzione Beta, "expovariate (lambd)" per la distribuzione esponenziale,

"gammavariate (alpha, beta)" per la distribuzione Gamma, "lognormvariate (mu,sigma)" per la distribuzione lognormale e "normalvariate (mu,sigma)" per la distribuzione normale[4].

2. Leggendo tale valore da un file precedentemente creato in cui sono stati inseriti una serie di valori da testare. In questo caso sarà necessario mettere in comunicazione tale file con Python; le funzioni da utilizzare variano in base al tipo di documento in cui sono stati salvati i dati ma il concetto di base rimane quello di dover scandire il file estraendo un valore alla volta, sostituirlo al file di input di HEC-RAS per, infine, eseguire una nuova simulazione. Se i dati sono stati salvati in un documento Excel è possibile utilizzare il modulo, già descritto nel capitolo 3, "OpenPyXl" con le relative funzioni, mentre se sono stati inseriti su un semplice documento di testo quale "Blocco Note" basta aprirlo in modalità lettura, scandire le varie righe salvando ogni volta in una variabile il dato e sostituirlo nel file di testo contente gli input HEC-RAS (dopo aver ricercato la posizione corretta con la funzione precedentemente descritta).

5.2.2 – INPUT riferiti all'immissione dei dati di portata

Altra tipologia di dato che può essere soggetto ad automazione è quello inerente alla portata. Premettendo che, in base allo scopo dell'analisi, possano essere effettuate simulazioni in moto permanente (portata costante nel tempo) oppure in moto vario (portata variabile nel tempo ed utilizzo di appositi idrogrammi come input), il concetto funzionale di base rimane il solito, quindi, individuazione del punto in cui effettuare la variazione del dato (mediante la funzione del paragrafo precedente), selezionare il nuovo input e sostituirlo nel apposito documento. Ciò che materialmente differenzia le due casistiche sono le tipologie di dato, infatti, se

[4] Per saperne di più su tali funzioni e gli argomenti che devono essere passati si può consultare il seguente sito web: https://docs.python.it/html/lib/module-random.html.

per analisi in moto permanente si ha a che fare con un solo dato di portata, per analisi in moto vario si ha un idrogramma e quindi una serie di valori di portata che variano nel tempo. Questo implica che per il primo caso, la modalità di variazione del valore è simile a quanto descritto per la scabrezza (varia ovviamente il file di testo contenente il dato) mentre per il secondo caso si devono sostituire più valori di portata mantenendo una corretta formattazione per evitare messaggi di errori di caricamento nel momento dell'apertura del file in HEC-RAS. Si riporta di seguito un esempio di file contenente come input un idrogramma.

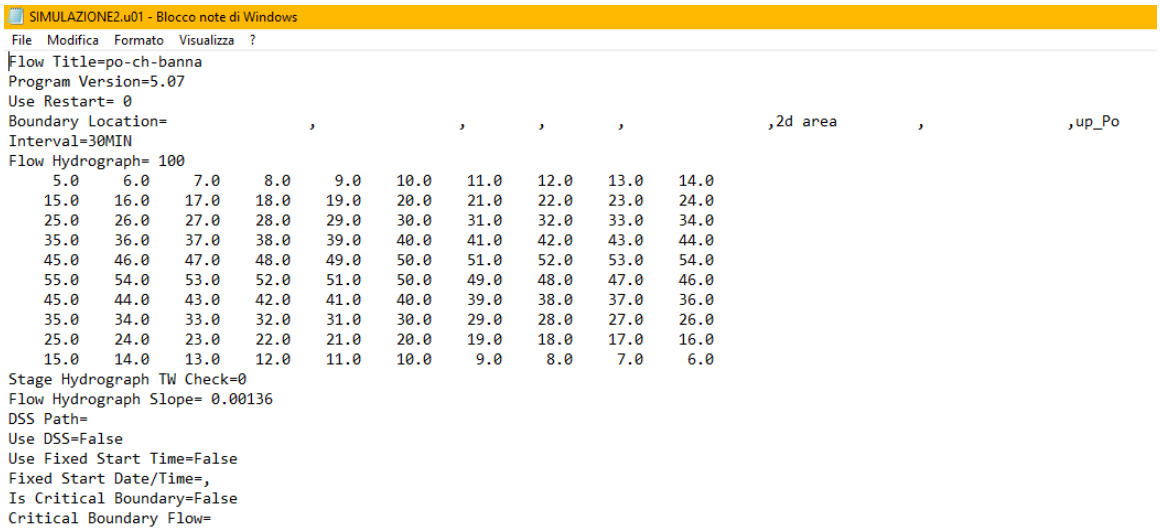

SIMULAZIONE2.u01 - Blocco note di Windows
File Modifica Formato Visualizza ?
Flow Title=po-ch-banna
Program Version=5.07
Use Restart= 0
Boundary Location= , , , , ,2d area , ,up_Po
Interval=30MIN
Flow Hydrograph= 100
5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 13.0 14.0
15.0 16.0 17.0 18.0 19.0 20.0 21.0 22.0 23.0 24.0
25.0 26.0 27.0 28.0 29.0 30.0 31.0 32.0 33.0 34.0
35.0 36.0 37.0 38.0 39.0 40.0 41.0 42.0 43.0 44.0
45.0 46.0 47.0 48.0 49.0 50.0 51.0 52.0 53.0 54.0
55.0 54.0 53.0 52.0 51.0 50.0 49.0 48.0 47.0 46.0
45.0 44.0 43.0 42.0 41.0 40.0 39.0 38.0 37.0 36.0
35.0 34.0 33.0 32.0 31.0 30.0 29.0 28.0 27.0 26.0
25.0 24.0 23.0 22.0 21.0 20.0 19.0 18.0 17.0 16.0
15.0 14.0 13.0 12.0 11.0 10.0 9.0 8.0 7.0 6.0
Stage Hydrograph TW Check=0
Flow Hydrograph Slope= 0.00136
DSS Path=
Use DSS=False
Use Fixed Start Time=False
Fixed Start Date/Time=,
Is Critical Boundary=False
Critical Boundary Flow=

Figura 5.2: Estratto del file contenente come input i dati di portata.

Come per quanto riguarda il file della geometria anche qui si è in presenza di un file strutturato, con estensione differente ".u" ad indicare Unsteady ovvero condizioni non permanenti. Dopo una serie di prime informazioni come il titolo del file, la versione del programma e le informazioni del punto a cui è riferito l'input (Boundary Location) vi sono i dati riguardanti la portata; in particolare "Interval" identifica il tempo che intercorre tra un dato e l'altro dell'idrogramma mentre "Flow Hydrograph" rappresenta l'idrogramma vero e proprio, a fianco viene indicato il numero di dati che lo compongono mentre nelle righe successive si riportano in dettaglio tutti i valori. Si può notare come siano disposti con un preciso formato, dieci valori ogni riga per un numero di righe variabile in base alla lunghezza dell'idrogramma; inoltre

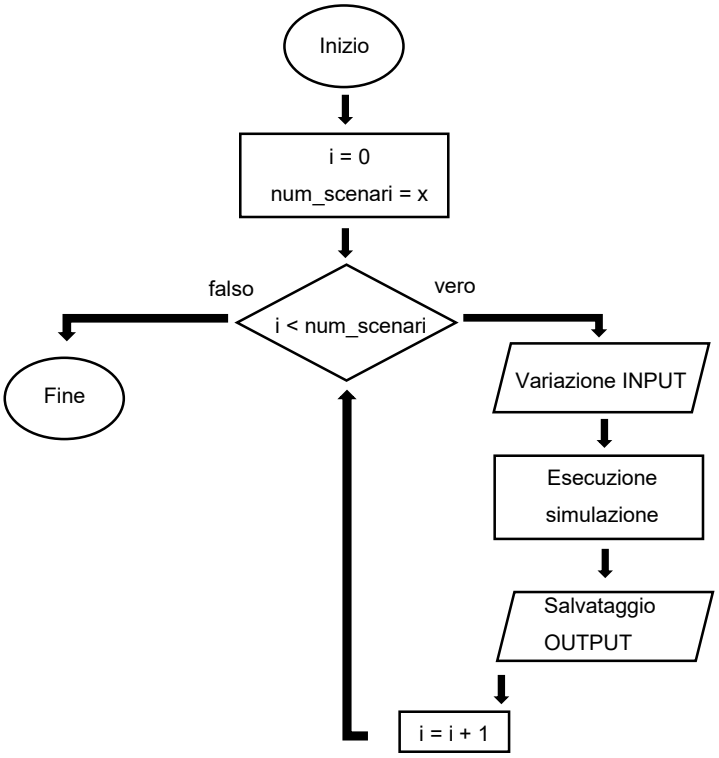
per ogni valore viene attribuito uno spazio pari a otto [SPACE] quindi al momento della sostituzione bisogna fare attenzione a mantenere tale formattazione. Per far ciò sono state predisposte delle righe di codice che vengono di seguito riportate:

```
file_in= open (r"C:\Users\Airola\Desktop\caso
studio\copia_input.txt","r")
file_out= open (r"C:\Users\Airola\Desktop\caso
studio\SIMULAZIONE2.u01","w")

num_linee = math.ceil(num_righe/10)
n=0
b=0
for riga in file_in:
      if count+1 <= n <= count+num_linee:
            for k in range (0,10):
                  a = vet[b,k]
                  cifre = len(str(a))
                  if cifre == 1:
                        file_out.write("       "+str(a))
                  elif cifre == 2:
                        file_out.write("      "+str(a))
                  elif cifre == 3:
                        file_out.write("     "+str(a))
                  elif cifre == 4:
                        file_out.write("    "+str(a))
                  elif cifre == 5:
                        file_out.write("   "+str(a))
                  elif cifre == 6:
                        file_out.write("  "+str(a))
                  elif cifre == 7:
                        file_out.write(" "+str(a))
            file_out.write("\n")
            b+=1
            n+=1
      else:
            file_out.write(riga)
            n+=1

file_in.close()
file_out.close()
```

Prima di tutto si devono predisporre e aprire due file: uno denominato “copia_input.txt” in modalità lettura che rimarrà sempre uguale e servirà da base per copiare quelle righe di file non interessate da variazioni e un secondo, “simulazione2.u01”, in modalità scrittura in cui verranno inseriti i nuovi valori costituenti l’idrogramma e aperto in HEC-RAS per l’esecuzione di una nuova simulazione. Con il primo ciclo “for” si iniziano a scandire le righe del file “copia_input”, subito dopo è stata inserita un’istruzione “if” che, in base al valore raggiunto dalla variabile contatore “n”, ha la funzione di determinare se si è all’interno dell’intervallo di righe in cui vi è l’idrogramma, definito in base al valore di “count” (determinato utilizzando la funzione ricerca_elemento) e alla somma di “count” con il numero di linee, calcolato precedentemente come il numero di valori dell’idrogramma diviso 10 ovvero il numero di valori che possono essere disposti su una riga. Partendo dal caso negativo, cioè di essere al di fuori del range, si finisce direttamente all’istruzione finale “else” che prevede la scrittura, sul file ”simulazione2”, della riga scansionata dal ciclo iniziale senza nessuna modifica. Qualora invece la variabile contatore “n” risultasse compresa nell’intervallo è necessario proseguire alla sostituzione dei valori; in questo caso viene utilizzato un secondo ciclo “for” annidato che insieme alla variabile contatore “b” determinano il corretto valore da estrarre da “vet”, matrice contente l’idrogramma (la costruzione di tale matrice verrà trattata nei capitoli successivi). Una volta estratto il valore e salvato nella variabile “a” è necessario verificare il numero di cifre di cui è composto al fine di allocarlo correttamente nello spazio a disposizione. Ad esempio, se il valore di portata è pari a 50.0 m^3/s, lo spazio occupato dal numero sarà di 4 caratteri quindi dovranno essere inseriti 4 spazi vuoti prima, al fine di colmare correttamente gli 8 spazi a disposizione. Nel codice sono stati ipotizzati più casi di lunghezza del valore, fino a sette cifre, e i relativi spazi vuoti da inserire al fine da rendere automatico il processo. Inserita la prima riga di valori, e quindi raggiunto il limite del secondo ciclo “for”, mediante il costrutto “\n” si va a capo, si incrementano le variabili contatore e si ritorna all’inizio scansionando una nuova linea. Tale procedimento, ripetuto fino al raggiungimento della fine del file “copia” permette quindi di creare un nuovo file “simulazione2.u” che verrà aperto da HEC-RAS come input.

5.3 – Automazione esecuzione simulazione

Una volta effettuata la variazione dei dati di input ed aperti i relativi file in HEC-RAS il passo successivo riguarda l'esecuzione delle simulazioni. Tale argomento è già stato trattato al paragrafo 4.7, si è già descritta la funzione che inserita in Python ne permette l'avvio ovvero "hec.Compute_CurrentPlan()" , pertanto ora si porrà l'attenzione sull'automazione. Il concetto base è in realtà piuttosto generale, tant'è che non si agisce puntualmente sulla funzione ma sul codice complessivo. Il processo di automatizzazione avviene ponendo nelle prime righe del codice delle istruzioni iterative che permettono la ripetizione di certe operazioni per un numero di volte preimpostato. Il processo logico delle operazioni, come già parzialmente riportato nello schema generale ad inizio capitolo, può essere così rappresentato:



Si riporta di seguito una bozza generale di codice in cui si vuol rappresentare il processo di automatizzazione nel suo complesso.

```
##SCHEMA GENERALE PROGRAMMA
#condizione ciclica iniziale
for i in range (0,num_scenari):

      ##PREPARAZIONE/MODIFICA FILE INPUT
      #vedi paragrafo 5.2

      ##ESECUZIONE SIMULAZIONE IN HECRAS
      hec.Project_Open(RASProject)
      NMsg,TabMsg,block = None,None,True
      hec.Compute_ShowComputationWindow()
      NMsg,TabMsg = hec.Compute_CurrentPlan(NMsg,TabMsg,block)
      hec.ShowRas()

      ##SALVATAGGIO OUTPUT
      #vedi paragrafo successivo
```

Si può quindi ben comprendere come la condizione ciclica iniziale sia alla base del processo di automatizzazione con tutte le altre operazioni che ricadono all'interno. Come si vedrà più avanti nel caso studio che sarà trattato il numero di simulazioni che si intende fare corrispondono ad un dato numero di scenari da analizzare.

5.4 – Salvataggio output

Passo successivo all'esecuzione della simulazione è il salvataggio dell'output, si tratta dell'ultima macro-operazione da eseguire, in modo tale da salvare i dati di interesse prima che si proceda con un'altra simulazione che porterebbe automaticamente alla perdita delle precedenti informazioni. Similmente a quanto presentato per gli INPUT anche in questo frangente vengono creati dei file, a fine simulazione, quali: "simulazione2.g0X.hdf" e "simulazione2.p0X.hdf". L'estensione ".hdf" sta per (Hierarchical Data Format File), i dati sono disposti in modo gerarchico all'interno un sistema di sottocartelle.

Facendo riferimento al file con estensione ".p" le informazioni che si possono reperire sono racchiuse in quattro macro cartelle come mostrato in Figura 5.3:

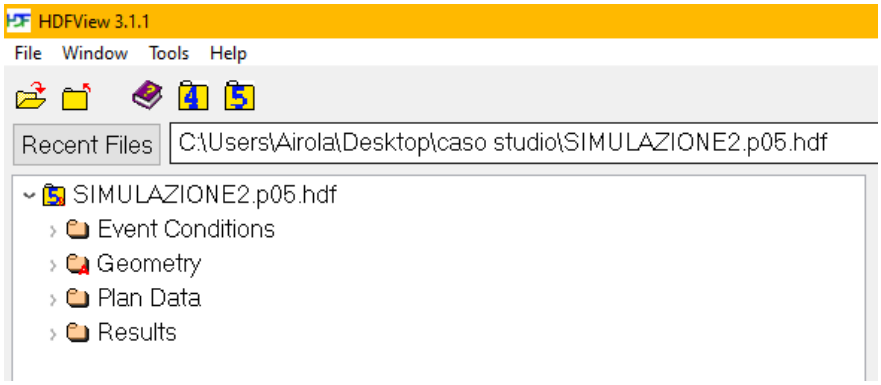
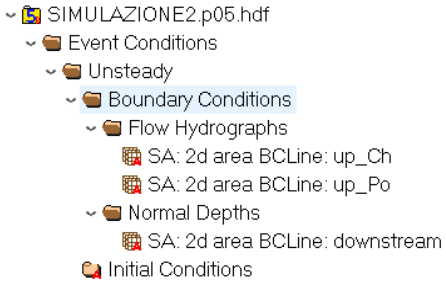


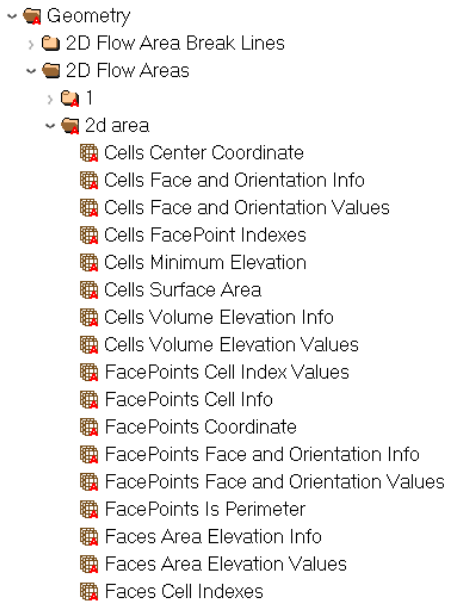
Figura 5.3: Le quattro macro cartelle contenute nel file ".p"

Dove:

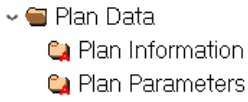
- In *Event Conditions* vi sono informazioni sulle condizioni al contorno di monte e di valle, si possono vedere in forma tabellare gli idrogrammi inseriti e vedere il valore di "normal depth" inserito.



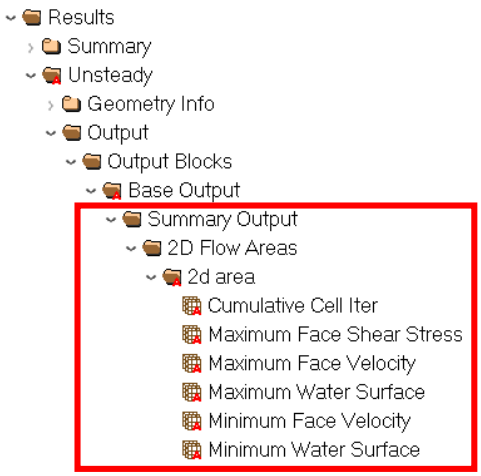
- In *Geometry* vi sono informazioni su tutti gli elementi geometrici che sono stati creati. In particolare vi sono tutta una serie di informazioni sulle celle della mesh come, in parte, rappresentato di fianco.



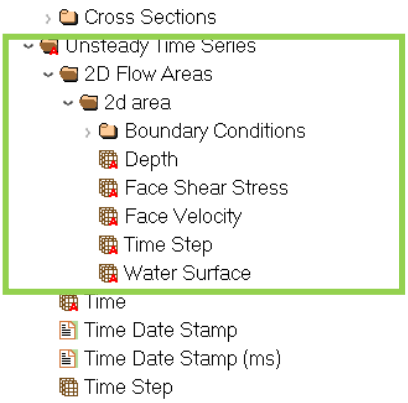
- In *Plan Data* e specialmente in “Plan Information” sono racchiusi i vari dati impostati nella finestra del Plan di HEC-RAS, ad esempio la risoluzione temporale della simulazione e la data/ora di inizio e fine della simulazione.



- In *Result* sono invece contenuti i risultati della fase di computazione. Questi si possono raggruppare in due macro-categorie: la prima si può individuare con la dicitura “Summary Output” (riquadro rosso) dove per ogni cella del dominio è stato calcolato il massimo e il minimo di una serie di grandezze quali la tensione di taglio esercitata sulla superficie, la velocità del flusso e la quota raggiunta dal pelo libero della corrente; la seconda (riquadro verde) individuabile con “Unsteady Time Series” porta ad una serie di risultati in cui per ogni cella è riportato tutto l’andamento temporale della specifica grandezza. L’andamento temporale dipende dal valore di “Base Output Interval” impostato nel Plan inizialmente (1, 5, 10, 20, 30 min



ecc). Le grandezze per cui è disponibile questa sequenza temporale sono: il tirante idrico (depth), la tensione di taglio esercitata sulla superficie della cella, la velocità della corrente nella cella, gli step temporali (in giorni) e il valore di quota s.l.m raggiunta dalla corrente.



Si riporta di seguito un esempio di estrazione di una determinata grandezza:

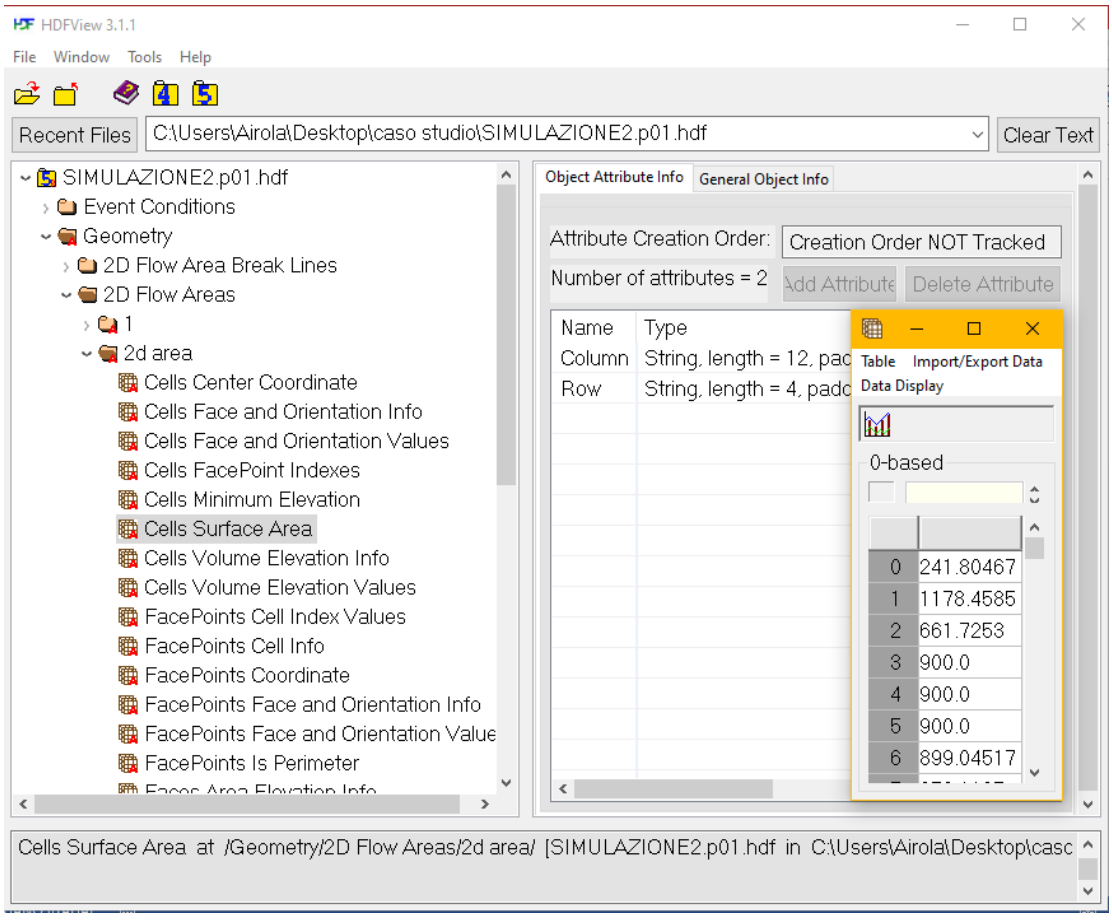
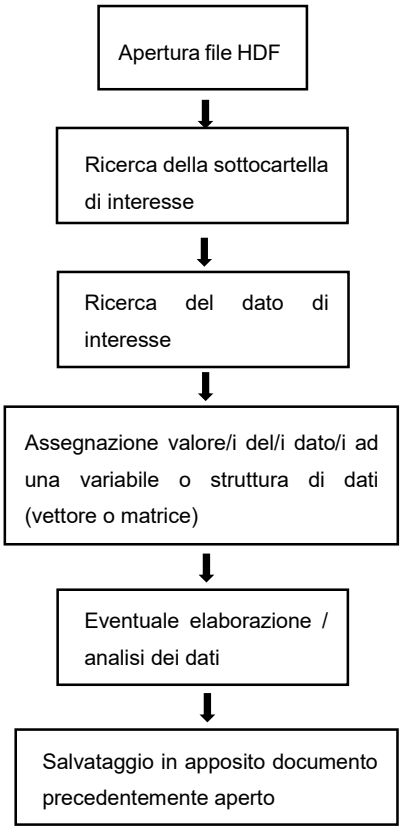


Figura 5.4: Esempio di apertura del file ".hdf" con HDFView

Come già fatto anche in precedenza il file output di HEC-RAS è stato aperto con il programma HDFView[5] ed è risultato molto utile al fine di comprendere dove e come sono archiviati determinati dati. Facendo riferimento all'immagine 5.4 si può notare l'apertura di parte della tabella con all'interno le superfici delle varie celle che compongono la mesh del dominio, che sarà oggetto di studio. Si può inoltre osservare il relativo percorso di sottocartelle che ha portato all'apertura della tabella voluta: Geometry→2D Flow Areas→2d area→Cells Surface Area.
Concentrandosi sul processo di automazione del salvataggio di un determinato output il flusso logico da seguire è:



[5] Maggiori informazioni sul programma e il relativo download sono possibili dal sito: https://www.hdfgroup.org/downloads/hdfview/

L'apertura del file avviene sfruttando il modulo "h5py"[6] con il quale, specificando il percorso che porta ad un determinato dato, è possibile estrarlo, eventualmente elaborarlo e poi salvarlo in un apposito documento che può essere un semplice documento di testo come "Blocco Note" o un foglio di calcolo Excel.
Si riporta nelle righe successive una bozza di codice inerente al flusso logico precedentemente descritto:

```
## APERTURA FILE HDF
with h5py.File(r"C:\Users\Airola\Desktop\caso
studio\SIMULAZIONE2.p01.hdf",       "r") as hdf:
      G1 = hdf.get('Geometry/2D Flow Areas/2d area')
      sup = np.array(G1.get('Cells Surface Area'))
nrows1, ncols1 = np.shape(sup)

##APERTURA FILE TXT
output = open(r"C:\Users\Airola\Desktop\caso studio\file_output.txt","a")

#SALVATAGGIO IN FILE
for w in range (0,ncols1):
      output.write(str(sup[w]))
output.close()
```

I percorsi riportati fanno riferimento a quanto mostrato in Figura 5.4 dove si era aperta la tabella contenente i dati della "Cells Surface Area"; in questo modo i dati vengono in un primo momento salvati nella matrice "sup" e successivamente introdotti nel file di testo "file_output.txt". Il file è aperto in modalità "append", questo fa sì che ad ogni simulazione eseguita i dati vengano inseriti in coda a quelli già presenti creando un documento unico che verrà post-processato per effettuarne un'analisi ed estrarre i risultati voluti.
La scrittura su file avviene mediante la sintassi "*output.write(str(sup[w]))*" che, introdotta in un ciclo "for" permette di scandire la matrice "sup" estraendo un valore alla volta e salvandolo nel file.

[6] Per maggiori informazioni sul modulo h5py accedere al sito https://www.h5py.org/ mentre per quanto riguarda l'installazione fare riferimento al paragrafo 3.4

6 – Caso studio

6.1 – Inquadramento territoriale e scopo delle analisi

In questo capitolo l'obiettivo è quello di fornire un esempio di applicazione dei concetti mostrati in precedenza. In particolar modo si vuol sfruttare l'automazione di HEC-RAS per analizzare diversi scenari di evento riguardanti il rischio alluvionale insistente sulla Città di Moncalieri dove vi è la confluenza tra il torrente Chisola e il fiume Po. L'idea è quella di valutare l'effetto che onde di piena di differente magnitudo hanno sul territorio limitrofo considerando anche l'aspetto temporale ovvero l'ipotesi di onde di piena temporalmente coincidenti alla confluenza oppure onde di piena sfalsate. Le simulazioni verranno quindi eseguite in serie secondo gli schemi già presentati e che verranno ripresi da un punto di vista più applicativo.
Dal punto di vista territoriale, come già accennato, lo studio interessa principalmente la Città di Moncalieri e il relativo territorio comunale, che si sviluppa verso sud fino ad interessare parte del Comune di La Loggia (Figura 6.1).

Figura 6.1: Area oggetto di studio (non in scala)

Il sistema fluviale dell'area è quindi composto dal fiume Po e dal torrente Chisola, entrambi scorrono da sud verso nord fino alla confluenza situata in corrispondenza della zona più edificata di Moncalieri. Si tratta di due corpi idrici aventi dimensioni dell'alveo non comparabili vista anche la dimensione del bacino idrologico da cui sono generati (Figura 6.2 e 6.3). Infatti il bacino del Po, considerando la sezione chiusura a Moncalieri, ha un'area pari a 5048.37 km^2 mentre quello del Chisola con sezione di chiusura a La Loggia è pari a 464.21 km^2. I dati e le immagini sono state reperite dal Atlante dei Bacini Imbriferi Piemontesi redatto da Gallo et al. (2013).

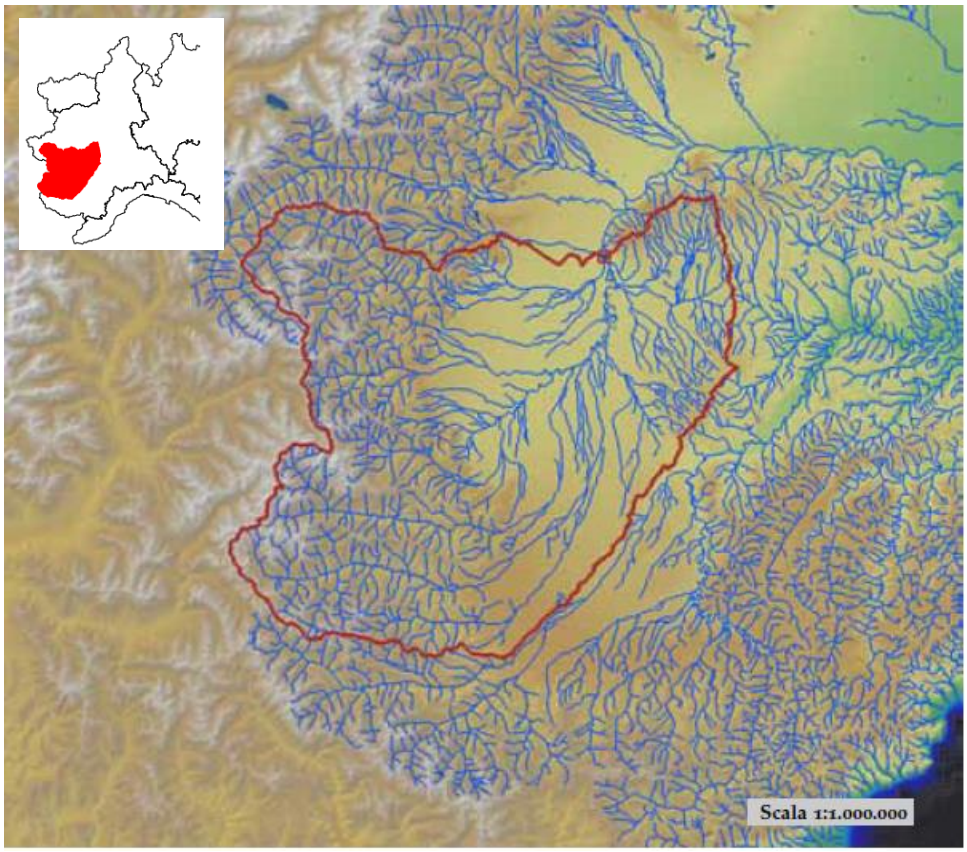


Figura 6.2: Bacino idrologico del Po con sezione di chiusura a Moncalieri (non in scala)

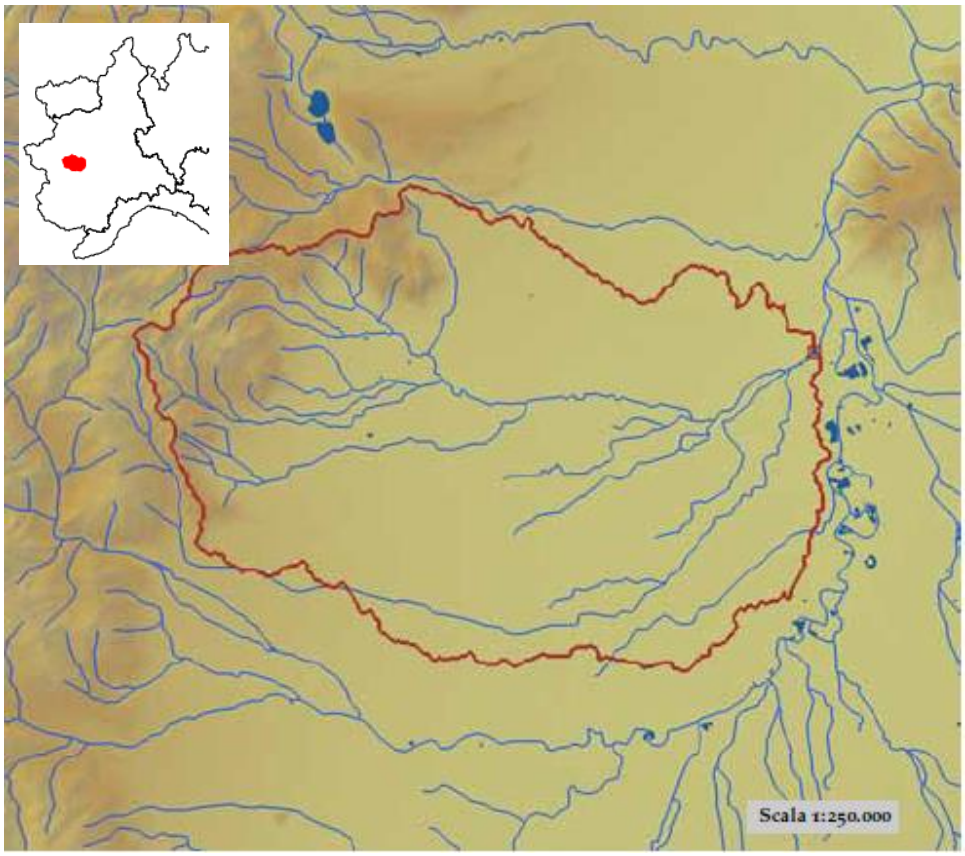


Figura 6.3: Bacino idrologico del Chisola a La Loggia (non in scala)

Osservando le due immagini risulta evidente la differenza sia in termini di dimensione areale sia di numerosità di apporti da corpi idrici secondari il che giustifica la modesta dimensione dell'alveo del Chisola rispetto a quella del Po anche in termini di portata transitabile. Nonostante ciò i disagi prodotti dalle piene del torrente Chisola durante eventi alluvionali sono stati molteplici; si vuol qui ricordare l'ultimo evento in ordine cronologico, quello del novembre 2016 quando a causa di una piena eccezionale si ebbe una rottura arginale che portò all'allagamento di una vasta area della Città di Moncalieri, Filippini (2016).

Quindi, in base agli obbiettivi descritti ad inizio capitolo, nei successivi paragrafi si ricostruirà il processo di azioni da mettere atto per poter avviare le simulazioni e per realizzare determinati output.

Si rimanda al *Allegato V* per una migliore rappresentazione dell'area di studio nel suo complesso.

6.2 – Operazioni preliminari all'esecuzione delle simulazioni

Ancor prima dell'esecuzione delle simulazioni devono essere eseguite una serie di operazioni quali:

- Costruzione geometria;
- Verifica dei valori di scabrezza dati;
- Creazione scenari ed idrogrammi da inserire nel modello;
- Inserimento condizioni al contorno e definizione del plan;

Come si può facilmente intuire si tratta di operazioni che esulano dal concetto di automazione trattato nei precedenti capitoli, in quanto devono essere eseguite inizialmente ed una volta sola. Si tratta di operazioni necessarie e fondamentali su cui si basa la corretta esecuzione delle simulazioni; pertanto nei successivi sotto paragrafi verranno descritte.

6.2.1 – Costruzione geometria

Come si è già detto al secondo capitolo del seguente lavoro, in HEC-RAS è possibile sviluppare differenti tipi di geometria: monodimensionale e bidimensionale (vi sarebbe anche la possibilità di avere una geometria mista mono e bidimensionale insieme). La geometria che si intende costruire in questo contesto è di tipo bidimensionale il che prevede l'utilizzo di uno strumento topografico su cui appoggiarsi. Si tratta del DTM o *Digital Terrain Model* ovvero una rappresentazione della distribuzione delle quote di un territorio in formato digitale[7]. Nello specifico si tratta di un raster ovvero un'immagine costituita da tanti pixel, ognuno di essi avente

[7] Si tratta di elaborati creati mediante tecniche di rilevamento che sfruttano sensori montati su satellite, aeromobile o stazione a terra. Il DTM non è da confondere con il DSM o *Digital Surface Model* che tiene conto di tutti gli oggetti presenti sul suolo come la vegetazione o gli edifici mentre il DTM riproduce l'andamento della superficie geodetica.

un determinato valore che rappresenta la quota del terreno. Si riporta in Figura 6.4 un estratto del DTM su cui verrà sviluppato il dominio di studio.

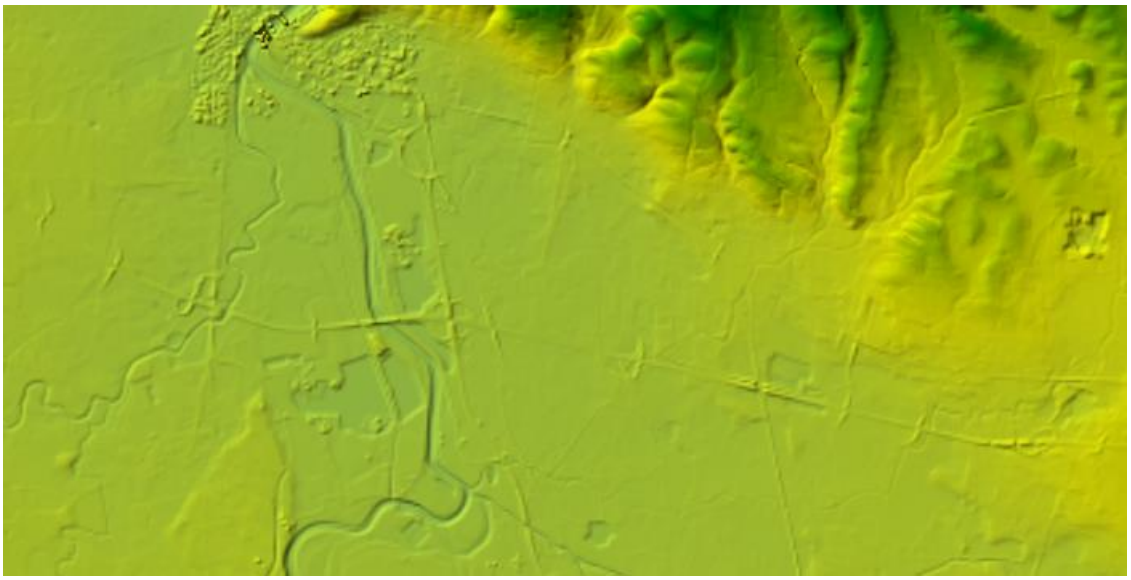


Figura 6.4: Estratto del DTM a disposizione

Per importarlo all'interno di HEC-RAS si deve seguire il seguente procedimento:

1. Accedere al RAS Mapper (Figura 6.5), si tratta di un ambiente GIS interno ad HEC-RAS nel quale è possibile importare il DTM, apportare modifiche alla geometria e visualizzare gli output anche in forma dinamica.

