Couplage Mage-BaM : analyse du code et paramétrage des fichiers

F. Mendez-Rios, Benjamin Renard

Centre
Lyon – Grenoble
Auvergne-Rhône-Alpes
Décembre 2022

INRAE

Couplage Mage-*BaM* : analyse du code et paramétrage des fichiers

FELIPE MENDEZ

Table des matières

List des figures

1. Introduction

L'objectif de ce rapport est d'expliquer le fonctionnement du code *BaM* pour des nouvelles intégrations des modèles à l'avenir. De même, l'idée est aussi de montrer la bonne façon de paramétrer non seulement les fichiers de configuration mais aussi l'IDE (CodeBlocks) qui s'utilise souvent pour lancer des scripts écrits en Fortran (langage de programmation), car actuellement peu d'information existe.

D'ailleurs, le rapport a aussi pour but de montrer la mise en place d'un modèle hydrodynamique 1D (Mage) sur le *Framework BaM* afin d'obtenir un calage automatique des coefficients de rugosité en prenant en compte des observations (campagnes de jaugeages et lignes d'eau).

Enfin, une documentation du RBaM (R package) se présente afin de faciliter l'écriture des fichiers de configuration.

2. Couplage Mage-*BaM*

BaM est une généralisation du code de calcul BaRatin qui a pour but l'estimation des paramètres d'un modèle avec leurs incertitudes associées et d'effectuer des prédictions, en prenant en compte leurs incertitudes, à partir du même modèle. C'est pour cela qu'il faut bien comprendre la façon comment il marche afin de pouvoir brancher un modèle hydrodynamique 1D (MAGE) pour son calage automatique.

D'après Renard, B (2017), dans *BaM* « un modèle est défini comme une fonction qui calcule une ou plusieurs variable(s) de sortie à partir d'une ou plusieurs variable(s) d'entrée. Cette fonction dépend d'un certain nombre de paramètres qui sont généralement inconnus et qui devront donc être estimés ». Donc, *BaM* a besoin d'un minimum des éléments listés ci-dessous :

- Nombre des variables d'entrée.
- Vecteur de paramètres nommé θ qui devra être estimé.
- Une fonction $\boldsymbol{M}\ (\boldsymbol{X};\boldsymbol{\theta})$ qui sert à calculer, à partir des variables d'entrée et des paramètres, les valeurs prises par les variables de sortie.
- Nombre de variables de sortie

De même, autres éléments peuvent être optionnels pour le pilotage de l'outil énoncés ci-après :

- Information de spécifique du modèle. Cela sera en fonction du type de modèle à intégrer. A titre d'exemple, il peut s'agir d'un fichier d'options, de conditions initiales, des données topographiques, entre autres.
- Paramètres dérivés. En d'autres termes, des paramètres calculés en fonction d'autres paramètres préalablement estimés.
- Variable d'état : ce sont des variables de sortie supplémentaires, leur particularité est qu'elles ne sont pas observables. Elles sont plutôt informatives sur l'état du modèle, elles ne sont pas utilisées pour l'estimation des paramètres du modèle.

En résumé, la schématisation ci-dessous montre à grosso modo la représentation interne de *BaM*.

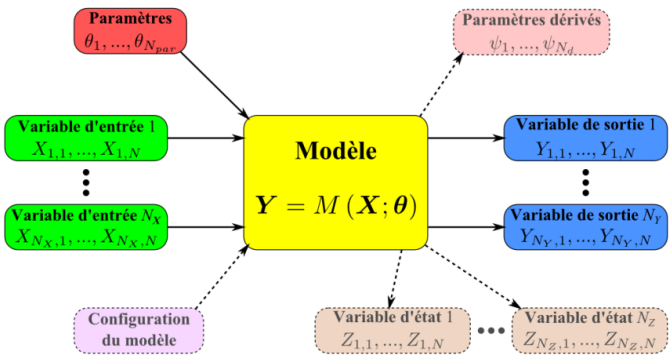


Figure 1. Schématisation du modèle manipulé par BaM (Renard, B (2017))

Sur la base de la compréhension d'un modèle sur *BaM*, il est donc temps de passer à l'estimation du vecteur de paramètres (θ). En général, le calcul se base sur la comparaison les résultats des simulations du modèle avec des observations de ses variables de sortie.

D'après Renard, B (2017), l'estimation des paramètres porte sur une analyse Bayésienne. Par conséquent, il se réalise à partir de la distribution a posteriori qui englobe l'information portée par les données quantifiées via la vraisemblance et la distribution a priori. Puisque le vecteur de paramètres à estimer peut avoir plus d'une dimension, *BaM* s'appuie sur l'algorithme de *MCMC*. Ensuite, l'objectif est d'évaluer si les valeurs des paramètres convergent vers les observations ou plutôt s'écartent.

Etant donné que *BaM* s'intéresse non seulement à l'estimation des paramètres, mais également à leurs incertitudes, l'outil considère trois sources d'erreurs (Figure 2) :

- **Erreur sur les données de sortie :** différence entre les valeurs observées (sortie du modèle) et les vraies valeurs.
- **Erreur sur les données d'entrée :** erreur liée aux observations des variables d'entrée du modèle.
- **Erreur structurelle :** différence entre la vraie valeur de la variable de sortie et la valeur simulée par le modèle.

