Dev doc – Développement

Documentation Interface MATLAB HEC-RAS, IMTLD

Dernière 07/09/2020 13:05:00 – version BETA 1_0_1

Ce document a pour objectif de vous donner tous les éléments et toutes les pistes déjà explorées qui pourraient vous aider dans la modification du script. Pour bien comprendre ce document, je vous recommande d'avoir lu le guide de l'utilisateur au préalable.

Comme il s'agit d'une énumération de pistes et conseils, il peut ne pas exister de lien logique entre un paragraphe et le suivant.

Pour obtenir des informations sur une fonction ou un objet en particulier, écrivez *help <fonction>* ou *help <objet>* dans la console MATLAB.

Table des matières

Notes générales

Abus de langage

J'appelle string des characters vectors, en raison des similitudes entre ces deux types.

Manière de programmer

J'ai privilégié les noms de variables longs et explicites car la saisie semi-automatique proposée par VS Code me permet de ne pas tout réécrire à chaque fois.

Si vous devez éditer le code, veillez à utiliser un éditeur qui propose cette fonctionnalité, sinon ce sera très laborieux pour vous.

Limitations de l'interface

HEC-RAS autorise le fait d'avoir plusieurs conditions initiales de même nom pour une seule river station, cela n'est PAS pris en charge par l'interface et ne sera pas forcément détecté, car c'est considéré comme une fonctionnalité non essentielle.

Note : le fait d'avoir plusieurs vannes sur une même XS est pris en charge, car elles sont numérotées par HEC-RAS. Par exemple, Gate #1 et Gate #2.

Toutefois, si vous souhaitez contourner cette limitation, il faudra trouver une solution pour que l'interface différencie les deux hydrogrammes. A priori il n'existe aucun moyen de les différencier hormis en comparant les données qu'ils contiennent, ce qui est dangereux, car le but est de modifier ces données.

Documentation du code

Les fonctions sont documentées au fil du code. Il y a même des informations de type pseudo code quand c'est nécessaire.

Pour accéder rapidement à la documentation succincte d'une fonction de l'interface, écrivez le nom de la fonction dans la console MATLAB puis appuyez sur F1.

Spécificités du HEC-RAS Controller dans MATLAB

Breaking étant basé sur une version antérieure d'HEC-RAS, et n'étant pas écrit en langage MATLAB, il existe des différences, certaines méthodes n'existent pas dans MATLAB.

Heureusement, grâce à la page de documentation MATLAB sur les actxserver (et les COM objects qui y sont associés), on peut détecter l'ensemble des méthodes disponibles grâce à la fonction methodsview.

Cela facilite grandement l'utilisation du HEC-RAS Controller.

Mais dans les faits, les arguments demandés en paramètres ne sont pas pertinents en langage MATLAB (par exemple des pointeurs). Donc il faut faire des tests pour vérifier que les arguments sont acceptés.

Par exemple Compute_CurrentPlan est indiqué comme supportant 4 arguments mais d'après les tests n'en accepte que 2, et seulement si le premier est 0 ou 1 et le deuxième est 0, en revanche le pointeur qu'elle aurait dû demander, mais qu'elle refuse, est bien retourné en output (sans être demandé en input)

Les safe array pointers doivent bien être remplacés par 0 (et rien d'autre sinon erreur, peut-être lié à la notion de pointeur NULL ?) et récupérés en sortie.

Handle, référence à un objet (plus haut niveau que le pointeur).

Donc dans methods view, handle est simplement l'objet lui-même.

Car rp.Methods(a) est équivalent à Methods(rp,a)

On peut considérer l'objet comme un paramètre classique nécessaire au bon fonctionnement de la méthode associée.

Si on appelle compute_current_plan alors qu'un plan est en cours de calcul, cela le ferme instantanément (avant même qu'il ait pu terminer les calculs).

Méthodologie de débogage

Cette méthodologie est utilisée pour faire en sorte que l'interface dialogue correctement avec HEC-RAS.

Première partie : les outils

Vous allez exécuter le script de très nombreuses fois, je vous conseille donc d'utiliser un projet HEC-RAS très rapide (temps de calcul de quelques secondes, 5 pas suffisent largement). A titre d'exemple, celui utilisé dans le tutoriel vidéo 1 – Exemple simple d'automatisation est parfaitement approprié.