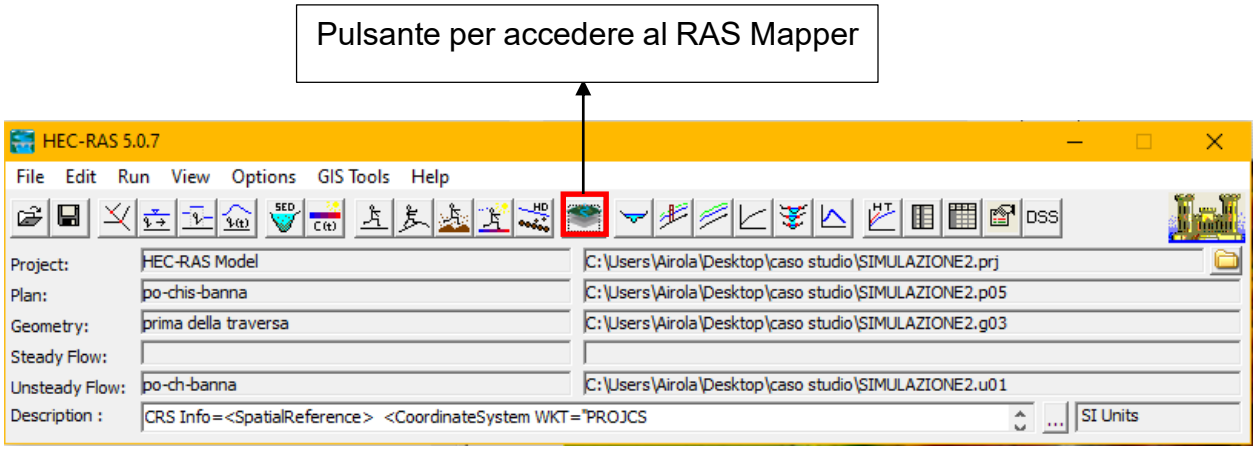


Figura 6.5: Finestra principale di HEC-RAS con evidenziato il pulsante per accedere al RAS Mapper

2. Da Tools selezionare “*Set Projection for Project”*: si apre una finestra (Figura 6.6) in cui si deve caricare il file contenente le informazioni sulle coordinate[8].

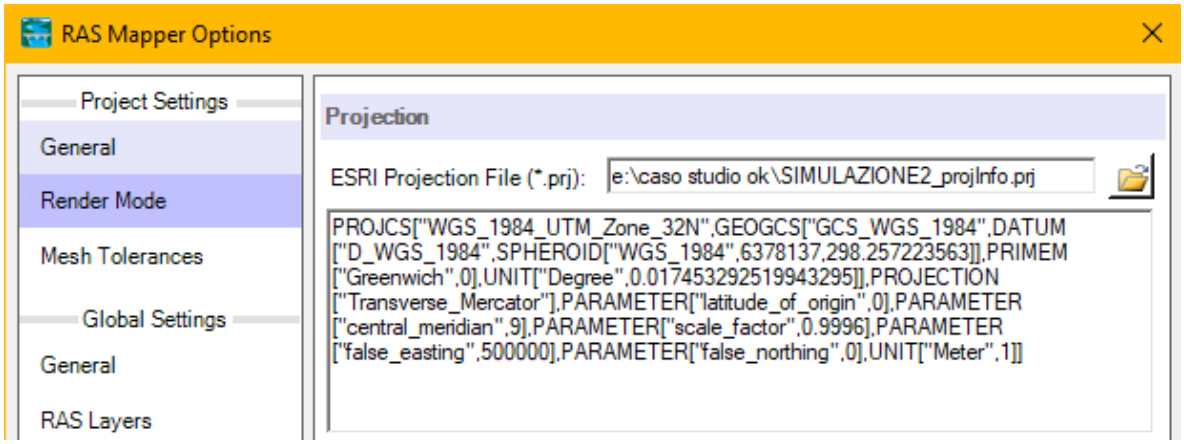


Figura 6.6: FInestra per l’inserimento del file contenente il sistema di riferimento

3. Da Tools selezionare “*New Terrain”*: si apre una finestra (Figura 6.7) in cui andare a selezionare il DTM. Una volta selezionato cliccare su *Create.*

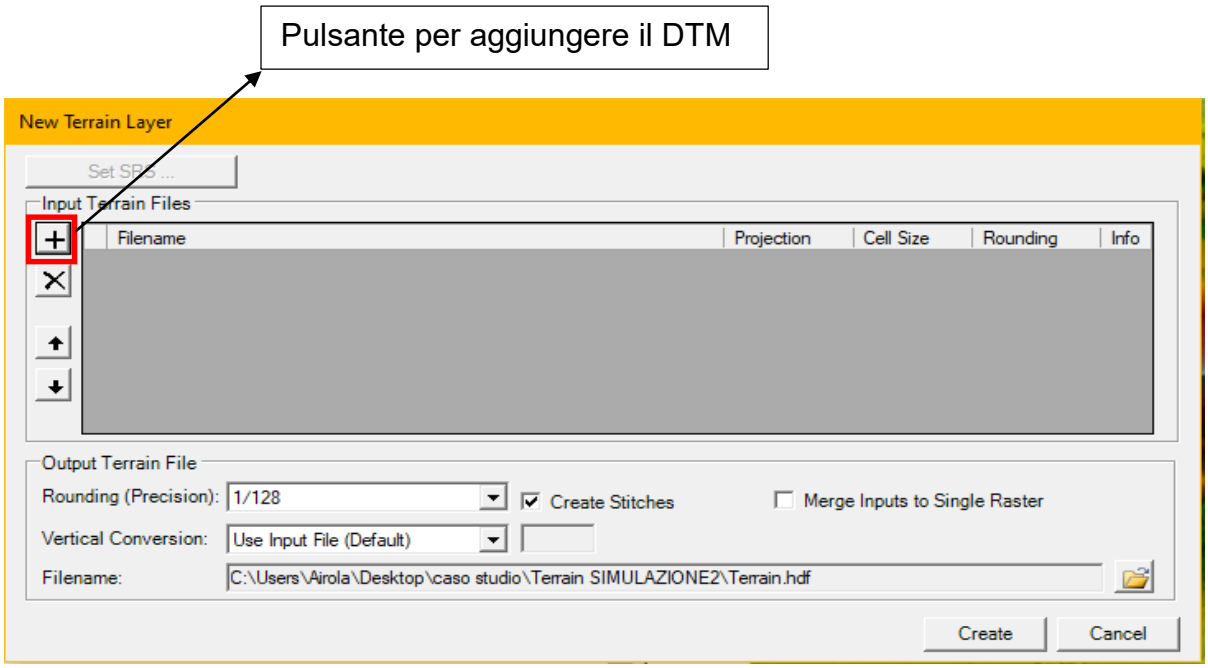


Figura 6.7: Finestra da cui è possibile ricercare il DTM

[8] Si tratta di un file con estensione “.prj” (ESRI projection file) che contiene il corretto sistema di riferimento e che può essere creato mediante ArcGIS.

Finito il caricamento il DTM compare nella finestra principale del RAS Mapper. A questo punto effettuando il salvataggio il DTM compare automaticamente anche nella geometria classica di HEC-RAS dalla quale si può procedere alla definizione del dominio di studio.

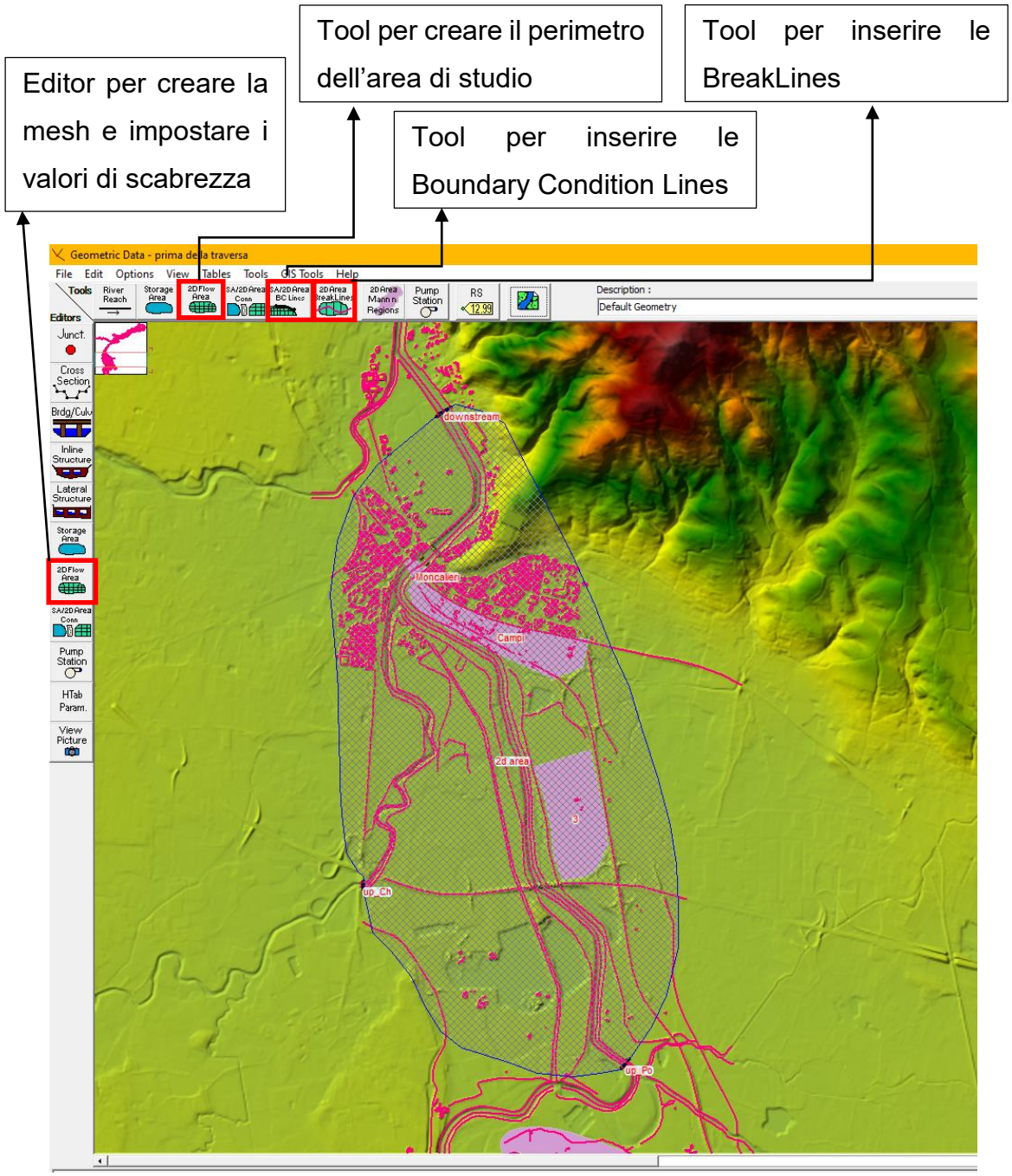


Figura 6.8: Perimetro dell'area di studio

4. Facendo riferimento all'immagine 6.8, dopo aver aperto la geometria, il primo passo è quello di determinare il perimetro dell'area di studio cliccando sul Tool "*2D Flow Area*" e creando una spezzata chiusa. Successivamente è opportuno inserire le BreakLines sfruttano il Tool "*2D Area Breaklines*", si tratta di linee da inserire laddove vi sono elementi quali edifici, rilevati arginali o stradali che, nel momento della creazione della mesh, ne permette una definizione migliore ed in fase di simulazione permette un calcolo più accurato. Inserite le breaklines si può procedere creando la mesh cliccando sul Editor "*2D Flow Area*" (Figura 6.9).

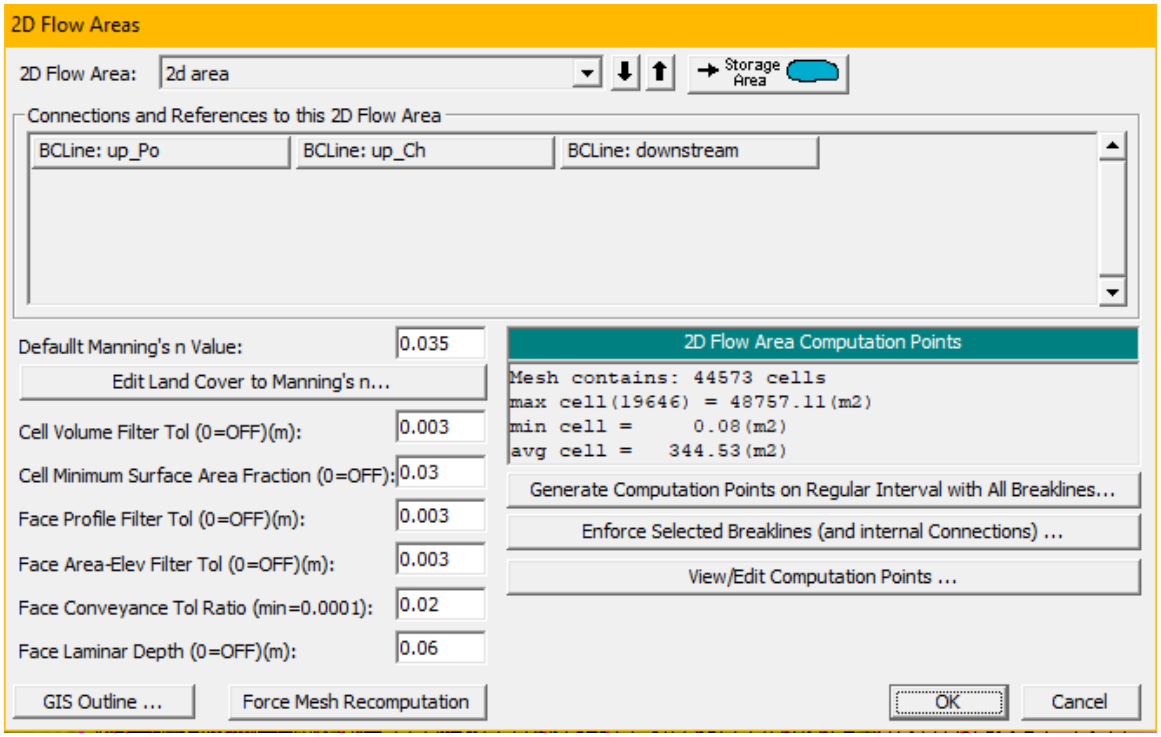


Figura 6.9: Finestra da cui generare la mesh e impostare la scabrezza

All'interno di questa finestra si deve impostare il valore di scabrezza "*Default Manning's n Value*" e inserire la dimensione delle celle della mesh cliccando su "*Generate Computation Points..*" (Figura 6.10). L'inserimento della dimensione delle celle e la generazione della mesh può essere anche eseguita da RAS Mapper abilitando la modifica dell'area 2D, i risultati saranno visibili sulla geometria classica.

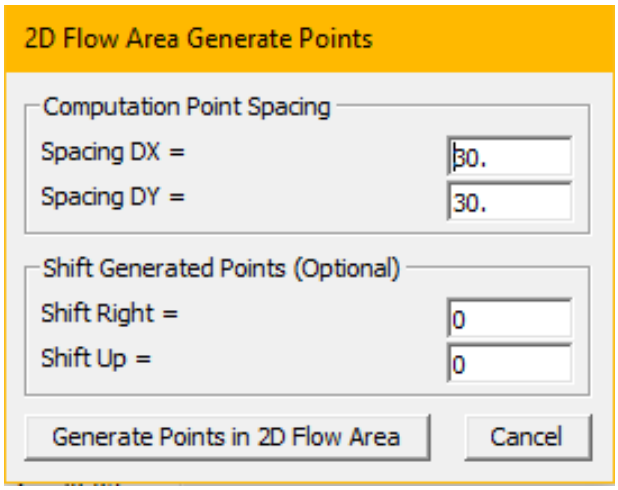


Figura 6.10: Finestra in cui inserire la dimensione delle celle

Cliccando quindi su "*ok*" (Figura 6.9) viene generata la mesh che, in corrispondenza delle breaklines risulterà più fine ed articolata come si può vedere in Figura 6.11.

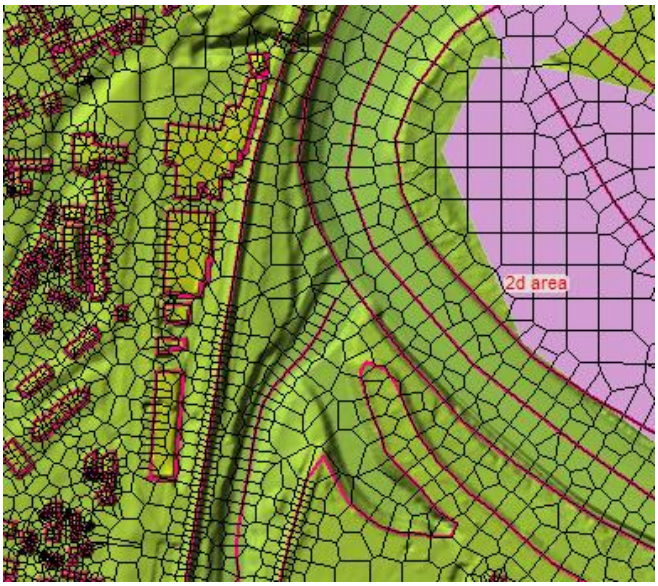


Figura 6.11: Breaklines e mesh

Può capitare che in seguito alla realizzazione della mesh vi siano degli errori, che si manifestano con dei punti rossi sulle celle contenenti l'errore. Questi errori possono essere dovuti a due motivi principalmente: il primo è che una cella abbia più di otto lati (che è il massimo consentito), oppure che i lati di due celle contigue si incrocino. In questo caso si deve intervenire manualmente laddove necessita

andando ad inserire altre celle (Edit – Add Point) o rimuovendone alcune (Edit – Remove Point) in modo tale da rispettare le varie regole imposte da HEC-RAS.
Infine, sempre facendo riferimento alla Figura 6.8, è necessario inserire le "*Boundary Condition Lines*" ovvero le sezioni di ingresso e di uscita del flusso idrico dal dominio; l'inserimento crea un collegamento diretto con lo unsteady flow editor dove si dovranno impostare le varie condizioni al contorno.

6.2.3 – Verifica dei valori di scabrezza

Altro punto importante riguarda l'inserimento dei dati relativi alla scabrezza. In questo lavoro non è stata eseguita una vera è propria calibrazione del modello in quanto si era in possesso di dati derivanti da elaborazioni già eseguite e quindi si è proceduti esclusivamente ad una verifica. L'obiettivo consiste nel verificare che, impostato un certo valore di scabrezza, la massima portata fluita nella sezione di uscita coincidesse, approssimativamente, con la massima portata dell'idrogramma della sezione di misura più vicina.
Come input sono stati inseriti gli idrogrammi del Po a Carignano e del Chisola a La Loggia inerenti all'evento del 2016 (Figura 6.12).

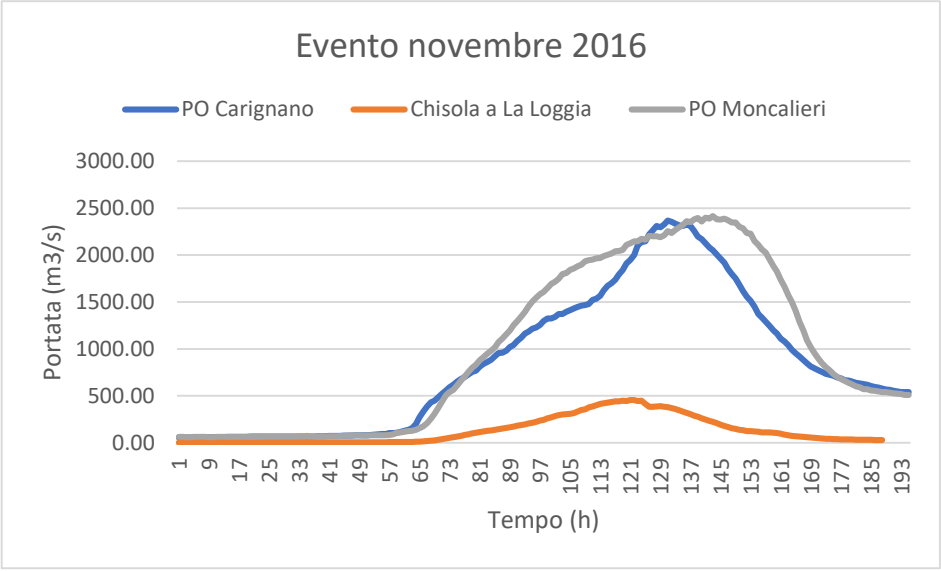


Figura 6.12: idrogrammi evento avvenuto nel novembre 2016

La portata che interessa in questo caso, oggetto di confronto, è la massima registrata in corrispondenza della sezione di misura di Moncalieri (2414 m^3/s), a valle della confluenza tra il Po e il Chisola; zona in cui è stata posta la sezione di uscita del modello. Il valore di scabrezza dato ed inserito (Figura 6.9) equivale a 0.035 $m/s^{1/3}$.

Eseguita la simulazione si sono estratti i valori massimi di portata registrati nella sezione d'uscita (Figura 6.13) accedendo al file avente estensione ".p0X.hdf" (Figura 6.14).

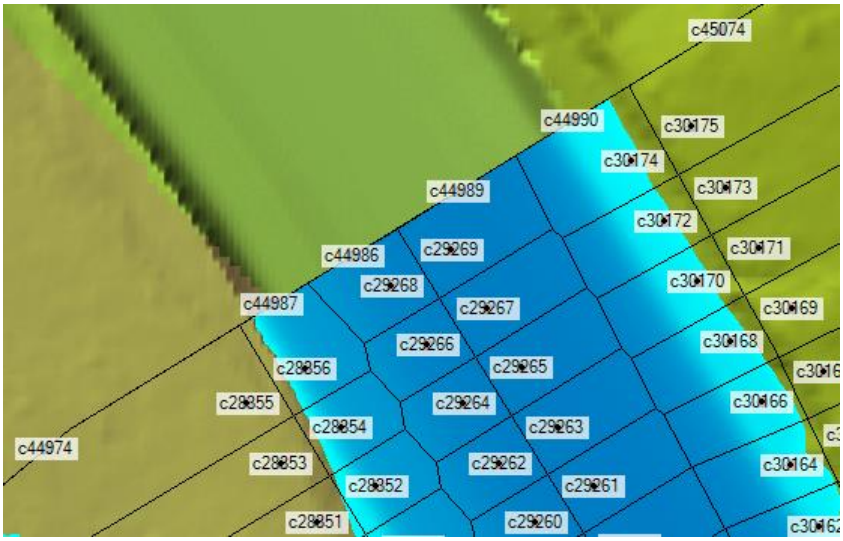
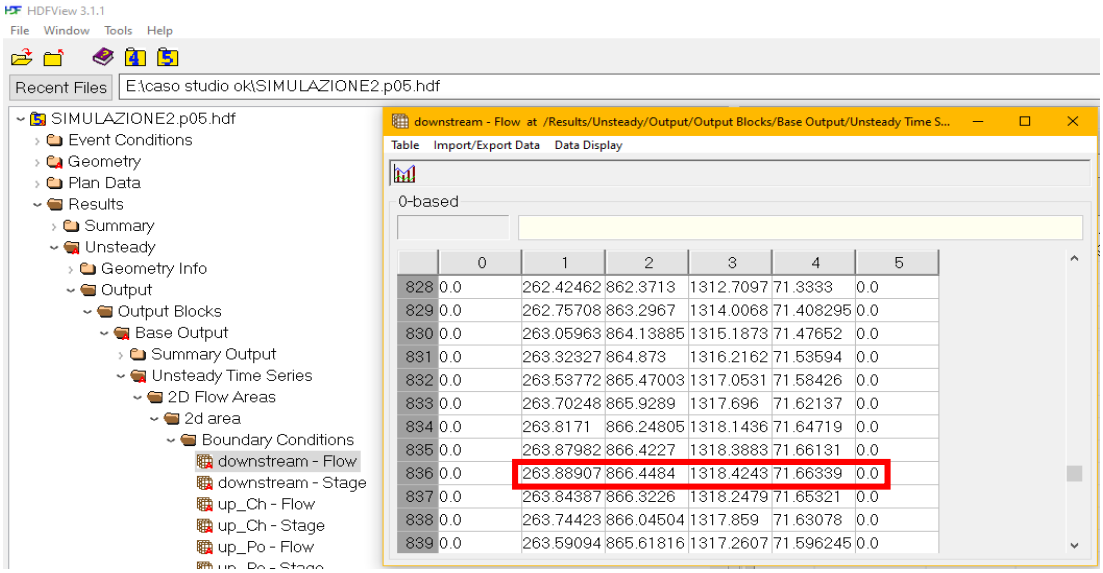


Figura 6.13: Downstream boundary del modello.



	0	1	2	3	4	5
828	0.0	262.42462	862.3713	1312.7097	71.3333	0.0
829	0.0	262.75708	863.2967	1314.0068	71.408295	0.0
830	0.0	263.05963	864.13885	1315.1873	71.47652	0.0
831	0.0	263.32327	864.873	1316.2162	71.53594	0.0
832	0.0	263.53772	865.47003	1317.0531	71.58426	0.0
833	0.0	263.70248	865.9289	1317.696	71.62137	0.0
834	0.0	263.8171	866.24805	1318.1436	71.64719	0.0
835	0.0	263.87982	866.4227	1318.3883	71.66131	0.0
836	0.0	263.88907	866.4484	1318.4243	71.66339	0.0
837	0.0	263.84387	866.3226	1318.2479	71.65321	0.0
838	0.0	263.74423	866.04504	1317.859	71.63078	0.0
839	0.0	263.59094	865.61816	1317.2607	71.596245	0.0

Figura 6.14: File ".hdf" – tabella contenente I valori di Q (m^3/s) defluita nei vari tratti dalla sezione di uscita.

La somma dei valori massimi di portata defluiti attraverso i vari tratti della sezione si fondo restituisce un valore di circa 2519 m^3/s che risulta più elevato di quello derivante dall'idrogramma e pari a 2414 m^3/s. Questa differenza può essere in realtà giustificata dal fatto che la sezione d'ingresso del Po a Carignano è stata posta circa 7 km più a valle della reale posizione quindi si sono escluse potenziali zone di laminazione che avrebbero portato ad un'ulteriore diminuzione della portata in uscita dal modello. In conclusione si può ritenere accettabile tale differenza e quindi valido il valore di 0.035 $m/s^{1/3}$.

Per quanto espresso appena sopra in merito alla non corrispondenza tra la sezione di misura del Po a Carignano e la sezione di ingresso nel dominio, al fine di rappresentare al meglio l'evento, si è dovuto porre particolare attenzione alle tempistiche di ingresso dell'idrogramma nel modello stesso. Si rende necessario, quindi, ritardare l'arrivo dell'onda di piena di un tempo pari a circa un'ora, calcolato come di seguito:

Distanza sezione di misura - sezione di ingresso	7	km
	7000	m

Velocità di deflusso dell'acqua (stima da letteratura)	2	m/s

Tempo di percorrenza tra i due punti in questione	3500	s
	0.97	**h**

Per far ciò, avendo dati di portata ogni 30 min, si sono inseriti all'inizio dell'idrogramma due valori di portata di base in modo da provocare il ritardo previsto.

6.2.4 – Creazione scenari ed idrogrammi da inserire nel modello

I macro-scenari che si intendono analizzare sono:

- Onde di piena del Po e Chisola temporalmente coincidenti;
- Onde di piena del Po e del Chisola non temporalmente coincidenti, con onda di piena del Po in arrivo prima dell'onda di piena del Chisola;
- Onde di piena del Po e del Chisola non temporalmente coincidenti, con onda di piena del Chisola in arrivo prima dell'onda di piena del Po;

Inoltre, per ciascuno scenario sono state considerate onde di piena aventi tempi di ritorno differenti; ad esempio onda di piena del Po con T 100 anni e T 200 anni e analogamente per il Chisola e, di conseguenza, sono state eseguite più simulazioni considerando le possibili combinazioni che potrebbero effettivamente verificarsi:

- T 100 Po – T 100 Ch;
- T 100 Po – T 200 Ch;
- T 200 Po – T 100 Ch;
- T 200 Po – T 200 Ch.

Queste sono le combinazioni che si intendono analizzare per i tre scenari esposti precedentemente; si tratta quindi di 12 simulazioni che saranno eseguite in serie con la modalità espressa nei precedenti capitoli e che sarà ulteriormente ripresa successivamente.

Per quanto riguarda la costruzione degli idrogrammi è stata fatta inizialmente un'analisi dei dati di portata disponibili per le stazioni idrometriche del Po a Carignano e Moncalieri, del Chisola a La Loggia e del Banna a Santena. Si tratta di dati disposti in file di testo con un preciso formato (Figura 6.15):

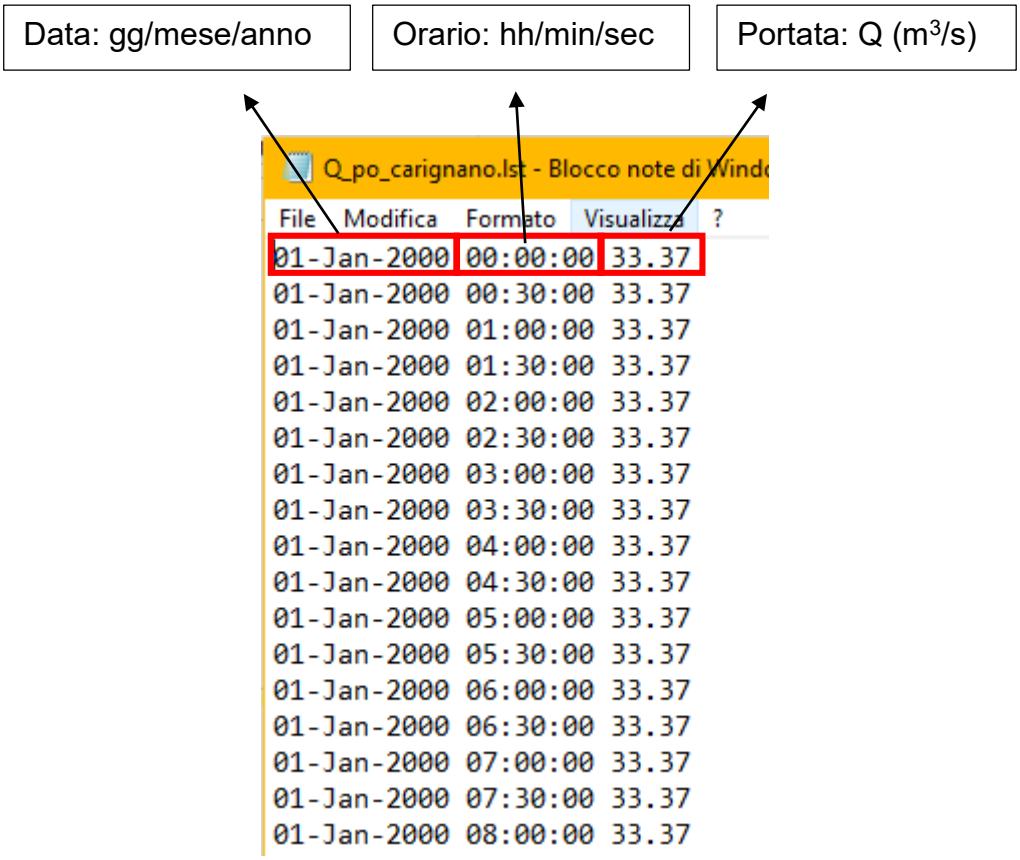


Figura 6.15: Estratto file dati di monitoraggio

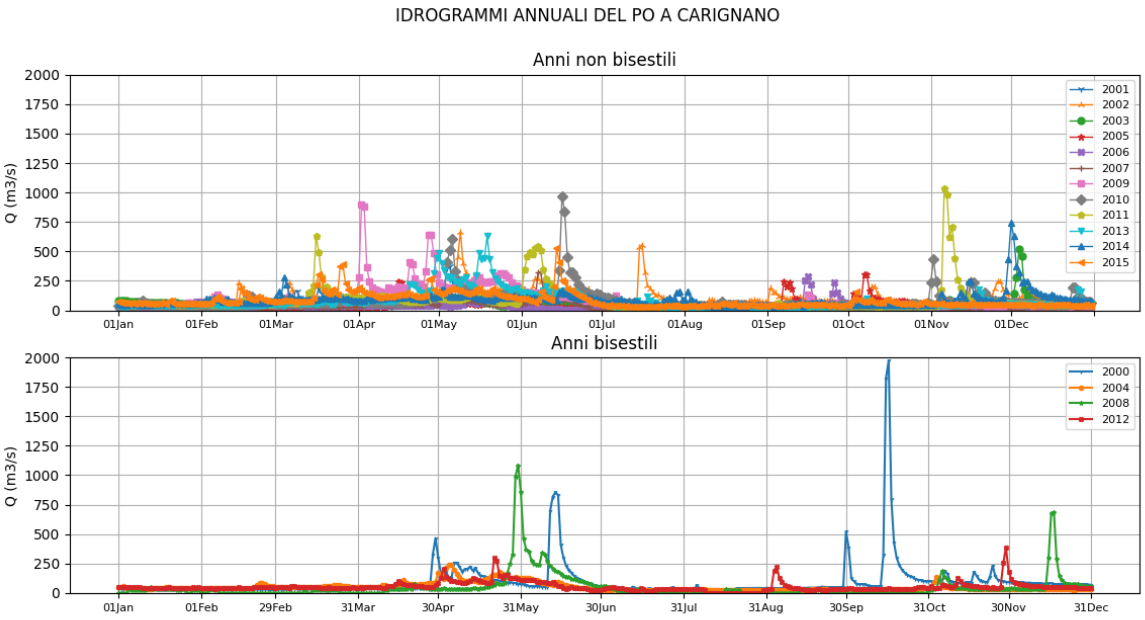
Per quanto riguarda i dati a disposizione si hanno:

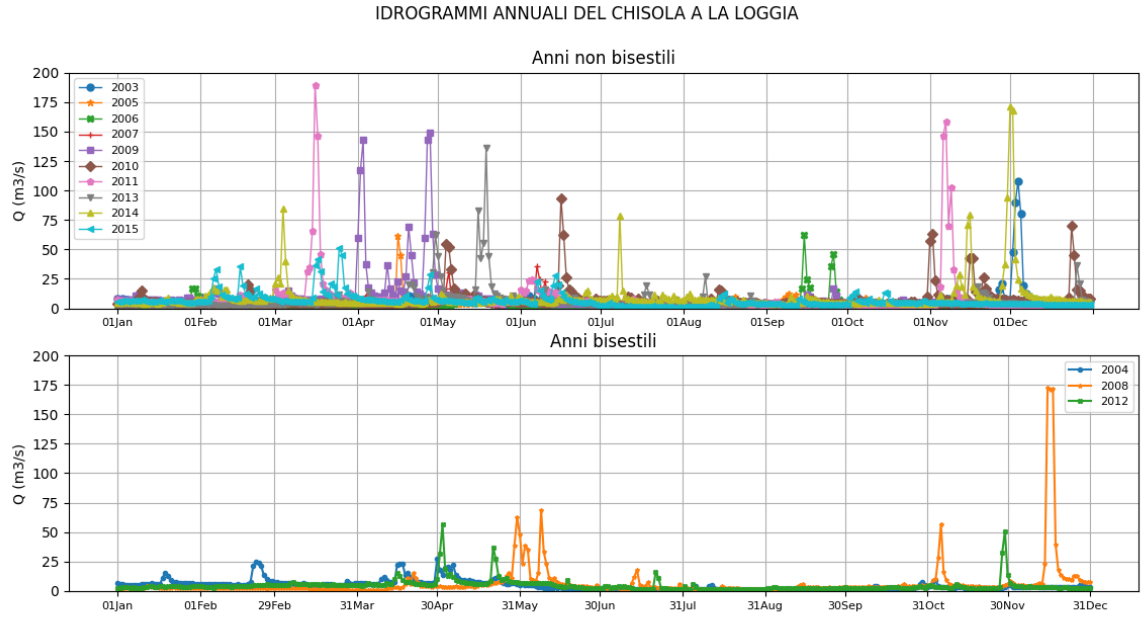
- Per il Po a Carignano la serie di portate va da Gennaio 2000 fino a Dicembre 2015 con un passo temporale di 30 minuti;
- Per il Po a Moncalieri la serie va da Gennaio 2007 a Dicembre 215 con un passo temporale di 30 min
- Per il Chisola i dati forniti vanno da Gennaio 2003 a Dicembre 2015 con passo temporale di 30 min;
- Per il Banna i dati forniti vanno da Gennaio 2003 a Ottobre 2015 con passo temporale di 30 min.

Si può quindi intuire come si abbia a che fare con un'elevata mole di dati che difficilmente può essere analizzata con semplici fogli di calcolo quali Excel in quanto sarebbe molto complicato gestire fisicamente tutti i dati. Pertanto, in questo caso, è

stato ancora utilizzato Python che, come visto già in precedenza, permette di gestire ed analizzare con relativa semplicità i file di testo.