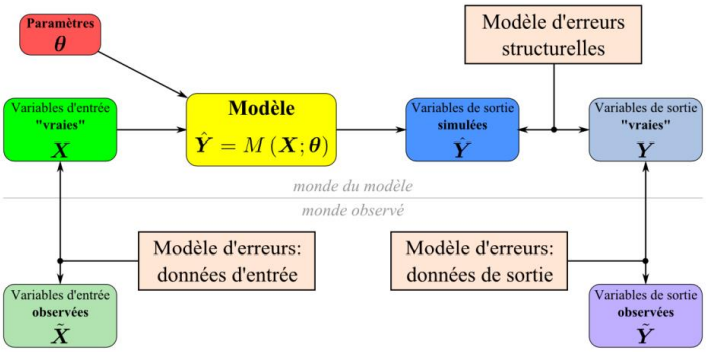


Figure 2. Modèles d'erreurs intégrés sur BaM (Renard, B (2017))

Après avoir abouti l'étape de calage, le modèle est prêt pour réaliser des prédictions. En d'autres termes, il prendra les valeurs des variables de sortie comme des variables d'entrée et ensuite relancer l'estimation.

Grâce à la versatilité et la structuration de *BaM*, plusieurs situations peuvent être représentées en fonction de la variable d'entrée fournie. Dans notre cas d'étude, l'utilité serait pour la **prévision** autrement dit, les variables d'entrée correspondent à des valeurs futures. En variant des conditions de la marée, les conditions initiales de la ligne d'eau, il est possible de pouvoir prédire le comportement du débit futur avec leur incertitude.

Finalement, après avoir expliqué à grosso modo le fonctionnement de *BaM*, il est donc temps de se concentrer sur le modèle hydrodynamique 1D à intégrer sur l'outil.

Pour le premier essai d'intégration du modèle MAGE, un modèle simplifié de la Seine aval a été implémenté. En effet, l'objectif est d'avoir un modèle léger qui se lance en quelques seconds afin d'obtenir les premiers résultats et ensuite pouvoir réviser la stabilité numérique des simulations. C'est pour cela qu'afin de réduire le temps de calcul, plusieurs astuces ont été adoptées comme la réduction des profils transversaux, homogénéisation des certaines valeurs de frottement par tronçon (réduction de la quantité de Ks) et réduction de la période d'analyse (focus sur la première campagne de jaugeages de mars 2017).

S'il fallait classer les possibles modifications sur un modèle pour réduire son temps de calcul, j'ai l'impression que la réduction de la période d'analyse serait le paramètre primordial à modifier afin de réduire le temps de calcul propre du MAGE. Ensuite, la réduction des tronçons qui seront affectées par un coefficient de rugosité, car cela permettra de réduire le vecteur de paramètres sur BaM, plus précise, la quantité de paramètres à enlever s'élève 2 fois chaque tronçon parce qu'il faut lui donner le Ks du lit mineur et lit majeur. Enfin, le nombre des profils transversaux sans aller à endommager ou perdre de précision du comportement de la ligne d'eau.

Ce nouveau modèle simplifié compte avec le même paramétrage pour le premier bief. En d'autres termes, 34 profils et un seul

coefficient de rugosité. En revanche, le deuxième bief passe de 568 à 194 profils et par rapport aux coefficients de Strickler, il passe de 12 à 7 tronçons. Finalement, le temps final de simulation passe de 124 jours 23 h à 1 jour 17 h 30 minutes. Cela est possible de faire parce qu'on a conclu précédemment que les conditions initiales étaient suffisamment bonnes pour diffuser leurs impacts sur la modélisation.

En conclusion, on passe d'un temps de calcul de 16 à environ 0.5 secondes. Ce qui est le souhaitable pour les premiers tests puisque la méthode à utiliser pour le calage se base sur *MCMC*, il aura besoin d'environ 100 000 simulations pour s'assurer de la convergence des simulations.

Les sous-sections ci-après seront divisés en deux grands groupes. L'un a pour but de montrer la façon de brancher un modèle hydrodynamique 1D sur BaM, notamment MAGE et le deuxième présentent l'explication des codes utilisés du *Framework*.

2.1. Paramétrage sur *BaM*

Le code de calcul est contrôlé par des fichiers de configuration au format texte pour piloter les différentes fonctionnalités de l'outil. Dans le présent rapport, il s'explique les fichiers configurations propres du modèle MAGE parce que le guide détaillé des fichiers se trouve sur B. Renard. BaM ! (Bayesian Modeling): Un code de calcul pour l'estimation d'un modèle quelconque et son utilisation en prédiction. irstea. 2017, pp.90.

Avant d'entrer sur les fichiers de configuration, ça vaut la peine d'expliquer le nécessité de la présence des certains dossiers comme la Figure 3 illustre.

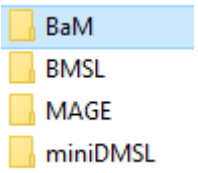


Figure 3. Dossiers impératifs pour le lancement du BaM

Le dossier *BaM* aura tous les fichiers de configuration qui vont piloter le *Framework*. Pour être lancé, il a besoin de s'appuyer sur des fonctionnalités développées dans les dossiers BMSL et miniDMSL. En même temps, pour l'intégration du modèle hydrodynamique 1D, il faut lui rendre l'exécutable qui sera répertorié dans le dossier MAGE.

2.1.1. Contrôleur général (Config_BaM.txt)

Ce fichier permet d'établir l'espace de travail où sont stockés non seulement les fichiers de configuration propre au modèle en question, mais aussi les résultats. *BaM* a forcément besoin de ce fichier pour être lancé. Seulement ce fichier sera localisé dans le même dossier que l'exécutable. En revanche, tous les autres fichiers de configuration seront placés sur l'espace de travail.