Pour connaitre les valeurs à modifier dans Ras, un outil indispensable est le comparateur de code, qui prend en charge tous les fichiers textes quel que soit leur extension. J'en ai programmé un assez facilement, mais les fonctionnalités avancées des comparateurs graphiques, que je n'avais pas programmé, sont très pratiques. En effet, s'il y a un décalage d'une ligne, le comparateur va détecter ce décalage et le signaler. Le comparateur est aussi capable de détecter les changements dans les caractères invisibles (par exemple, celui de retour à la ligne qui causait des soucis, car celui utilisé par Ras est différent de celui utilisé par MATLAB).

Parmi ces comparateurs « graphiques » (i.e. qui ne s'utilisent pas via ligne de commande, mais qui ont une interface graphique) j'ai utilisé celui intégré à MATLAB > Menu HOME > Bouton COMPARE.

J'ai aussi essayé celui de VS Code (logiciel gratuit), et il fonctionne très bien aussi. J'ai juste trouvé l'interface plus complexe à comprendre au premier abord, car il n'affiche pas de texte d'aide et ne masque aucune ligne. Contrairement à MATLAB qui est très efficace sur les grands fichiers car il masque toutes les lignes identiques (et redonne les lignes qui précèdent et suivent la ligne différente, pour donner du contexte). De plus, MATLAB produit des bilans sous forme de tableaux qui listent les fichiers et qui sont très pratiques.

Deuxième partie : comment procéder

1ère étape : identifier les modifications nécessaires dans les fichiers d'HEC-RAS

Pour effectuer les trois manipulations détaillées ci-dessous, je vous conseille de faire un projet HEC-RAS de référence, de copier 3 fois le dossier contenant le projet, et de faire chacune des manipulations dans une copie différente du projet.

a. Réglez le script sur debug=1, puis faites des sauvegardes (backup) dès que possible. Le script a un système de gestion avancée des sauvegardes, qui évite que vos sauvegardes précédentes soient effacées. Sous réserve que ce système fonctionne bien, toute nouvelle sauvegarde devrait avoir un dossier différent de ceux existant déjà.

b. Faites ensuite la même simulation, mais manuellement, en agissant directement dans HEC-RAS sans utiliser MATLAB. En faisant des pauses à chaque pas. Sauvegardez avant et après

chaque pas l'état des fichiers d'HEC-RAS dans un nouveau dossier à l'aide du script BATCH de sauvegarde.

Pour apprendre à mettre en pause Ras, cf. tutoriel de développement 1 – Simulation manuelle pas à pas.
Attention : il faut sauvegarder les fichiers Ras, par exemple à l'aide des scripts BATCH de sauvegarde, avant et après chaque pas de simulation, i.e. copier le dossier contenant le projet RAS dans un nouveau dossier portant le nom de la sauvegarde. Par exemple : AprèsPas1 ou encore AvantPas3.

c. Faites ensuite la même simulation, manuellement dans HEC-RAS, mais sans faire de pause. Sauvegardez avant et après la simulation l'état des fichiers d'HEC-RAS dans un nouveau dossier à l'aide du script BATCH de sauvegarde. Disponible dans le dossier BATCH code du dossier contenant une version de l'interface (par exemple BETA 1_0_1).

Comparez ensuite les fichiers de la même étape entre eux, à l'aide de l'outil de comparaison évoqué plus haut.

Comparez également les fichiers entre deux étapes successives. Vous pouvez obtenir des résultats similaires à la figure suivante par exemple.

Comparing folder AfterInit vs. folder BeforeRestart1

Left file list	Contents of folder F:\CanalSteps12_Backup\AfterInit
Right file list	Contents of folder F:\CanalSteps12_Backup\BeforeRestart1