Primo passo dell'analisi è stato quello di estrarre gli eventi più importanti avvenuti negli anni a disposizione, sono quindi stati creati degli idrogrammi annuali, come quelli mostrati in Figura 6.16, 6.17, 6.18 e 6.19:





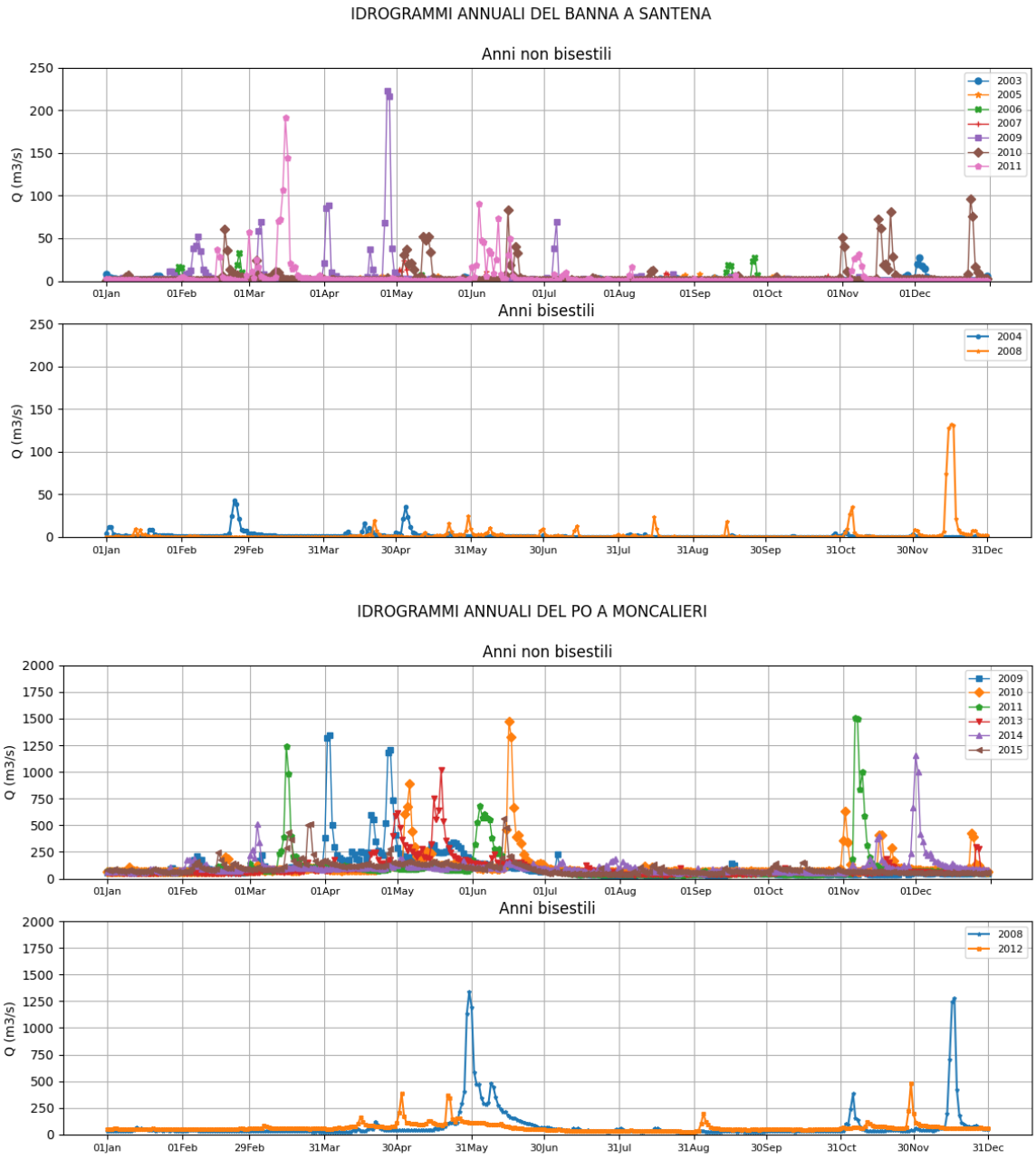
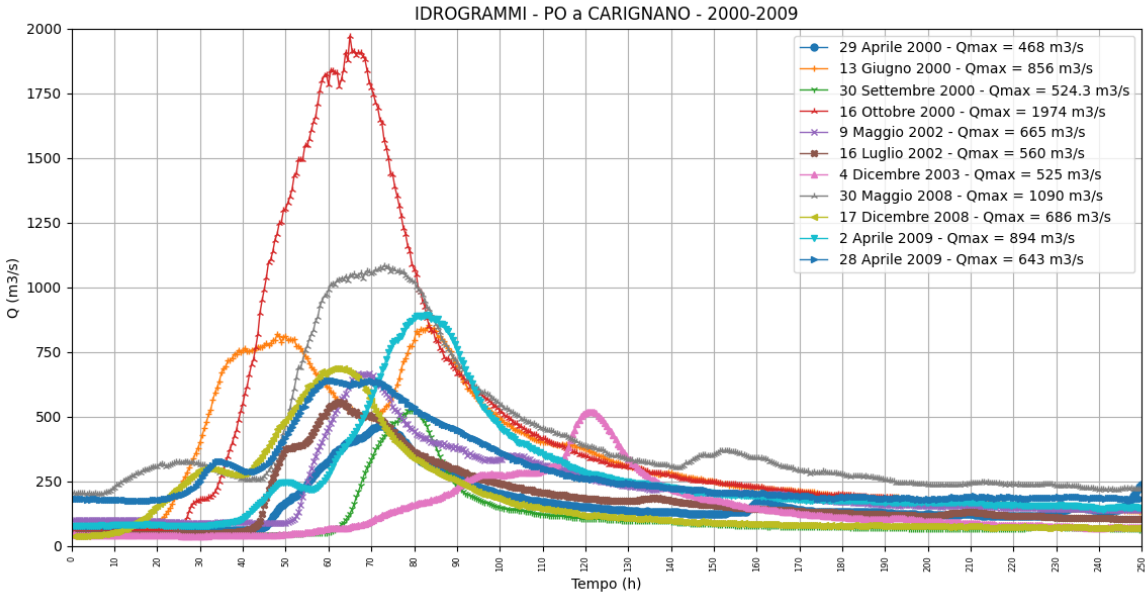


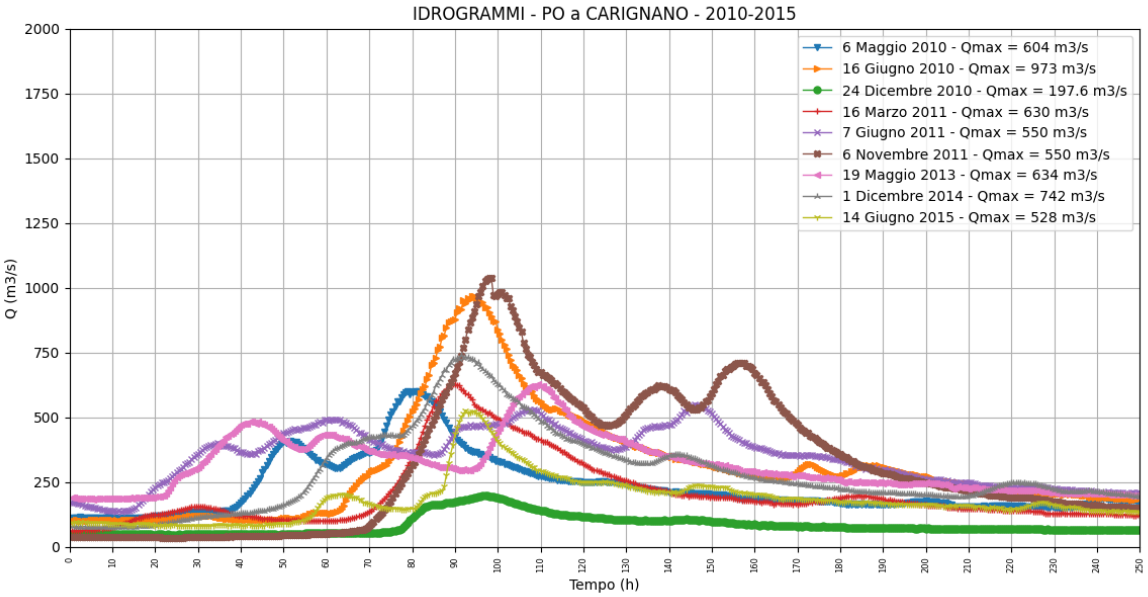
Figura 6.16, 6.17, 6.18 e 6.19: Idrogrammi annuali nelle varie stazioni di interesse

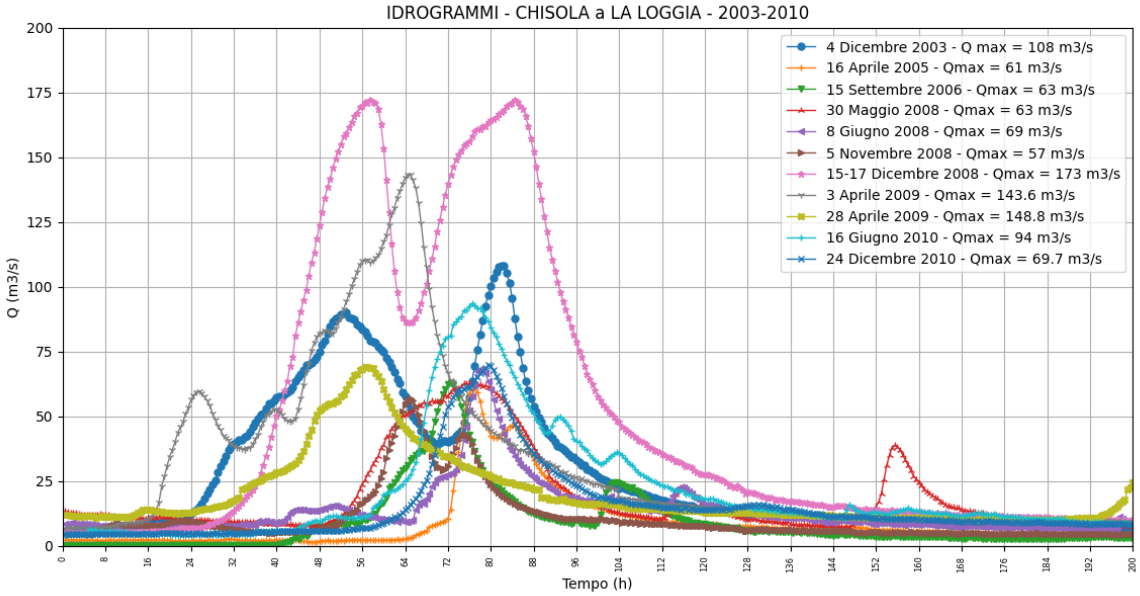
Si sono separati gli anni bisestili semplicemente per motivi di costruzione del grafico, visto che la presenza di un giorno in più non ne permetteva la corretta sovrapposizione; inoltre la mancanza di idrogrammi annuali del Banna è dovuta all'assenza di dati che non li rende sovrapponibili.

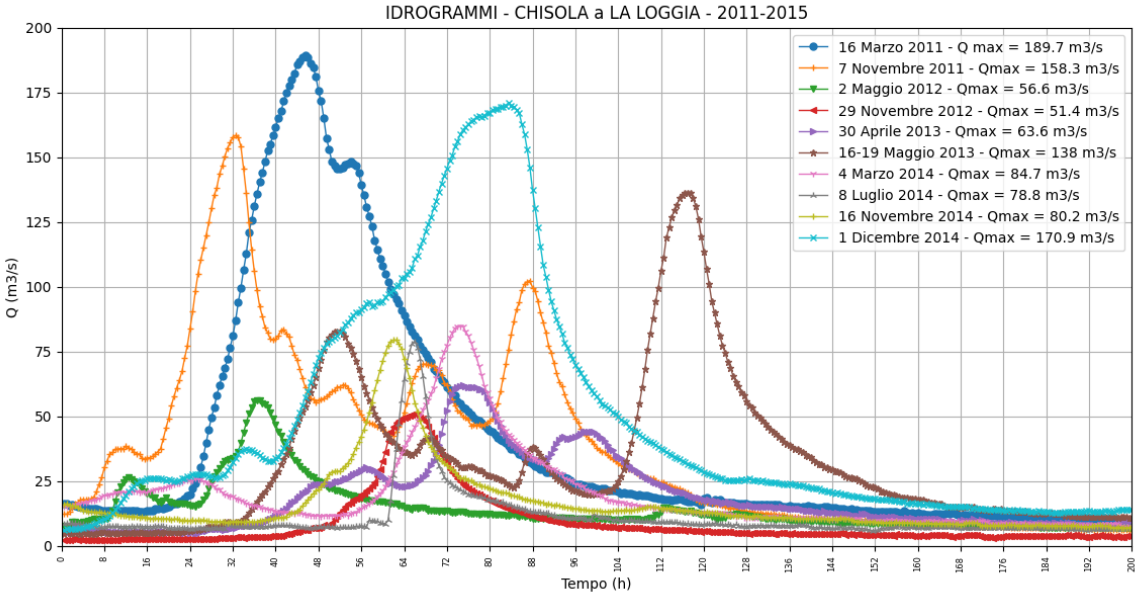
Da tali idrogrammi si può notare, in linea generale, come gli eventi più importanti si concentrino nel periodo primaverile (aprile – giugno) ed autunnale (settembre – novembre) con piccoli ed isolati eventi estivi. Andando più nello specifico ed estraendo gli eventi più importanti si è potuto notare un'ottima sincronia per gli eventi con picchi in assoluto più elevati, ciò è dovuto al fatto che nonostante la dimensione dei bacini sia notevolmente diversa, la loro vicinanza fa sì che un evento meteorico importante e diffuso con elevata probabilità coinvolga entrambi producendo un aumento delle portate in alveo. Vi sono poi eventi di media entità che non sempre provocano un aumento sincronizzato, differentemente da quanto detto prima ciò può essere dovuto alla concentrazione della precipitazione in una zona più ristretta e limitata.

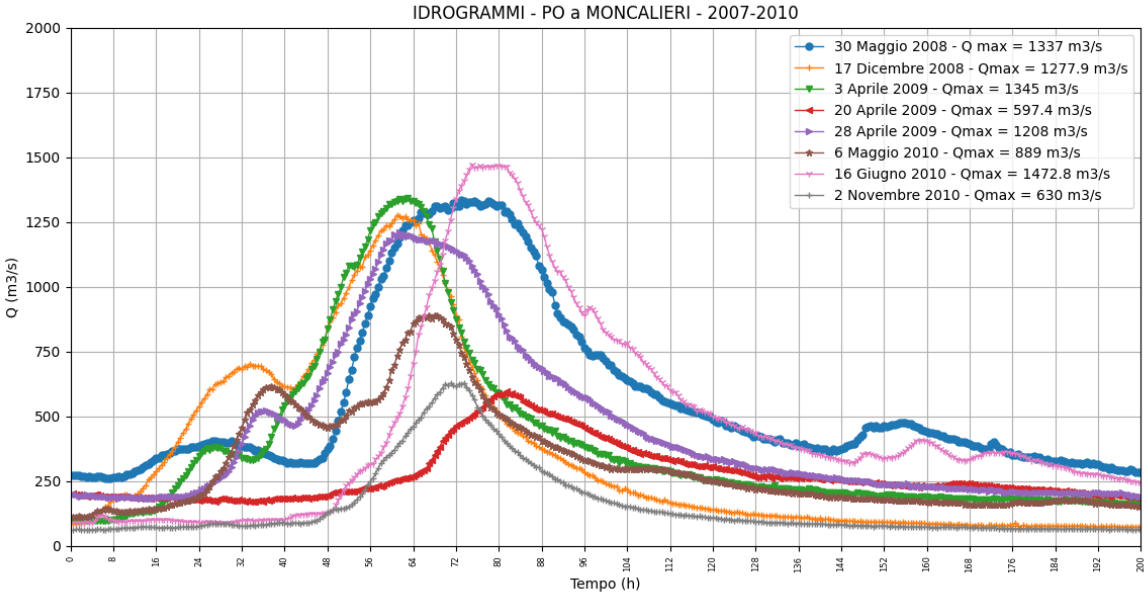
Si riportano di seguito nelle Figure 6.20, 6.21, 6.22, 6.23, 6.24 e 6.25 gli eventi maggiori estrapolati. Si sono suddivisi in due grafici per ogni stazione di misura per evitare un eccessivo sovrapporsi di linee.

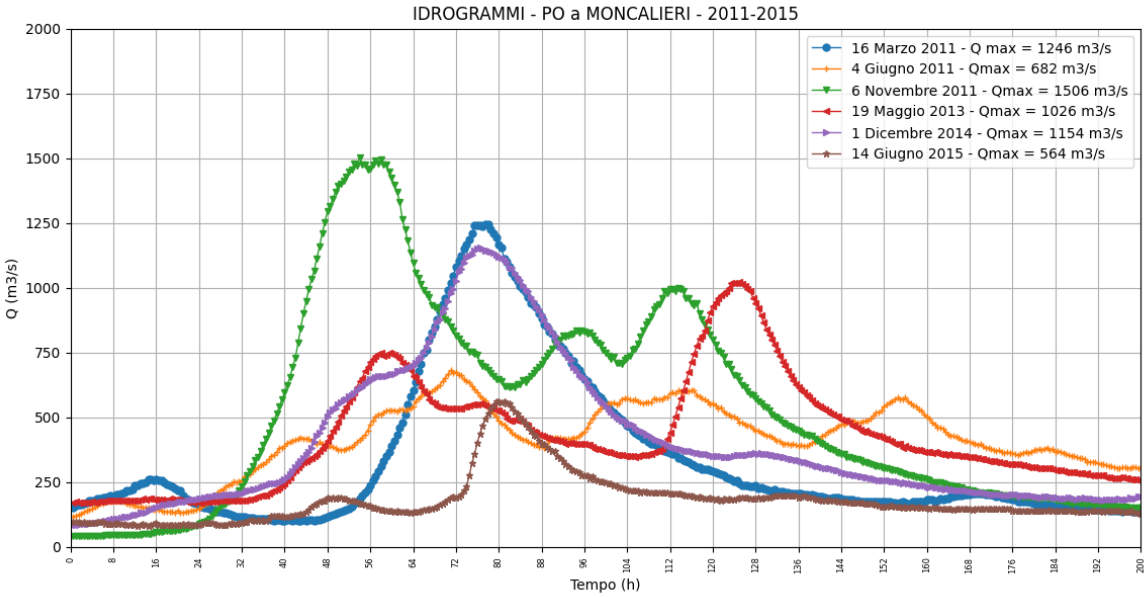



IDROGRAMMI - PO a CARIGNANO - 2010-2015
Q (m3/s)
Tempo (h)
6 Maggio 2010 - Qmax = 604 m3/s
16 Giugno 2010 - Qmax = 973 m3/s
24 Dicembre 2010 - Qmax = 197.6 m3/s
16 Marzo 2011 - Qmax = 630 m3/s
7 Giugno 2011 - Qmax = 550 m3/s
6 Novembre 2011 - Qmax = 550 m3/s
19 Maggio 2013 - Qmax = 634 m3/s
1 Dicembre 2014 - Qmax = 742 m3/s
14 Giugno 2015 - Qmax = 528 m3/s


IDROGRAMMI - CHISOLA a LA LOGGIA - 2003-2010
Q (m3/s)
Tempo (h)
4 Dicembre 2003 - Q max = 108 m3/s
16 Aprile 2005 - Qmax = 61 m3/s
15 Settembre 2006 - Qmax = 63 m3/s
30 Maggio 2008 - Qmax = 63 m3/s
8 Giugno 2008 - Qmax = 69 m3/s
5 Novembre 2008 - Qmax = 57 m3/s
15-17 Dicembre 2008 - Qmax = 173 m3/s
3 Aprile 2009 - Qmax = 143.6 m3/s
28 Aprile 2009 - Qmax = 148.8 m3/s
16 Giugno 2010 - Qmax = 94 m3/s
24 Dicembre 2010 - Qmax = 69.7 m3/s


IDROGRAMMI - CHISOLA a LA LOGGIA - 2011-2015
Q (m3/s)
Tempo (h)
16 Marzo 2011 - Q max = 189.7 m3/s
7 Novembre 2011 - Qmax = 158.3 m3/s
2 Maggio 2012 - Qmax = 56.6 m3/s
29 Novembre 2012 - Qmax = 51.4 m3/s
30 Aprile 2013 - Qmax = 63.6 m3/s
16-19 Maggio 2013 - Qmax = 138 m3/s
4 Marzo 2014 - Qmax = 84.7 m3/s
8 Luglio 2014 - Qmax = 78.8 m3/s
16 Novembre 2014 - Qmax = 80.2 m3/s
1 Dicembre 2014 - Qmax = 170.9 m3/s


IDROGRAMMI - PO a MONCALIERI - 2007-2010
Q (m3/s)
Tempo (h)
30 Maggio 2008 - Q max = 1337 m3/s
17 Dicembre 2008 - Qmax = 1277.9 m3/s
3 Aprile 2009 - Qmax = 1345 m3/s
20 Aprile 2009 - Qmax = 597.4 m3/s
28 Aprile 2009 - Qmax = 1208 m3/s
6 Maggio 2010 - Qmax = 889 m3/s
16 Giugno 2010 - Qmax = 1472.8 m3/s
2 Novembre 2010 - Qmax = 630 m3/s



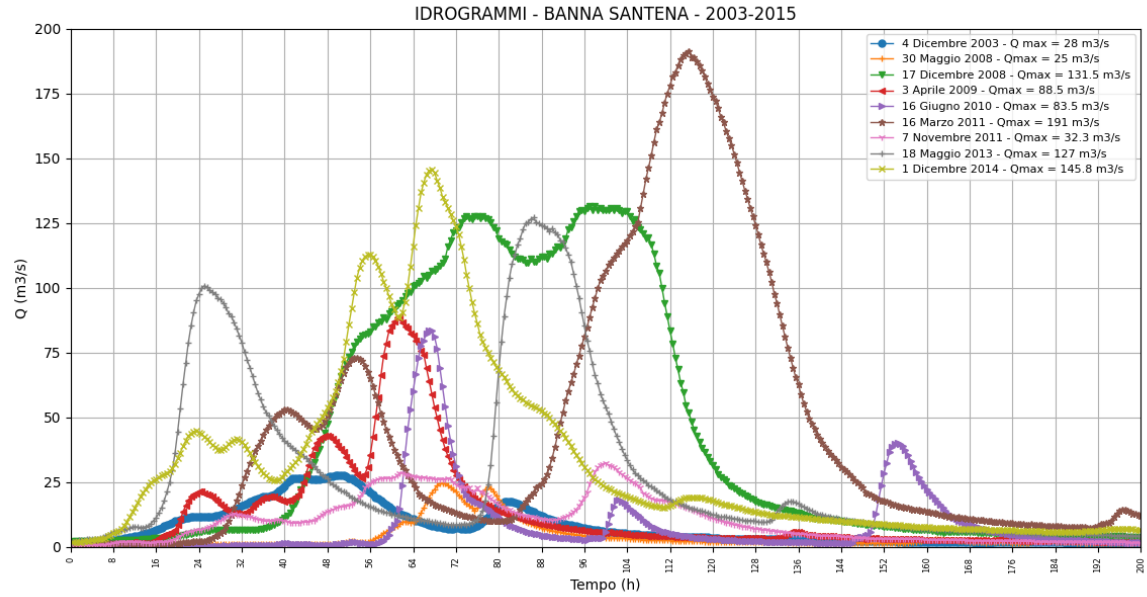


Figura 6.20, 6.21, 6.22, 6.23, 6.24 e 6.25: Idrogrammi degli eventi più gravosi nelle tre stazioni di misura.

In aggiunta agli idrogrammi riportati precedentemente è importante considerare anche quello inerente all'evento del novembre 2016 che ha provocato allagamenti importanti nella città di Moncalieri. Tale evento non è compreso nella serie di dati dalla quale sono stati estratti i precedenti idrogrammi in quanto si ferma al 2015 ma si hanno a disposizione altri dati specifici dell'evento che si riportano di seguito:

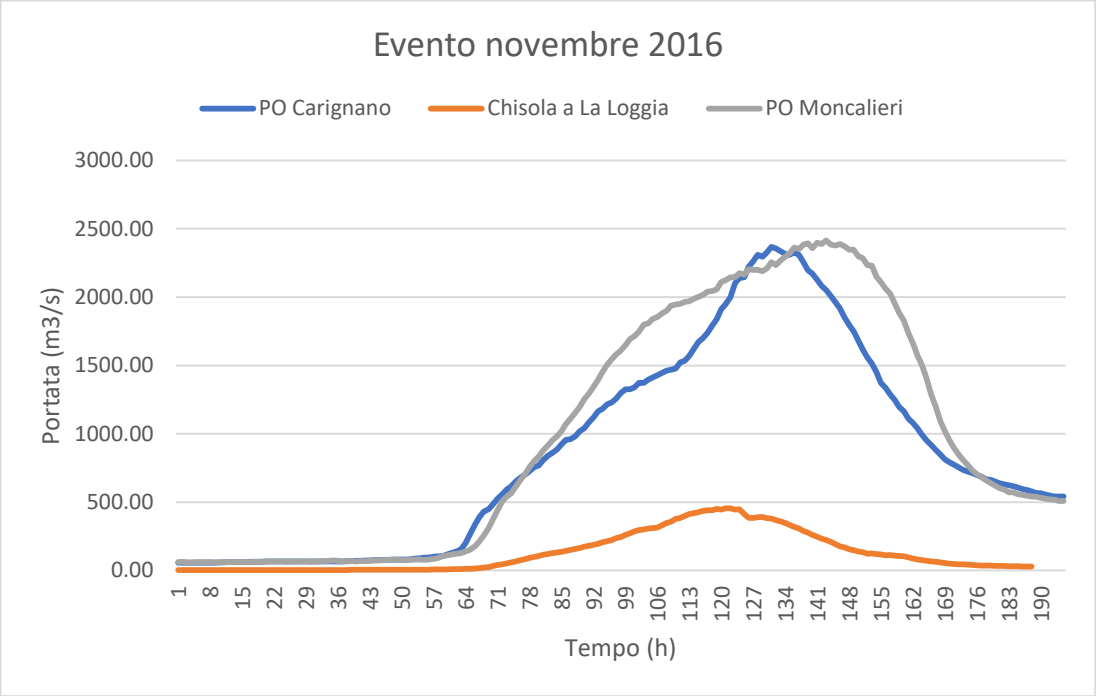


Figura 6.26: Idrogrammi del Po a Carignano e Moncalieri e del Chisola a La Loggia

Il secondo passo dell'analisi è quello di determinare i tempi di ritorno associati alle portate massime specialmente degli idrogrammi del Po a Carignano e del Chisola che rappresentano i due input di monte. Questo con l'intento di determinare un idrogramma "medio" scalabile poi in base al valore di Q_T, ovvero la portata associata ad un determinato tempo di ritorno, e quindi creare appositi idrogrammi da inserire nel modello.

Per la determinazione dei tempi di ritorno si è fatto riferimento alle curve di crescita reperite dal geoportale dell'Arpa Piemonte. Nello specifico si sono utilizzate quelle determinate mediante il metodo ARPIEM che sfrutta la distribuzione lognormale a tre parametri[9]. Si riporta di seguito la curva di crescita riferita al Po a Carignano (Tabella 6.1 e Figura 6.27) dalla quale mediante interpolazione lineare sono stati ricavati i tempi di ritorno per determinati eventi (Tabella 6.2)

Tab. 6.1: Dati curva di crescita – Po a Carignano

T (anni)	Q_T (m^3/s)
2	540
5	916
10	1202
20	1502
50	1926
100	2272
200	2641
500	3169
1000	3600

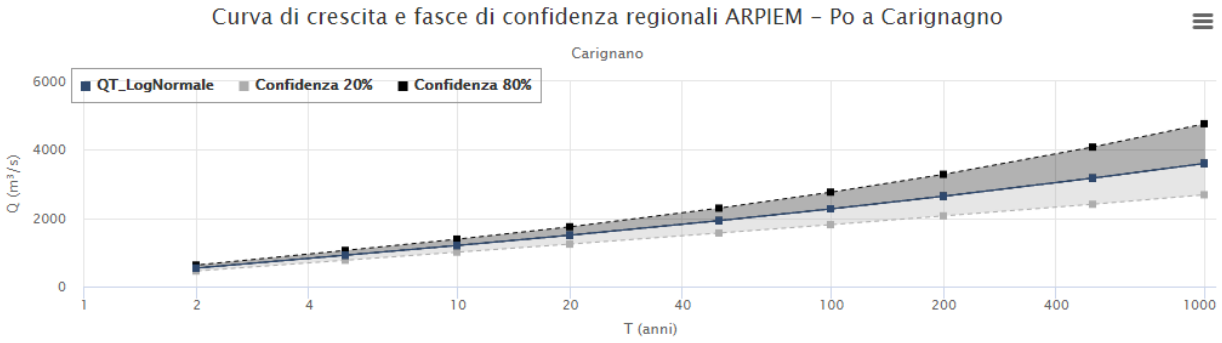


Figura 6.27: Curva di crescita del Po a Carignano

[9]Sito:https://webgis.arpa.piemonte.it/agportal/apps/opsdashboard/index.html#/19fc2f223314452cb5bd7548d67885b3. Svariati studi hanno dimostrato come la distribuzione lognormale a tre parametri sia applicabile ad una gran varietà di fenomeni idrologici, specialmente quando le variabili hanno limite inferiore. L'applicabilità si estende dalla modellazione di picchi di portata fino alla descrizione delle concentrazioni di inquinanti in atmosfera.

Tab. 6.2: Stima del tempo di ritorno - eventi Po a Carignano

Evento	16-Ott-00	9-Mag-02	4-Dic-03	30-Mag-08	17-Dic-08	3-Apr-09
Q_T (m^3/s)	1973	665	525	1090	686	894
T (anni)	57	3	2	6	3	5

Evento	16-Giu-10	16-Mar-11	6-Nov-11	19-Mag-13	1-dic-14	23-Nov-16
Q_T (m^3/s)	973	630	1038	634	742	2368
T (anni)	5	3	6	3	4	126

Anche per il Chisola si riportano in Tabella 6.3 e Figura 6.28 i dati riguardanti la curva di crescita mentre in Tabella 6.4 vi è la stima dei tempi di ritorno per gli eventi più importanti.

Tab. 6.3: Dati curva di crescita – Chisola a La Loggia

T (anni)	Qt (m^3/s)
2	117
5	181
10	241
20	313
50	429
100	536
200	661
500	858
1000	1033

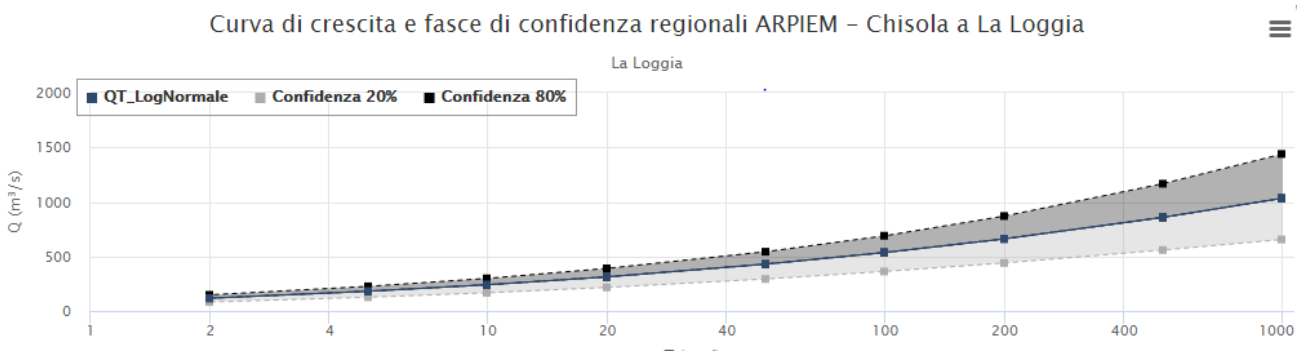


Figura 6.28: Curva di crescita del Chisola a La Loggia

Tab. 6.4: Stima del tempo di ritorno - eventi Chisola a La Loggia

Evento	17-Dic-08	3-Apr-09	28-Apr-09	16-Mar-11
Q_T (m^3/s)	173	144	149	189
T (anni)	**5**	**3**	**3**	**6**

Evento	7-Nov-11	19-Mag-13	1-Dic-14	23-Nov-16
Q_T (m^3/s)	158	138	171	650
T (anni)	**4**	**3**	**5**	**191**

Osservando i risultati ottenuti per le due stazioni di misura si è notato che la maggior parte degli eventi presentano un tempo di ritorno inferiore ai 10 anni, pertanto l'intenzione iniziale di creare un idrogramma "medio", da scalare poi a tempi di ritorno superiori (50, 100, 200 anni), non si è ritenuta valida. Si è quindi pensato di utilizzare l'idrogramma più grande registrato in ogni sezione di misura e normalizzarlo in modo tale da creare delle onde di piena di riferimento scalabili in base ai valori di portata di picco estraibili dalle curve di crescita. Gli idrogrammi saranno quindi costruibili sfruttando la seguente formula:

Formula 6.1: espressione per la costruzione degli idrogrammi

$$Q(t) = Q_T \cdot q(t)$$

Dove:

- Q_T = picco per un determinato tempo di ritorno, ottenuto dalla stima della curva di crescita;
- q(t) = idrogramma di riferimento normalizzato.

Per quanto riguarda il torrente Banna si riportano esclusivamente i dati della curva di crescita.

Tab. 6.5: Dati curva di crescita – Banna a Santena

T (anni)	QT (m3/s)
2	60
5	100
10	134
20	174
50	234
100	288
200	348
500	441
1000	520

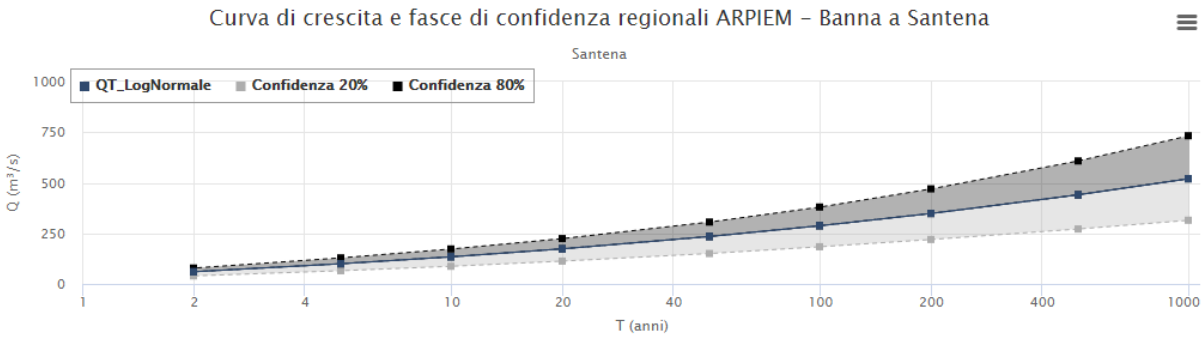


Figura 6.29: Curva di crescita del Banna a Santena

Considerando i dati a disposizione gli eventi più gravosi per le tre sezioni di misura di monte sono:

- Po a Carignano – Novembre 2016 (Q_{MAX} = 2368 m^3/s);
- Chisola a La Loggia – Novembre 2016 (Q_{MAX} = 650 m^3/s);
- Banna a Santena – Novembre-Dicembre 2014 (Q_{MAX} = 145 m^3/s);

La normalizzazione degli idrogrammi, effettuata rispetto al valore massimo (Formula 6. 2) ha permesso di ricavare le forme delle onde di piena (Figura 6.30):

Formula 6.2: espressione per la normalizzazione degli idrogrammi

$$q(t) = \frac{Q(t)}{Q_{MAX}}$$

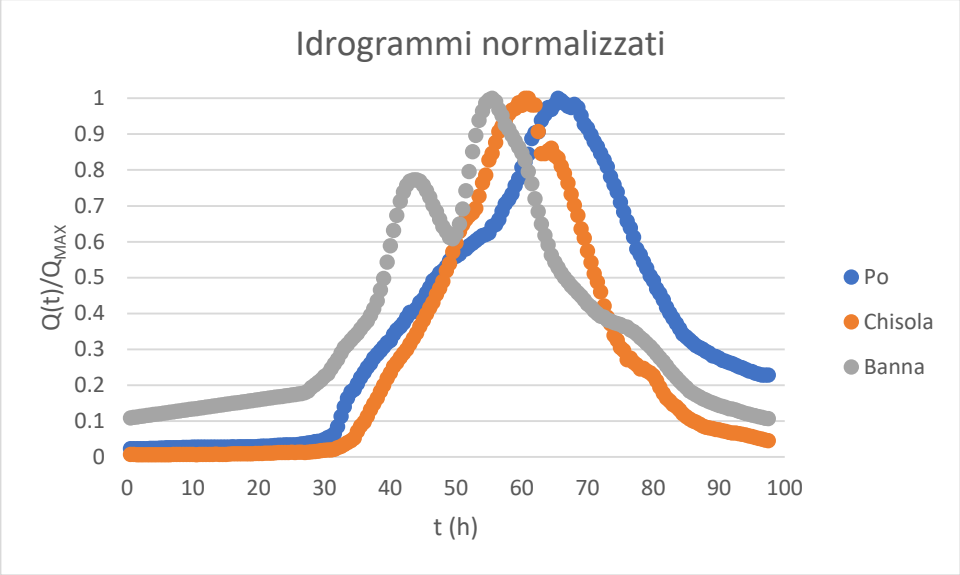


Figura 6.30: Idrogrammi normalizzati

Da essi, sfruttando la Formula 6.2, si sono ricavati gli idrogrammi per diversi tempi di ritorno (Figura 6.31 e 6.32). Gli idrogrammi del Po e del Banna sono stati sommati tra di loro in modo da crearne uno unico, ciò è stato fatto in quanto la sezione di ingresso del Po e nelle strette vicinanze della confluenza con il Banna ed inoltre il contributo in termini di portata non è molto significativo vista la ridotta dimensione dell'alveo e del bacino in generale.

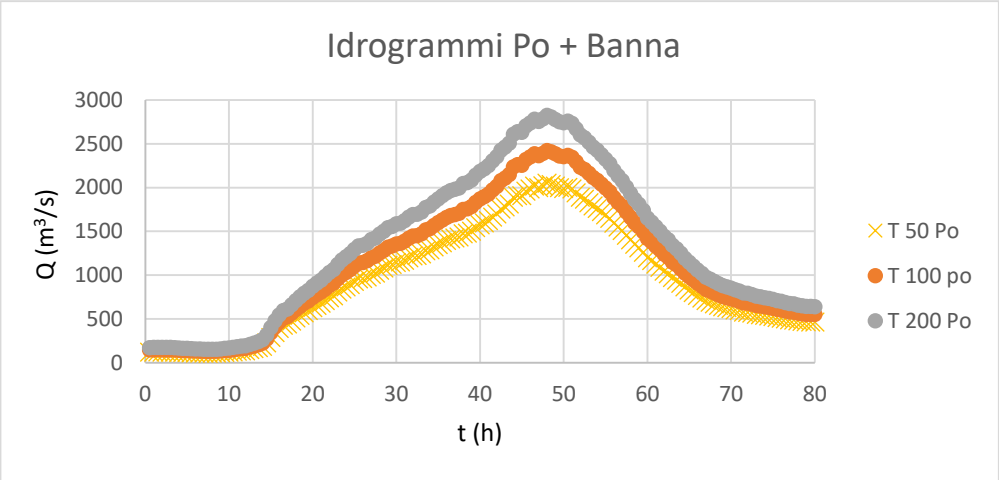


Figura 6.31: Idrogrammi Po + Banna per diversi “T”

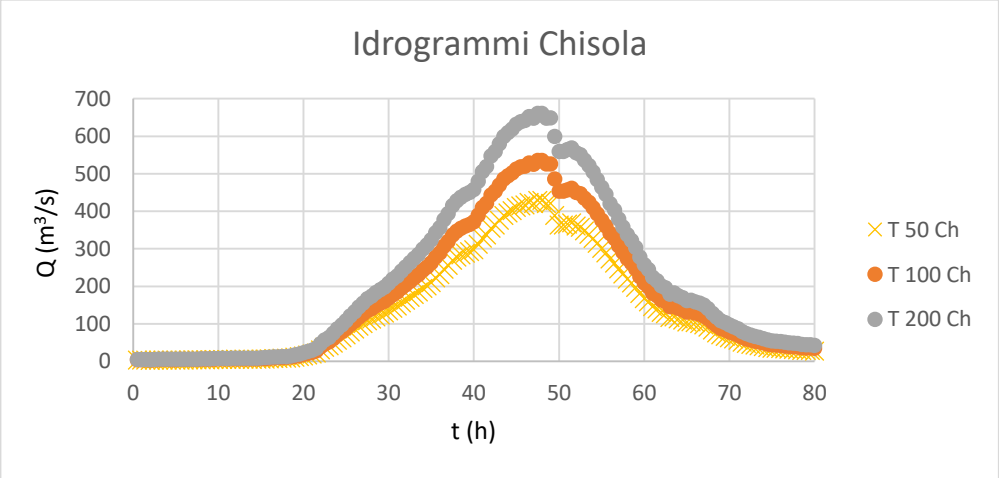


Figura 6.32: Idrogrammi Chisola per diversi “T”

Una volta creati gli idrogrammi sono stati formati i vari scenari andando a mettere insieme le onde di piena del Po e del Chisola e traslandole in modo da determinare tempistiche di arrivo differenti alla confluenza.