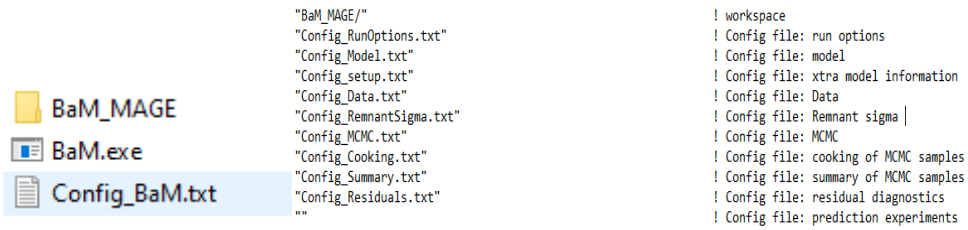


Figure 4. Espace de travail (à gauche) et contenu du fichier Confi_BaM.txt (à droite)

Ce qu'il faut toujours retenir dans ce fichier est que la première ligne fait référence à l'espace du travail du modèle à lancer.

2.1.2. Contrôle des actions que *BaM* doit exécuter (Config_RunOptions.txt)

Les actions à effectuer sur *BaM* seront affichées sur ce fichier. Il faut lui donner une réponse vrai/faux afin d'activer ou désactiver une action. Il faut respecter le format (.true./.false.) pour que l'outil comprenne bien. La signification de chaque ligne se fait ci-après :

- Ligne 1 : simulation MCMC des paramètres à estimer ;
- Ligne 2 : résumé statistique des simulations MCMC ;
- Ligne 3 : analyse des résidus, i.e. comparaison des simulations du modèle avec les données de calibration
- Ligne 4 : prédictions et propagation des incertitudes

```
.true.    ! Do MCMC?
.true.    ! Do MCMC summary?
.true.    ! Do Residual diagnostics?
.false.   ! Do Predictions?
```

Figure 5. Contenu du fichier Config_RunOptions.txt étape de calage

En général, le lancement de *BaM* s'effectue en deux étapes. La première étape réalise l'estimation, ce qui indique l'activation des trois premières lignes. La deuxième étape porte sur la prédiction et pour ce faire, il faut désactiver les trois premières lignes et activer seulement la dernière. Il faut toujours faire dans ce sens, car la deuxième étape a besoin des fichiers MCMC préalablement crées.

2.1.3. Configuration du couplage MAGE-*BaM* (Config_setup.txt et fichiers MAGE)

Afin d'intégrer le nouveau modèle dans *BaM*, il faut passer par ce fichier qui permettra de faire le lien entre l'outil et MAGE. En effet, la Figure 6 illustre à grosso modo les chemins d'accès qu'il faut lui fournir pour que *BaM* puisse lancer l'exécutable MAGE (première ligne) et ensuite, tous les fichiers issus de MAGE seront sauvegardés dans un dossier du projet qui est créé tout seul (deuxième ligne).

Jusqu'ici, *BaM* a repéré les fichiers nécessaires pour lancer MAGE et le dossier résultats sans aucune autre intervention. Le lien sera tissu à partir de la troisième ligne (*REP file* (Figure 7)). Effectivement, les rectangles en rouge indiquent les fichiers dont *BaM* a besoin pour lancer l'analyse Bayésienne.

```
'/home/brenard/BEN/GitHub/MAGE/mage'                          ! Mage executable
'/home/brenard/BEN/GitHub/BaM/tests/BaM_MAGE/Mage_Seineav3_0/' ! Project directory
'Seineav3.REP'                                                ! REP file
```

Figure 6. Contenu du fichier Confi_setup.txt

```
confirmation=non
TAL Seineav3.TAL
MIN Seineav3.MIN
TIT Seineav3.TIT
GEO Seineav3.GEO
DEV Seineav3.DEV
NUM Seineav3.NUM
RUG Seineav3.RUG
HYD Seineav3.HYD
INI Seineav3.INI
LIM Seineav3.LIM
TRA Seineav3.TRA
BIN Seineav3.BIN
ERR Seineav3.ERR
ENV Seineav3.ENV
CSV R_comp_29m_q_2_41568_1_300
```

Figure 7. Contenu du REP file

La figure ci-dessous illustre les contenus des fichiers qui servent à caler le modèle via la campagne de jaugeages de mars 2017. Etant donné que le vecteur paramètre s'agit du coefficient de Strickler, *BaM* modifiera à chaque itération ses valeurs jusqu'à qu'il atteint la quantité d'itérations définie. C'est ainsi que le calage automatique du modèle est réalisé. Puisque le calcul du débit est

directement lié avec les coefficients de rugosité de chaque tronçon, on aura un vecteur résultat (débits) estimé pendant toute la période de modélisation.

```
*************** STRICKLERS *********************
*Bif        x deb      x fin      K min      K moy
*---=======----------==========----------==========
K  1             0.     5064.      39.83      10.00
K  2          5064.    16823.      19.63      10.00
K  2         16823.    51186.      26.11      10.00
K  2         51186.    65055.      30.33      10.00
K  2         65055.    97091.      53.70      10.00
K  2         97091.   111227.      52.27      10.00
K  2        111227.   137983.      37.26      10.00
```

Temps	Débits
0.000	0.000
0.083	-20.957
0.167	-95.319
0.250	-149.710
0.333	-196.478
0.417	-229.715
0.500	-252.923
0.583	-275.966
0.667	-298.085
0.750	-318.447

Figure 8. Contenu du RUG file (à gauche) et CSV file (à droite)

2.1.4. Configuration du modèle (Config_Model.txt)

Ce fichier aura toutes les propriétés du modèle à caler. Chaque ligne sera répertoriée afin de discuter plus en détail :