Click on a column header to sort the table

		In left list (folder AfterInit)		In right list (folder BeforeRestart1)		
Type	**File Name**	**Size (bytes)**	**Last Modified Date**	**Size (bytes)**	**Last Modified Date**	**Difference Summary**
O02 File	CANAL.IC.O02 (open: left | right)	1669760	2020-06-17 10:57:59	1669760	2020-06-17 10:57:59	*identical*
O02 File	CANAL.O02 (open: left | right)	4748800	2020-06-17 10:58:02	4748800	2020-06-17 10:58:02	*identical*
B02 File	CANAL.b02 (open: left | right)	3209	2020-06-17 10:57:52	3209	2020-06-17 10:57:52	*identical*
BCO02 File	CANAL.bco02 (open: left | right)	2076	2020-06-17 10:57:59	2076	2020-06-17 10:57:59	*identical*
C02 File	CANAL.c02 (open: left | right)	1448292	2020-06-17 10:57:57	1448292	2020-06-17 10:57:57	*identical*
DSS File	CANAL.dss (open: left | right)	448000	2020-06-17 10:57:59	448000	2020-06-17 10:57:59	*identical*
G02 File	CANAL.g02 (open: left | right)	778271	2020-06-16 11:32:36	778271	2020-06-16 11:32:36	*identical*
HDF File	CANAL.g02.hdf (open: left | right)	472365	2020-06-17 10:57:57	472365	2020-06-17 10:57:57	*identical*
P02 File	CANAL.p02 (open: left | right)	4916	2020-06-17 10:57:06	4916	2020-06-17 11:01:04	contents changed (compare)
RST File	CANAL.p02.22JAN2020 0100.rst (open: left | right)	126135	2020-06-17 10:57:59	126135	2020-06-17 10:57:59	*identical*
BLF File	CANAL.p02.blf (open: left | right)	82	2020-06-17 10:57:59	82	2020-06-17 10:57:59	*identical*
HDF File	CANAL.p02.hdf (open: left | right)	1278484	2020-06-17 10:58:03	1278484	2020-06-17 10:58:03	*identical*
PRJ File	CANAL.prj (open: left | right)	560	2020-06-17 10:31:08	560	2020-06-17 10:31:08	*identical*
R02 File	CANAL.r02 (open: left | right)	1104613	2020-06-17 10:57:59	1104613	2020-06-17 10:57:59	*identical*
U01 File	CANAL.u01 (open: left | right)	44126	2020-06-17 10:56:20	44173	2020-06-17 11:00:59	contents changed (compare)
X02 File	CANAL.x02 (open: left | right)	937507	2020-06-17 10:57:52	937507	2020-06-17 10:57:52	*identical*

Figure 1- Comparaison après initialisation et avant le premier redémarrage

Dans la figure ci-dessus, on a ''photographié'' la manière dont étaient enregistrés les changements faits par l'utilisateur entre deux pas de simulation.

Que faut-il comparer ?

Le schéma ci-dessous synthétise notre démarche, modifications par MATLAB correspond à la manipulation a) et modifications par l'utilisateur correspond à la modification b) :