Come riportato all'inizio del paragrafo le simulazioni che si intendono effettuare sono:

Tab. 6.6: casistiche da simulare

Scenari	T Po	T Ch
Scenario 1: picchi onde di piena temporalmente coincidenti	100	100
	100	200
	200	100
	200	200
Scenario 2: picco onda di piena del Po prima di quello del Chisola	100	100
	100	200
	200	100
	200	200
Scenario 3: picco onda di piena del Po dopo di quello del Chisola	100	100
	100	200
	200	100
	200	200

Le combinazioni su grafico per lo scenario 1 sono le seguenti:

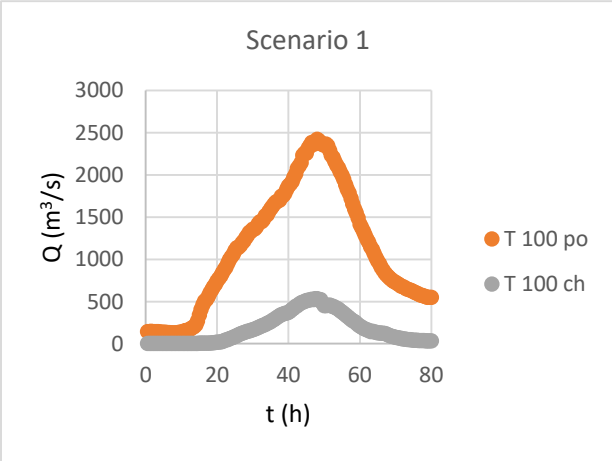


Figura 6.33: Scenario 1 – T 100-100

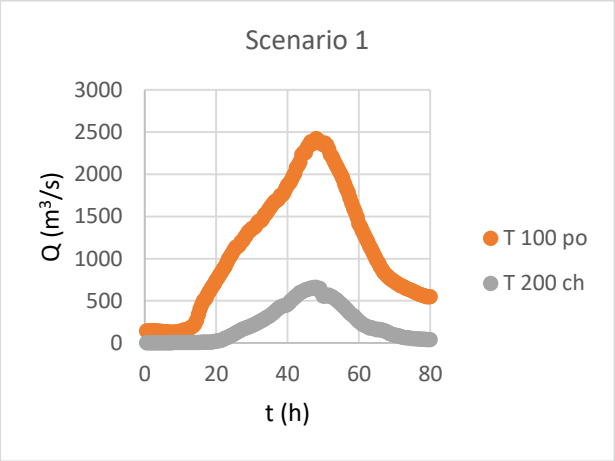


Figura 6.34: Scenario 1 – T 100-200

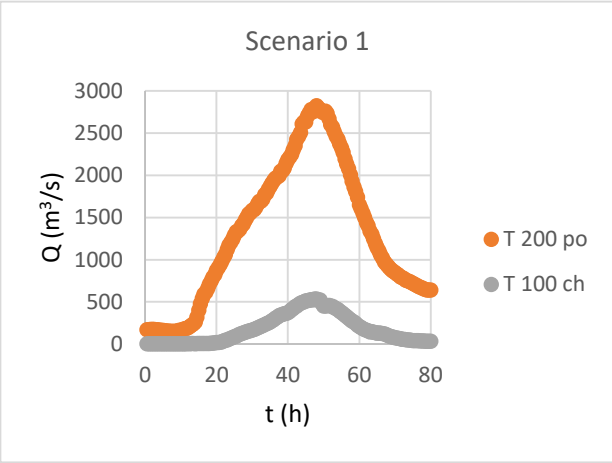


Figura 6.35: Scenario 1 – T 200-100

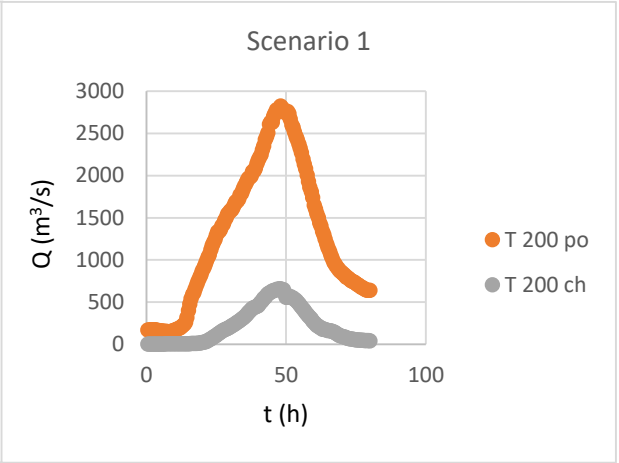


Figura 6.36: Scenario 1 – T 200-200

I colmi delle onde di piena non sono esattamente coincidenti bensì i picchi degli idrogrammi del Chisola sono traslati leggermente verso destra, questo perché al fine di ottenere la coincidenza delle onde di piena alla confluenza è necessario tenere in considerazione le diverse distanze che intercorrono tra le sezioni di ingresso al modello e la confluenza che significa anche diversi tempi di arrivo.

Calcolo sfasamento per ottenere la coincidenza degli idrogrammi

CHISOLA		
Dist. Ingresso - Confluenza	4000	m
Velocità media di percorrenza	2	m/s
t_CH	2000	s
	0.6	h

PO		
Dist. Ingresso - Confluenza	7000	m
Velocità media di percorrenza	2	m/s
t_PO	3500	s
	1.0	h

I calcoli mostrati presentano una stima del tempo che impiega il flusso d'acqua a percorrere i due tratti considerando una velocità media di percorrenza pari a 2 m/s (grandezza presa da letteratura). Attraverso i calcoli mostrati risulta quindi necessario posticipare l'arrivo del picco dell'onda del Chisola di 0.4 h ovvero circa 30 min.

Le combinazioni su grafico per lo scenario 2 sono le seguenti:

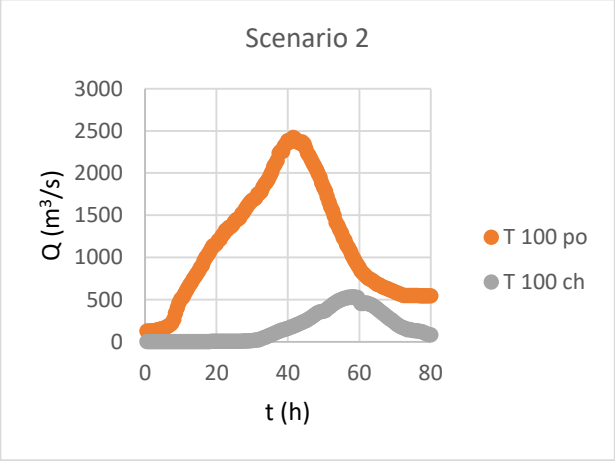


Figura 6.37: Scenario 2 – T 100-100

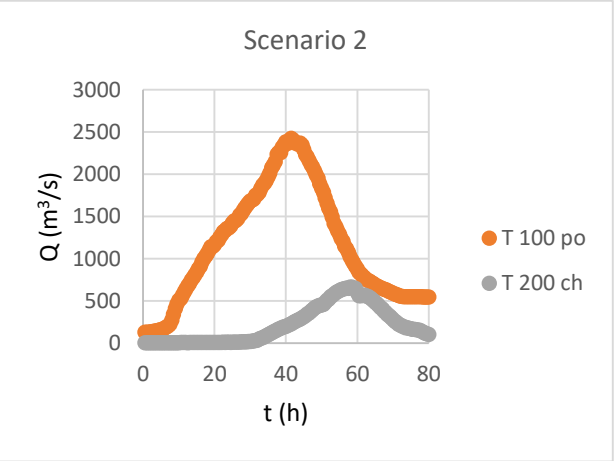


Figura 6.38: Scenario 2 – T 100-200

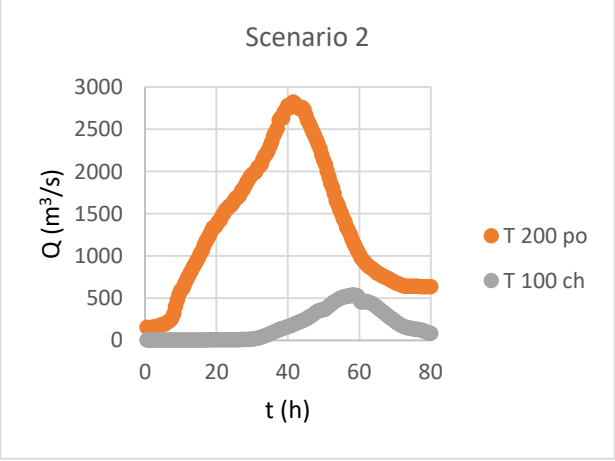


Figura 6.39: Scenario 2 – T 200-100

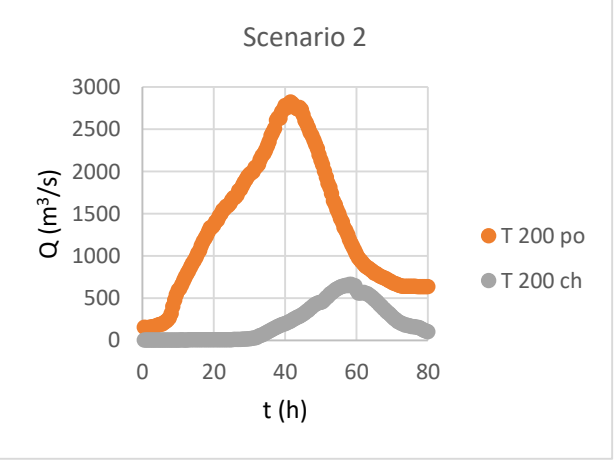


Figura 6.40: Scenario 2 – T 200-200

Le combinazioni su grafico per lo scenario 3 sono le seguenti:

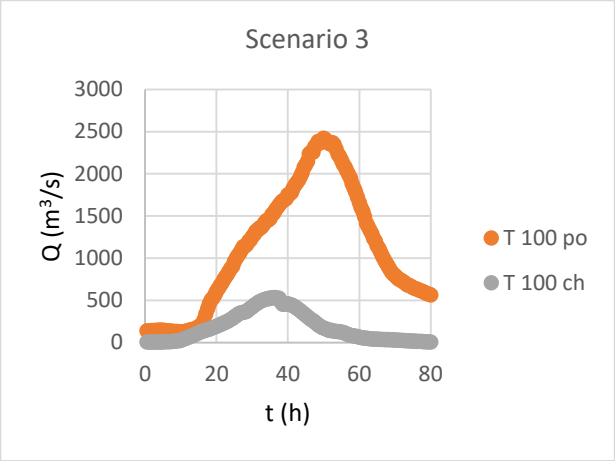


Figura 6.41: Scenario 3 – T 100-100

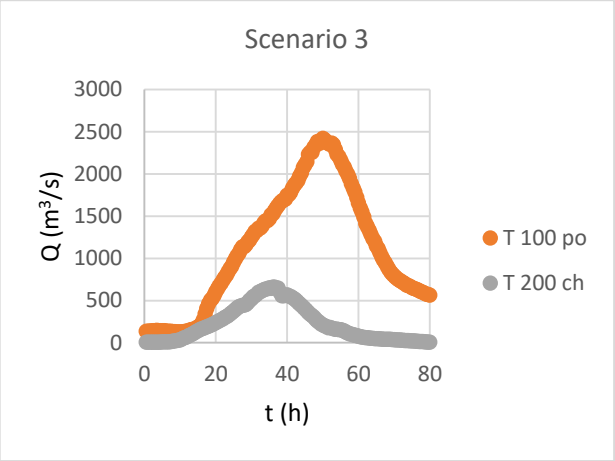


Figura 6.42: Scenario 3 – T 100-200

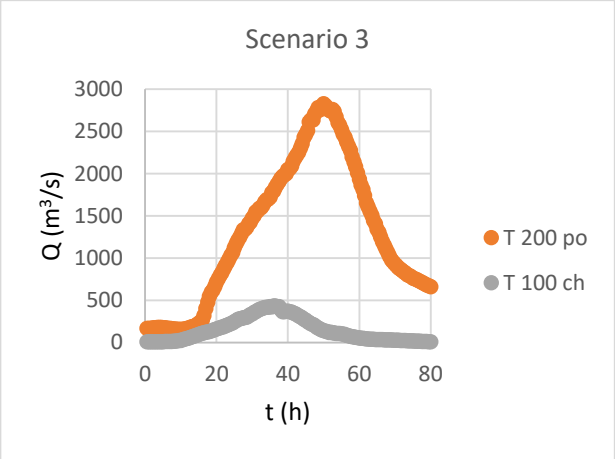


Figura 6.43: Scenario 3 – T 200-100

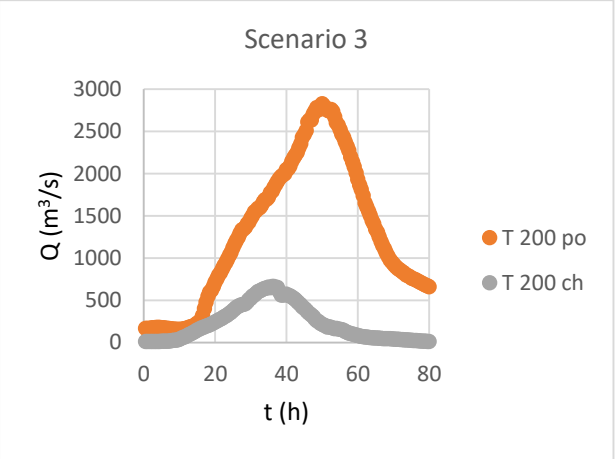


Figura 6.44: Scenario 3 – T 200-200

Definiti gli scenari e tutte le combinazioni che si simuleranno in HEC-RAS, si devono ora disporre i seguenti idrogrammi in due file Excel, uno contenente quelli del Po+Banna e l'altro quelli del Chisola. Particolare importanza assumono le prime due righe in cui si inseriscono i numeri corrispondenti agli scenari e quelli riferiti ai tempi di ritorno (Figura 6.45); tant'è che con il codice in Python si accederà a quelle celle

per copiarne i valori e, con uno specifico formato, salvarli nel file di testo contenente gli output prima dell'inserimento dei dati della simulazione.

1		2		3	
100	200	100	200	100	200
144.6	171.78	127.91	154.76	138.58	164.5
145.83	173.26	128.74	152.04	137.72	163.53
147.91	175.71	129.57	152.97	140.36	166.65
147.28	174.95	130.39	153.9	142.77	169.56
147.5	175.15	133.97	158.07	144.6	171.78
146.5	173.87	137.96	162.74	145.83	173.26
145.12	172.13	142.46	168.03	147.91	175.71
141.62	167.9	147.14	173.61	147.28	174.95
139.82	165.66	156.11	184.13	147.5	175.15
138.12	163.53	159.9	188.63	146.5	173.87
136.5	161.5	163.74	193.18	145.12	172.13
133.4	157.75	177.06	208.79	141.62	167.9
132.96	157.15	190.28	224.26	139.82	165.66
130.99	154.76	203.57	239.88	138.12	163.53
128.74	152.04	223.63	263.39	136.5	161.5
129.57	152.97	270.59	318.15	133.4	157.75
130.39	153.9	342.19	401.6	132.96	157.15
133.97	158.07	407.11	477.27	130.99	154.76
137.96	162.74	463.04	542.4	128.74	152.04
142.46	168.03	507.23	593.92	129.57	152.97
147.14	173.61	525.18	614.91	130.39	153.9

Linea contenente il numero degli scenari

Linea contenente il valore dei "T"

Linee contenenti gli idrogrammi

Figura 6.45: Estratto degli idrogrammi riferiti al Po

6.2.5 – Inserimento condizioni al contorno e definizione del plan

Altra operazione preliminare consiste nell'impostazione delle condizioni al contorno. Come già accennato nel paragrafo inerente la costruzione della geometria, una volta inserite le "*boundary condition lines*", ovvero le sezioni di entrata e uscita del flusso idrico dal dominio, si devono poi impostare delle opportune condizioni al contorno accedendo al "Unsteady Flow Editor" (Figura 6.46). La prima volta quest'operazione dev'essere svolta manualmente andando ad inserire per le sezioni di ingresso e per la sezione d'uscita opportune condizioni tra quelle disponibili, ovvero:

- Stage Hydrograph: utilizzata generalmente come condizione di valle nei casi in cui il tratto studiato termini in una zona lacustre o marina e quindi si conosce con ragionevole precisione l'altezza idrica finale.
- Flow Hydrograph: utilizzata nella maggior parte dei casi come condizione di monte e prevede l'inserimento di un idrogramma. Si può usare anche come condizione di valle quando si hanno a disposizione dati di portata nel tempo registrati nel tempo e si vuol ricostruire un determinato evento.
- Rating Curve o Scala di Deflusso: si tratta di una curva monotona crescente che lega altezza idrica e portata. Condizione particolarmente indicata per canali artificiali e strutture idrauliche dove questa relazione può essere determinata con precisione. Nei casi di alvei naturali si deve porre molta attenzione all'utilizzo in quanto la geometria muta nel tempo e quindi tale relazione dev'essere periodicamente rivalutata.
- Normal Depth: tipica condizione di valle nella quale andare inserire la pendenza dell'alveo. A partire da tale grandezza mediante l'equazione di Chezy si determina l'altezza idrica; si può intuire come si tratti di una semplificazione in quanto la validità dell'equazione è per correnti uniformi.

Nel caso trattato le condizioni usate sono "*Flow Hydrograph*" per le sezioni di monte e "*Normal Depth*" per quella di valle, come mostrato in Figura 6.46.

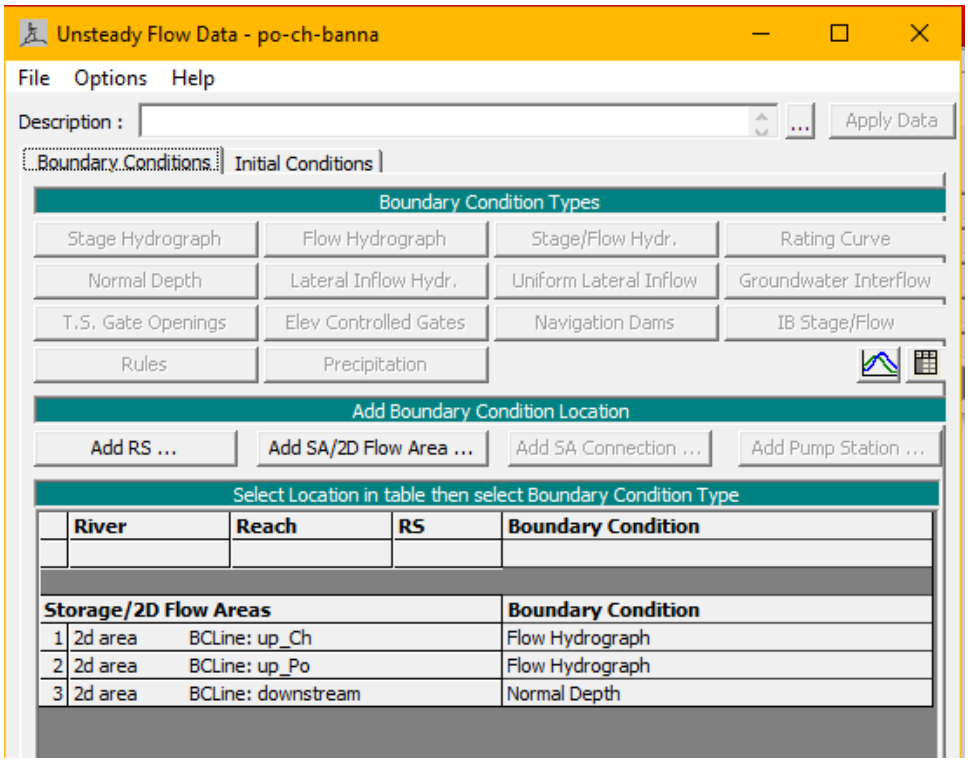


Figura 6.46: Inserimento condizioni al contorno

Per quanto riguarda i "*Flow Hydrographs*" aprendo la relativa finestra (Figura 6.47) si deve:

- Impostare un determinato valore di "*Data time interval*", in questo caso pari a 30 min e stabilito in base all'intervallo temporale tra un dato e l'altro realmente misurato nelle varie stazioni di misura;
- Impostare data e ora d'inizio della simulazione. In caso di evento ipotizzato possono essere inseriti casualmente;
- Inserire un idrogramma copiandolo tra quelli presenti nel file Excel, bisogna assicurarsi che le lunghezze coincidano impostando l'esatto numero attraverso "No. Ordinates";
- Inserire il valore di "*EG Slope*", in questo caso assunto da simulazioni già eseguite da terzi.

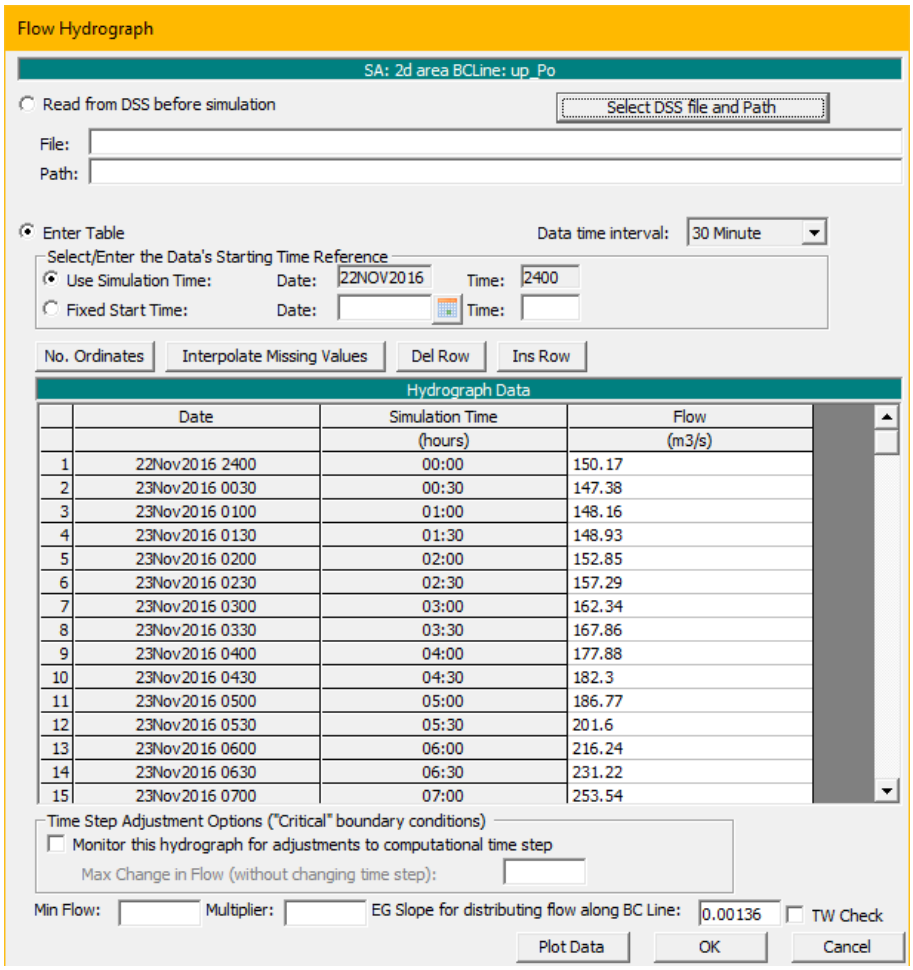


Figura 6.47: Finestra in cui impostare i valori inerenti alla boundary condition “Flow Hydrograph”

Per la condizione di valle si deve invece inserire il valore di “Friction Slope” anche in questo caso già a disposizione (Figura 6.48).

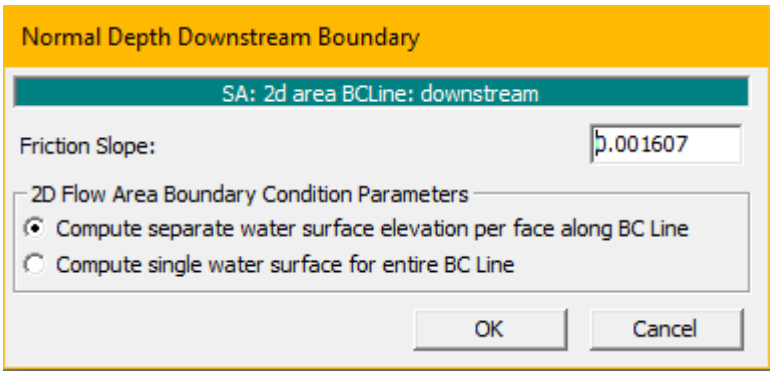


Figura 6.48: Finestra della boundary condition di valle

Dopo aver inserito tutti i dati necessari si deve salvare, in questo modo verrà generato un file con estensione ".u" all'interno della cartella contenente il progetto generale di HEC-RAS (Figura 6.49).

SIMULAZIONE2.u01 - Blocco note di Windows

File Modifica Formato Visualizza ?

```
Flow Title=po-ch-banna
Program Version=5.07
Use Restart= 0
Boundary Location=                ,                ,        ,        ,               ,2d area         ,                ,up_Po
Interval=30MIN
Flow Hydrograph= 147
  150.17  147.38  148.16  148.93  152.85  157.29  162.34  167.86  177.88   182.3
  186.77   201.6  216.24  231.22  253.54  304.87  383.04  453.92  514.77  563.06
  582.89  625.84  671.33  710.32  749.78   781.0  824.33  859.06  893.41  932.08
  975.95 1008.13 1064.12 1111.38 1146.81 1181.64 1226.72 1265.58 1271.36  1293.5
 1332.01 1350.07 1390.08 1419.76 1458.27 1470.98 1496.87  1503.8 1527.58 1564.58
 1592.08 1596.85 1620.95 1670.51 1688.79 1730.97  1768.1 1801.54 1833.93 1860.49
 1874.07 1888.76 1936.58 1944.99 1971.85 2023.45 2063.28 2089.94  2121.4 2171.64
  2215.7 2284.75 2316.44 2357.94 2451.03  2475.9 2469.57 2536.11 2566.17  2604.9
 2580.73  2608.4 2644.19 2624.96 2600.02 2575.13 2564.08 2581.65 2556.91 2498.81
 2434.36 2404.09 2355.08 2305.14 2269.38 2220.44 2173.42 2124.88 2055.36 1997.41
  1946.6  1877.1 1808.07 1743.63 1694.75 1629.45 1545.98 1503.34 1453.02  1406.7
 1350.89 1314.23  1254.9 1219.33 1170.33 1117.21 1073.09  1035.0  992.54  955.13
  915.03  891.65  867.75  845.56  823.89  810.34  796.79  779.86  766.83  746.01
  740.78  728.01  712.44  704.21  695.57  687.79  675.91  663.62  655.44  643.18
  631.55  627.26  616.08  608.36  600.64  599.86  591.86     nan     0.0     0.0
Stage Hydrograph TW Check=0
Flow Hydrograph Slope= 0.00136
DSS Path=
Use DSS=False
Use Fixed Start Time=False
```

Figura 6.49: Estratto del file ".u" generato al momento del salvataggio dell'unsteady flow editor

Questo file, già nominato nei precedenti capitoli, è di fondamentale importanza in quanto viene richiamato all'interno del codice per accedervi ed effettuare la sostituzione automatica degli idrogrammi.

Ultima finestra da impostare è quella relativa al Plan (Figura 6.50).

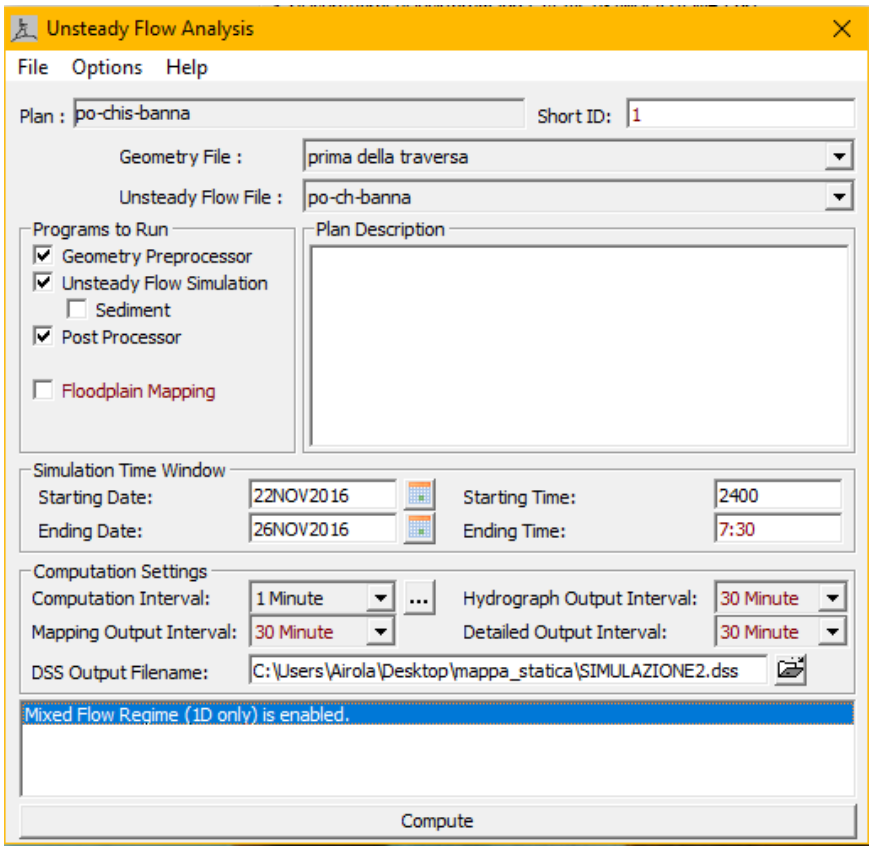


Figura 6.50: Finestra da cui avviare manualmente le simulazioni

Le operazioni da fare su questa finestra sono:

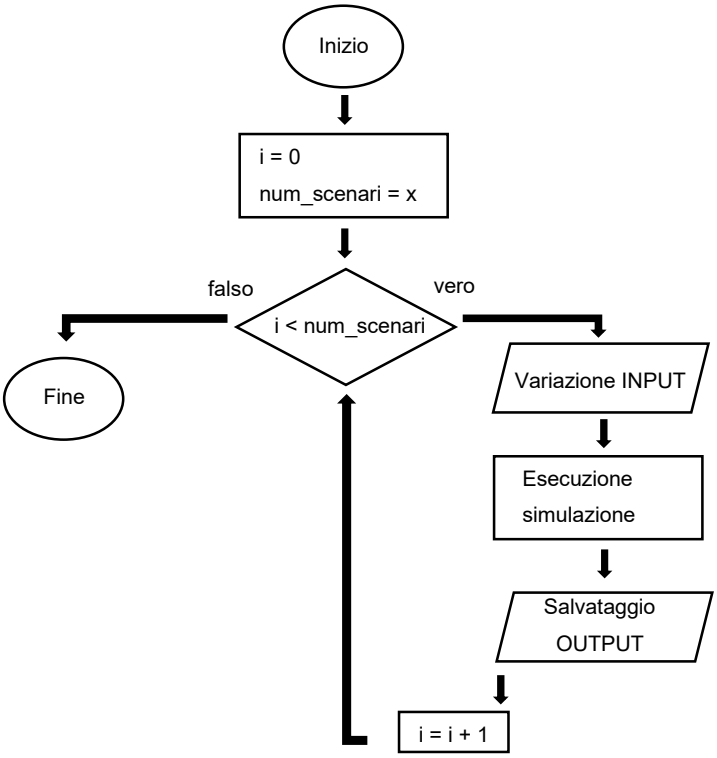
- Nel riquadro "*Programs to Run*" settare *Geometry Preprocessor*, *Unsteady Flow Simulation* e *Post Processor*. In questo modo all'avvio della simulazione viene effettuata un'analisi sulla geometria verificando che non ci siano errori, successivamente parte la vera e propria fase di computazione ed infine vengono generati gli output (lavorando con geometria bidimensionale gli output sono da ricercarsi nel file .p0X.hdf generato);
- Nel riquadro "*Simulation Time Window*" inserire data e ora di inizio e fine della simulazione, tale tempistica dipende dalla lunghezza dell'idrogramma e dallo step temporale tra un dato e l'altro dello stesso,
- In "*Computation Settings*" impostare delle tempistiche per le quattro grandezze indicate; in cui il "*Computation Interval (1 min)*" rappresenta l'intervallo temporale con cui viene effettuata la simulazione mentre i restanti

tre fanno riferimento alle tempistiche con cui vengono presentati i dati di output (in questo caso si è impostato 30 min).

Infine è necessario salvare; anche in questo caso viene creato un file, posizionato nella cartella in cui è contenuto il progetto generale, con estensione “.p” e contenente tutte le informazioni inserite. Terminata la simulazione viene generato file con estensione “.p0x.hdf” contenente tutta una serie di output che verrà aperto per poter estrarre quelli di interesse.

6.3 – Esecuzione simulazioni

Una volta che tutte le operazioni preliminari sono state eseguite e quindi sono stati inseriti tutti gli input al modello, si può proseguire all'esecuzione delle simulazioni



Schema 6.1: Schema alla base dell'esecuzione delle simulazioni

Come riportato nello schema precedente, che è lo stesso riportato nel Capitolo 5 inerente all'automatizzazione delle simulazioni, il processo di simulazione prevede una serie operazioni eseguite in serie e ripetute un numero "x" di volte, pari agli scenari che si sono ipotizzati e che si vogliono analizzare. Come già rimarcato nei primi capitoli lo scopo primario è quello di rendere automatico questo processo e ciò è possibile utilizzando un linguaggio di programmazione che riesca a dialogare con il software. Sfruttando Python si è quindi redatto un codice il cui schema di base è ancora una volta quello riportato sopra (Schema 6.1).

Di seguito quindi si analizzeranno le varie parti facendo stretto riferimento al codice e a tutto ciò che ad esso è collegato.

6.3.1 – Inizializzazione delle variabili

La prima fase consiste nell'inizializzazione di alcune variabili che sono indispensabili per il corretto funzionamento del codice e su cui si basano le iterazioni; si riporta di seguito l'estratto del codice in cui si effettuano le inizializzazioni:

```
hec = win32com.client.Dispatch("RAS507.HECRASController")
RASProject = os.path.join(os.getcwd(),r'C:\Users\Airola\Desktop\caso
      studio\SIMULAZIONE2.prj')
cartella_input = load_workbook(filename="idrogrammi_Po.xlsx")
cartella2_input = load_workbook(filename="idrogrammi_Chisola.xlsx")
foglio_input = cartella_input.active
foglio2_input = cartella2_input.active

num_scenari = 3
num_T_Po = 2
num_T_Ch = 2
```

Quindi si ha:

- "*hec*", variabile da utilizzare per richiamare le funzioni e subroutines del HEC-RAS Controller;
- "*RASProject*", variabile in cui si riporta il percorso di salvataggio del progetto che verrà successivamente aperto;
- "*cartella_input*", variabile a cui viene associato il file Excel contenente gli idrogrammi del Po;
- "*cartella2_input*", variabile a cui viene associato il file Excel contenente gli idrogrammi del Chisola;
- "*foglio_input*", variabile in cui specificare in quale foglio del file Excel, già associato alla variabile cartella_input, sono presenti i dati;

- "*foglio2_input*", variabile in cui specificare in quale foglio del file Excel, già associato alla variabile cartella2_input, sono presenti i dati;
- "*num_scenari*", variabile in cui inserire il numero di scenari che si intende analizzare (es. num_scenari = 3, considerando scenario con onde di piena temporalmente coincidenti e onde di piena sfalsate con Po prima del Chisola e viceversa);
- "*num_T_Po*", variabile in cui inserire il numero di tempi di ritorno del Po per cui si vuol fare l'analisi (es. num_T_Po = 2, ipotizzando di voler analizzare idrogrammi aventi T_{100} e T_{200});
- "*num_T_Ch*", variabile in cui inserire il numero di tempi di ritorno del Chisola per cui si vuol fare l'analisi (es. num_T_Ch = 2, ipotizzando di voler analizzare idrogrammi aventi T_{100} e T_{200});

Infine è ancora necessario ricavare il numero di valori costituenti l'idrogramma che può essere effettuato mediante il seguente pezzo di codice:

```
w=0
j=0
num_righe=0
while w == 0:
    if foglio_input.cell(j+2,1).value != None:
        num_righe+=1
        j+=1
    else:
        w=1
```

La variabile in questione è "*num_righe*"; conoscendo la disposizione per colonne degli idrogrammi all'interno dei file Excel (Figura 6.45) si è predisposto un ciclo "while" all'interno del quale scandire la prima colonna, cella per cella, e incrementare il valore della variabile fino a che si arriva ad una cella vuota, ciò vuol dire che si è alla fine dell'idrogramma. La lunghezza dei vari idrogrammi deve essere uguale.

6.3.2 – Inserimento condizioni iterative

Le quattro variabili *num_scenari*, *num_T_Po*, *num_T_Ch* e *num_righe* sono alla base del processo iterativo, infatti sono state utilizzate come limite massimo all'interno dei cicli "for" che scandiscono l'andamento automatico delle simulazioni. Si riporta di seguito la parte di codice inerente l'inserimento delle condizioni iterative:

```
## INSERIMENTO CICLI FOR
for s in range(0,num_scenari):
    for i in range(0,num_T_Po):
        z=0
        for v in range (0, num_T_Ch):
            #inizializzazione vettori
            vet = np.empty([num_righe], dtype=float)
            vet2 = np.empty([num_righe], dtype=float)
            #estrazione idrogrammi e popolazione vettori
            for j in range(1,num_righe+1):
                vet[z] = foglio_input.cell(j+2, i+(s*num_T_Po)).value
                vet2[z] = foglio2_input.cell(j+2, v+(s*num_T_Ch)).value
                z+=1
            #stampa a video degli idrogrammi
            vet.resize(num_lines, 10)
            vet2.resize(num_lines, 10)
            print(vet)
            print(vet2)
```

Come si può notare vi sono quattro cicli "for" annidati, di cui i primi tre sono necessari per poter analizzare tutti gli scenari e le varie casistiche di interesse, mentre il quarto serve per scandire, riga per riga, le varie colonne dei file Excel contenenti gli idrogrammi. La scansione eseguita da quest'ultimo ciclo "for" permette di estrarre un idrogramma del Po e un idrogramma del Chisola e salvarlo rispettivamente in due vettori "vet" e "vet2" precedentemente inizializzati. Questi due vettori saranno successivamente utilizzati per effettuare la sostituzione dei valori degli idrogrammi nel file ".u" di HEC-RAS. Le ultime righe di codice riportate nel riquadro fanno riferimento alla trasformazione dei vettori in matrici, attraverso la funzione "resize", aventi dieci valori per riga e successivamente stampate a video mediante la funzione "print". Questo ultimo passaggio è stato fatto semplicemente per avere un

riscontro visivo immediato della correttezza di estrazione degli idrogrammi che verranno poi caricati in HEC-RAS (Figura 6.51).