- Ligne 1 : identifiant du modèle à lancer, car plusieurs modèles sont disponibles sur l'outil ;
- Ligne 2 : nombre de variables d'entrée. Dans le cas d'étude, il n'y a aucun variable d'entrée. Néanmoins, la façon du codage de l'outil, il ne permet pas d'avoir la valeur Nulle comme entrée. Même si le modèle comme tel n'a pas besoin de variable, il faut lui spécifier 1 ;
- Ligne 3 : nombre de variables de sortie. Il s'agit des observations auxquelles les simulations seront comparées. En premier lieu, il a été affecté la valeur de 1, car le calage s'est fait seulement avec les débits jaugés dans les campagnes des jaugeages ;
- Ligne 4 : nombre de paramètres. En d'autres termes, ce sont les paramètres à caler via une approche Bayésienne. Pour le cas d'étude, il faut se souvenir qu'il y aura toujours deux coefficients de Strickler (lit mineur et lit majeur) à caler pour chaque tronçon défini. Le cas utilisé pour le premier calage utilise seulement 7 tronçon, c'est pour cela que le nombre de paramètres est égal à 14 ;
- Ligne 5 : pour chaque paramètre, un bloc de 4 lignes composé de :
    a) Nom du paramètre. $K_{\mathrm{min_i}}$ pour le coefficient de rugosité du lit mineur du tronçon « i » et $K_{\mathrm{maj_i}}$ pour le lit majeur ;
    b) Valeur initiale du coefficient de Strickler ;
    c) Distribution a priori. Plusieurs distributions sont disponibles sur BaM, mais il semble que pour un modèle hydrodynamique 1D, la distribution LN Normale semble plutôt une bonne approche ;
    d) Paramètres de la distribution a priori (séparés par des virgules). Puisque la distribution est un Log Normal, les a priori doivent être calés à la fonction de probabilité ;
    Toutefois, il ne semble pas évident d'affecter les valeurs. C'est pour cela qu'il est plus simple de penser que si on souhaite un a priori du Ks de 40, qui est plus évident, il faut lui appliquer le logarithme népérien à 40 (Ln (40)) qui est égal à environ 3.7 ;

Dans *BaM*, il est aussi disponible l'option de fixer un paramètre à sa valeur initiale via la pseudo-distribution 'FIX'. Il faut remarquer qu'après la pseudo-distribution, il est nécessaire de laisser une ligne blanche dans la quatrième ligne de chaque bloc paramètre.

Ces étapes se répètent autant des fois qu'indiqué par le nombre de paramètres.

```
"MAGE"                  ! Model ID
1                       ! nX: number of input variables
1                       ! nY: number of output variables
14                      ! nPar: number of parameters theta
"Kmin1"                 ! Parameter Name
11.0                     ! Initial guess
'LogNormal'             ! Prior distribution
3.7,0.5                 ! Prior parameters
"Kmaj1"                 ! Parameter Name
10.                     ! Initial guess
'FIX'                   ! Prior distribution
                        ! Prior parameters
```

Figure 9. Contenu du fichier Confi_Model.txt

2.1.5. Configuration de l'inférence (Config_Data.txt et Config_RemnantSigma.txt)

Pour l'estimation des paramètres, il faut établir pour chaque variable de sortie, les données utilisées avec leur incertitude associée. Une liste plus détaillée se montre ci-après :

- Ligne 1 : chemin d'accès au fichier des variables de sortie. En général, ce fichier contient les valeurs des observations accompagnées ou pas de leurs incertitudes qui servent à comparer aux simulations.
- Ligne 2 : nombre de lignes d'en-tête ;
- Ligne 3 : nombre d'observations dans le fichier. Il faut surveiller que toutes les observations ont la même taille ;
- Ligne 4 : nombre de colonnes dans le fichier ;
- Ligne 5 : colonne(s) contenant les variables d'entrée (autant de colonnes que de variables d'entrée, séparées par des virgules). Dans le présent cas d'étude, le temps a été sélectionné (même s'il n'est pas vraiment) comme variable pour organiser l'information des jaugeages ;
- Ligne 6 : colonne(s) contenant l'écart-type des erreurs non-systématiques. La valeur 0 peut être utilisée si l'on souhaite faire l'hypothèse que toutes ces erreurs sont nulles ;
- Ligne 7 : colonne(s) contenant l'écart-type des erreurs systématiques. La valeur 0 peut être utilisée;
- Ligne 8 : colonne(s) contenant l'index des erreurs systématiques. La valeur 0 peut être utilisée;
- Ligne 9 : colonne(s) contenant les variables de sortie. Pour le premier calage, il s'agit seulement du débit jaugé issus de la campagne de jaugeages ;
- Ligne 10 : colonne(s) contenant l'écart-type des erreurs non-systématiques. Elle correspond à l'incertitude relative du jaugeage. Néanmoins, il n'y avait pas d'information là-dessus et c'est pourquoi une incertitude relative de 5% a été affectée sur l'ensemble des jaugeages;
- Ligne 11 : colonne(s) contenant l'écart-type des erreurs systématiques. La valeur 0 peut être utilisée;
- Ligne 12 : colonne(s) contenant l'index des erreurs systématiques. La valeur 0 peut être utilisée.

```
'/home/brenard/BEN/GitHub/BaM/tests/BaM_MAGE/Seineav3.BAD'              !!! Absolute path to data file
1       !!! number of header lines
498     !!! Nobs, number of rows in data file (excluding header lines)
4       !!! number of columns in the data file
1       !!! columns for X (observed inputs) in data file - comma-separated if several
0       !!! columns for Xu (random uncertainty in X, EXPRESSED AS A STANDARD DEVIATION - use 0 for a no-error assumption)
0       !!! columns for Xb (systematic uncertainty in X, EXPRESSED AS A STANDARD DEVIATION - use 0 for a no-error assumption)
0       !!! columns for Xb_indx (index of systematic errors in X - use 0 for a no-error assumption)
3       !!! columns for Y (observed outputs) in data file - comma-separated if several
4       !!! columns for Yu (uncertainty in Y, EXPRESSED AS A STANDARD DEVIATION - use 0 for a no-error assumption)
0       !!! columns for Yb (systematic uncertainty in Y, EXPRESSED AS A STANDARD DEVIATION - use 0 for a no-error assumption)
0       !!! columns for Yb_indx (index of systematic errors in Y - use 0 for a no-error assumption)
```