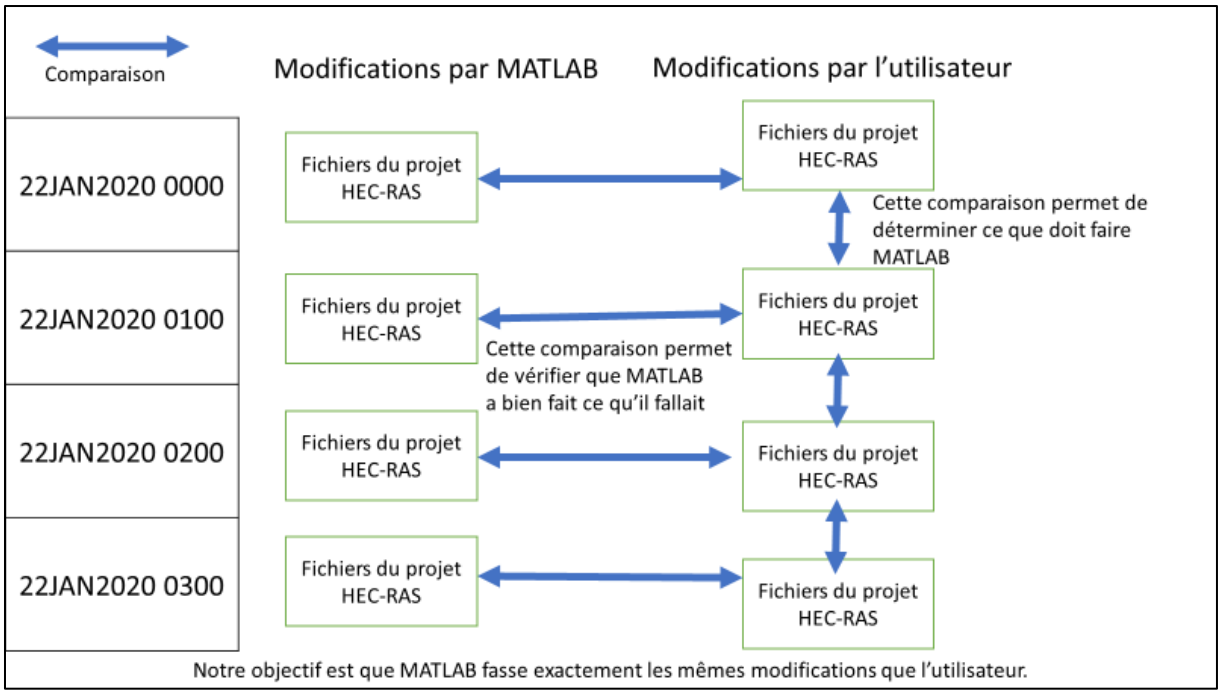

Figure 2 - Schéma de la méthodologie

2e étape : implémenter les modifications trouvées

Dans cette étape, vous allez pouvoir programmer les modifications que vous avez détectées comme étant nécessaires. Le code des fonctions est commenté autant que possible. Et vous pouvez vous aider de la commande MATLAB *help* pour en savoir plus à propos d'un objet ou d'une fonction.

De plus, voici quelques pistes pour vous permettre de savoir quels scripts vous devez modifier :

- la partie du master qui vous intéresse est contenue dans le for, il s'agit particulièrement des lignes de code entourant les appels de la fonction update_param_first
- les fonction update_param_first et param_delete permettent à elles deux de faire toutes les modifications nécessaires dans HEC-RAS pour permettre l'automatisation de la mise en pause et redémarrage d'HEC-RAS

3e étape : vérifier les modifications

Pour vérifier qu'il n'y a pas d'erreur cachée, je vous invite à comparer vos résultats obtenus en faisant les manipulations a), b) et c). Puis en suivant la méthode présentée dans le tutoriel 1 - exemple simple d'automatisation afin de tester votre script en conditions réelles.

Méthodologie de programmation

Ajouter des fonctionnalités

Si vous souhaitez davantage maitriser l'écriture dans les fichiers d'HEC-RAS, par exemple pour faire quelque chose qu'il n'est pas possible de faire avec les scripts d'input et d'output. Vous pouvez utiliser le script d'interface d'une autre manière.

Tout d'abord, je vous invite à lire la partie de cette documentation présentant le fonctionnement global du script master, afin d'identifier quelle partie du script vous souhaitez modifier.

Ensuite, familiarisez-vous avec l'arborescence des fonctions. En effet, les fonctions sont triées par type dans des dossiers et sous-dossiers. Cela vous permet de choisir de quelles fonctions vous allez avoir besoin, lesquelles vous allez modifier, etc. Le script étant décomposé en de nombreuses fonctions, essayez de réutiliser au maximum les fonctions fournies, cela vous évitera d'écrire du code qui existe déjà. A ce propos, si vous remarquez que du code pourrait être amélioré, n'hésitez pas à nous contacter pour faire part de vos suggestions. Nous vous remercions par avance pour votre aide !

Enfin, il ne vous reste plus qu'à programmer afin de concrétiser votre idée. Pour cela les paragraphes suivants présentent des méthodes et conseils dans le cadre de cas classiques de modification de script.

Créer une version personnalisée

Il existe trois types de versions ''officielles'' de l'interface :

- ALPHA (versions non publiées, type utilisé lors du début de la construction de l'interface)
- BETA (instable)
- RELEASE (stable)

Vous pouvez créer un nouveau type de version très simplement, il suffit que le dossier de votre version soit nommé de la manière suivante pour être correctement reconnu par le launcher :

Soit X un chiffre quelconque, soit Y un nombre quelconque

<le nom de votre type> Y MATLAB HECRAS interface

<le nom de votre type> X_X_X MATLAB HECRAS interface

Par défaut, le launcher lance la dernière version, quel que soit son type, du moment que c'est une chaine de caractères. Donc si vous souhaitez pouvoir lancer par défaut votre version personnalisée, il suffit de lui donner un numéro supérieur à celui des versions officielles.

Remarque : X_X_X est interprété comme le numéro XXX

e.g. 1_0_1 est compté comme 101

donc vous pouvez nommer votre version 102 ou 1_0_2 pour qu'elle soit lancée prioritairement vis-à-vis de 1_0_1

Autre remarque : MATLAB en version 2020 ne supporte pas les dossiers dont le nom contient des '.' Cela explique l'absence de 1.0.1, qui aurait pourtant été plus pratique.