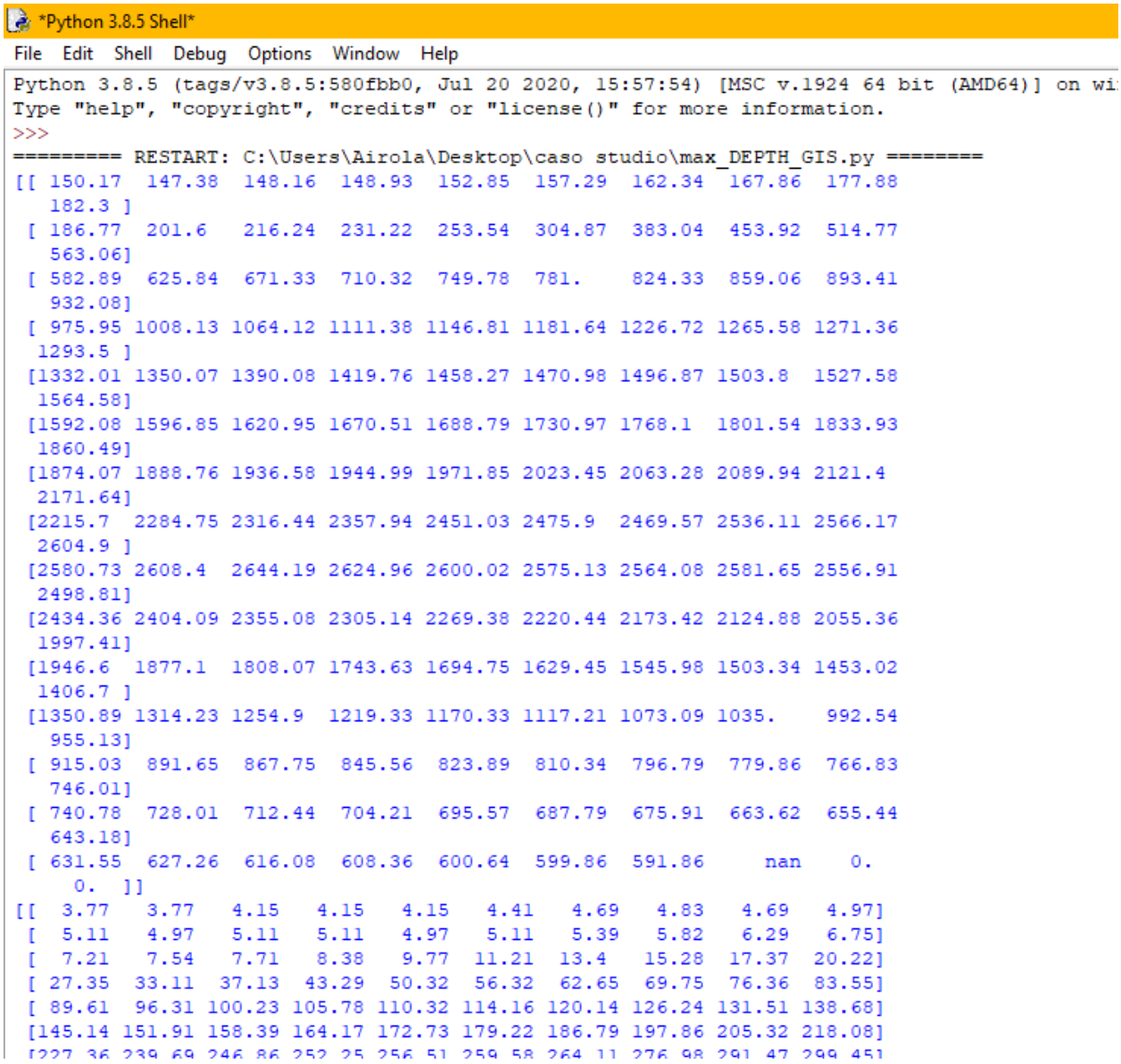


Figura 6.51: Idrogrammi restituiti a video dopo l'estrazione dal file Excel

6.3.3 – Sostituzione idrogrammi

Le fasi successive del codice si sviluppano all'interno del terzo ciclo "for". Una di queste fasi è la modifica dell'idrogramma contenuto nel file ".u" (Figura 6.52) che dev'essere eseguita ad ogni iterazione prima che il software HEC-RAS inizi la fase di computazione.

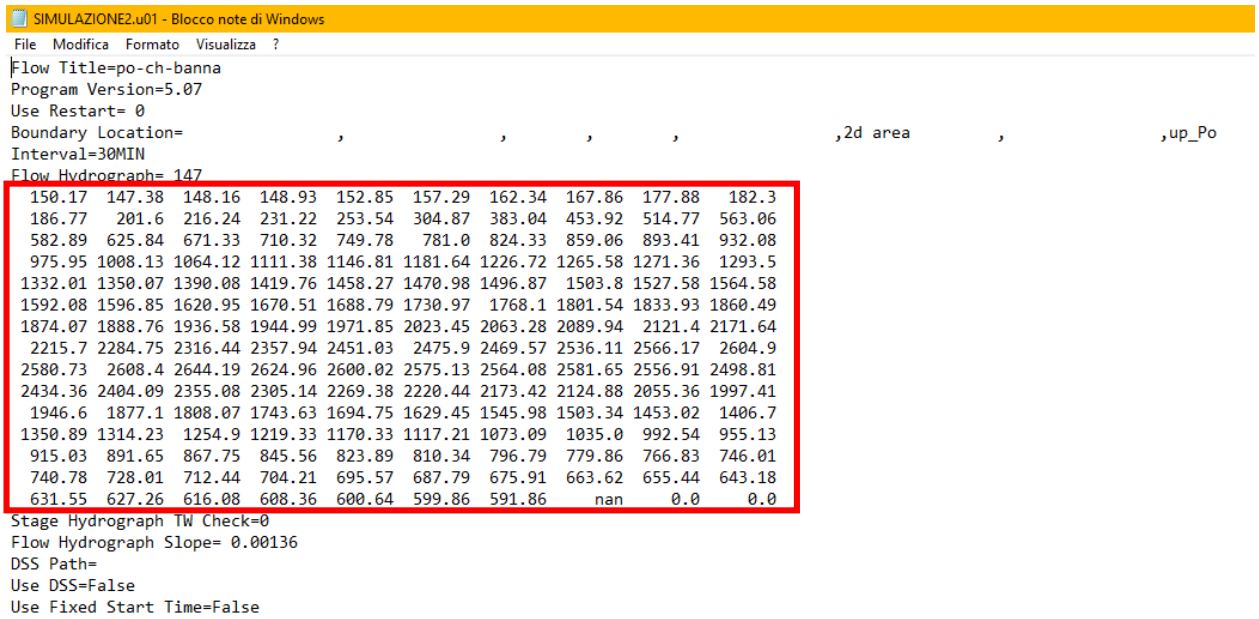


Figura 6.52: esempio di file ".u" contenente l'idrogramma (incorniciato in rosso)

Per poter effettuare questa modifica è stato necessario creare una copia del file ".u" (precedentemente generato – vedi par. 6.2.5), chiamato "copia_input" (salvato nella stessa cartella) che servirà come base per copiare quelle righe del file che non sono soggette a variazione (si deve porre attenzione a ripetere la copiatura di tale file tutte le volte in cui si apportano delle modifiche nell'unsteady flow editor).
Si riporta di seguito la parte di codice sviluppata per effettuare questa operazione, già menzionata nel paragrafo 2 del capitolo 5 e a cui si rimanda per una più completa spiegazione.

```
##APERTURA FILE
file_in= open (r"C:\Users\Airola\Desktop\caso
studio\copia_input.txt","r")
file_out= open (r"C:\Users\Airola\Desktop\caso
studio\SIMULAZIONE2.u01","w")

num_linee = math.ceil(num_righe/10)
n=0
b=0
##CREAZIONE NUOVO FILE ".u"
for riga in file_in:
      #sostituzione idrogramma
      if count+1 <= n <= count+num_linee:
            for k in range (0,10):
                  a = vet[b,k]
                  cifre = len(str(a))
                  if cifre == 1:
                        file_out.write("       "+str(a))
                  elif cifre == 2:
                        file_out.write("      "+str(a))
                  elif cifre == 3:
                        file_out.write("     "+str(a))
                  elif cifre == 4:
                        file_out.write("    "+str(a))
                  elif cifre == 5:
                        file_out.write("   "+str(a))
                  elif cifre == 6:
                        file_out.write("  "+str(a))
                  elif cifre == 7:
                        file_out.write(" "+str(a))
            file_out.write("\n")
            b+=1
            n+=1
      #inserimento line che non variano
      else:
            file_out.write(riga)
            n+=1
#CHIUSURA FILE
file_in.close()
file_out.close()
```

Sostanzialmente ciò che viene fatto è scandire riga per riga il file "copia_input" da cui copiare, nel nuovo file ".u" che verrà generato, le righe che non necessitano di modifica, mentre quando si è nell' intervallo di righe in cui vi è l'idrogramma devono

essere inseriti i nuovi valori estraendoli dal vettore “vet” precedentemente creato e popolato. La corretta posizione della riga in cui inserire i nuovi valori è stata determinata attraverso una funzione denominata “ricerca_elemento” che va a ricercare una determinata espressione e salva in una variabile il numero della riga in cui è stata ritrovata. Il codice di tale funzione già descritta nel capitolo 5 viene di seguito riportato:

```
##CREAZIONE FUNZIONE
def ricerca_elemento(file,elem):
    riga=" "
    count=0
    #ciclo che scandisce le righe del file
    while(riga):
        riga=file.readline()
            #condizione di ricerca
            if elem in riga:
            #assegnazione num riga in cui si è trovata corrispondenza
            return count
        #incremento variabile se non soddisfo la condizione di ricerca
        count+=1
```

Osservando la Figura 6.52 si nota come l’espressione da ricercare sia “Flow Hydrograph” al di sotto del quale vi sono i valori costituenti l’idrogramma.
La funzione in realtà non è parte integrante del programma, dev’essere posizionata nelle prime righe di codice e poi richiamata laddove necessario mediante la seguente sintassi:

```
#APERTURA FILE IN CUI EFFETTUARE LA RICERCA
file_in= open (r"C:\Users\Airola\Desktop\caso
studio\copia_input.txt","r")
#elemento da ricercare
elem = "Flow Hydrograph="
#variabile a cui assegnare la posizione
count = ricerca_elemento(file_in, elem)
#chiusura file
file_in.close()
```

Nella variabile count sarà quindi salvato il numero corrispondente alla riga in cui vi è l'espressione ricercata e grazie al quale si intuisce la corretta posizione in cui iniziare ad effettuare l'inserimento dei nuovi valori.

Considerando che nel modello costruito si hanno due input ovvero due corsi d'acqua in ingresso, anche gli idrogrammi da variare saranno due. Infatti, come si può notare in Figura 6.53, il file ".u" generato inizialmente contiene al suo interno due idrogrammi; il primo riferito al Po + Banna mentre il secondo al Chisola. Pertanto, dal punto di vista del codice, la seconda sostituzione può essere effettuata ricercando la posizione del secondo "Flow Hydrograph" sempre mediante la funzione sopra riportata e quindi sostituendo i nuovi valori dell'idrogramma con il pezzo di codice riportato ad inizio paragrafo. Per maggior chiarezza sul codice nel suo complesso consultare l'*Allegato I*.

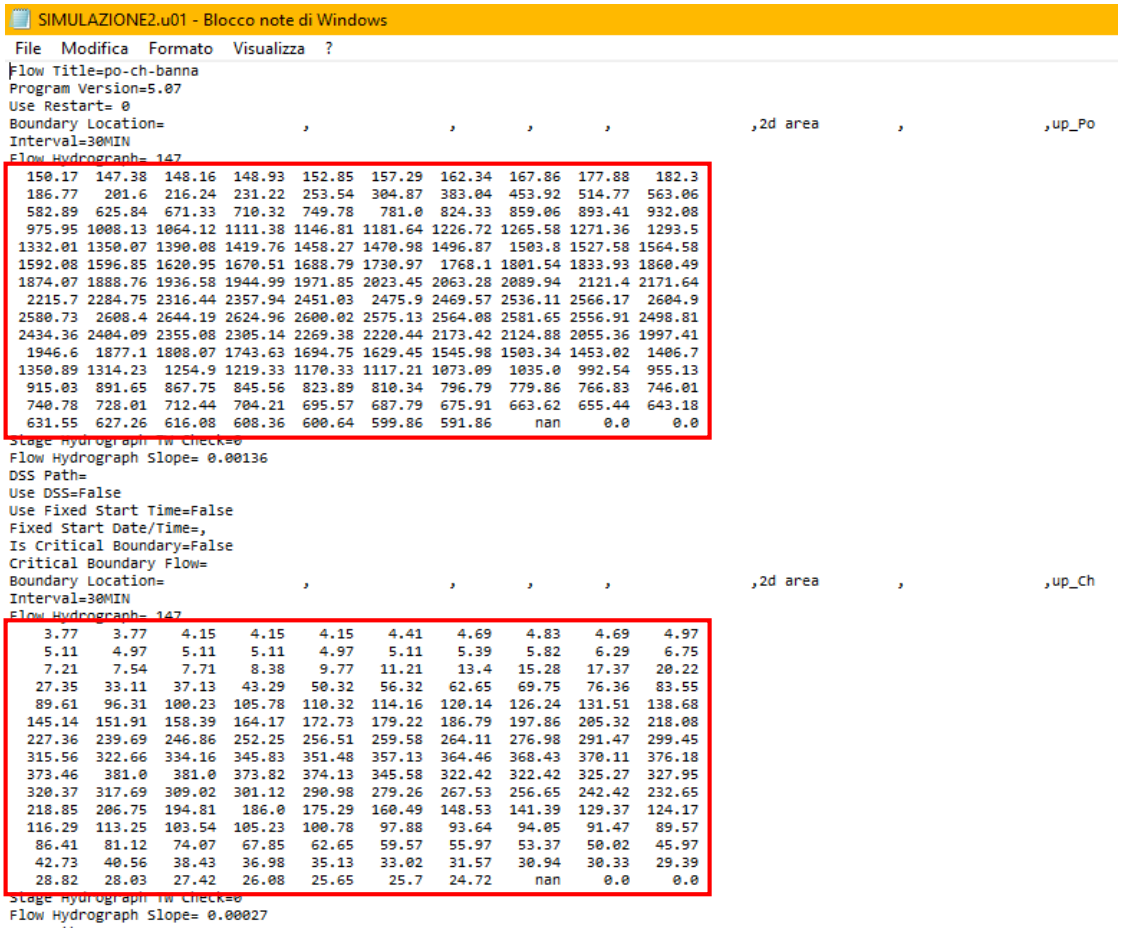


Figura 6.53: estratto del file ".u" contenente i due idrogrammi

6.3.4 – Avvio computazione da parte di HEC-RAS

Una volta generato il nuovo file ".u" è possibile avviare la computazione vera e propria. Ciò avviene automaticamente aggiungendo ai pezzi di codice sopra riportati il seguente:

```
##APERTURA PROGETTO HEC-RAS
hec.Project_Open(RASProject)
#inizializzazione variabili
NMsg,TabMsg,block = None,None,True
#inserimento funzione per aprire la finestra in cui si vede l'avanzamento
della simulazione
hec.Compute_ShowComputationWindow()
#avvio fase di computazione
v1,NMsg,TabMsg,v2 = hec.Compute_CurrentPlan(NMsg,TabMsg,block)
```

Aperto il progetto con "Project_Open" l'inizio della fase di computazione avviene con l'inserimento della funzione "Compute_CurrentPlan". Inizia così la fase di calcolo da parte di HEC-RAS che può durare anche parecchio tempo, ciò può dipendere principalmente dalla dimensione del dominio, dal numero delle celle e dalla risoluzione temporale impostata inizialmente che insieme alla lunghezza temporale degli idrogrammi influenzano notevolmente il tempo necessario.

Nel caso specifico si ha:

- Numero di celle pari a 50779;
- Lunghezza temporale degli idrogrammi di circa 3 giorni e risoluzione temporale di 1 minuto.

Il tempo complessivo impiegato per concludere la computazione è di circa 50 min; supponendo quindi di voler effettuare in serie 12 simulazioni (3 scenari x 2 T_{Po} x 2 T_{Ch}) il tempo totale è di circa 7 ore.

6.4 – Estrazione e salvataggio output

Terminata una simulazione si devono estrarre i dati necessari per la generazione degli output previsti, che per il seguente caso studio sono:

- Generazione di una o più mappe statiche, relative ai diversi scenari in cui riportare il massimo tirante idrico raggiunto nelle celle e quindi individuare le aree allagabili sul dominio;
- Estrazione della sequenza temporale di una certa grandezza, nel caso in questione si intende estrarre la “water surface elevation” per una cella o un gruppo di celle, questo è importante per stimare il tempo che una certa area può rimanere allagata.

Eseguendo simulazioni in serie è importante che l’estrazione avvenga prima dell’inizio di una nuova fase di computazione che porterebbe all’aggiornamento dei dati e conseguentemente alla perdita di quelli precedenti. Quindi dal punto di vista del codice questa fase si posiziona subito dopo le linee mostrate nel paragrafo precedente in merito all’avvio della simulazione e sempre indentate all’interno del terzo ciclo “for” posto inizialmente.

Come già anticipato alla fine del Capitolo 5, HEC-RAS sfrutta una modalità di salvataggio gerarchica attraverso file con estensione “.hdf” (Hierarchical Data Format File). Si era già visto come all’interno del file generato al termine della simulazione e avente estensione ”.p0X.hdf” si possano trovare i valori assunti da alcune grandezze quali: il tirante idrico (depth), la tensione di taglio esercitata sulla superficie della cella, la velocità della corrente nella cella, gli step temporali (in giorni) e il valore di quota s.l.m (water surface) raggiunta dalla corrente.

Con riferimento specialmente alla creazione della mappa statica le grandezze che si son volute estrarre per le varie celle sono: i valori di quota massima del pelo libero, le coordinate dei centri e le quote minime del terreno.

L’estrazione e il salvataggio in un apposito documento di testo è stato fatto con le righe di codice riportate di seguito in cui si è sfruttato il modulo h5py, appositamente installato in python e che permette di lavorare con i file HDF.

```
#APERTURA FILE .p0X.hdf
with
h5py.File(r"C:\Users\Win10\Desktop\mappa_statica\SIMULAZIONE2.p05.hdf",
      "r") as hdf:
      #Ricerca della sottocartella contenente il primo dato da estrarre
      G1 = hdf.get('Results/Unsteady/Output/Output Blocks/Base
      Output/Unsteady Time Series/2D Flow Areas/2d area')
      #Estrazione della grandezza voluta e salvataggio in matrice
      wse = np.array(G1.get('Water Surface'))
      #Ricerca della sottocartella contenente il secondo dato da estrarre
      G2 = hdf.get('Geometry/2D Flow Areas/2d area')
      #Estrazione della grandezza voluta e salvataggio in matrice
      coord = np.array(G2.get('Cells Center Coordinate'))
      #Ricerca della sottocartella contenente il terzo dato da estrarre
      G3 = hdf.get('Geometry/2D Flow Areas/2d area')
      #Estrazione della grandezza voluta e salvataggio in vettore
      min_elev = np.array(G3.get('Cells Minimum Elevation'))
#DETERMINAZIONE DELLA DIMENSIONE DELLE MATIRCI E DEL VETTORE
nrows1, ncols1 = np.shape(wse)
nrows2, ncols2 = np.shape(coord)
nrows3 = np.shape(min_elev)

#RICAVO IL MAX DI WSE PER OGNI CELLA
num_id = 0
for w in range (0,ncols1):
      flag = 0
      for h in range (0,nrows1):
            val_wse = wse[h,w]
            if val_wse > flag:
                  max_wse = val_wse
                  time = h
                  flag = max_wse
             elif flag == 0:
                  max_wse=0

#CALCOLO LA MAX. WATER DEPTH SULLA CELLA
      depth = max_wse - min_elev[w]

#SCRIVO SU FILE
      output.write(str(num_id))
      output.write(", ")
      output.write(str(coord[w,0]))
      output.write(", ")
      output.write(str(coord[w,1]))
      output.write(", ")
```

```
        output.write(str(max_wse))
        output.write(", ")
        output.write(str(min_elev[w]))
        output.write(", ")
        output.write(str(depth))
        output.write("\n")
        num_id += 1
```

In questo modo si costruisce un file txt avente su ogni riga le grandezze riferite ad una cella della mesh come riportato in Figura 6.54:

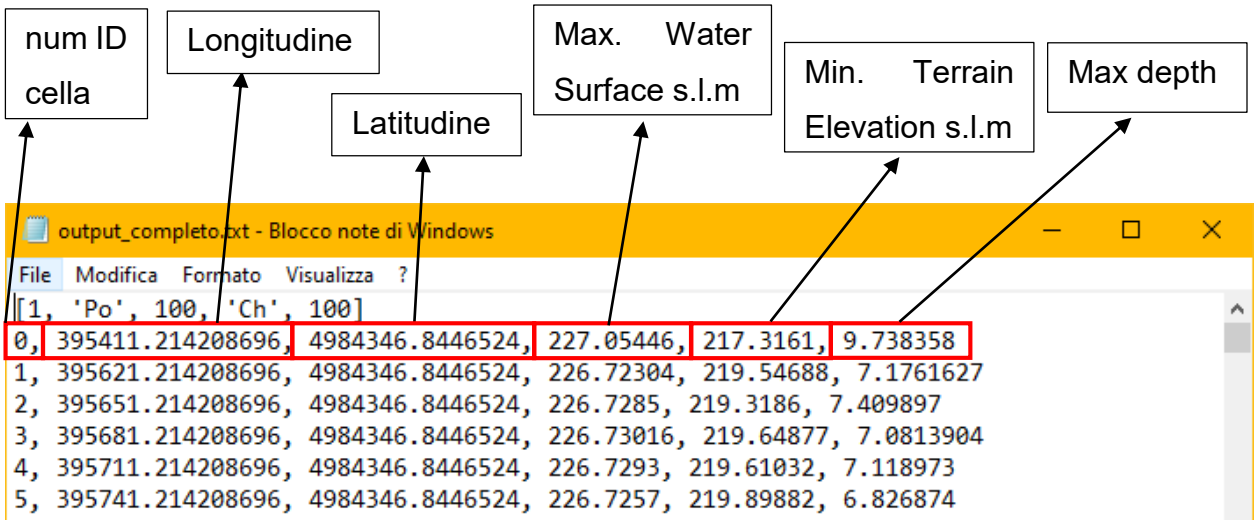


Figura 6.54: Estratto del file txt contenente gli output estratti dal file HDF

6.5 – Creazione mappa statica

Le grandezze estratte e salvate nel file *"output_completo.txt*" sono propedeutiche alla costruzione di una mappa dove, in base al valore di max water depth di ogni cella, andare ad individuare le aree potenzialmente allagabili all'interno del dominio di studio.

Prima di riportare i vari passaggi necessari alla creazione della mappa si deve ancora effettuare un'operazione di post-processing della mole di dati salvati nel file di testo. Questo perché in esso sono contenuti i risultati di tutte le simulazioni effettuate. Come si può notare dalla Figura 6.54 del precedente paragrafo, per differenziare i dati si è deciso di introdurre una riga identificativa all'inizio di ogni blocco di dati, dove:

- il primo numero rappresenta il tipo di scenario;
- il secondo numero rappresenta il tempo di ritorno associato all'idrogramma del Po
- il terzo numero rappresenta il tempo di ritorno associato all'idrogramma del Chisola.

In questo modo si riesce ad individuare univocamente un determinato evento simulato, che dev'essere ulteriormente estratto e salvato in un altro file di testo. Questo perché per le successive operazioni di creazione della mappa è necessario avercelo singolarmente. Questa operazione è stata effettuata creando un script in Python dove l'utente sceglie l'evento di interesse in base ai tre valori elencati in precedenza e il programma estrae i relativi dati salvandoli in un altro file denominato "*output_gis.txt*" (il codice completo è riportato in *Allegato II*).

La parte di codice inerente la scelta dell'evento da estrarre viene di seguito riportato:

```
# INIZIALIZZAZIONE VARIABILI
num_scenari = 3
vet_scenari = np.arange(1,num_scenari,1+1)
tempi_ritorno = [100, 200]
tempi_ritorno_Po = [100, 200]
tempi_ritorno_Ch = [100, 200]

# INSERIMENTO SCENARIO CON VERIFICA
```

```
print("Digitare uno dei seguenti valori in base allo scenario di
interesse:")
print("1 - Onde di piena del Po e Chisola coincidenti")
print("2 - Onda di piena del Po prima dell'onda di piena del Chisola")
print("3 - Onda di piena del Po dopo l'onda di piena del Chisola")

scenario = int(input("scenario n: "))
# verifica correttezza valore inserito
flag = 1
while flag != 0:
    for s in range (0, len(vet_scenari)):
        if scenario == vet_scenari[s]:
            flag = 0
        else: continue
    if flag == 1:
        print("Errore, ridigitare valore")
        scenario = int(input("scenario n: "))

# INSERIMENTO TEMPO DI RITORNO PO CON VERIFICA
print("Inserire T_Po (100, 200)")

T_Po = int(input("T_Po: "))
# verifica correttezza valore inserito
flag = 1
while flag != 0:
    for s in range (0, len(tempi_ritorno)):
        if T_Po == tempi_ritorno[s]:
            flag = 0
        else: continue
    if flag == 1:
        print("Errore, ridigitare valore")
        T_Po = int(input("T_Po: "))

# INSERIMENTO TEMPO DI RITORNO CHISOLA CON VERIFICA
print("Inserire T_Chisola (100, 200)")

T_Ch = int(input("T_Ch: "))

# verifica correttezza valore inserito
flag = 1
while flag != 0:
    for t in range (0, len(tempi_ritorno)):
        if T_Ch == tempi_ritorno[t]:
            flag = 0
        else: continue
```

```
    if flag == 1:
        print("Errore, ridigitare valore")
        T_Ch = int(input("T_Ch: "))
```

Le prime linee servono per inizializzare delle variabili che serviranno successivamente come parametri di verifica. Infatti dopo aver inserito il valore, mediante la funzione “input” di Python, è opportuno eseguire una verifica della correttezza del valore ed eventualmente reinserirlo se errato. I valori inseribili sono inoltre mostrati a video prima dell’inserimento dei valori utilizzando la funzione “print” come mostrato in Figura 6.55.

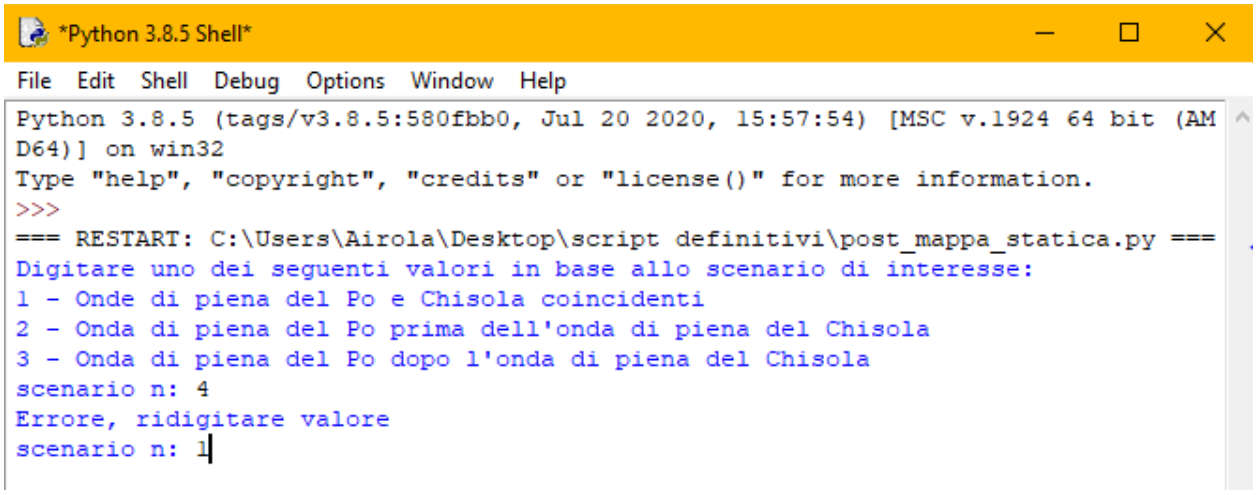


Figura 6.55: Esempio di inserimento del numero dello scenario da estrarre.

Passo successivo è l’estrazione dei dati vera e propria che avviene dopo aver trovato la giusta posizione iniziale nel file “output_generale.txt” contenente tutti i dati, mediante la funzione ricerca_elemento menzionata già più volte e descritta nel capitolo 5. Si riporta di seguito la seconda parte del codice inerente la scrittura sul nuovo file, “output_gis”, dei dati specifici.

```
#APERTURA FILE CONTENENTE TUTTI I DATI
output_generale = open(r"C:\Users\Airola\Desktop\mappa_statica
      \output_completo.txt","r")

#APERTURA FILE IN CUI SALVARE LO SCENARIO DI INTERESSE
```

```
output_gis = open(r"C:\Users\Airola\Desktop\mappa_statica
      \output_gis.txt","w")

#FUNZIONE PER LA RICERCA DELLA RIGA DA CUI INIZIARE L'ESTRAZIONE
elem = str([scenario, "Po", T_Po, "Ch", T_Ch])
count = ricerca_elemento(output_generale, elem)

#CONDIZIONE ITERATIVA PER L'ESTRAZIONE DEI DATI E SCRITTURA SUL NUOVO
FILE
n=0
output_generale.seek(0)
for riga in output_generale:
    if count< n <=count+num_col:
        output_gis.write(riga)
    n=n+1

#CHIUSURA DEI FILE
output_gis.close()
output_generale.close()
```

Generato il file si può procedere alla vera e propria creazione della mappa che richiede l'utilizzo di un software GIS, nel caso specifico si è utilizzato ArcMap 10.7. I passaggi da svolgere in GIS sono i seguenti:

1. Apertura del software e generazione di un nuovo progetto;

2. Impostazione dell'ambiente di lavoro tramite Geoprocessing →Environment Setting, andando ad impostare il sistema di riferimento cartografico, la cartella in cui andare a salvare gli output e la maschera che permette di basare le elaborazioni successive soltanto all'interno del dominio di studio (Figura 6.56).La maschera ("Mask") si può ottenere accedendo all'ambiente GIS di HEC-RAS ovvero Ras-Mapper e da Geometry andare su Perimeter e con tasto destro selezionare Export to salvandolo come shapefile. In questo modo si genera un file caricabile su ArcMap e, di conseguenza, impostabile come maschera.

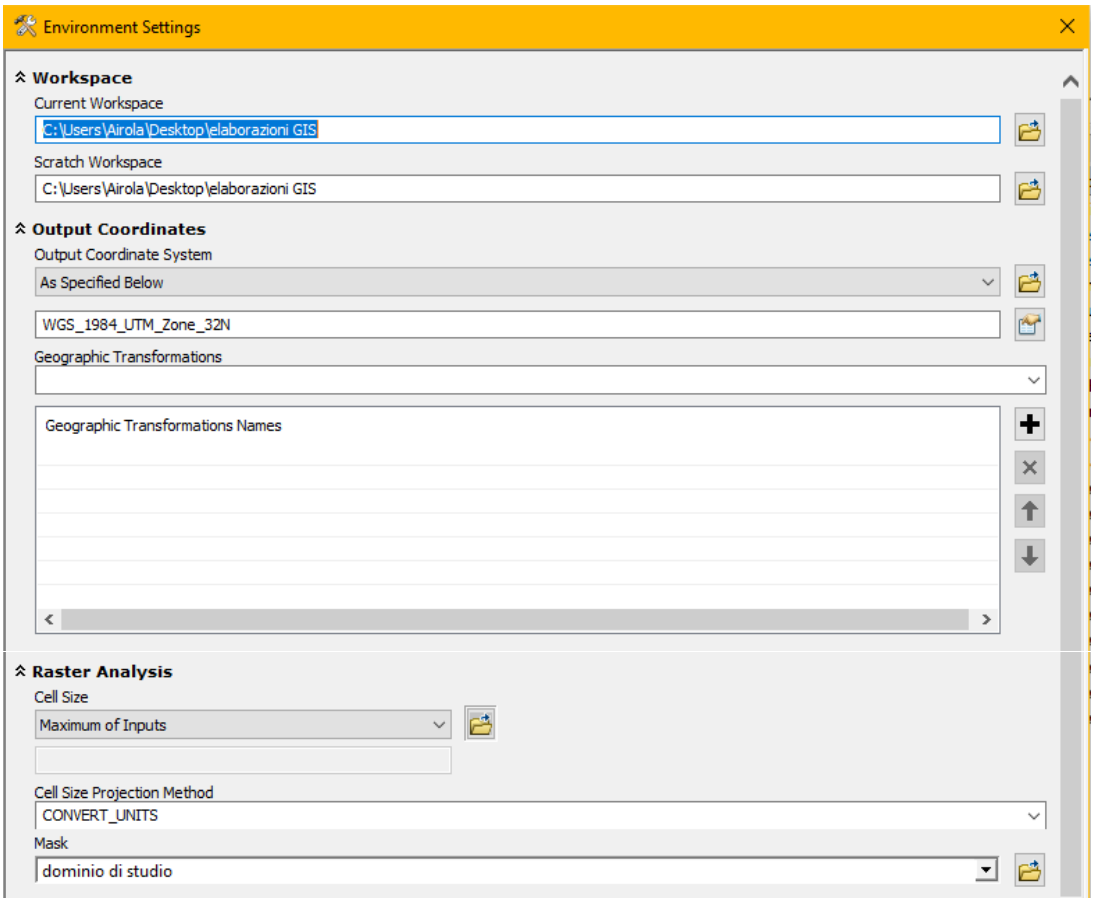
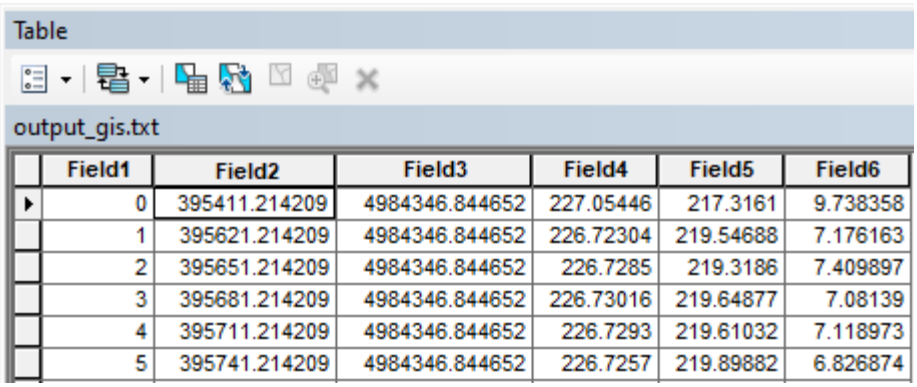


Figura 6.56: Impostazione ambiente di lavoro su ArcMap 10.7

3. Caricamento del file “output_gis.txt”, già aprendo la tabella allegata si può notare come la varie grandezze, che nel file di testo erano suddivise tramite una virgola (Figura 6.54), ora sono correttamente organizzate in colonne, pronte quindi per essere rielaborate (Figura 6.57);



Table

output_gis.txt

	Field1	Field2	Field3	Field4	Field5	Field6
▶	0	395411.214209	4984346.844652	227.05446	217.3161	9.738358
	1	395621.214209	4984346.844652	226.72304	219.54688	7.176163
	2	395651.214209	4984346.844652	226.7285	219.3186	7.409897
	3	395681.214209	4984346.844652	226.73016	219.64877	7.08139
	4	395711.214209	4984346.844652	226.7293	219.61032	7.118973
	5	395741.214209	4984346.844652	226.7257	219.89882	6.826874

Figura 6.57: Tabella associata al file output_gis

4. Estrazione dei punti, cliccando con tasto destro sulla tabella e selezionando “Display XY Data” si apre una finestra (Figura 6.58) dove andare ad inserire in “X Field” il campo contenente i valori di longitudine (Field 2) e in “Y Field” quello con i valori di latitudine (Field 3); il terzo campo ovvero quello corrispondente al valore altimetrico può non essere inserito in questo caso.

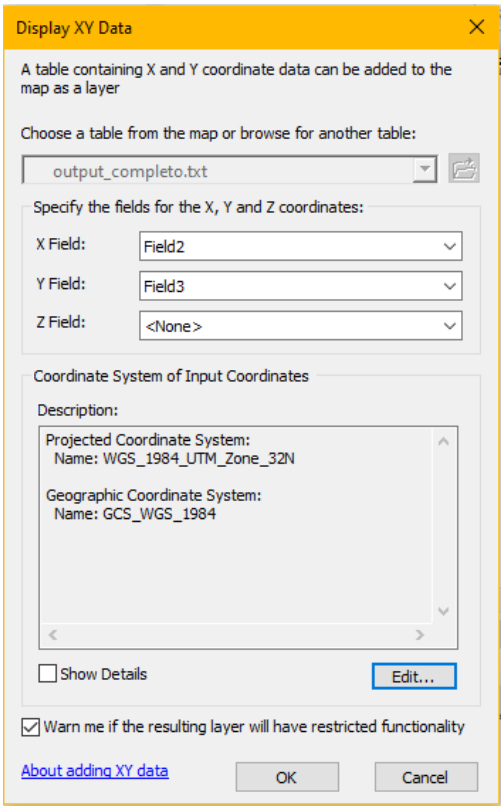


Figura 6.58: Finestra in cui impostare i campi di latitudine e longitudine

Cliccano su “OK” il software inizia ad elaborare restituendo tutti i vari punti come da Figura 6.59.

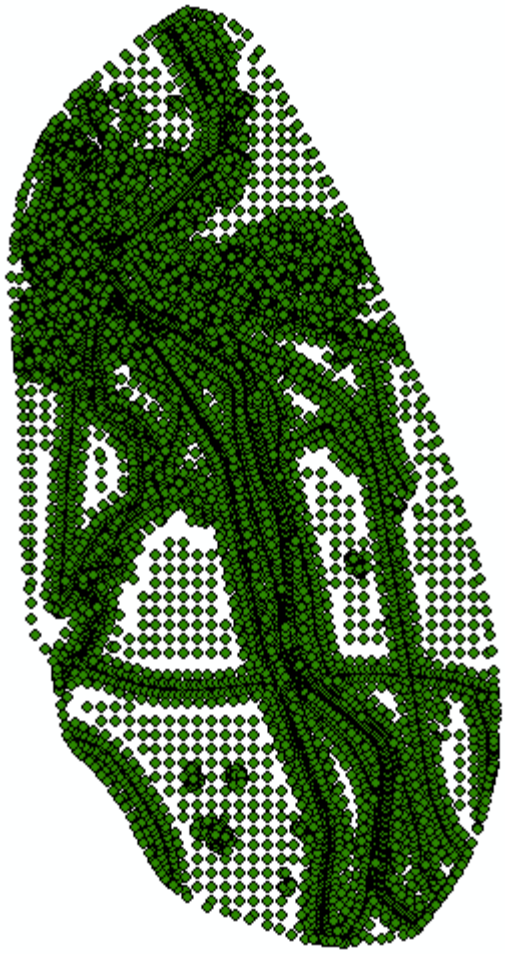


Figura 6.59: Rappresentazione dei punti corrispondenti ai centri delle celle della mesh

In seguito a tutto ciò viene generato un layer con estensione “.txt Events” dal quale è possibile estrarre lo shapefile dei punti, selezionandolo con tasto destro e andando su Data → Export Data.

5. Interpolazione in base ai valori di water depth. In ArcMap vi sono una serie di metodi di interpolazione che possono essere raggiunti cliccando su ArcToolbox→Spatial Analyst Tools→Interpolation (Figura 6.60)

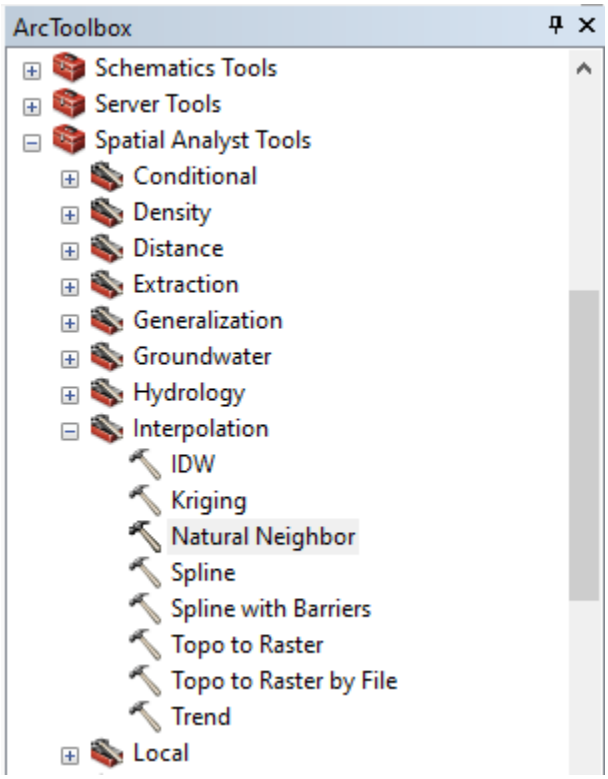


Figura 6.60: Percorso per arrivare alle funzioni di interpolazione

Tra le funzioni di interpolazione disponibili si è utilizzata Natural Neighbor che risulta particolarmente adatta per campioni di dati molto densi. Infatti, come si può notare dalla Figura 6.59 la zona a nord è particolarmente densa in quanto, a causa della presenza dell'abitato di Moncalieri, si è adottata una mesh più fine che fosse in grado di simulare l'andamento di un eventuale flusso d'acqua tra le abitazioni. Prima dell'avvio della fase di interpolazione è ancora necessario completare la compilazione di una finestra che si apre nel momento che si seleziona il metodo di interpolazione voluto (Figura 6.61). Si deve quindi impostare lo shapefile puntuale in input (in questo caso corrisponde ai punti creati al punto 4) e la

colonna della tabella contenente la grandezza per cui si vuole fare l'interpolazione che in questo caso corrisponde al max water depth situata nella sesta colonna (visibile aprendo la tabella associata allo shapefile).