Figure 10. Contenu du fichier Confi_Data.txt

Les variables de sorties font référence pour le premier calage aux débit jaugés à la campagne de jaugeages du mars 2017. A terme, l'objectif est de pouvoir caler en fonction non seulement des jaugeages, mais aussi du marégraphe dégradé sur les stations où l'information est disponible. Le contenu du fichier de ces variables se montre ci-après.

```
Temps;Débits;Debit_jauge;Incertitude_jaugeage
0;0;-9999;-9999
0.083;-20.975;-9999;-9999
0.167;-95.78;-9999;-9999
0.25;-149.303;-9999;-9999
0.333;-203.48;-9999;-9999
0.417;-237.651;-9999;-9999
0.5;-264.599;-9999;-9999
0.583;-291.399;-9999;-9999
0.667;-316.949;-9999;-9999
0.75;-340.351;-9999;-9999
0.833;-360.694;-9999;-9999
0.917;-375.797;-9999;-9999
1;-383.166;-9999;-9999
```

Figure 11. Contenu du fichier Seineav3.BAD

En ce qui concerne l'organisation du fichier, il s'appuie sur l'exportation des résultats issus de MAGE, c'est pour cela qu'il est similaire au fichier « .csv » précédemment mentionné. Même si *BaM* n'as pas besoin d'une variable d'entrée, il faut lui donner un paramètre qui permet d'organiser les observations, dans le cas présent, la variable affectée est le temps.

La colonne suivante n'est pas forcément utilisée. En revanche, la troisième et quatrième colonne font référence au débit jaugé et l'incertitude associée respectivement. *BaM* nécessite d'une information de la variable de sortie à chaque pas de temps, pourtant il n'y a pas assez de jaugeages pendant la période de simulation. C'est pourquoi une valeur nulle sera affectée qui sera compris par l'outil via la chiffre « -9999 ».

De plus, il faut spécifier un modèle d'erreurs structurelles pour chaque variable de sortie. L'explication s'effectue ci-après :

- Ligne 1 : identifiant du modèle d'erreurs structurelles ('Constant', 'Linear', 'Exponential' ou 'Gaussian') ;
- Ligne 2 : nombre de paramètres du modèle d'erreurs structurelles ;
- Ligne 3 : pour chaque paramètre, un bloc de 4 lignes composé de :
    a) Nom du paramètre ;
    b) Valeur initiale ;
    c) Distribution a priori ;
    d) Paramètres de la distribution a priori (séparés par des virgules).

```
'Linear'            ! Function f used in sdev=f(Qrc)
2                   ! Number of parameters gamma for f
"intercept"         ! Parameter Name
0.01                ! Initial Guess
'Uniform'           ! Prior distribution
0,100000            ! Prior parameters
"slope"             ! Parameter Name
0.60                ! Initial Guess
'Uniform'           ! Prior distribution
0,100000            ! Prior parameters
```

Figure 12. Contenu du fichier Config_RemnantSigma.txt

2.1.6. Configuration des actions de l'exécutable (Config_MCMC.txt et Config_Cooking.txt)

Le fichier Config_MCMC.txt sert à régler les propriétés de l'algorithme MCMC alors que l'autre fichier établi les conditions du brûlage et affinage des simulations. En d'autres termes, ça sert à enlever les premières simulations influencées par les conditions initiales et ensuite, l'affinage a pour objectif de justifier l'hypothèse d'avoir des simulations non corrélées. L'explication pour le premier fichier s'affiche ci-après en prenant l'information de Renard, B (2017) :

- Ligne 1 : nom du fichier où seront sauvegardées les simulations MCMC ;
- Ligne 2 : nombre de simulations entre chaque adaptation ;
- Ligne 3 : nombre de cycles d'adaptation ;
- Ligne 4 : nombre de cycles d'adaptation ;
- Ligne 5 : taux d'acceptation minimal ;
- Ligne 6 : taux d'acceptation maximal ;
- Ligne 7 : facteur de diminution de l'écart-type de saut ;
- Ligne 8 : facteur d'augmentation de l'écart-type de saut ;
- Ligne 9 : option pour la spécification de l'écart-type de saut initial. L'écart-type de saut initial est égal à un facteur fois la valeur absolue de la valeur initiale du paramètre. Si option=0, ce facteur est identique pour tous les paramètres. Si option=1, ce facteur est spécifié individuellement pour chaque paramètre ;
- Ligne 10 : ligne cosmétique ignorée ;
- Ligne 11 : facteur si option=0 ;
- Ligne 12 : facteurs individuels pour θ si option=1 ;
- Ligne 13 : facteurs individuels pour γ si option=1.

En ce qui concerne le post-traitement des simulations, il est divisé en deux étapes, le brûlage et l'affinage qui seront expliqués par la suite :

- Brulage (« burn »): on efface le début des simulations, afin de laisser le temps à l'algorithme MCMC de converger ;
- Affinage (« slim ») : sur les simulations restant après brulage, on ne conserve qu'une simulation toutes les X

Pour ce faire, il faut régler le fichier Config_Cooking.txt et l'explication s'illustre ci-dessous :

- Ligne 1 : nom du fichier où seront sauvegardées les simulations MCMC « dégraissées »;
- Ligne 2 : facteur de brulage ;
- Ligne 3 : facteur d'affinage.

```
"Results_MCMC.txt" ! File name
10  ! NAdapt
100 ! Ncycles
0.1 ! MinMoveRate
0.5 ! MaxMoveRate
0.9 ! DownMult
1.1 ! UpMult
0   ! Mode for setting the initial Std of the jump distribution
****      DEFINITION OF INITIAL JUMP STD         **** ! Cosmetics
0.1         ! MultFactor in default mode (ignored in manual mode)
0.1,0.1,0.1 ! RC MultFactor in manual mode (ignored in auto mode)
0.1,0.1     ! Remnant MultFactor in manual mode (ignored in auto mode)
```

```
"Results_MCMC_Cooked.txt" ! Result file
0.2  ! BurnFactor
8    ! Nslim
```