Réaliser une montée de version de la compatibilité avec HEC-RAS ou MATLAB

Le script est prévu pour fonctionner avec MATLAB 2020 et HEC-RAS 5.0.7. Il est donc susceptible de ne pas fonctionner sur des versions ultérieures.

Mais pour réaliser ce script, je me suis basé sur des scripts MATLAB et VBA (cf. document références) qui utilisaient des versions antérieures d'HEC-RAS. Je peux donc vous donner quelques conseils dans le cas où vous devez effectuer une montée de version d'HEC-RAS.

L'idée est d'utiliser la Méthodologie de débogage, mais dans un contexte différent. En effet, ici on va faire une simulation manuellement, en mettant en pause et en redémarrant avec le restart file, dont on va sauvegarder les modifications des fichiers après chaque itération.

Le tableau suivant synthétise les comparaisons de fichiers HEC-RAS à effectuer pour déterminer quelles modifications apporter au code.

Tableau 1 - comparaisons pour une montée de version

Quoi	Ceci	Comparé à ceci
Exécution manuelle d'HEC-RAS	En version d'HEC-RAS 5.0.7	Dans la nouvelle version d'HEC-RAS
Exécution automatisée via MATLAB	Dernière version de l'interface, sur la version 5.0.7	Dernière version de l'interface, sur la nouvelle version HEC-RAS **Remarque :** il devrait se produire de nombreuses erreurs ici. Pensez à les noter et les analyser.
Exécution automatisée VS manuelle	Exécution manuelle dans la nouvelle version d'HEC-RAS	Corrigez les erreurs en modifiant le script MATLAB (pensez à créer une nouvelle version, cf. Créer une version personnalisée). Puis exécutez votre script MATLAB.

Cependant, si le langage MATLAB subit des modifications importantes, il faudra utiliser la documentation MATLAB pour rendre compatible ce script. Je n'ai dû adapter qu'une seule fonction : dlmread, qui posait un problème léger car elle était d'une version antérieure de MATLAB, je n'en ai finalement pas eu besoin, mais la procédure de montée de version était bien expliquée dans la documentation de MATLAB.

Dépendances entre fonctions

Les scripts de l'interface sont interconnectés, comme le montre le graph des dépendances présenté plus loin. Cela peut poser un problème lors de la modification de fonctions de bas niveau, i.e. dont de nombreuses fonctions dépendent.

Méthode d'analyse des dépendances exécutée sur la version BETA 1_0_1

J'ai simplement suivi les instructions de la page Analyze Project Dependencies de la documentation interne à MATLAB. En créant un projet qui a pour dossier principal le dossier parent du launcher. Et pour script de démarrage le script launcher.

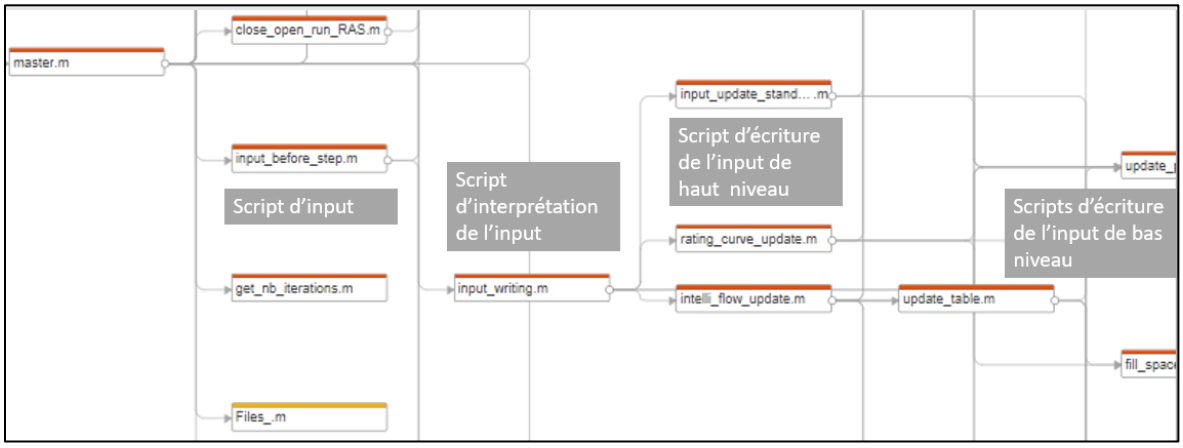

Figure 3 - Détail du graph des dépendances BETA 1_0_1 - script d'input

La figure ci-dessus met en évidence les nombreuses fonctions nécessaires à l'interprétation des données d'input avant leur écriture dans les fichiers de résultats.