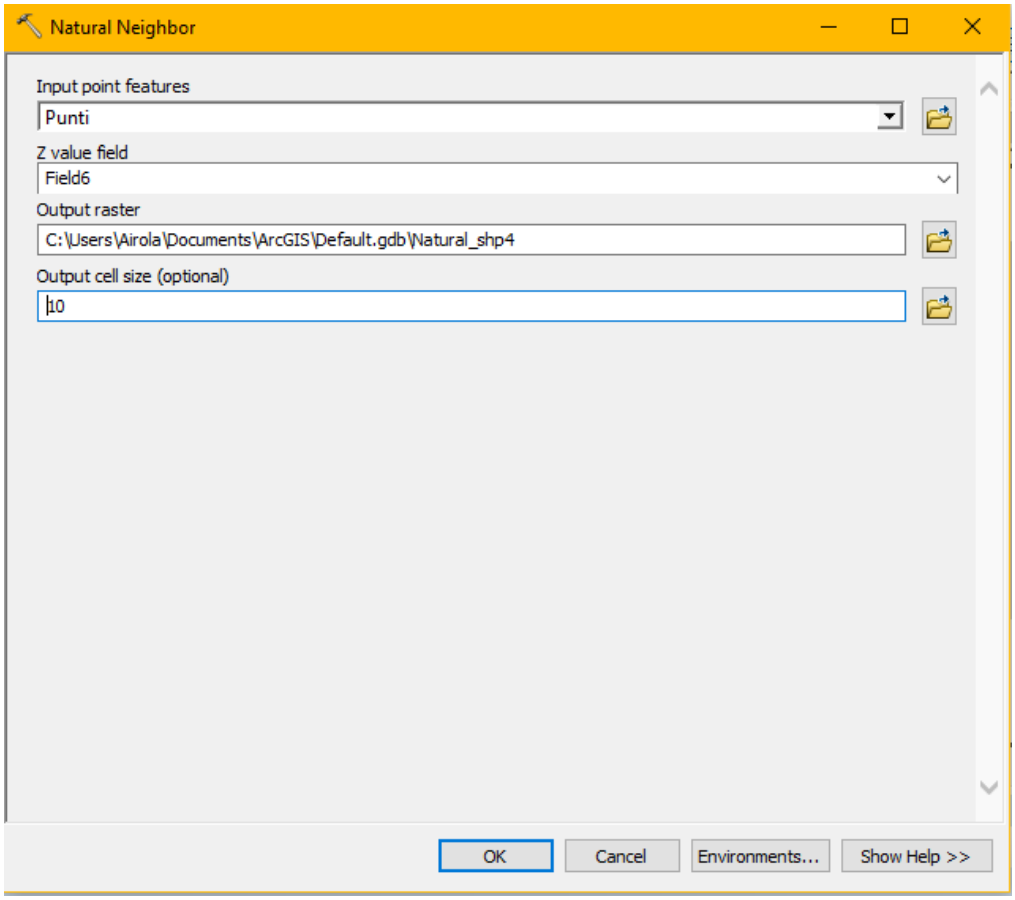


Figura 6.61: Finestra in cui impostare gli input per l'interpolazione

Premendo su OK inizia l'interpolazione da parte del software che dura qualche secondo.

Così facendo viene generato un nuovo shapefile in cui le varie profondità suddivise in range vengono rappresentate con una certa scala di colori creando così una carta tematica che, sovrapposta ad una mappa satellitare del luogo permette di individuare l'estensione dell'allagamento associato all'evento simulato. Un esempio di tali mappe sono riportate nell'*Allegato V*.

6.6 – *Estrazione sequenza temporale di una o più grandezze*

Altro output che può essere importante estrarre è l'andamento nel tempo di una certa grandezza tra quelle estraibili dal file HDF. Si pensi ad un'area che a seguito di un determinato evento viene allagata, può essere utile capire per quanto tempo rimane allagata e avere un'idea dell'andamento del tirante idrico nel tempo. Ciò può essere utile per svariati scopi come la valutazione della stabilità dei rilevati arginali che se esposti a lunghi periodi di allagamento, il generarsi di moti infiltrativi può comprometterne la stabilità.

L'obiettivo è quindi quello di estrarre per una cella o per un gruppo di celle l'andamento temporale di una certa grandezza la cui risoluzione dipende dal valore di "Base Output Interval" impostato nel Plan di HEC-RAS prima dell'inizio della fase di computazione.

Inizialmente si era pensato di procedere come per la generazione della mappa statica ovvero eseguire le simulazioni in serie, quindi analizzare diversi scenari, e salvare le varie grandezze su file di testo. A differenza di prima però il numero di righe necessarie per salvare i dati di una simulazione è ben più elevato in quanto per ogni cella si è interessati all'andamento delle varie grandezze per più istanti temporali, e questo ha fatto sì che il file di testo non fosse più in grado di supportare l'enorme mole di dati derivante dalle simulazioni in serie. Per ovviare a questa problematica si è dovuto fare in modo di diminuire la mole di dati da salvare e quindi si è pensato di non eseguire le simulazioni in serie ma di eseguirne una sola dando la possibilità all'utente di scegliere quale scenario e quali idrogrammi inserire in input.

Il codice completo è riportato nell'Allegato III mentre nei successivi sottoparagrafi si analizzeranno più nel dettaglio le varie fasi.

Uno schema generale della seguente fase viene di seguito riportato.

La prima fase di inserimento dei dati viene effettuata sfruttando le linee di codice riportate di seguito (corrispondo a quelle descritte al paragrafo 6.5 dove l'obiettivo era quello di estrarre determinati dati dal file contenente tutti gli output per poi creare la mappa statica).

```
#INSERIMENTO VARIABILI NECESSARIE PER I SUCCESSIVI CONTROLLI
num_scenari = 3
vet_scenari = np.arange(1,num_scenari,1+1)
tempi_ritorno = [100, 200]

#inserimento scenario con verifica
print("Digitare uno dei seguenti valori in base allo scenario di
interesse:")
print("1 - Onde di piena del Po e Chisola coincidenti")
print("2 - Onda di piena del Po prima dell'onda di piena del Chisola")
print("3 - Onda di piena del Po dopo l'onda di piena del Po")

scenario = int(input("scenario n: "))

flag = 1
while flag != 0:
    for s in range (0, len(vet_scenari)):
        if scenario == vet_scenari[s]:
            flag = 0
        else: continue
    if flag == 1:
        print("Errore, ridigitare valore")
        scenario = int(input("scenario n: "))

#inserimento tempo di ritorno Po con verifica
print("Inserire T_Po (100, 200 )")
T_Po = int(input("T_Po: "))

flag = 1
while flag != 0:
    for s in range (0, len(tempi_ritorno)):
        if T_Po == tempi_ritorno[s]:
            flag = 0
        else: continue
    if flag == 1:
        print("Errore, ridigitare valore")
        T_Po = int(input("T_Po: "))

#inserimento tempo di ritorno Chisola con verifica
print("Inserire T_Chisola (100, 200 )")
T_Ch = int(input("T_Ch: "))

flag = 1
while flag != 0:
    for t in range (0, len(tempi_ritorno)):
```

```
        if T_Ch == tempi_ritorno[t]:
            flag = 0
        else: continue
    if flag == 1:
        print("Errore, ridigitare valore")
        T_Ch = int(input("T_Ch: "))
```

La seconda fase invece si riferisce alla ricerca della colonna nei file excel in cui si trovano gli idrogrammi da estrarre ed inserire nel file ".u". Per far ciò sono state create le seguenti linee di codice:

```
##DETERMINAZIONE POSIZIONE SCENARIO NEI FILE EXCEL
n = 1
flag = 0
while flag == 0:
    if foglio_input.cell(1,n).value != scenario:
        n += 1
    else:
        col_scenario = n
        flag = 1

##DETERMINAZIONE POSIZIONE DELLA COLONNA DEL T_PO
n = col_scenario
flag = 0
while flag == 0:
    if foglio_input.cell(2,n).value != T_Po:
        n += 1
    else:
        col_T_Po = n
        flag = 1

##DETERMINAZIONE POSIZIONE DELLA COLONNA DEL T_CH
n = col_scenario
flag = 0
while flag == 0:
    if foglio2_input.cell(2,n).value != T_Ch:
        n += 1
    else:
        col_T_Ch = n
        flag = 1
```

Come si può notare vi sono tre parti ciascuna riferita alla ricerca di uno dei tre input precedentemente inseriti. La sintassi è simile per ciascuna parte e prevede la presenza di una condizione ciclica all'interno della quale viene effettuato il confronto tra il contenuto di una determinata cella e il valore ricercato, e viene incrementata una variabile contatore "n" fino a quando non si ottiene l'uguaglianza. Questa ricerca prevede la conoscenza della struttura del file Excel tant'è che lo scenario viene ricercato nella prima riga mentre il T_Po e T_Ch nella seconda e si sa che gli idrogrammi partono dalla terza riga. In questo modo quindi si determinano le posizione delle celle da cui partire ad estrarre i due idrogrammi.
Si passa quindi alla terza fase dove vengono effettivamente estratti i valori degli idrogrammi e salvati temporaneamente in due vettori "*vet e vet2*":

```
##ESTRAZIONE DATI E SALVATAGGIO IN VETTORI
z=0
vet = np.empty([num_righe], dtype=float)
vet2 = np.empty([num_righe], dtype=float)
for j in range(1,num_righe+1):
    vet[z] = foglio_input.cell(j+2, col_T_Po).value
    vet2[z] = foglio2_input.cell(j+2, col_T_Ch).value
    z+=1
vet.resize(num_lines, 10)
vet2.resize(num_lines, 10)
print(vet)
print(vet2)
```

L'estrazione avviene all'interno del ciclo "for" dove l'operazione viene ripetuta per un numero di volte pari a "*num_righe*" che rappresenta il numero di valori che costituiscono l'idrogramma e determinato come di seguito:

```
##DETERMINAZIONE LUNGHEZZA IDROGRAMMA
w=0
num_righe=0
j=0
while w==0:
    if foglio_input.cell(j+2,1).value != None:
        num_righe+=1
        j+=1
```

```
    else:
        w=1
```

Anche in questo caso si sfrutta un ciclo "*while*" all'interno del quale si pone la condizione di verifica che in questo caso corrisponde a controllare che il contenuto della cella sia diverso da "*None*" ovvero che la cella non sia vuota andando così ad incrementare una variabile contatore "*num_righe*" fino a quando non si raggiunge la prima cella vuota e quindi la fine dell'idrogramma.
I due passi successivi che riguardano l'inserimento nel file ".u" dei due idrogrammi e poi l'esecuzione della simulazione viene effettuata con le stesse linee di codice già presentate nei paragrafi precedenti e qui riportate per continuità del discorso:

```
##MODIFICARE IL FILE CONTENENTE L'IDROGRAMMA
file_in = open(r"C:\Users\Win-
10\Desktop\seq_temporale\copia_input.txt","r")
file_out= open(r"C:\Users\Win-
10\Desktop\seq_temporale\SIMULAZIONE2.u02","w")

n=0
b=0
y=0
for riga in file_in:
    if count+1 <= n <= count+num_lines:
        for k in range (0,10):
            a = vet[b,k]
            cifre = len(str(a))
            if cifre == 1:
                file_out.write("       "+str(a))
            elif cifre == 2:
                file_out.write("      "+str(a))
            elif cifre == 3:
                file_out.write("     "+str(a))
            elif cifre == 4:
                file_out.write("    "+str(a))
            elif cifre == 5:
                file_out.write("   "+str(a))
            elif cifre == 6:
                file_out.write("  "+str(a))
            elif cifre == 7:
                file_out.write(" "+str(a))
```

```
        file_out.write("\n")
        b+=1
        n+=1
    elif count+count2+2 <= n <= count+count2+num_lines+1:
        for k in range (0,10):
            a = vet2[y,k]
            cifre = len(str(a))
            if cifre == 1:
                file_out.write("       "+str(a))
            elif cifre == 2:
                file_out.write("      "+str(a))
            elif cifre == 3:
                file_out.write("     "+str(a))
            elif cifre == 4:
                file_out.write("    "+str(a))
            elif cifre == 5:
                file_out.write("   "+str(a))
            elif cifre == 6:
                file_out.write("  "+str(a))
            elif cifre == 7:
                file_out.write(" "+str(a))
        file_out.write("\n")
        y+=1
        n+=1
    else:
        file_out.write(riga)
        n+=1

file_in.close()
file_out.close()
z=0

##ESEGUIRE LA SIMULAZIONE IN HECRAS
hec.Project_Open(RASProject)
NMsg,TabMsg,block = None,None,True
hec.ShowRas()
hec.Compute_ShowComputationWindow()
v1,NMsg,TabMsg,v2 = hec.Compute_CurrentPlan(NMsg,TabMsg,block)
```

Terminata la simulazione vi è l'ultima fase ovvero il salvataggio dei dati. Come già anticipato l'obiettivo ora è quello di salvare per ogni cella tutta la sequenza temporale di una determinata grandezza. Ciò è stato fatto con le seguenti righe di codice:

```
##APRO FILE HDF E ESTRAGGO GLI OUTPUT DI INTERESSE
#salvataggio in vettori/matrici
with
h5py.File(r"C:\Users\Win10\Desktop\seq_temporale\SIMULAZIONE2.p01.hdf",
      "r") as hdf:
    G1 = hdf.get('Results/Unsteady/Output/Output Blocks/Base
      Output/Unsteady Time Series/2D Flow Areas/2d area')
      wse = np.array(G1.get('Water Surface'))
    G2 = hdf.get('Geometry/2D Flow Areas/2d area')
      min_elev = np.array(G2.get('Cells Minimum Elevation'))
    G3 = hdf.get('Geometry/2D Flow Areas/2d area')
      coord = np.array(G3.get('Cells Center Coordinate'))

#determinazione dimensioni vettori/matrici
nrows1, ncols1 = np.shape(wse)
nrows2 = np.shape(min_elev)
nrows3, ncols3 = np.shape(coord)

##APERTURA FILE DI TESTO E INSERIMENTO DATI
output=open(r"C:\Users\Win-
10\Desktop\seq_temporale\output_completo.txt","w")
num_id = 0
delta_t = 0
for i in range(0,ncols1):
    for j in range(0,nrows1):
        output.write(str(num_id))          #num cella
        output.write(", ")
        output.write(str(coord[i,0]))      #longitudine
        output.write(", ")
        output.write(str(coord[i,1]))      #latitudine
        output.write(", ")
        output.write(str(wse[j,i]))        #water surface
        output.write(", ")
        output.write(str(min_elev[i]))     #min elevation DTM
        output.write(", ")
        depth = wse[j,i] - min_elev[i]     #calcolo tirante
        output.write(str(depth))           #tirante
        output.write(", ")
```

```
        output.write(str(delta_t))          #delta t (30 min)
        output.write("\n")
        delta_t += 0.5
    delta_t = 0
    num_id +=1

output.close()
```

Dopo aver aperto il file HDF, raggiunto le cartelle che contengono i dati di interesse e salvati in apposite strutture (vettori o matrici) si deve procedere alla scrittura su file. La grandezza principale che si vuol estrarre in questo caso è la "*Water Surface*" che si presenta salvata nel file .hdf sotto forma di matrice (Figura 6.62); pertanto per poterla scansionare e poter estrarre tutti i valori è stato necessario inserire due cicli "*for*" annidati in modo da estrarre per ogni cella tutta la sequenza temporale.

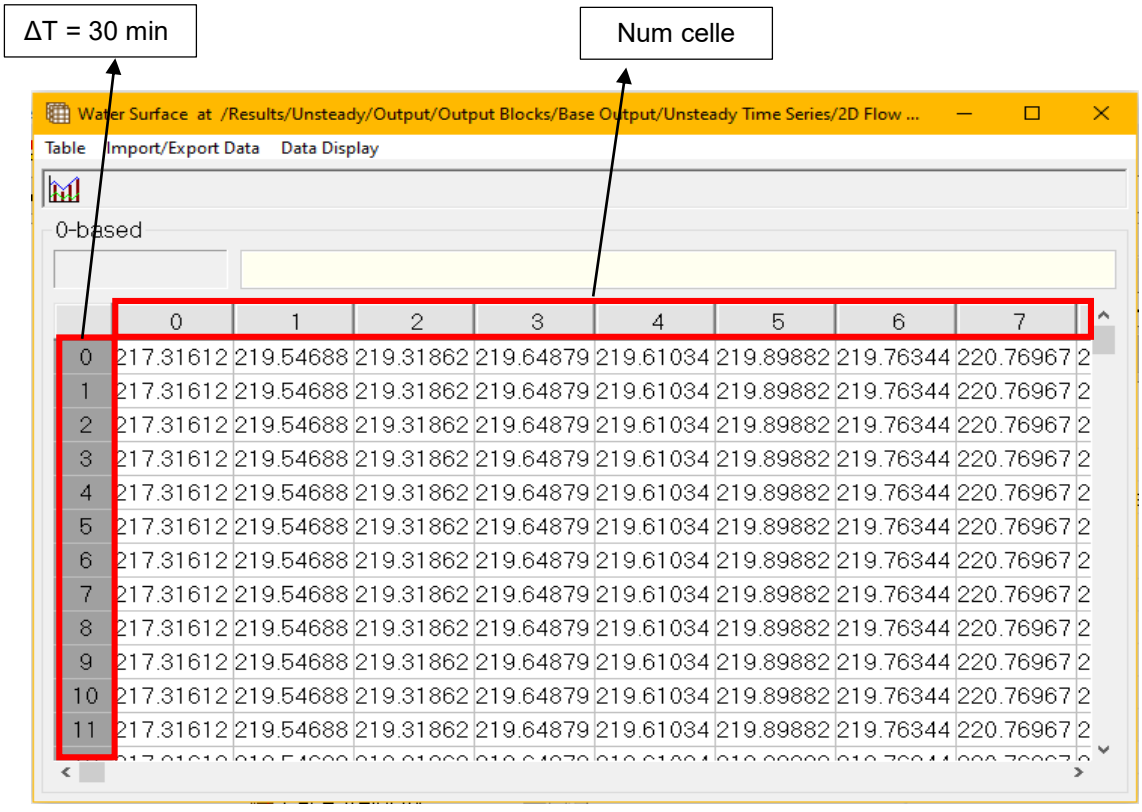


Figura 6.62: Estratto della tabella contenente i dati di Water Surface

Si riporta in Figura 6.63 un estratto del file di testo così creato.

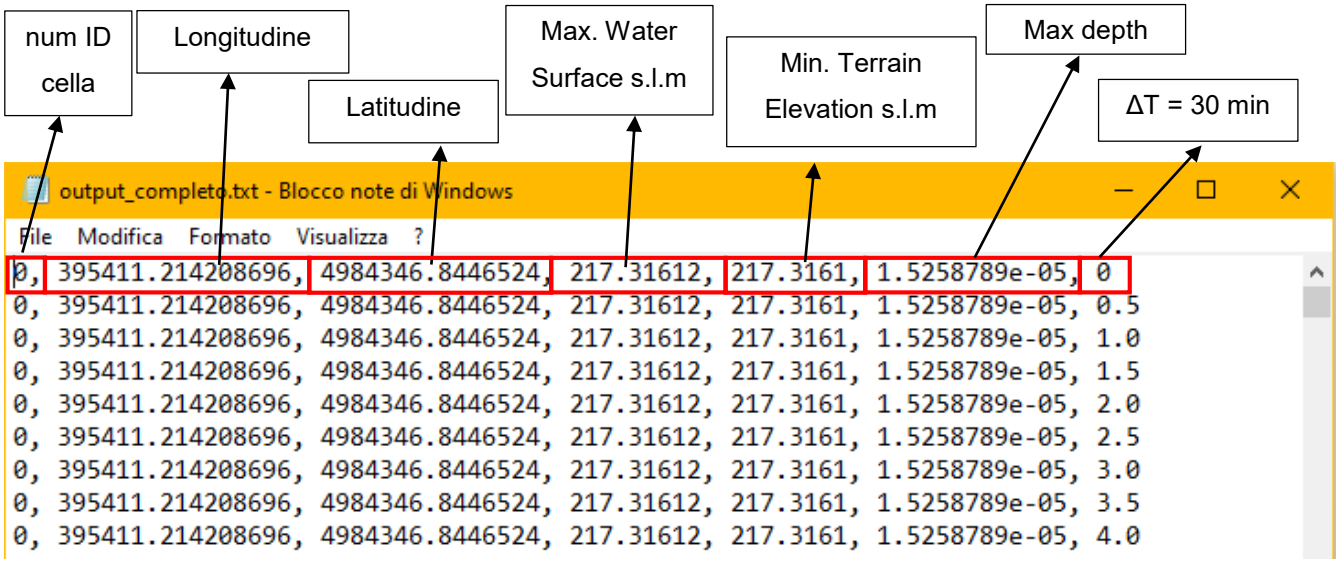
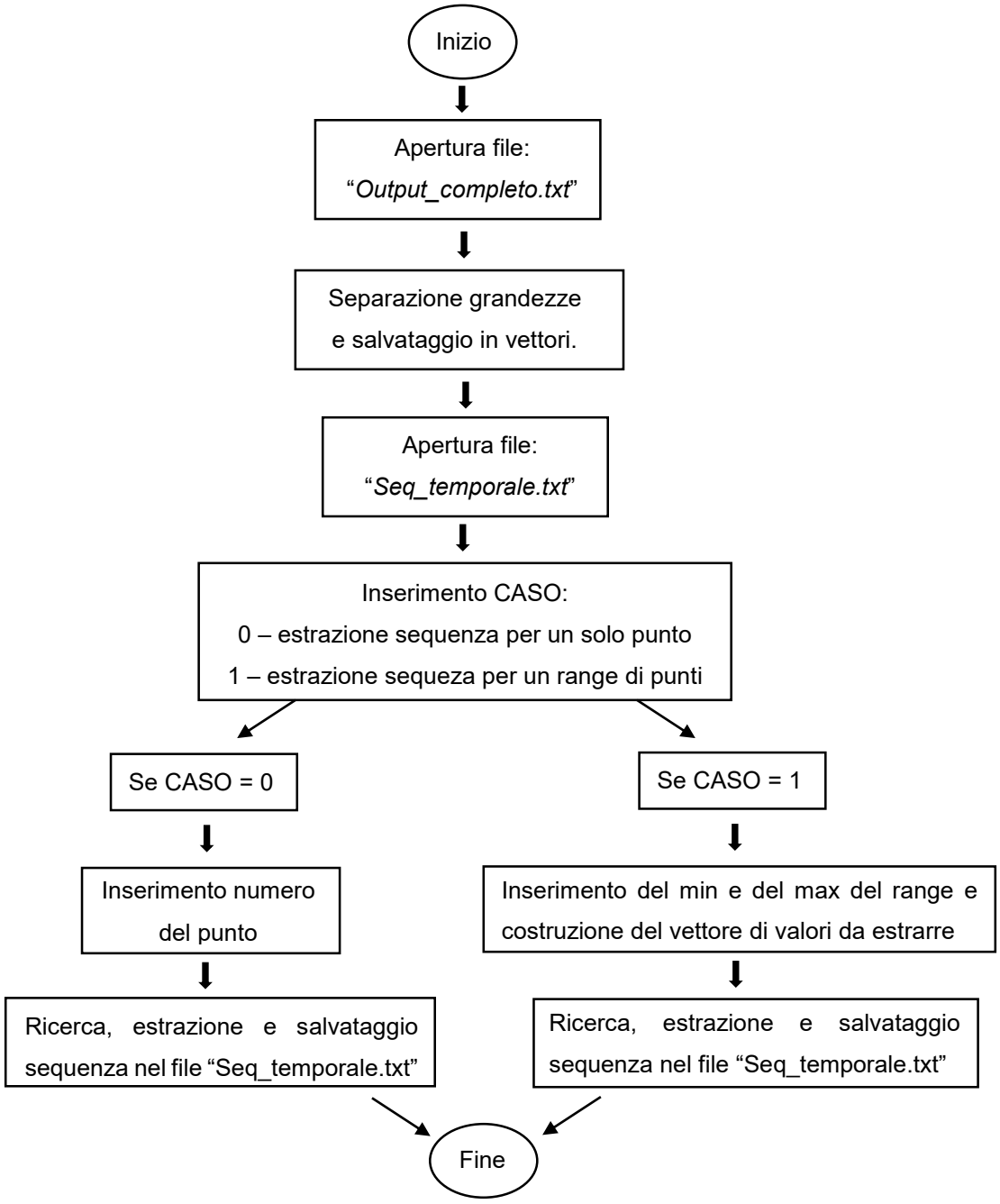


Figura 6.63: Estratto del file di testo contenente le grandezze

6.6.1 – Post-processing dati e salvataggio sequenza per determinati punti del dominio

Una volta salvati tutti i dati sul file di testo "*output_completo.txt*" si è creato un ulteriore codice che permettesse di estrarre da esso la sequenza solamente per alcune celle di particolare interesse in modo tale da rendere più agevole la ricerca e la consultazione dell'output che altrimenti sarebbe difficoltosa vista la numerosità di righe presenti nel file.

Si presenta di seguito uno schema esemplificativo del funzionamento del codice:



Di seguito invece si riportano le linee di codice:

```
#INSERIMENTO MODULI
import numpy as np
import h5py
import math

#inizializzazione vettori
num_id = []
lon = []
lat = []
wse = []
min_elev = []
depth = []
delta_t = []

#apertura file contenente la sequenza per tutte le celle
output_generale = open(r"C:\Users\Win-10\Desktop\seq_temporale
      \output_completo.txt","r")

#separazione grandezze e salvataggio in vettori
cont=0
for riga in output_generale:
    splitLine = riga.split(", ")
    num_id.append(str(splitLine[0]))
    wse.append(str(splitLine[3]))
    min_elev.append(str(splitLine[4]))
    depth.append(str(splitLine[5]))
    delta_t.append(str(splitLine[6]))
    cont+=1

#file txt contenente l'output delle sequenze volute
output_seq_temp = open(r"C:\Users\Win-10\Desktop\seq_temporale
      \output_seq_temp.txt","w")

#inserimento manuale della cella o delle celle per cui interessa estrarre
la sequenza temporale
print("Se si vuole estrarre un solo valore digitare 0")
print("Se si vuole estrarre un range di valori digitare 1")
caso = int(input("Caso (0 o 1): "))

#condizione ciclica in cui si effettua l'estrazione in base al caso
scelto oppure si reinserisce il valore se inserito erratamente
flag = 1
```

```
while flag != 0:
      #se il caso è 0 inserire il valore del punto
    if caso == 0:
        print("Inserire il numero del punto per cui si vuol estrarre la
            sequenza temporale.")
        punto = str(input("Punto/i: "))
        #ricerca del punto ed estrazione
        for j in range(0,cont):
            if punto == num_id[j]:
                output_seq_temp.write(str(num_id[j]))
                output_seq_temp.write(", ")
                wse_def = round(float(wse[j]),2)
                output_seq_temp.write(str(wse_def))
                output_seq_temp.write(", ")
                min_elev_def = round(float(min_elev[j]),2)
                output_seq_temp.write(str(min_elev_def))
                output_seq_temp.write(", ")
                depth = float(wse[j]) - float(min_elev[j])
                depth_def = str(round(depth,2))
                output_seq_temp.write(str(depth_def))
                output_seq_temp.write(", ")
                output_seq_temp.write(str(delta_t[j]))
            else:
                continue
        flag = 0
      #se il caso è 1 inserire i due valori (min e max)
    elif caso == 1:
        print("Inserire il min (Punto 1) e il max (Punto 2)")
        punto_1 = int(input("Punto 1: "))
        punto_2 = int(input("Punto 2: "))
        #generazione vettore di punti da estrarre
        vet_punti = np.arange(punto_1,punto_2+1)
        for i in range(0,len(vet_punti)):
            for j in range(0,cont):
                if str(vet_punti[i]) == num_id[j]:
                    output_seq_temp.write(str(num_id[j]))
                    output_seq_temp.write(", ")
                    wse_def = round(float(wse[j]),2)
                    output_seq_temp.write(str(wse_def))
                    output_seq_temp.write(", ")
                    min_elev_def = round(float(min_elev[j]),2)
                    output_seq_temp.write(str(min_elev_def))
                    output_seq_temp.write(", ")
                    depth = float(wse[j]) - float(min_elev[j])
                    depth_def = str(round(depth,2)
```

```
                    output_seq_temp.write(str(depth_def))
                    output_seq_temp.write(", ")
                    output_seq_temp.write(str(delta_t[j]))
        flag = 0
      #reinserimento valore se errato
    elif flag==1:
        caso = int(input("Errore, ridigitare valore (0 o 1): "))

#chiusura file
output_generale.close()
output_seq_temp.close()
```

Seguendo lo schema riportato sopra il codice, si può intuire come all'inizio si debbano separare le varie grandezze e salvarle in appositi vettori precedentemente inizializzati. La separazione avviene utilizzando la funzione "*split*" ed in particolare la seguente sintassi:

```
splitLine = riga.split(", ")
```

Dove si indica che la riga del file dev'essere separata in corrispondenza della virgola (vedere Figura 6.63). Una volta effettuata la separazione le grandezze devono essere salvate in appositi vettori, sfruttando la seguente sintassi (esempio in cui si popola il vettore dei numeri delle celle "*num_id*", dove 0 indica il primo elemento della separazione):

```
num_id.append(str(splitLine[0]))
```

Questa operazione effettuata per tutte le righe del file permette di creare dei vettori con la quale si lavorerà e necessari per effettuare le successive operazioni.
Le righe seguenti del codice vedono l'inserimento da parte dell'utente di un input che può essere "0" se si vuol estrarre la sequenza per un solo punto oppure "1" per effettuare l'estrazione per una serie di punti. Si entra quindi in una condizione iterativa in cui se l'input inserito è uguale a "0" viene richiesto l'inserimento del numero del punto da ricercare ed estrarre mentre se uguale a "1" si richiede l'inserimento del minimo e del massimo del range, si costruisce un vettore con tutti

i valori da estrarre e infine si ricercano tali valori e si estraggono nel file di testo apposito. Il vettore viene creato automaticamente sfruttando la funzione "*arange*" del modulo "*numpy*" che dati gli estremi di un intervallo ricostruisce i valori intermedi basandosi sullo step dato (in questo caso +1).

```
vet_punti = np.arange(punto_1,punto_2+1)
```

Le fasi successive di ricerca e salvataggio su file di testo avvengono confrontando il valore inserito singolarmente o i valori presenti nel vettore con quelli dei punti presenti nel vettore "*num_id*", una volta soddisfatta l'uguaglianza (in base alla posizione raggiunta nel vettore) si estraggono di conseguenza le altre grandezze che si trovano alla stessa posizione. L'estrazione prosegue fino a che l'uguaglianza non è più soddisfatta, generando un file di testo con molti meno dati e quindi più facilmente consultabile.

Il numero della cella o del gruppo di celle può essere determinato aprendo in GIS il file contenente la mappa statica creata in precedenza e visualizzando i numeri dei centri delle celle (andando sulle proprietà del layer e aggiungendo l'etichetta corrispondente al numero della cella) come in Figura 6.64.

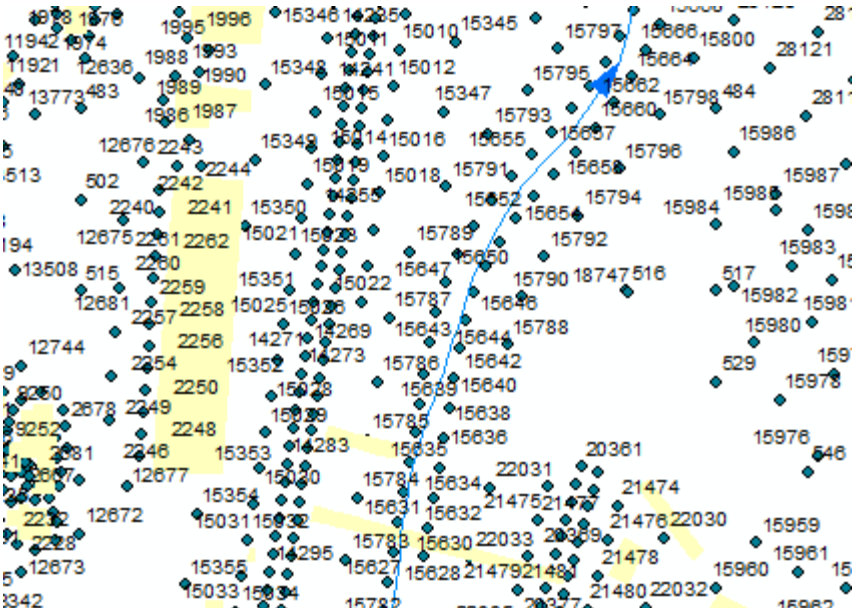


Figura 6.64: Esempio di rappresentazione dei numeri dei punti rappresentanti i centri delle celle della mesh

7 - Conclusioni

L'utilizzo della modellazione idraulica in ambito fluviale è oramai normale pratica da alcuni anni ed è uno strumento molto importante e necessario per la valutazione delle aree soggette a rischio alluvionale. Ad oggi sono presenti svariati software che consentono di effettuare tali analisi e tra questi vi è HEC-RAS; si tratta di un software gratuito e liberamente scaricabile che permette di fare simulazioni in campo monodimensionale e bidimensionale. Normalmente si lavora mediante simulazioni singole avviate manualmente; questa operazione va sicuramente bene per tutti quei casi in cui le simulazioni da eseguire sono limitate ma può diventare lunga e onerosa in caso contrario ed è proprio in questi casi che può essere utile introdurre il concetto di automazione delle simulazioni.

Il processo di controllo e automazione di HEC-RAS è reso possibile dalle presenza del "HEC-RAS Controller" ovvero di una raccolta di funzioni implementabili all'interno di un codice generato con un generico linguaggio di programmazione. In questa tesi si è proposto l'utilizzo di Python che presenta una serie di vantaggi, tra i più importanti quello di accedere con facilità ai file con estensione HDF (contenenti i dati output di HEC-RAS) per estrarre ed analizzare determinati dati di interesse.

Nei primi capitoli sono state presentate le funzioni principali che permettono il controllo e i concetti generali alla base dell'automazione. Successivamente è stata proposta un'applicazione analizzando un determinato caso studio; l'idea di base consisteva nell'analizzare diversi scenari d'evento che potessero potenzialmente interessare i territori del Comune di Moncalieri dal punto di vista del rischio idraulico, data la presenza del torrente Chisola e del fiume Po con annessa confluenza a nord dello stesso territorio comunale, ipotizzando onde di piena con diversi tempi di ritorno e considerando anche l'aspetto temporale ovvero onde di piena coincidenti alla confluenza oppure no. Si tratta quindi di più simulazioni eseguibili in serie in cui andare a variare ogni volta gli idrogrammi in ingresso e salvare opportuni dati in

uscita, utili per successive rielaborazioni. Alcuni di questi dati sono stati utilizzati sia per la generazione di output grafici come mappe statiche (generate mediante software GIS e utili per individuare l'entità degli allagamenti per i vari casi simulati) sia di output numerici come l'estrazione della sequenza temporale di una certa grandezza come ad esempio la "Water Surface Elevation" per determinate celle del dominio di studio (utile per stimare la durata dell'allagamento).

Per eseguire tutto ciò sono stati realizzati quattro codici: il primo consente l'esecuzione in serie delle simulazioni variando gli input e salvando per ognuna di essa determinati dati in un apposito file di testo; il secondo è stato realizzato per poter estrarre determinati dati dal file creato e popolato precedentemente e utili per la realizzazione della mappa statica; il terzo per l'esecuzione della simulazione dalla quale estrarre la sequenza temporale di determinate grandezze per tutti i punti del dominio ed infine il quarto per estrapolare la sequenza esclusivamente per determinati punti, in modo da non dover ricercare manualmente i dati di interesse in quanto contenuti in un file di testo di migliaia di righe.

Dal punto di vista della funzionalità e della stabilità del codice in fase di esecuzione è importante fare due semplici ma importanti precisazioni: la prima è che, durante l'esecuzione delle simulazioni, il computer non deve andare in sospensione quindi si consiglia di controllare ed eventualmente cambiare le impostazioni generali; la seconda riguarda sempre la fase di esecuzione, in particolare si devono lasciare le finestre aperte sul desktop (sia quelle di Python sia quelle di HEC-RAS) e non ridurle ad icona. In entrambi i casi nel caso in cui non si rispettino tali regole il programma si bloccherà mostrando un messaggio di errore nel momento in cui vi è il passaggio tra una simulazione e l'altra.

I risultati così ottenuti possono essere sicuramente molto utili a fini pratici per una migliore valutazione del rischio idraulico dell'area, inoltre la possibilità di analizzare dati come ad esempio il tirante idrico per diversi istanti temporali in una determinata zona permette di fare delle valutazioni in merito al tempo che una determinata area

rimane allagata e di conseguenza valutare possibili rischi ad esso associato. In questo senso tale valutazione può essere fatta per verificare la stabilità dei rilevati arginali, che se esposti a lunghi periodi di allagamento i moti di filtrazione che si innescano possono minare la stabilità degli stessi.

In conclusione si ritiene che questi concetti possano essere ancora ampliamente sviluppati ed affinati, applicandoli ad altri casi in cui il concetto di automazione può essere di particolare aiuto.

Bibliografia

1. D. Ganora, F.Laio, P.Claps (2014) *Valutazione probabilistica delle piene in Piemonte e Valle d'Aosta*. Progetto di ricerca FLORA tra Politecnico di Torino e ARPA Piemonte.
2. Di Bello Bonaventura (2019) *Programmare con Python for dummies*. HOEPLI
3. Dysarz Tomasz (2018) *Application of Python Scripting Techniques for Control and Automation of HEC-RAS Simulations*. Department of Hydraulic and Sanitary Engineering, Poznan University of Life Sciences, Poland.
4. E. Gallo, D. Ganora, F. Laio, A. Masoero, P. Claps (2013) *Atlante dei bacini imbriferi piemontesi*. Progetto RENERFOR Regione Piemonte.
5. Filippini Guglielmo (2016) *Relazione sintetica sull'evento Alluvionale del 23-26 Novembre 2016*. Città metropolitana di Torino.
6. Goodell Christopher, P.E., D.WRE (2014) *Breaking the HEC-RAS Code, A User's Guide to Automating HEC-RAS*. USA
7. US Army Corps of Engineers – Hydrologic Engineering Center (2016) *HEC-RAS River Analysis System Hydraulic Reference Manual*, USA.
8. US Army Corps of Engineers – Hydrologic Engineering Center (2018) *HEC-RAS River Analysis System Supplemental User's Manual Version 5.0.4*, USA.