Figure 13. Contenu du fichier Config_MCMC.txt (à gauche) et Config_Cooking.txt (à droite)

2.1.7. Résumé des simulations MCMC (Config_Summary.txt et Config_Residuals.txt)

Après avoir effectué l'analyse Bayésienne, il se propose de réaliser des statistiques des simulations afin de décrire les résultats. Ce pour cela que ce fichier a seulement une ligne qui spécifie le fichier où sera sauvegardé le résumé.

```
"Results_Summary.txt" ! Result file
```

Figure 14. Contenu du fichier Config_Summary.txt

Le fichier résultats aura les suivants statistiques : Nombre de simulations, valeur minimum, valeur maximale, range, moyenne, médiane, premier et dernier percentile, premier et troisième quantile, déviation standard, variance, covariance, asymétrie (Skewness), aplatissement (Kurtosis) et Maximum A Posteriori (MAP).

	Kmin1
N	0.500000E+03
Minimum	0.103506E+02
Maximum	0.161284E+03
Range	0.150933E+03
Mean	0.441898E+02
Median	0.398576E+02
Q10%	0.207693E+02
Q25%	0.281303E+02
Q75%	0.542953E+02
Q90%	0.712995E+02
St.Dev.	0.232132E+02
Variance	0.538854E+03
CV	0.525308E+00
Skewness	0.167711E+01
Kurtosis	0.427405E+01
MaxPost	0.302558E+02

Figure 15. Contenu du fichier Results_Summary.txt

Enfin, une comparaison entre les variables de sortie simulées par le modèle avec celles observées est proposée. En ce qui concerne le fichier configuration, il semble à celui d'avant, car il y aura seulement une ligne qui va créer le fichier où les résultats seront sauvés.

```
"Results_Residuals.txt" ! Result file
```

Figure 16. Contenu du fichier Config_Residuals.txt

Ci-dessous, il s'explique l'organisation du fichier résultats.

- N_X colonnes contenant les variables d'entrée observées;
- N_X colonnes contenant les variables d'entrée estimées vraies. Si les données d'entrée sont supposées sans erreur, alors, la variable d'entrée vraie est égale à la variable d'entrée observée;
- N_Y colonnes contenant les variables de sortie observées;
- N_Y colonnes contenant les variables de sortie débiaisées;
- N_Y colonnes contenant les variables de sortie simulées ;

- N_Y colonnes contenant les résidus pour chaque variable de sortie. Ces résidus sont calculés en considérant les observations débiaisées ;
- N_Y colonnes contenant les résidus standardisés pour chaque variable de sortie. Si les modèles d'erreurs sont corrects (à la fois pour les erreurs de mesure et les erreurs structurelles), ces résidus standardisés devraient suivre une loi normale centrée réduite N (0,1).

X1_obs	X1_true	Y1_obs	Y1_unbiased	Y1_sim	Y1_res	Y1_stdres
0.000000E+00	0.000000E+00	-0.999990E+05	-0.999990E+05	0.000000E+00	-0.999990E+05	-0.100000E+01
0.830000E-01	0.830000E-01	-0.999990E+05	-0.999990E+05	-0.209680E+02	-0.999780E+05	-0.999790E+00
0.167000E+00	0.167000E+00	-0.999990E+05	-0.999990E+05	-0.956270E+02	-0.999034E+05	-0.999043E+00
0.250000E+00	0.250000E+00	-0.999990E+05	-0.999990E+05	-0.157434E+03	-0.998416E+05	-0.998425E+00
0.333000E+00	0.333000E+00	-0.999990E+05	-0.999990E+05	-0.206596E+03	-0.997924E+05	-0.997934E+00
0.417000E+00	0.417000E+00	-0.999990E+05	-0.999990E+05	-0.235412E+03	-0.997636E+05	-0.997645E+00
0.500000E+00	0.500000E+00	-0.999990E+05	-0.999990E+05	-0.260405E+03	-0.997386E+05	-0.997396E+00
0.583000E+00	0.583000E+00	-0.999990E+05	-0.999990E+05	-0.285361E+03	-0.997136E+05	-0.997146E+00
0.667000E+00	0.667000E+00	-0.999990E+05	-0.999990E+05	-0.309607E+03	-0.996894E+05	-0.996903E+00
0.750000E+00	0.750000E+00	-0.999990E+05	-0.999990E+05	-0.331924E+03	-0.996671E+05	-0.996680E+00
0.833000E+00	0.833000E+00	-0.999990E+05	-0.999990E+05	-0.351459E+03	-0.996475E+05	-0.996485E+00
0.917000E+00	0.917000E+00	-0.999990E+05	-0.999990E+05	-0.367015E+03	-0.996320E+05	-0.996329E+00
0.100000E+01	0.100000E+01	-0.999990E+05	-0.999990E+05	-0.377489E+03	-0.996215E+05	-0.996225E+00

Figure 17. Contenu du fichier Results_Residuals.txt

2.1.8. Configuration pour la prédiction et la propagation d'incertitude (Config_Pred_Master.txt)

Après avoir réalisé l'étape de calage, on poursuit avec la prédiction et la propagation des incertitudes. Pour ce faire, il faut désactiver les trois premières options du fichier « Config_RunOptions.txt » et activer la dernière de sorte que l'outil fasse la propagation à partir des résultats obtenus dans l'étape précédente.

```
.false.          ! Do MCMC sampling?
.false.          ! Do MCMC summarizing?
.false.          ! Do residual analysis?
.true.           ! Do prediction experiments?
```