Le fichier GRAPHML dont un extrait est présenté ci-dessus est disponible dans le dossier EN resources.

Comment améliorer la détection d'erreur côté HEC-RAS

Citation du document Dépannage :

> *Erreurs RAS cachées*
>
> *Si RAS rencontre une erreur lors du lancement de la simulation, il peut ne pas le dire à MATLAB et rester en boucle infinie. Le plus souvent, il a écrit un message d'erreur dans le fichier txt ''[…]compute msgs '', mais dans le cadre d'une utilisation totalement automatique, MATLAB restera indéfiniment en attente d'HEC-RAS.*

Il est assez difficile de faire interpréter le fichier compute msgs par MATLAB, car le mot erreur n'y est pas écrit

La détection est donc basée sur un test de longueur du fichier après un temps où normalement tout type de simulation qui fonctionne devrait avoir un fichier d'une longueur plus grande que cette valeur.

Cette méthode reste très rudimentaire et peut très facilement mal se comporter. Ici ce n'est pas gênant car il s'agit simplement de montrer le fichier texte contenant le message d'erreur à l'utilisateur.

Mais si vous souhaitez utiliser cela pour détecter une erreur, il faudra plutôt penser à concevoir une méthode de détection plus efficace, voire plus personnalisée. En effet, si vous connaissez le temps moyen que prend votre simulation, vous pouvez mettre en place une condition qui détecte que la simulation ne s'est pas terminée dans le temps imparti. Cela fonctionnera uniquement pour votre projet, et avec des réglages inchangés.

Dépendances entre paramètres

HEC-RAS permet que plusieurs conditions limites différentes soient appliquées sur une coupe en travers (XS). Si elles ont un nom différent, c'est supporté par l'interface MATLAB HEC-RAS.

Mais elles ne sont pas sauvegardées dans le même ordre que celui affiché dans l'interface graphique d'HEC-RAS. Le plus simple est d'observer l'organisation d'un fichier Ras pour comprendre :