Sitografia

1. https://www.youtube.com/watch?v=T_I0OiyciFk – NiktorTheNat – Corso completo per principianti sul Python [tutorial] – ultimo accesso: settembre 2020
2. https://www.youtube.com/watch?v=R795eunWB_E&t=413s – Navid Jadidoleslam – 2D Flow Modeling Using HEC-RAS 5.0 – ultimo accesso: settembre 2020
3. https://www.youtube.com/watch?v=dlN9W54GbmE – PyMike – Tutorial Python 3 - 19 - Come Installare Moduli di Terze Parti con PIP - Programmare In Python – ultimo accesso: settembre 2020
4. https://www.youtube.com/watch?v=I0HZ77SyCQk – NiktorTheNat - Tutorial Python su come leggere dati da una cartella di lavoro Excel – ultimo accesso: settembre 2020
5. https://www.youtube.com/watch?v=9-F9e4HLDJY – NiktorTheNat - Python 3.x - Tutorial 33 - Le Funzioni – ultimo accesso: ottobre 2020
6. https://www.youtube.com/watch?v=113u1trnSXs – PyMike - Tutorial Python 3 - Come Creare, Leggere e Scrivere File Testuali - Programmare In Python – ultimo access: novembre 2020
7. https://www.youtube.com/watch?v=xuWB_byi-6Q&pbjreload=101 – Noureddin Sadawi - 4/10- HDF5 with Python: How to Read HDF5 Files – ultimo accesso: novembre 2020
8. https://docs.python.it/html/lib/module-random.html - ultimo accesso: novembre 2020
9. https://www.hdfgroup.org/downloads/hdfview/ - ultimo accesso: novembre 2020
10. https://www.geoportale.piemonte.it/cms/ - ultimo accesso: gennaio 2021
11. https://webgis.arpa.piemonte.it/agportal/apps/opsdashboard/index.html#/19fc2f223314452cb5bd7548d67885b3 - ultimo accesso: gennaio 2021

Allegati – Codici in Python

Allegato I

Esecuzione simulazioni in serie e salvataggio dati in file di testo

```
##INSERIMENTNO MODULI
from openpyxl import load_workbook      #per leggere dati da Excel
import numpy as np                      #per poter lavorare con vettori
import os                               #per poter lavorare con file
import win32com.client                  #per usare l'HEC-RAS Controller
import h5py                             #per lavorare con file HDF
import math                             #per usare funzioni matematiche

##CREAZIONE FUNZIONE PER RICERCARE PAROLE O FRASI ALL'INTERNO DEI FILE
#inserimento nome funzione e relativi argomenti
def ricerca_elemento(file,elem):
    #inizializzazione variabili
    riga=" "
    count=0
    count2=0
    n=0
    #condizione ciclica per scandire le righe del file
    while(riga):
        riga=file.readline()
        #condizione di ricerca
        if elem in riga:
            n+=1
            if n==1:
                return count
            elif n==2:
                return count2
        count+=1
        count2+=1


##INIZIO PROGRAMMA
#variabile hec per usare le funzioni dell'hecras control
hec = win32com.client.Dispatch("RAS507.HECRASController")
#caricamento del percorso del progetto
RASProject = os.path.join(os.getcwd(),r'C:\Users\Win-
      10\Desktop\mappa_statica\SIMULAZIONE2.prj')
```

```
#caricamento del file contenente tutti gli idrogrammi
cartella_input = load_workbook(filename="idrogrammi_Po.xlsx")
cartella2_input = load_workbook(filename="idrogrammi_Chisola.xlsx")
#individuazione foglio Excel contenente gli idrogrammi
foglio_input = cartella_input.active
foglio2_input = cartella2_input.active

#inizializzazione variabili
num_scenari = 3
num_T_Po = 2
num_T_Ch = 2

#det. numero valori degli idrogrammi
w=0
num_righe=0
j=0
while w==0:
    if foglio_input.cell(j+2,1).value != None:
        num_righe+=1
        j+=1
    else:
        w=1
#numero di righe occupate dall'idrogramma ne file ".u"
num_lines = math.ceil((num_righe/10)-1)

m=0
#inserimento condizioni iterative
for s in range(0,num_scenari):
    for i in range(1,num_T_Po+1):
        z=0
        for v in range (1, num_T_Ch+1):
            #iniz. vettori in cui salvare gli idrog. estratti da Excel
            vet = np.empty([num_righe], dtype=float)
            vet2 = np.empty([num_righe], dtype=float)
            #estrazione idrogrammi
            for j in range(1,num_righe+1):
                vet[z] = foglio_input.cell(j+2, i+(s*num_T_Po)).value
                vet2[z] = foglio2_input.cell(j+2, v+(s*num_T_Ch)).value
                z+=1
            #trasformo il vettore in una matrice num_lines x 10
            vet.resize(num_lines, 10)
            vet2.resize(num_lines, 10)
            print(vet)        #stampo nella finestra della shell
            print(vet2)
```

```
#CERCARE FLOW HYDRO. IN FILE CON FUNZIONE
file_in= open (r"C:\Users\Win-10\Desktop\mappa_statica
\copia_input.txt","r")
elem = "Flow Hydrograph="
count = ricerca_elemento(file_in, elem)
count2 = ricerca_elemento(file_in, elem)
file_in.close()

#MODIFICARE IL FILE CONTENENTE L'IDROGRAMMA
#caricamento file per copiare le righe che non cambiano
file_in= open (r"C:\Users\Win-10\Desktop\mappa_statica
\copia_input.txt","r")
#file aggiornato da usare per la simulazione
file_out= open (r"C:\Users\Win-10\Desktop\mappa_statica
\SIMULAZIONE2.u01","w")

#contatore per il numero di righe del file
n=0
#contatore per le righe della matrice da inserire nel file
b=0
y=0
#scansiono riga x riga il file_in
for riga in file_in:
    #se sono nel primo range di righe da cambiare
    if count+1 <= n <= count+num_lines:
        #ciclo per le colonne della matrice
        for k in range (0,10):
        #salvo il singolo valore del vettore in una variabile
            a = vet[b,k]
            #valuto il numero di cifre di cui è composto
            cifre = len(str(a))
            #se a ha 1 cifra aggiungo 7 spazi vuoti
            if cifre == 1:
                file_out.write("       "+str(a))
            #se a ha 2 cifre aggiungo 6 spazi vuoti
            elif cifre == 2:
                file_out.write("      "+str(a))
            #se a ha 3 cifre aggiungo 5 spazi vuoti
            elif cifre == 3:
                file_out.write("     "+str(a))
            #se a ha 4 cifre aggiungo 4 spazi vuoti
            elif cifre == 4:
                file_out.write("    "+str(a))
            #se a ha 5 cifre aggiungo 3 spazi vuoti
            elif cifre == 5:
```

```
                    file_out.write("   "+str(a))
                #se a ha 6 cifre aggiungo 2 spazi vuoti
                elif cifre == 6:
                    file_out.write("  "+str(a))
                #se a ha 7 cifre aggiungo 1 spazi vuoti
                elif cifre == 7:
                    file_out.write(" "+str(a))
            #vado a capo dopo aver inserito una riga
            file_out.write("\n")
            b+=1
            n+=1
        #se sono nel secondo range di righe da cambiare
        elif count+count2+2 <= n <= count+count2+num_lines+1:
            for k in range (0,10):
                a = vet2[y,k]
                cifre = len(str(a))
                if cifre == 1:
                    file_out.write("       "+str(a))
                elif cifre == 2:
                    file_out.write("      "+str(a))
                elif cifre == 3:
                    file_out.write("     "+str(a))
                elif cifre == 4:
                    file_out.write("    "+str(a))
                elif cifre == 5:
                    file_out.write("   "+str(a))
                elif cifre == 6:
                    file_out.write("  "+str(a))
                elif cifre == 7:
                    file_out.write(" "+str(a))
            file_out.write("\n")
            y+=1
            n+=1
        else:
            file_out.write(riga)
            n+=1

file_in.close()
file_out.close()
z=0
```

```
#ESEGUIRE LA SIMULAZIONE IN HECRAS
hec.Project_Open(RASProject)
NMsg,TabMsg,block = None,None,True
hec.ShowRas()
hec.Compute_ShowComputationWindow()
v1,NMsg,TabMsg,v2 =
      hec.Compute_CurrentPlan(NMsg,TabMsg,block)

#SALVARE OUTPUT SU FILE DI TESTO
#apro file hdf
with h5py.File(r"C:\Users\Win-10\Desktop\mappa_statica
\SIMULAZIONE2.p05.hdf","r") as hdf:
    #ricerca cartella contenente l'output
    G1 = hdf.get('Results/Unsteady/Output/Output Blocks/Base
    Output/Unsteady Time Series/2D Flow Areas/2d area')
    #salvataggio temporaneo in vettore
    wse = np.array(G1.get('Water Surface'))
    G2 = hdf.get('Geometry/2D Flow Areas/2d area')
    coord = np.array(G2.get('Cells Center Coordinate'))
    G3 = hdf.get('Geometry/2D Flow Areas/2d area')
    min_elev = np.array(G3.get('Cells Minimum Elevation'))
nrows1, ncols1 = np.shape(wse)
nrows2, ncols2 = np.shape(coord)
nrows3 = np.shape(min_elev)

#mettere riferimenti nel file Excel di output
output = open(r"C:\Users\Win-10\Desktop\mappa_statica
\output_completo.txt","a")

#estrazione T_Po e T_Ch dal file contenente gli idrogrammi
T_Po = foglio_input.cell(2,i+(s*num_T_Po)).value
T_Ch = foglio2_input.cell(2,v+(s*num_T_Ch)).value

#scrittura espressione per riconoscere le varie simulazioni
output.write("[")
output.write(str(s+1))
output.write(", ")
output.write("'Po', ")
output.write(str(T_Po))
output.write(", ")
output.write("'Ch', ")
output.write(str(T_Ch))
output.write("]")
output.write("\n")
```

```
#ricavo il max di wse per ogni cella e salvo in vettore
num_id = 0
for w in range (0,ncols1):
    flag = 0
    for h in range (0,nrows1):
        val_wse = wse[h,w]
        if val_wse > flag:
            max_wse = val_wse
            time = h
            flag = max_wse
        elif flag == 0:
            max_wse=0

    depth = max_wse - min_elev[w]

    #scrittura su file di output
    output.write(str(num_id))
    output.write(", ")
    output.write(str(coord[w,0]))
    output.write(", ")
    output.write(str(coord[w,1]))
    output.write(", ")
    output.write(str(max_wse))
    output.write(", ")
    output.write(str(min_elev[w]))
    output.write(", ")
    output.write(str(depth))
    output.write("\n")
    num_id += 1


m+=1
output.close()
```

Allegato II

Estrazione evento simulato per cui costruire la mappa statica

```
##INSERIMENTNO MODULI
import h5py                             #per lavorare con i file HDF
import numpy                            #per lavorare con vettori e matrici

##INSERIMENTO FUNZIONE PER RICERCRARE LA GIUSTA POSIZIONE DELLO SCENARIO
DA ESTRARRE
def ricerca_elemento(file,elem):
    riga=" "
    count=0
    while(riga):
        riga=file.readline()
        if elem in riga:
                return count
        count+=1

##PARTE PER SCEGLIERE LO SCENARIO DI INTERESSE
#iniz. variabili
num_scenari = 3
vet_scenari = np.arange(1,num_scenari,1+1)
tempi_ritorno = [100, 200]
tempi_ritorno_Po = [100, 200]
tempi_ritorno_Ch = [100, 200]

#apertura file HDF e ricerca cartella con dati di WSE
with h5py.File(r"C:\Users\Win-10\Desktop\mappa_statica
        \SIMULAZIONE2.p05.hdf","r") as hdf:
    G1 = hdf.get('Results/Unsteady/Output/Output Blocks/Base
        Output/Unsteady
    Time Series/2D Flow Areas/2d area')
    water_surf = np.array(G1.get('Water Surface'))

#variabili utili successivamente
num_righe, num_col = np.shape(water_surf)

#inserimento scenario con verifica
print("Digitare uno dei seguenti valori in base allo scenario di
interesse:")
print("1 - Onde di piena del Po e Chisola coincidenti")
```

```
print("2 - Onda di piena del Po prima dell'onda di piena del Chisola")
print("3 - Onda di piena del Po dopo l'onda di piena del Chisola")
scenario = int(input("scenario n: "))

flag = 1
while flag != 0:
    for s in range (0, len(vet_scenari)):
        if scenario == vet_scenari[s]:
            flag = 0
        else: continue
    if flag == 1:
        print("Errore, ridigitare valore")
        scenario = int(input("scenario n: "))

#inserimento tempo di ritorno Po con verifica
print("Inserire T_Po (100, 200)")
T_Po = int(input("T_Po: "))

flag = 1
while flag != 0:
    for s in range (0, len(tempi_ritorno)):
        if T_Po == tempi_ritorno[s]:
            flag = 0
        else: continue
    if flag == 1:
        print("Errore, ridigitare valore")
        T_Po = int(input("T_Po: "))

#inserimento tempo di ritorno Chisola con verifica
print("Inserire T_Chisola (100, 200)")
T_Ch = int(input("T_Ch: "))

flag = 1
while flag != 0:
    for t in range (0, len(tempi_ritorno)):
        if T_Ch == tempi_ritorno[t]:
            flag = 0
        else: continue
    if flag == 1:
        print("Errore, ridigitare valore")
        T_Ch = int(input("T_Ch: "))
```

```
#apertura file txt contenente l'output delle simualzioni
output_generale = open(r"C:\Users\Win-10\Desktop\mappa_statica
    \output_completo.txt","r")

#ricerca posizione stringa da cui iniziare ad estrarre l'evento simulato
che si vuole estrarre
elem = str([scenario, "Po", T_Po, "Ch", T_Ch])
count = ricerca_elemento(output_generale, elem)

#apertura file txt in cui estraggo solo lo scenario di interesse
output_gis = open(r"C:\Users\Win-10\Desktop\mappa_statica
       \output_gis.txt","w")
n=0
#puntatore ad inizio file
output_generale.seek(0)
#condizione iterativa in cui avviene l'estrazione
for riga in output_generale:
       #confronto varibili per capire se si è nel intervallo corretto
    if count< n <=count+num_col:
        output_gis.write(riga)
    n=n+1    #variabile contatore riferita alle righe del file generale

#chiusura file
output_gis.close()
output_generale.close()
```

Allegato III

Esecuzione simulazione per estrazione sequenza temporale

```
##INSERIMENTO MODULI
from openpyxl import load_workbook      #per leggere dati da excel
import numpy as np                      #per poter lavorare con vettori
import os                               #per poter lavorare con file
import win32com.client                  #per usare l' HEC-RAS Controller
import h5py                             #per lavorare con file HDF
import math                             #per usare funzioni matematiche

##CREAZIONE FUNZIONE PER RICERCARE PAROLE O FRASI ALL'INTERNO DEI FILE
#inserimento nome funzione e relativi argomenti
def ricerca_elemento(file,elem):
    #inizializzazione variabili
    riga=" "
    count=0
    count2=0
    n=0
    #condizione ciclica per scandire le righe del file
    while(riga):
        riga=file.readline()
        #condizione di ricerca
        if elem in riga:
            n+=1
            if n==1:
                return count
            elif n==2:
                return count2
        count+=1
        count2+=1

##INIZIO PROGRAMMA
#variabile hec per usare le funzioni del HEC-RAS Control
hec = win32com.client.Dispatch("RAS507.HECRASController")
#caricamento del percorso del progetto
RASProject = os.path.join(os.getcwd(),r'C:\Users\Win-
      10\Desktop\mappa_statica\SIMULAZIONE2.prj')

#caricamento del file contenente tutti gli idrogrammi
cartella_input = load_workbook(filename="idrogrammi_Po.xlsx")
cartella2_input = load_workbook(filename="idrogrammi_Chisola.xlsx")
```

```
#individuazione foglio Excel contenente gli idrogrammi
foglio_input = cartella_input.active
foglio2_input = cartella2_input.active

#PARTE PER SCEGLIERE L'EVENTO DA SIMULARE
#variabili necessarie per il controllo succesivo
num_scenari = 3
vet_scenari = np.arange(1,num_scenari,1+1)
tempi_ritorno = [100, 200]

#inserimento scenario con verifica
print("Digitare uno dei seguenti valori in base allo scenario di
interesse:")
print("1 - Onde di piena del Po e Chisola coincidenti")
print("2 - Onda di piena del Po prima dell'onda di piena del Chisola")
print("3 - Onda di piena del Po dopo l'onda di piena del Po")

scenario = int(input("scenario n: "))

flag = 1
while flag != 0:
    for s in range (0, len(vet_scenari)):
        if scenario == vet_scenari[s]:
            flag = 0
        else: continue
    if flag == 1:
        print("Errore, ridigitare valore")
        scenario = int(input("scenario n: "))

#inserimento tempo di ritorno Po con verifica
print("Inserire T_Po (100, 200 )")
T_Po = int(input("T_Po: "))

flag = 1
while flag != 0:
    for s in range (0, len(tempi_ritorno)):
        if T_Po == tempi_ritorno[s]:
            flag = 0
        else: continue
    if flag == 1:
        print("Errore, ridigitare valore")
        T_Po = int(input("T_Po: "))
#inserimento tempo di ritorno Chisola con verifica
print("Inserire T_Chisola (100, 200 )")
T_Ch = int(input("T_Ch: "))
```

```
flag = 1
while flag != 0:
    for t in range (0, len(tempi_ritorno)):
        if T_Ch == tempi_ritorno[t]:
            flag = 0
        else: continue
    if flag == 1:
        print("Errore, ridigitare valore")
        T_Ch = int(input("T_Ch: "))

#determinazione posizione scenario in file Excel (1 e 2)
n=1
flag=0
while flag == 0:
    if foglio_input.cell(1,n).value != scenario:
        n += 1
    else:
        col_scenario = n
        flag = 1

#determinazione posizione della colonna del T_Po
n=col_scenario
flag=0
while flag == 0:
    if foglio_input.cell(2,n).value != T_Po:
        n += 1
    else:
        col_T_Po = n
        flag = 1

#determinazione posizione della colonna del T_Ch
n=col_scenario
flag=0
while flag == 0:
    if foglio2_input.cell(2,n).value != T_Ch:
        n += 1
    else:
        col_T_Ch = n
        flag = 1


#determinazione lunghezza idrogramma
w=0
num_righe=0
```

```
j=0
while w==0:
    if foglio_input.cell(j+2,1).value != None:
        num_righe+=1
        j+=1
    else:
        w=1

#numero di righe occupate dall'idrogramma ne file ".u"
num_lines = math.ceil(num_righe/10)

#SALVATAGGIO IDROGRAMMI SCELTI IN VETTORI
z=0
#definisco il vettore per salvare l'idrogramma del Po
vet = np.empty([num_righe], dtype=float)
#definisco il vettore per salvare l'idrogramma del Chisola
vet2 = np.empty([num_righe], dtype=float)
#entro nel file excel e scansiono riga per riga l'idrogramma
for j in range(1,num_righe+1):
    #estraggo il valore (Po) e lo salvo nel vettore
    vet[z] = foglio_input.cell(j+2, col_T_Po).value
    #estraggo il valore (Ch) e lo salvo nel vettore
    vet2[z] = foglio2_input.cell(j+2, col_T_Ch).value
    z+=1
#trasformo i vettori in matrici num_line x 10
vet.resize(num_lines, 10)
vet2.resize(num_lines, 10)
#stampo a video
print(vet)
print(vet2)

##CERCARE FLOW HYDRO. IN FILE CON FUNZIONE
file_in= open (r"C:\Users\Win-
10\Desktop\seq_temporale\copia_input.txt","r")
elem = "Flow Hydrograph="
#riga nel file ".u" in cui si trova il primo Flow Hydrograph
count = ricerca_elemento(file_in, elem)
#riga nel file ".u" in cui si trova il secondo Flow Hydrograpyh
count2 = ricerca_elemento(file_in, elem)
file_in.close()


#MODIFICARE IL FILE CONTENENTE L'IDROGRAMMA
#apertura file per copiare le righe che non cambiano
```

```
file_in= open (r"C:\Users\Win-
10\Desktop\seq_temporale\copia_input.txt","r")
#apertura file che verrà aggiornato
file_out= open (r"C:\Users\Win-10\Desktop\seq_temporale
    \SIMULAZIONE2.u02","w")
#contatore per il numero di righe del file
n=0
#contatore per le righe della matrice 1 da inserire nel file
b=0
#contatore per le righe della matrice 2 da inserire nel file
y=0
#scansiono riga per riga il file_in
for riga in file_in:
    #se sono nel range di righe da cambiare
    if count+1 <= n <= count+num_lines:
        #ciclo per le colonne della matrice
        for k in range (0,10):
            #salvo il singolo valore del vettore in una variabile
            a = vet[b,k]
            #valuto il numero di cifre di cui è composto
            cifre = len(str(a))
            #se a ha 1 cifra aggiungo 7 spazi vuoti
            if cifre == 1:
                file_out.write("       "+str(a))
            #se a ha 2 cifre aggiungo 6 spazi vuoti
            elif cifre == 2:
                file_out.write("      "+str(a))
            #...3 ....5
            elif cifre == 3:
                file_out.write("     "+str(a))
            #...4 ....4
            elif cifre == 4:
                file_out.write("    "+str(a))
            #...5 ....3
            elif cifre == 5:
                file_out.write("   "+str(a))
            #...6 ....2
            elif cifre == 6:
                file_out.write("  "+str(a))
            #...7 ....1
            elif cifre == 7:
                file_out.write(" "+str(a))
        #vado a capo dopo aver inserito una riga
        file_out.write("\n")
        b+=1
```

```
            n+=1
        #ripeto operazione per il secondo idrogramma
    elif count+count2+2 <= n <= count+count2+num_lines+1:
            for k in range (0,10):
                a = vet2[y,k]
                cifre = len(str(a))
                if cifre == 1:
                    file_out.write("       "+str(a))
                elif cifre == 2:
                    file_out.write("      "+str(a))
                elif cifre == 3:
                    file_out.write("     "+str(a))
                elif cifre == 4:
                    file_out.write("    "+str(a))
                elif cifre == 5:
                    file_out.write("   "+str(a))
                elif cifre == 6:
                    file_out.write("  "+str(a))
                elif cifre == 7:
                    file_out.write(" "+str(a))
            file_out.write("\n")
            y+=1
            n+=1
    else:
            #inserisco le altre righe che non necessitano di modifiche
            file_out.write(riga)
            n+=1

file_in.close()
file_out.close()
z=0

##ESEGUIRE LA SIMULAZIONE IN HECRAS
#apro il progetto
hec.Project_Open(RASProject)
#variabili da inizializzare
NMsg,TabMsg,block = None,None,True
#mostra la finestra principale di HEC-RAS
hec.ShowRas()
#mostra la finestra in cui si vede l'avanzamento delle simulazioni
hec.Compute_ShowComputationWindow()
#fa partire la simulazione
v1,NMsg,TabMsg,v2 = hec.Compute_CurrentPlan(NMsg,TabMsg,block)
##APRO FILE HDF E ESTRAGGO GLI OUTPUT DI INTERESSE
##apertura file HDF e ricerca cartelle con dati di interesse
```

```
with h5py.File(r"C:\Users\Win-10\Desktop\seq_temporale
      \SIMULAZIONE2.p01.hdf","r") as hdf:
    G1 = hdf.get('Results/Unsteady/Output/Output Blocks/Base
      Output/Unsteady Time Series/2D Flow Areas/2d area')
    wse = np.array(G1.get('Water Surface'))
    G2 = hdf.get('Geometry/2D Flow Areas/2d area')
    min_elev = np.array(G2.get('Cells Minimum Elevation'))
    G3 = hdf.get('Geometry/2D Flow Areas/2d area')
    coord = np.array(G3.get('Cells Center Coordinate'))
nrows1, ncols1 = np.shape(wse)
nrows2 = np.shape(min_elev)
nrows3, ncols3 = np.shape(coord)

#apertura file per salvare i dati di output
output = open(r"C:\Users\Win-10\Desktop\seq_temporale
      \output_completo.txt","w")
num_id = 0
delta_t = 0
#cicli per scandire la matrice contenenti wse per vari istanti temporali
for i in range(0,ncols1):
    for j in range(0,nrows1):
        #scrittura su file
        output.write(str(num_id))
        output.write(", ")
        output.write(str(wse[j,i]))
        output.write(", ")
        output.write(str(min_elev[i]))
        output.write(", ")
        depth = wse[j,i] - min_elev[i]
        output.write(str(depth))
        output.write(", ")
        output.write(str(delta_t))
        output.write("\n")
        #dati output ogni 30 min
        delta_t += 0.5
    delta_t = 0
    num_id +=1

output.close()
```

Allegato IV

Estrazione sequenza temporale per determinate celle

```
#INSERIMENTO MODULI
import numpy as np
import h5py
import math

#inizializzazione vettori
num_id = []
lon = []
lat = []
wse = []
min_elev = []
depth = []
delta_t = []

#apertura file contenente la sequenza per tutte le celle
output_generale = open(r"C:\Users\Win-10\Desktop\seq_temporale
      \output_completo.txt","r")

#separazione grandezze e salvataggio in vettori
cont=0
for riga in output_generale:
    splitLine = riga.split(", ")
    num_id.append(str(splitLine[0]))
    wse.append(str(splitLine[3]))
    min_elev.append(str(splitLine[4]))
    depth.append(str(splitLine[5]))
    delta_t.append(str(splitLine[6]))
    cont+=1

#file txt contenente l'output delle sequenze volute
output_seq_temp = open(r"C:\Users\Win-10\Desktop\seq_temporale
      \output_seq_temp.txt","w")

#inserimento manuale della cella o delle celle per cui interessa estrarre
la sequenza temporale
print("Se si vuole estrarre un solo valore digitare 0")
print("Se si vuole estrarre un range di valori digitare 1")
caso = int(input("Caso (0 o 1): "))
```

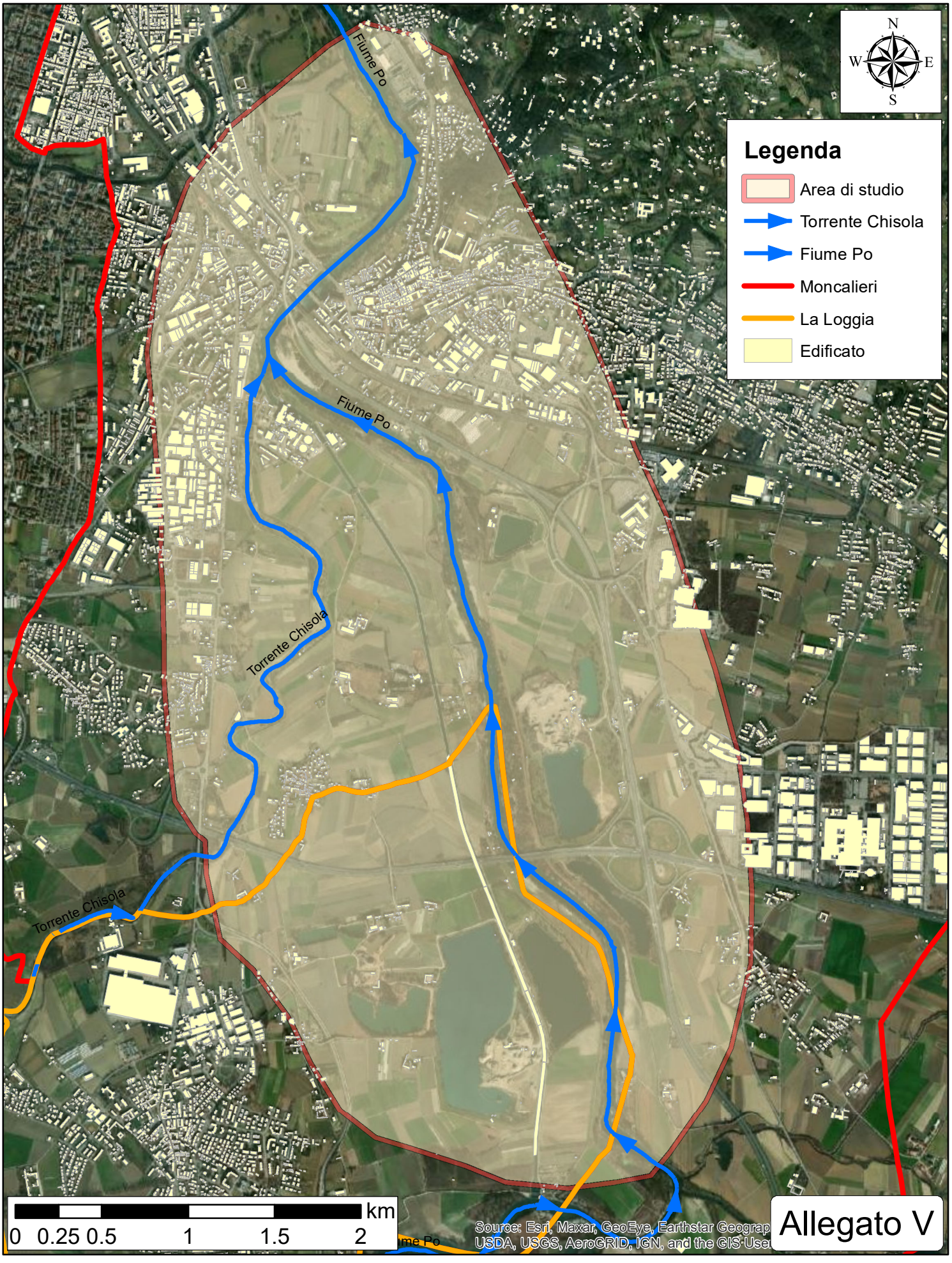
```
#condizione ciclica in cui si effettua l'estrazione in base al caso
scelto oppure si reinserisce il valore se inserito erratamente
flag = 1
while flag != 0:
      #se il caso è 0 inserire il valore del punto
    if caso == 0:
        print("Inserire il numero del punto per cui si vuol estrarre la
            sequenza temporale.")
        punto = str(input("Punto/i: "))
        #ricerca del punto ed estrazione
        for j in range(0,cont):
            if punto == num_id[j]:
                output_seq_temp.write(str(num_id[j]))
                output_seq_temp.write(", ")
                wse_def = round(float(wse[j]),2)
                output_seq_temp.write(str(wse_def))
                output_seq_temp.write(", ")
                min_elev_def = round(float(min_elev[j]),2)
                output_seq_temp.write(str(min_elev_def))
                output_seq_temp.write(", ")
                depth = float(wse[j]) - float(min_elev[j])
                depth_def = str(round(depth,2))
                output_seq_temp.write(str(depth_def))
                output_seq_temp.write(", ")
                output_seq_temp.write(str(delta_t[j]))
            else:
                continue
        flag = 0
      #se il caso è 1 inserire i due valori (min e max)
    elif caso == 1:
        print("Inserire il min (Punto 1) e il max (Punto 2)")
        punto_1 = int(input("Punto 1: "))
        punto_2 = int(input("Punto 2: "))
        #generazione vettore di punti da estrarre
        vet_punti = np.arange(punto_1,punto_2+1)
        for i in range(0,len(vet_punti)):
            for j in range(0,cont):
                if str(vet_punti[i]) == num_id[j]:
                    output_seq_temp.write(str(num_id[j]))
                    output_seq_temp.write(", ")
                    wse_def = round(float(wse[j]),2)
                    output_seq_temp.write(str(wse_def))
                    output_seq_temp.write(", ")
                    min_elev_def = round(float(min_elev[j]),2)
                    output_seq_temp.write(str(min_elev_def))
```

```
                    output_seq_temp.write(", ")
                    depth = float(wse[j]) - float(min_elev[j])
                    depth_def = str(round(depth,2)
                    output_seq_temp.write(str(depth_def))
                    output_seq_temp.write(", ")
                    output_seq_temp.write(str(delta_t[j]))
        flag = 0
      #reinserimento valore se errato
    elif flag==1:
        caso = int(input("Errore, ridigitare valore (0 o 1): "))

#chiusura file
output_generale.close()
output_seq_temp.close()
```

Allegati – Elaborazioni grafiche

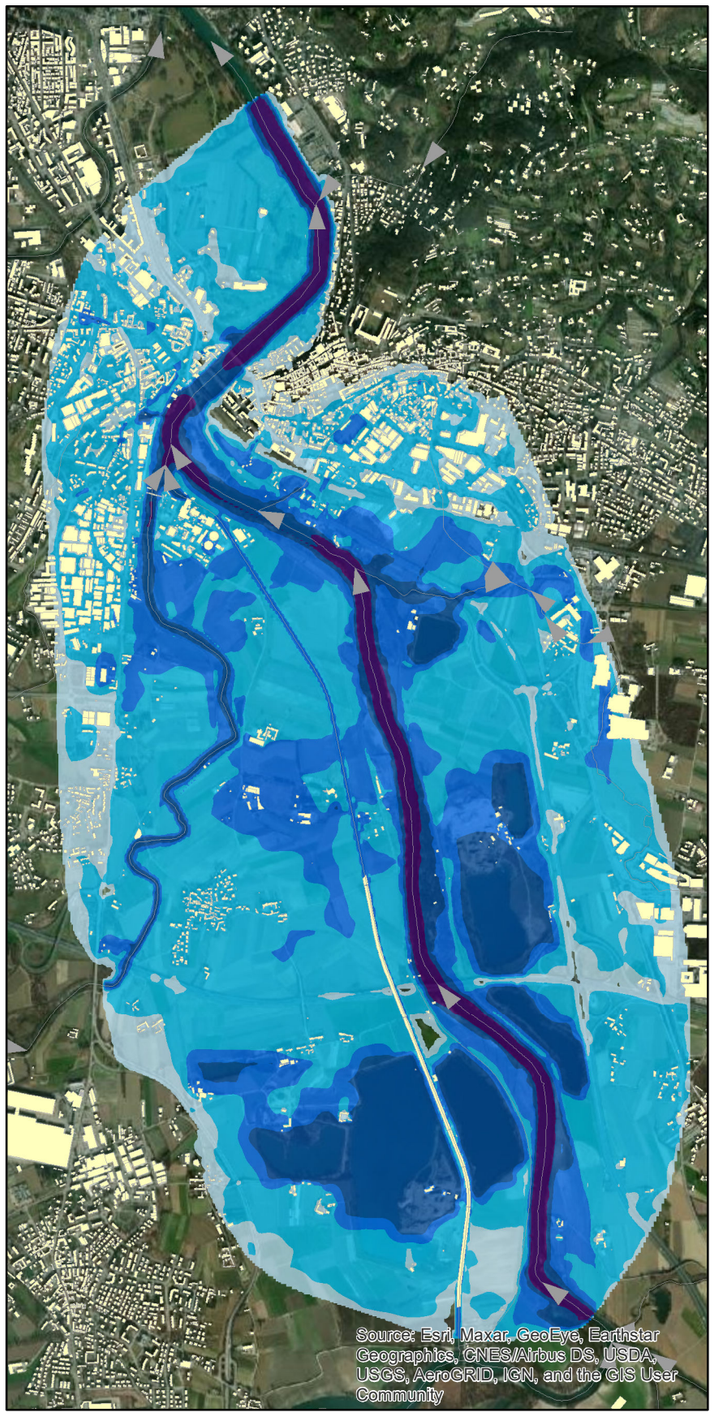
Area oggetto di studio

Comuni di Moncalieri e La Loggia

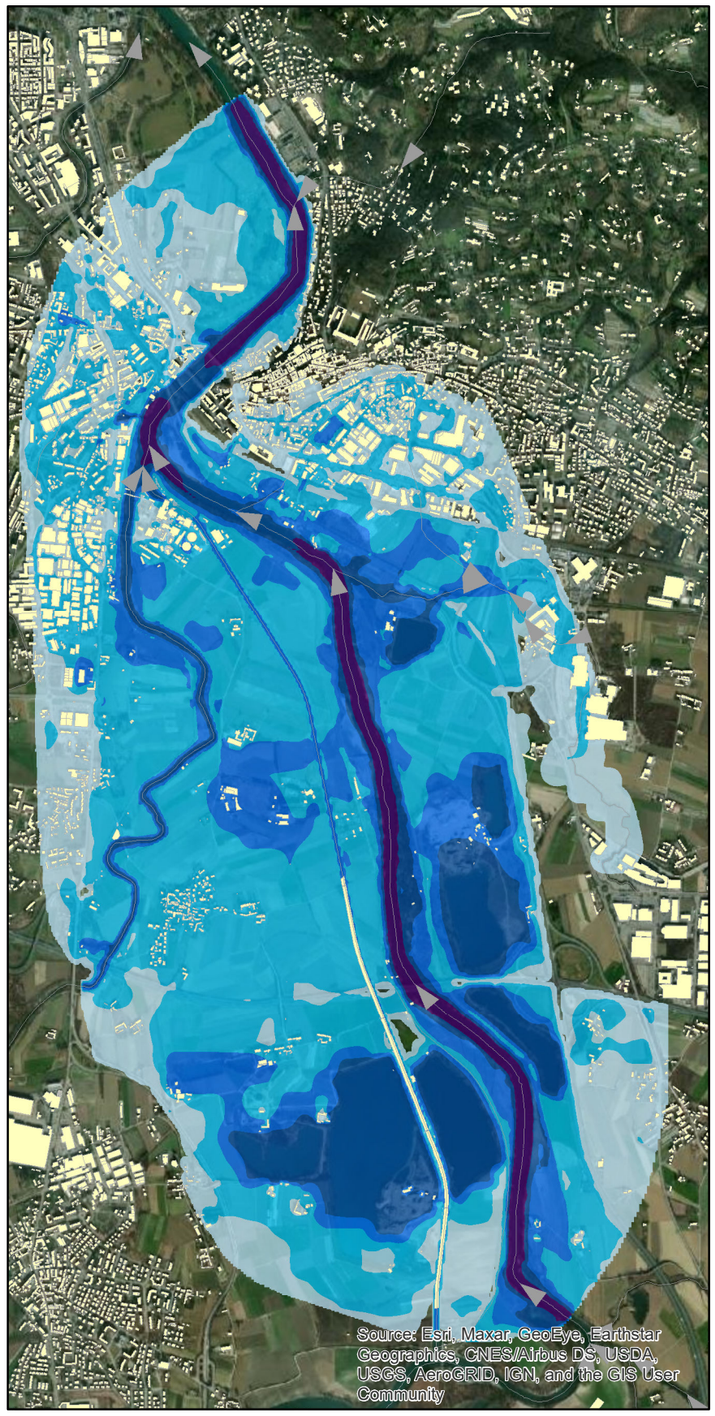


Evento simulato: T Po = 100 anni T Chisola = 100 anni

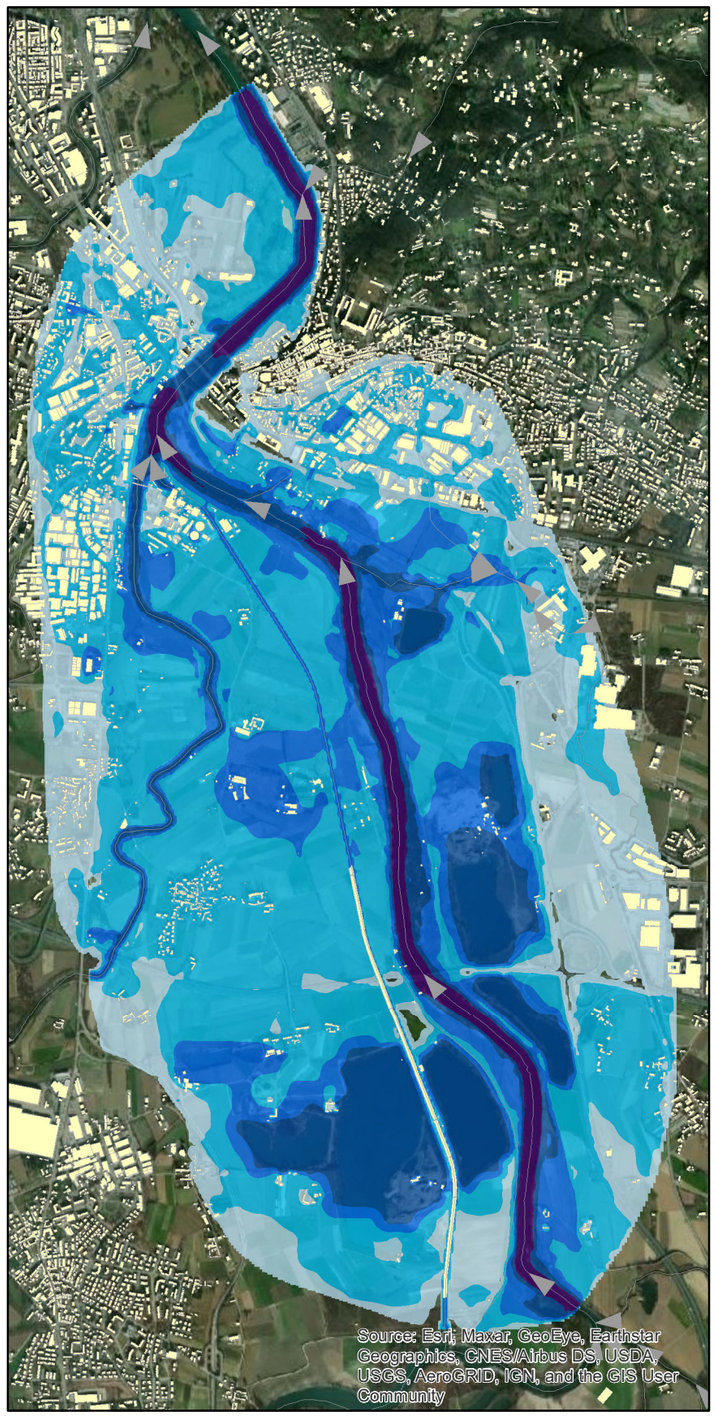
Onde di piena coincidenti



Onda di piena non coincidenti, Po prima del Chisola



Onda di piena non coincidenti, Chisola prima del Po



N
W E
S

Legenda

Ret. idrografico

Edificato

Aree allagabili (m)