Figure 18. Contenu du fichier Config_RunOptions.txt étape de prédiction

Ensuite, le fichier principal, par rapport aux prédictions, est « Config_Pred_Master.txt ». Il permettra de piloter la quantité de prédictions à réaliser et les noms des fichiers à lire avec leur propre paramétrage. L'explication de chaque ligne s'expose ci-après :

- Ligne 1. Nombre d'expérience de prédictions à réaliser ;
- Ligne 2. Fichier de configuration pour l'expérience 1 ;
- Ligne 3. Fichier de configuration pour l'expérience 2 ;
- Ligne 4. etc.

```
4                          ! Number of prediction experiments
"Config_Pred_Prior.txt"    ! Config file for experiment 1
"Config_Pred_ParamU.txt"   ! Config file for experiment 2
"Config_Pred_Maxpost.txt"  ! Config file for experiment 3
"Config_Pred_TotalU.txt"   ! Config file for experiment 4
```

Figure 19. Contenu du fichier Config_Pred_Master.txt

2.1.9. Configuration des fichiers de prédiction

Autant fichiers de configuration doivent être écrits que le nombre d'expérience de prédiction. Le format de l'écriture se conserve d'une prédiction à l'autre. Le format à respecter est le suivant :

- Ligne 1. Chemins d'accès aux spaghettis pour chaque variable d'entrée : dans BaM, c'est l'utilisateur qui doit fournir les valeurs des variables d'entrée utilisées en prédiction. De plus, il est possible de fournir des spaghettis pour ces variables d'entrée, i.e. plusieurs réplications représentant l'incertitude (chaque spaghetti étant une colonne dans le fichier de données). Utiliser un unique spaghetti revient à ignorer toute incertitude sur la variable d'entrée.
- Ligne 2. Nombre d'observations dans chaque spaghetti : nombre de lignes dans chaque fichier de spaghettis. Ce nombre

est commun à toutes les variables d'entrée.

- Ligne 3. Nombre de spaghettis pour chaque variable d'entrée : nombre de colonnes dans chaque fichier de spaghettis.
- Ligne 4. Propagation de l'incertitude paramétrique ?
- Ligne 5. Propagation de l'incertitude structurelle ? A spécifier pour chaque variable de sortie.
- Ligne 6. Nombre de simulations dans le cas d'une prédiction a priori. Pour une simulation à partir de la distribution a posteriori, mettre un nombre négatif. Les paramètres seront alors issus des simulations MCMC. Un nombre positif sera interprété comme une demande pour réaliser une simulation à partir de la distribution a priori, et donnera le nombre de simulations réalisées.
- Ligne 7. Fichiers résultats pour chaque variable de sortie (spaghettis). Autant de fichiers que de variables de sortie.
- Ligne 8. Transposer les spaghettis ? Par défaut les spaghettis sont écrits « à l'envers » dans le fichier résultat : chaque ligne est un spaghetti (pour des raisons liées à l'algorithme de prédiction). Cette action permet de remettre les spaghettis à la verticale. A spécifier pour chaque variable de sortie.
- Ligne 9. Calculer les enveloppes d'incertitude ? A spécifier pour chaque variable de sortie.
- Ligne 10. Fichiers résultats pour chaque variable de sortie (enveloppes). Autant de fichiers que de variables de sortie.
- Ligne 11. Afficher une « barre de progrès » en cours de calcul ?
- Ligne 12. Effectuer la prédiction pour les variables d'état ?
- Ligne 13. Fichiers résultats pour chaque variable d'état (spaghettis). Autant de fichiers que de variables d'état.
- Ligne 14. Transposer les spaghettis ? A spécifier pour chaque variable d'état.
- Ligne 15. Calculer les enveloppes d'incertitude? A spécifier pour chaque variable d'état.
- Ligne 16. Fichiers résultats pour chaque variable d'état (enveloppes). Autant de fichiers que de variables d'état.

```
"/home/brenard/BEN/GitHub/BaM/tests/BaM_MAGE/X1.pred" ! Files containing spaghettis for each input variable (size nX)
498                  ! Nobs, number of observations per spaghetti (common to all files!)
1                    ! Nspag, number of spaghettis for each input variable (size nX)
.true.               ! Propagate parametric uncertainty?
.true.               ! Propagate remnant uncertainty for each output variable? (size nY)
-1                   ! Nsim[prior]. If <=0: posterior sampling (nsim is given by mcmc sample); if >0: sample nsim replicates from prior distribution
"q_TotalU_Rouen.spag"! Files containing spaghettis for each output variable (size nY)
.true.               ! Post-processing: transpose spag file (so that each column is a spaghetti)? (size nY)
.true.               ! Post-processing: create envelops? (size nY)
"q_TotalU_Rouen.env" ! Post-processing: name of envelop files (size nY)
.true.               ! Print progress in console during computations?
.false.              ! Do state prediction? (size nState)
                     !  Files containing spaghettis for each state variable (size nState)
                     ! Post-processing: transpose spag file (so that each column is a spaghetti)? (size nState)
                     ! Post-processing: create envelops? (size nState)
                     ! Post-processing: name of envelop files (size nState)
```

Figure 20. Contenu du fichier Config_Pred_Prior.txt

2.2. Analyse du code source de *BaM*

Cette section est divisée en deux parties, la première porte sur l'explication de l'implémentation d'un nouveau modèle notamment le cas du MAGE dans *BaM*. La deuxième partie fait référence au paramétrage sur CodeBlocks afin d'avoir une trace pour les utilisateurs qui vont collaborer à l'intégration des nouveaux modèles au *Framework* à l'avenir.

Avant de commencer, ça vaut la peine de rappeler que le répertoire du code source de *BaM* se trouve sur le GitHub : https://github.com/BaM-tools/BaM.