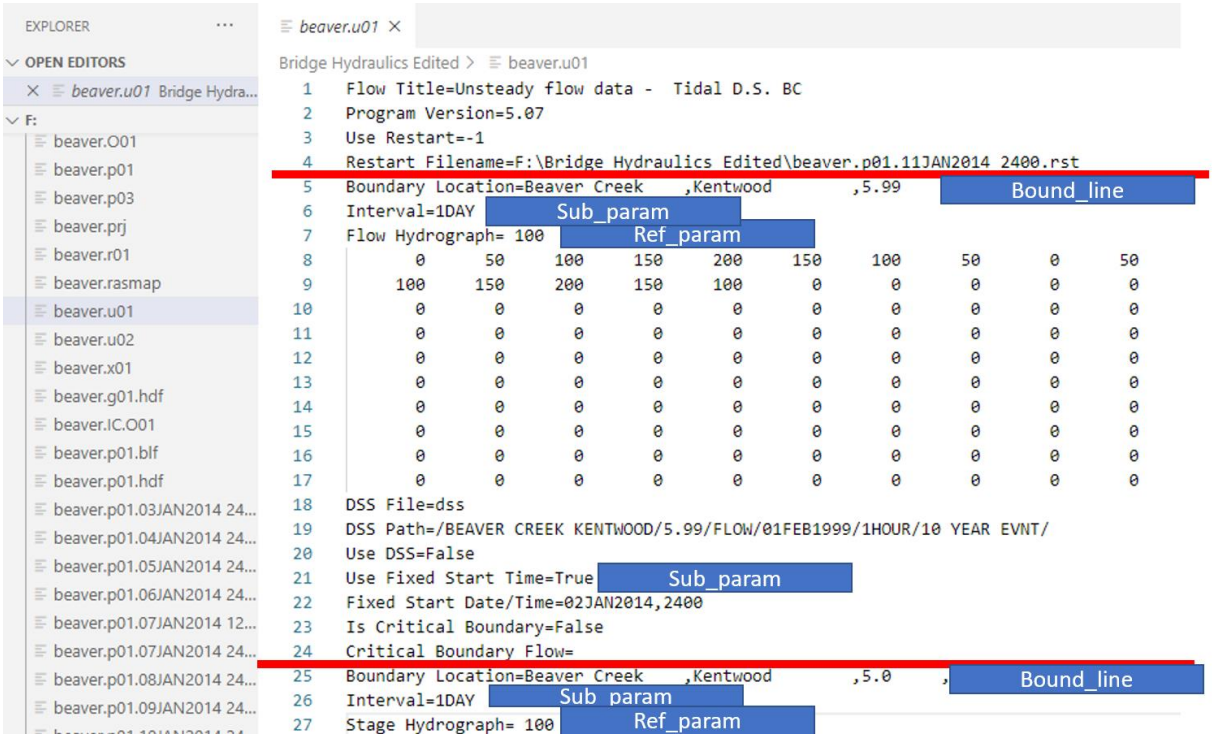

Figure 4 - Capture d'écran modifiée d'un extrait d'un fichier de débits inconstants

Définitions

- ref_param est, conformément à la limitation de l'interface énoncée plus haut, défini comme un paramètre unique pour une cross section donnée
- sub_param est défini comme un paramètre, qui est unique pour un ref_param donné.

Dans l'exemple ci-dessus Figure 4, si le Stage Hydrograph et le Flow Hydrograph avaient été deux conditions limites de la même cross section, il y aurait eu pour une même cross section deux sub_param Interval que l'on aurait pu confondre. Mais tant que l'on sait à quel ref_param est lié chaque sub_param, on peut facilement trouver et modifier ce sub_param sans risquer de le confondre avec un autre qui a un nom identique.

Présentation de scripts notables

Description générale du script launcher

En fonctionnement classique de l'interface, le launcher est appelé avant chaque lancement d'une version de l'interface.

Cleanup

Il s'agit de vider les variables globales et de fermer les fichiers laissés ouverts par inadvertance.

Puis d'enlever du chemin MATLAB les versions précédemment lancées.

Attention : cela effacera aussi les addpath que vous avez éventuellement fait avant l'appel du launcher.

Init

Ici le script demande à l'utilisateur quelle version il veut lancer et va appeler des fonctions qui lancent une version de l'interface selon la réponse de l'utilisateur.

Fonctions de lancement

Si c'est une version ALPHA, le launcher change le dossier MATLAB courant (en raison de la manière dont sont programmées les versions ALPHA). Et copie les fichiers modifiés par l'utilisateur dans le dossier de la version.

Sinon il ajoute au chemin MATLAB le dossier contenant la version, afin que la fonction master soit appelable depuis launcher.

Description générale du dossier Installation

Il s'agit d'un ensemble d'utilitaires de compression et décompression du format ZIP, de détection des versions installées et de vérification de ces versions.

Description générale de la détection d'erreur HEC-RAS

La fonction detect_RAS_error cherche dans compute_msgs plusieurs informations :

- Contient-il le mot « error », si oui, arrêt de la simulation, proposition de sortie de la boucle MATLAB. Si acceptée, affichage des résultats
- Contient-il le mot « Progress » qui doit apparaitre dès le début de la simulation, si ce n'est pas le cas, HEC-RAS est souvent en erreur bloquante dès le début

Description générale de la fonction master

MATLAB init

Récupère les réglages et les applique en initialisant des variables qui convertissent le paramètre donné en informations compréhensibles par HEC-RAS.

Ensuite, le caractère de fin de ligne propre à HEC-RAS est récupéré.

Enfin, une fonction qui réalise une boucle similaire à la boucle for principale, mais à vide, compte le nombre d'itérations qu'il va être nécessaire de faire. Et l'utilisateur a la possibilité de gérer les fichiers de projet HEC-RAS s'il a activé le debug.

Loop

On fait nb_of_iterations itérations. En appelant à chaque fois les scripts input et output adéquats.

Si c'est le premier pas.

Alors on initialise le projet HEC-RAS, c'est-à-dire qu'on fait la simulation du premier pas de temps, en demandant à HEC-RAS de créer un restart file à la fin.

Sinon si c'est le deuxième pas de temps.

Alors on configure HEC-RAS pour qu'il démarre sur le restart file créé au pas précédent, on met à jour les paramètres utiles, et on lance le deuxième pas.

Sinon

Alors c'est le troisième pas ou plus, on met à jour les paramètres utiles, et on lance un nouveau pas.

End

Fin du programme, affichage des résultats et restauration des fichiers HEC-RAS pour qu'il soit prêt à faire une simulation complète, sans pause, sur la durée que l'on vient de simuler.