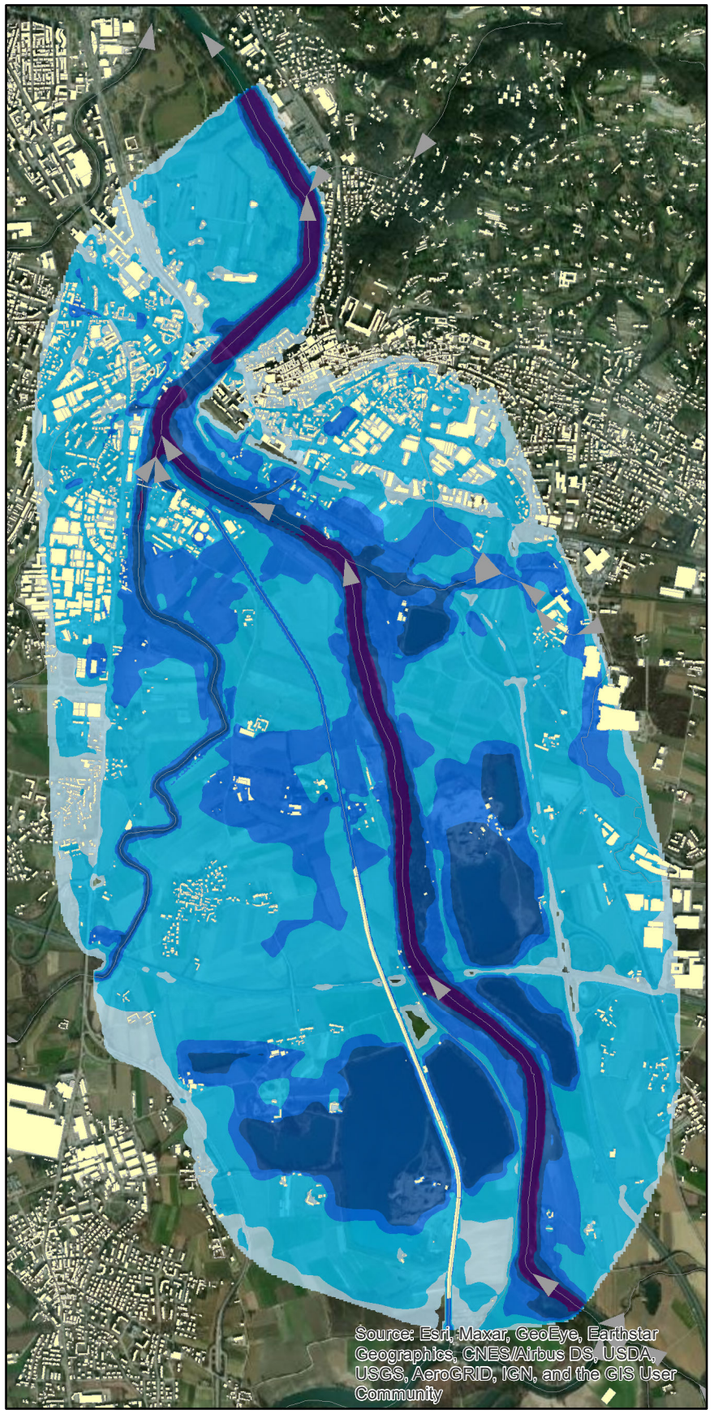
- 0 - 0
- 0.1 - 1.4
- 1.5 - 3.3
- 3.4 - 5.9
- 6 - 9
- 9.1 - 13.6

0 0.5 1 2 3 4 5 km

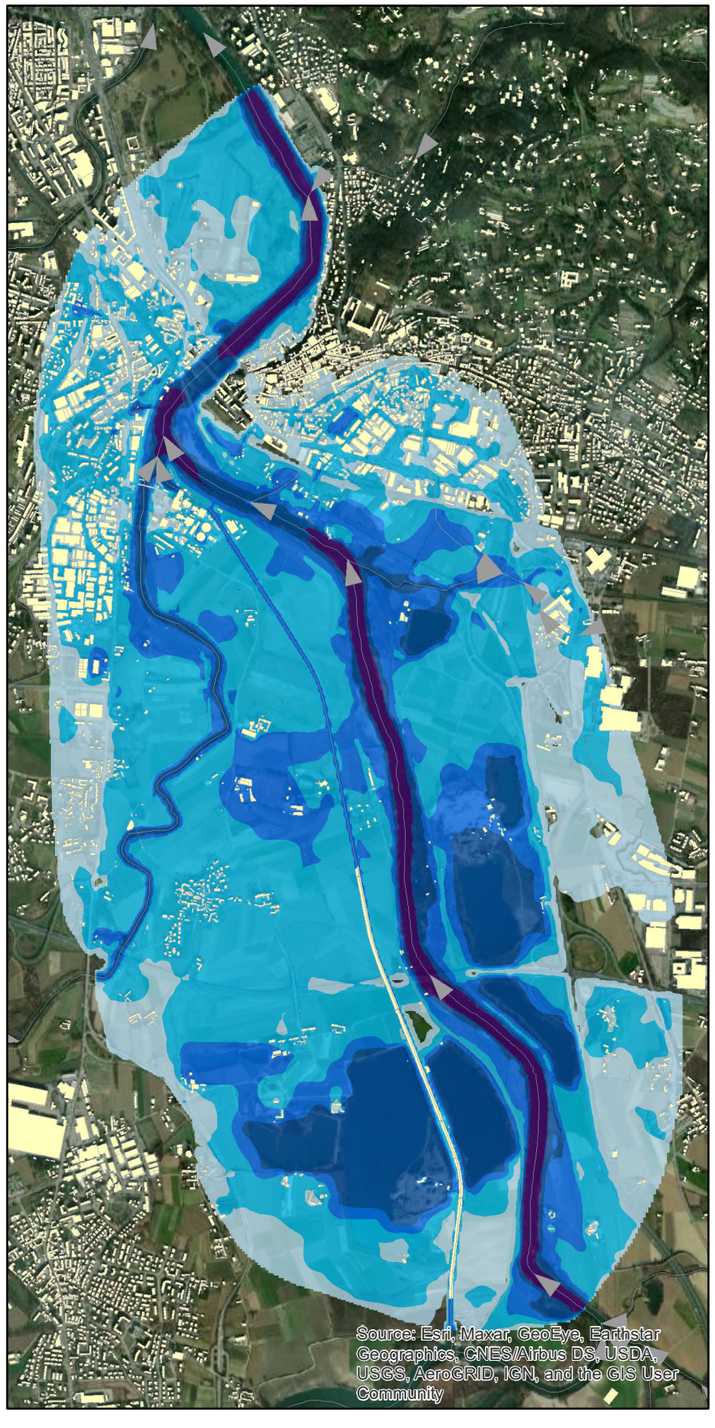
Allegato VI

Evento simulato: T Po = 100 anni T Chisola = 200 anni

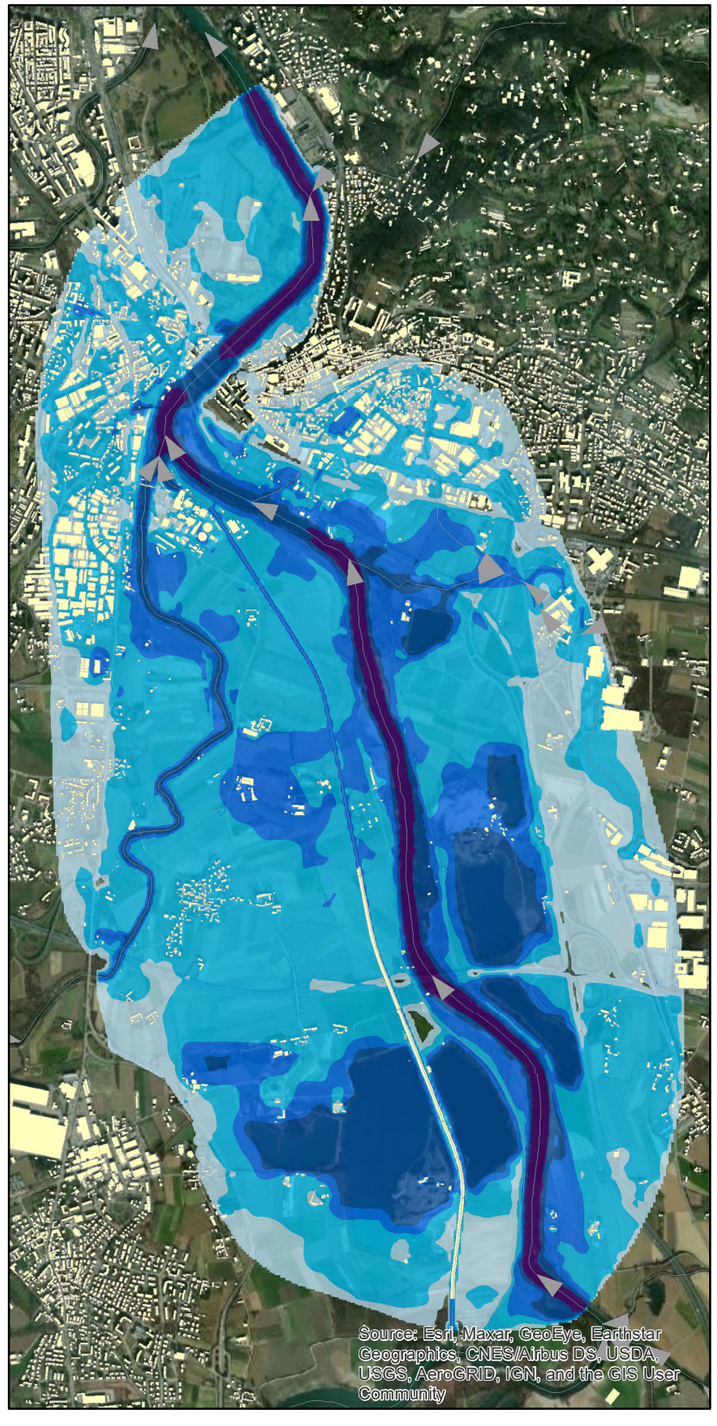
Onde di piena coincidenti

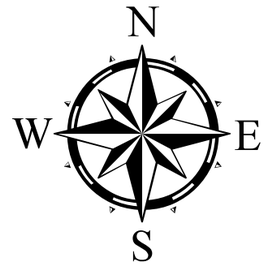


Onda di piena non coincidenti, Po prima del Chisola



Onda di piena non coincidenti, Chisola prima del Po





Legenda

Ret. idrografico

Edificato

Aree allagabili

(m)

- 0 - 0
- 0.1 - 1.4
- 1.5 - 3.3
- 3.4 - 5.9
- 6 - 9
- 9.1 - 13.7

0 0.5 1 2 3 4 5 km

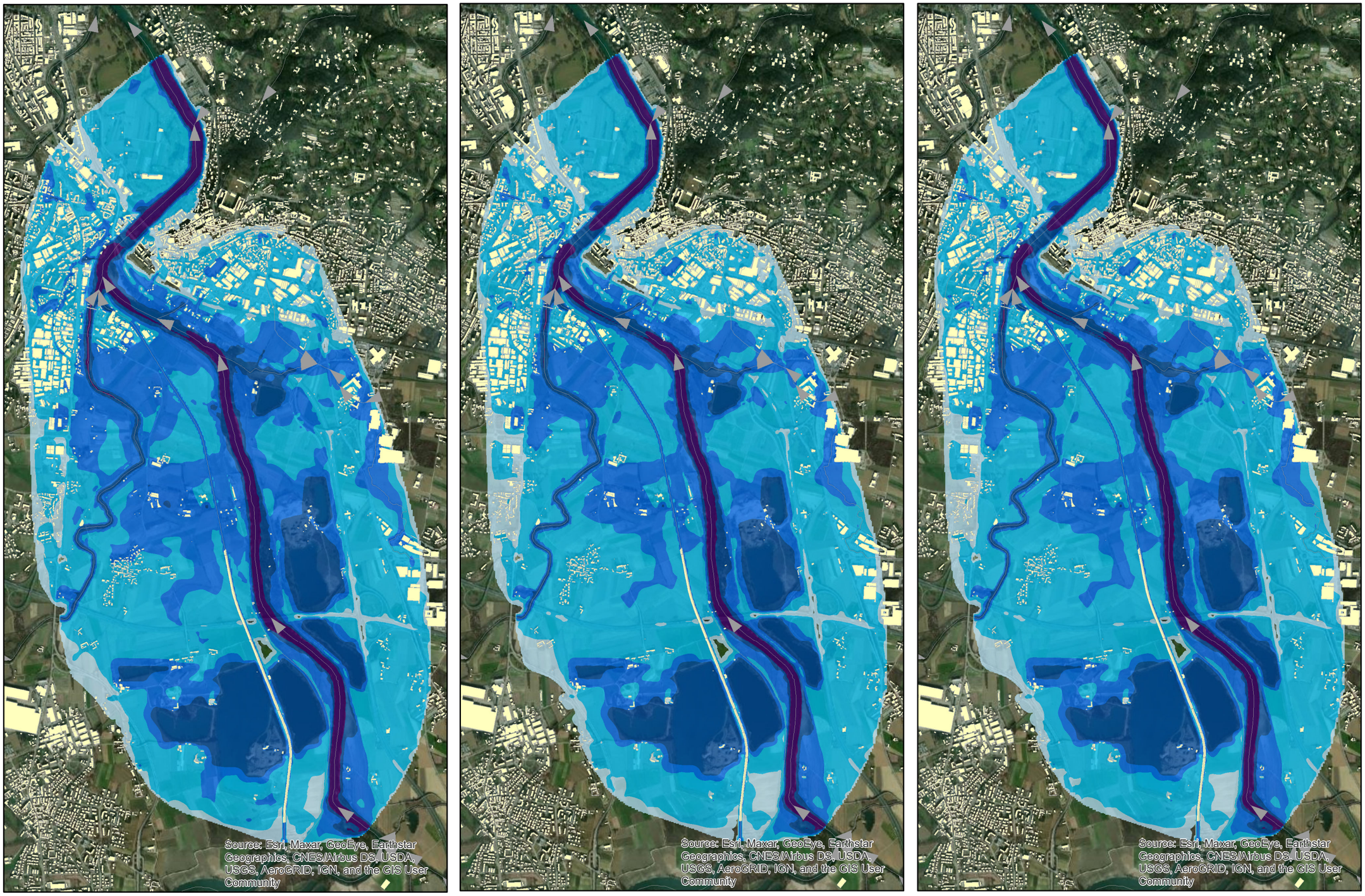
Allegato VII

Evento simulato: T Po = 200 anni T Chisola = 100 anni

Onde di piena coincidenti

Onda di piena non coincidenti, Po prima del Chisola

Onda di piena non coincidenti, Chisola prima del Po



N
W E
S

Legenda

Ret. idrografico

Edificato

Aree allagate

(m)

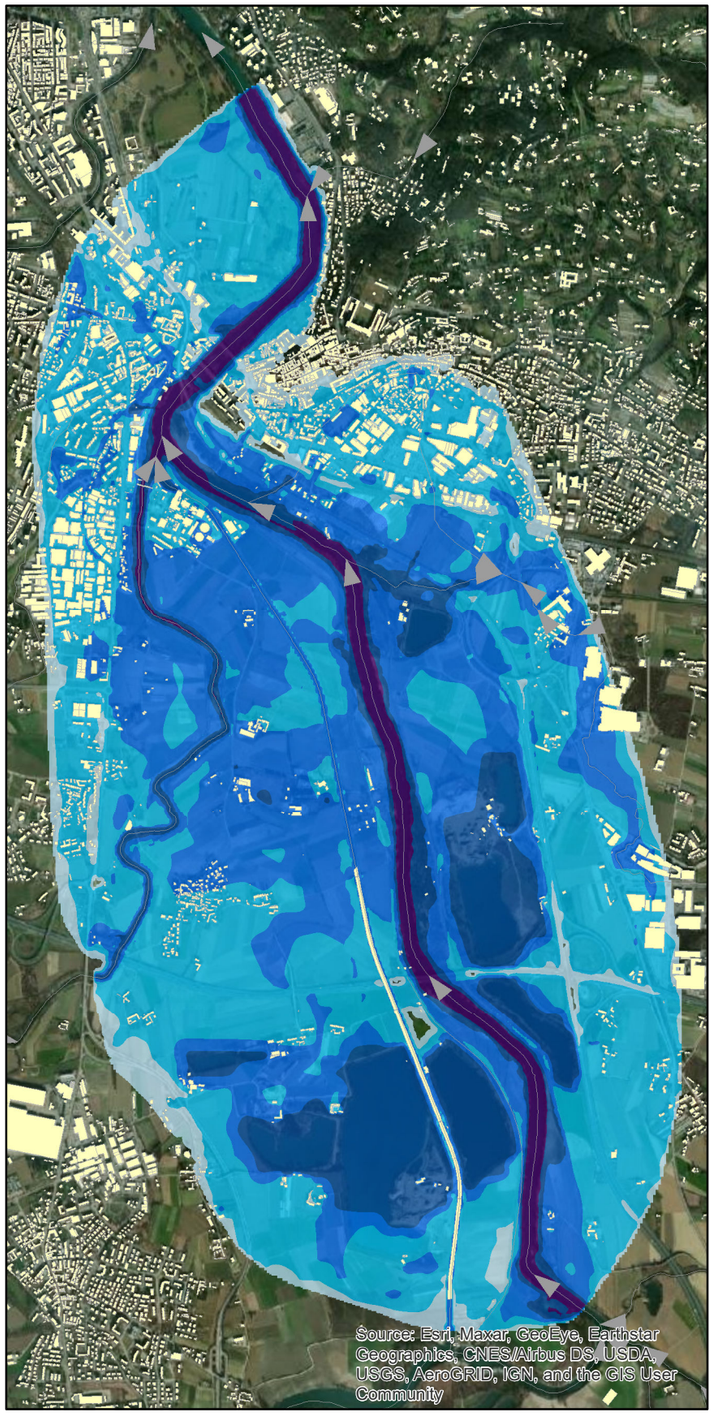
- 0 - 0
- 0.1 - 1.4
- 1.5 - 3.3
- 3.4 - 5.9
- 6 - 9
- 9.1 - 13.8

0 0.5 1 2 3 4 5 km

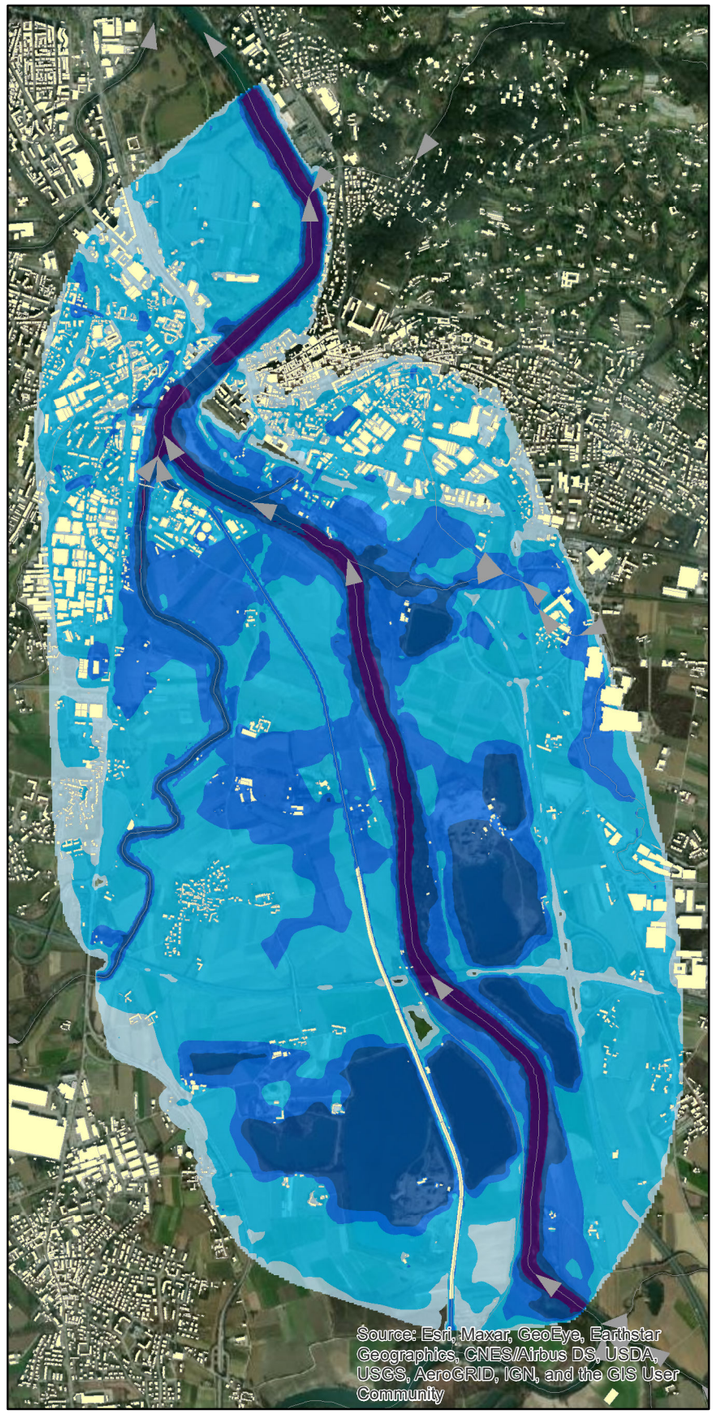
Allegato VIII

Evento simulato: T Po = 200 anni T Chisola = 200 anni

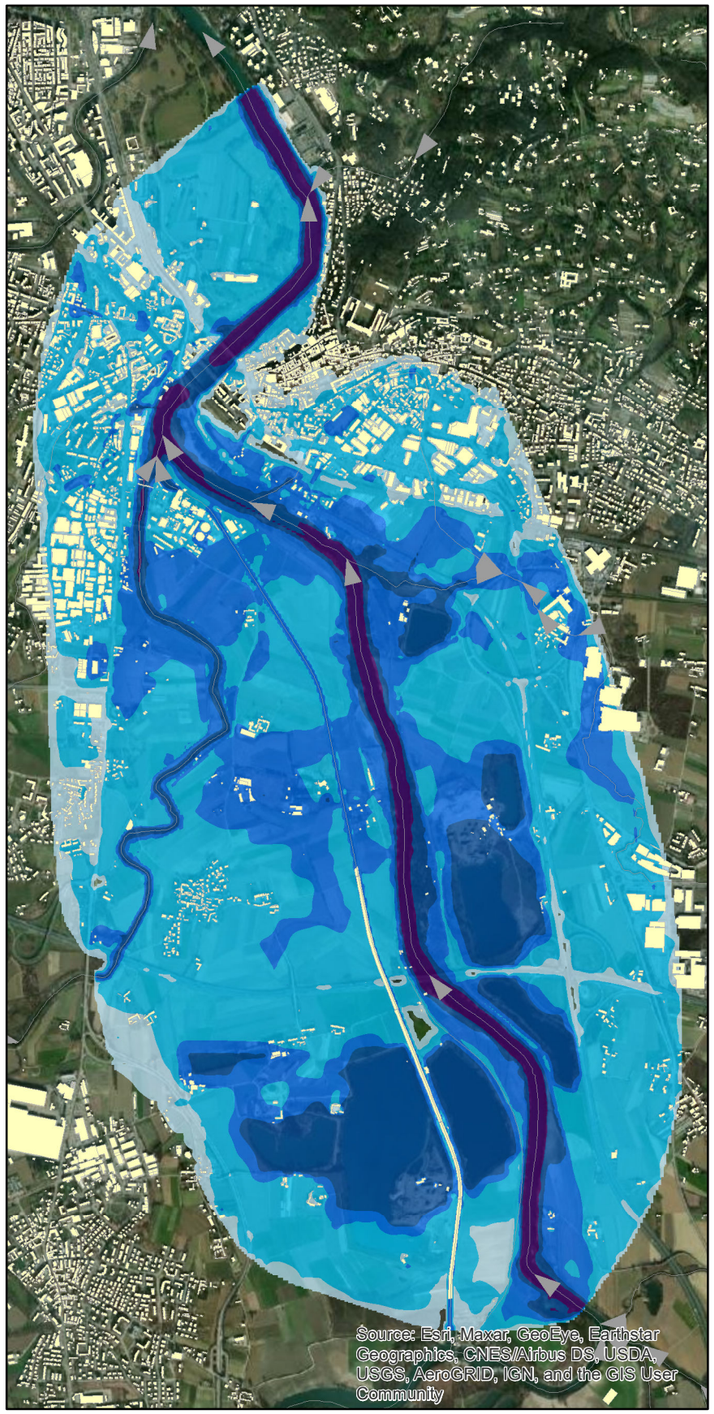
Onde di piena coincidenti



Onda di piena non coincidenti, Po prima del Chisola



Onda di piena non coincidenti, Chisola prima del Po



Legenda

Ret. idrografico

Edificato

Aree allagate

(m)

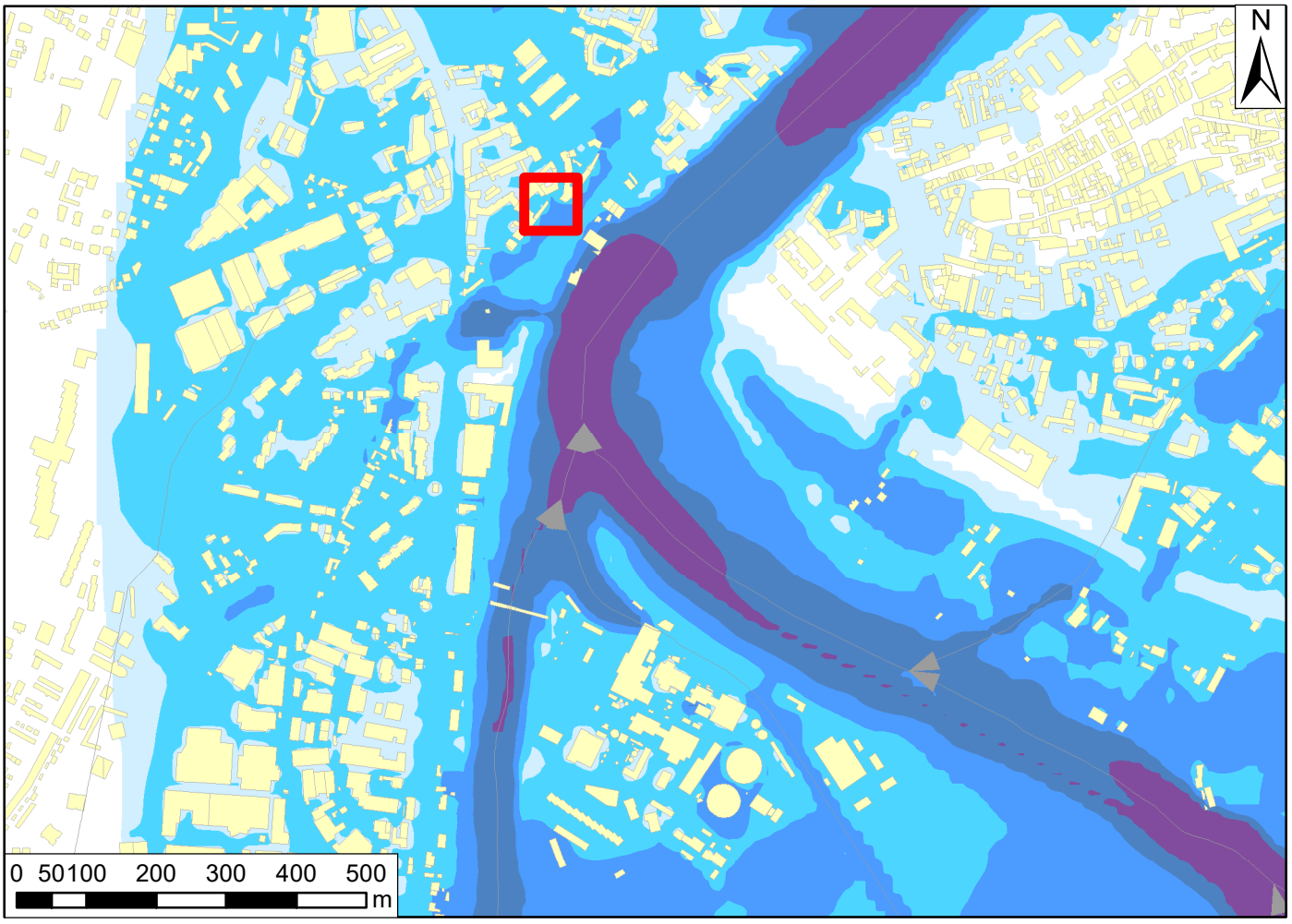
- 0 - 0
- 0.1 - 1.4
- 1.5 - 3.3
- 3.4 - 5.9
- 6 - 9
- 9.1 - 13.9

0 0.5 1 2 3 4 5 km

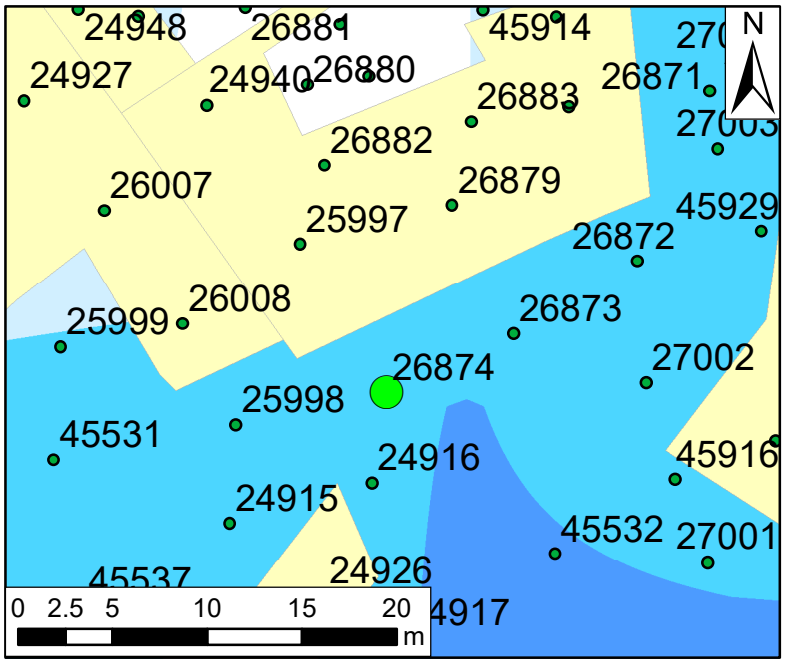
Allegato IX

Estrazione sequenza temporale

Evento con T Po 200 , T Ch 200 e onde di piena coincidenti



Zoom riquadro rosso



Legenda

- Punto di interesse
- Punti (centro celle)
- Ret. idrografico
- Edificato

Aree allagabili

(m)

- 0 - 0
- 0.1 - 1.4
- 1.5 - 3.3
- 3.4 - 5.9
- 6 - 9
- 9.1 - 13.6

Allegato X

Punto	WSE (m s.l.m)	Quota terreno (m s.l.m)	Tirante (m)	Tempo (h)	Punto	WSE (m s.l.m)	Quota terreno (m s.l.m)	Tirante (m)	Tempo (h)
26874	219.94	219.94	0	0	26874	219.94	219.94	0	23.5
26874	219.94	219.94	0	0.5	26874	219.94	219.94	0	24
26874	219.94	219.94	0	1	26874	219.94	219.94	0	24.5
26874	219.94	219.94	0	1.5	26874	219.94	219.94	0	25
26874	219.94	219.94	0	2	26874	219.94	219.94	0	25.5
26874	219.94	219.94	0	2.5	26874	219.94	219.94	0	26
26874	219.94	219.94	0	3	26874	219.94	219.94	0	26.5
26874	219.94	219.94	0	3.5	26874	220.09	219.94	0.15	27
26874	219.94	219.94	0	4	26874	220.38	219.94	0.44	27.5
26874	219.94	219.94	0	4.5	26874	220.6	219.94	0.66	28
26874	219.94	219.94	0	5	26874	220.72	219.94	0.78	28.5
26874	219.94	219.94	0	5.5	26874	220.82	219.94	0.88	29
26874	219.94	219.94	0	6	26874	220.95	219.94	1.01	29.5
26874	219.94	219.94	0	6.5	26874	221.1	219.94	1.16	30
26874	219.94	219.94	0	7	26874	221.22	219.94	1.28	30.5
26874	219.94	219.94	0	7.5	26874	221.31	219.94	1.37	31
26874	219.94	219.94	0	8	26874	221.36	219.94	1.42	31.5
26874	219.94	219.94	0	8.5	26874	221.38	219.94	1.44	32
26874	219.94	219.94	0	9	26874	221.42	219.94	1.49	32.5
26874	219.94	219.94	0	9.5	26874	221.48	219.94	1.54	33
26874	219.94	219.94	0	10	26874	221.53	219.94	1.59	33.5
26874	219.94	219.94	0	10.5	26874	221.57	219.94	1.63	34
26874	219.94	219.94	0	11	26874	221.61	219.94	1.67	34.5
26874	219.94	219.94	0	11.5	26874	221.64	219.94	1.7	35
26874	219.94	219.94	0	12	26874	221.69	219.94	1.75	35.5
26874	219.94	219.94	0	12.5	26874	221.75	219.94	1.81	36
26874	219.94	219.94	0	13	26874	221.8	219.94	1.86	36.5
26874	219.94	219.94	0	13.5	26874	221.86	219.94	1.92	37
26874	219.94	219.94	0	14	26874	221.93	219.94	1.99	37.5
26874	219.94	219.94	0	14.5	26874	221.98	219.94	2.04	38
26874	219.94	219.94	0	15	26874	222.05	219.94	2.11	38.5
26874	219.94	219.94	0	15.5	26874	222.13	219.94	2.19	39
26874	219.94	219.94	0	16	26874	222.21	219.94	2.28	39.5
26874	219.94	219.94	0	16.5	26874	222.3	219.94	2.36	40
26874	219.94	219.94	0	17	26874	222.38	219.94	2.44	40.5
26874	219.94	219.94	0	17.5	26874	222.45	219.94	2.51	41
26874	219.94	219.94	0	18	26874	222.52	219.94	2.58	41.5
26874	219.94	219.94	0	18.5	26874	222.58	219.94	2.64	42
26874	219.94	219.94	0	19	26874	222.64	219.94	2.7	42.5
26874	219.94	219.94	0	19.5	26874	222.69	219.94	2.75	43
26874	219.94	219.94	0	20	26874	222.74	219.94	2.8	43.5
26874	219.94	219.94	0	20.5	26874	222.79	219.94	2.85	44
26874	219.94	219.94	0	21	26874	222.83	219.94	2.89	44.5
26874	219.94	219.94	0	21.5	26874	222.86	219.94	2.92	45
26874	219.94	219.94	0	22	26874	222.89	219.94	2.95	45.5
26874	219.94	219.94	0	22.5	26874	222.92	219.94	2.98	46
26874	219.94	219.94	0	23	26874	222.95	219.94	3.01	46.5

26874	222.97	219.94	3.03	47	26874	221.95	219.94	2.01	63.5
26874	223	219.94	3.06	47.5	26874	221.85	219.94	1.91	64
26874	223.02	219.94	3.08	48	26874	221.74	219.94	1.81	64.5
26874	223.04	219.94	3.1	48.5	26874	221.63	219.94	1.7	65
26874	223.06	219.94	3.12	49	26874	221.53	219.94	1.6	65.5
26874	223.1	219.94	3.16	49.5	26874	221.45	219.94	1.51	66
26874	223.14	219.94	3.2	50	26874	221.39	219.94	1.45	66.5
26874	223.17	219.94	3.24	50.5	26874	221.35	219.94	1.41	67
26874	223.2	219.94	3.26	51	26874	221.32	219.94	1.38	67.5
26874	223.22	219.94	3.28	51.5	26874	221.31	219.94	1.37	68
26874	223.23	219.94	3.29	52	26874	221.31	219.94	1.37	68.5
26874	223.24	219.94	3.3	52.5	26874	221.31	219.94	1.37	69
26874	223.24	219.94	3.3	53	26874	221.3	219.94	1.36	69.5
26874	223.24	219.94	3.3	53.5	26874	221.28	219.94	1.34	70
26874	223.22	219.94	3.28	54	26874	221.27	219.94	1.33	70.5
26874	223.2	219.94	3.26	54.5	26874	221.25	219.94	1.31	71
26874	223.17	219.94	3.23	55	26874	221.23	219.94	1.29	71.5
26874	223.14	219.94	3.2	55.5	26874	221.22	219.94	1.28	72
26874	223.1	219.94	3.16	56	26874	221.2	219.94	1.26	72.5
26874	223.05	219.94	3.11	56.5	26874	221.19	219.94	1.25	73
26874	223	219.94	3.07	57	26874	221.17	219.94	1.24	73.5
26874	222.95	219.94	3.01	57.5	26874	221.16	219.94	1.22	74
26874	222.89	219.94	2.95	58	26874	221.15	219.94	1.21	74.5
26874	222.82	219.94	2.89	58.5	26874	221.14	219.94	1.2	75
26874	222.75	219.94	2.81	59	26874	221.12	219.94	1.19	75.5
26874	222.68	219.94	2.74	59.5	26874	221.11	219.94	1.18	76
26874	222.6	219.94	2.66	60	26874	221.1	219.94	1.16	76.5
26874	222.51	219.94	2.57	60.5	26874	221.09	219.94	1.15	77
26874	222.42	219.94	2.49	61	26874	221.08	219.94	1.14	77.5
26874	222.34	219.94	2.4	61.5	26874	221.07	219.94	1.14	78
26874	222.24	219.94	2.31	62	26874	221.07	219.94	1.13	78.5
26874	222.15	219.94	2.21	62.5	26874	221.06	219.94	1.12	79
26874	222.05	219.94	2.12	63	26874	221.05	219.94	1.11	79.5