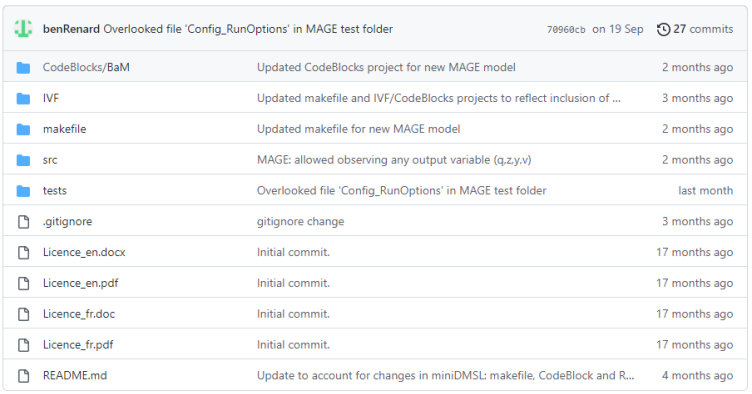


Figure 21. GitHub de BaM

Il est proposé un projet sur CodeBlocks qui a déjà tous les modèles chargés et les librairies dont il a besoin pour être lancé.

2.2.1. Implémentation du modèle MAGE

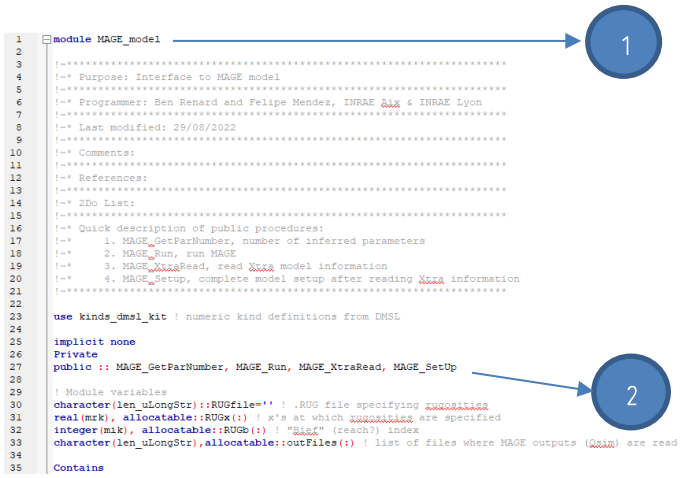
Dans le cas général, le branchement de n'importe quel modèle s'effectue en deux étapes. La première est la création d'un nouveau module qui sera répertorié dans le chemin : BaM/src/Models et ensuite, annoncer l'existence du modèle à *BaM* via le fichier « ModelLibrary_tools.f90. ». Cette dernière étape se réalisé en plusieurs pas montrés ci-après :

a) Annoncer qu'un nouveau modèle est disponible. Pour ce faire, il faut écrire la ligne suivante de code : *use MAGE_model ;*
b) Le nom du modèle doit être déclaré de la manière suivante : *MDL_MAGE="MDL_MAGE" ;*
c) Ajouter un « case » pour appeler (call) la fonction *MAGE_GetParNumber* codé dans la sous-routine *GetModelParNumber* ;
d) Ajouter un « case » pour appeler (call) la fonction *MAGE_Run* codé dans la sous-routine *ApplyModel* ;
e) Ajouter un « case » pour appeler (call) la fonction *MAGE_XtraRead* codé dans la sous-routine *XtraRead ;*
f) Ajouter un « case » pour définir le nombre et le nom des paramètres dérivés et variables d'état (pour le moment, il n'y en a pas). Toutefois, il a été codé et pour l'appeler (call) la fonction *MAGE_setUp* codé dans la sous-routine *XtraSetup*.

A partir de ce moment, le modèle est branché au *Framework BaM*. Maintenant, il faut passer à l'explication des fonctions mentionnées précédemment pour bien comprendre son fonctionnement interne (première étape).

- **Déclaration des variables et des fonctions :**

  - Le plus simple pour démarrer et avoir une idée de la structure à suivre pour l'implémentation d'un nouveau modèle est de copier un modèle existant (par exemple : Linear_model.f90) ;
  - Ensuite il faut renommer le module comme le fichier au début et à la fin du code (circule 1 dans la Figure 22) ;
  - Mettre à jour les commentaires ;
  - Trier les fonctions dont on a besoin. Pour le cas d'étude, il est nécessaire de 4 fonctions pour lancer le modèle et son calage (circule 2 dans la Figure 22) ;
  - **Contains** : définition des fonctions.




```
module MAGE_model

!~*****************************************************************
!~* Purpose: Interface to MAGE model
!~*****************************************************************
!~* Programmer: Ben Renard and Felipe Mendez, INRAE Aix & INRAE Lyon
!~*****************************************************************
!~* Last modified: 29/08/2022
!~*****************************************************************
!~* Comments:
!~*****************************************************************
!~* References:
!~*****************************************************************
!~* 2Do List:
!~*****************************************************************
!~* Quick description of public procedures:
!~*      1. MAGE_GetParNumber, number of inferred parameters
!~*      2. MAGE_Run, run MAGE
!~*      3. MAGE_XtraRead, read Xtra model information
!~*      4. MAGE_Setup, complete model setup after reading Xtra information
!~*****************************************************************

use kinds_dmsl_kit ! numeric kind definitions from DMSL

implicit none
Private
public :: MAGE_GetParNumber, MAGE_Run, MAGE_XtraRead, MAGE_SetUp

! Module variables
character(len_uLongStr)::RUGfile='' ! .RUG file specifying roughness
real(mrk), allocatable::RUGx(:) ! x's at which roughness are specified
integer(mik), allocatable::RUGb(:) ! "Risk" (reach?) index
character(len_uLongStr),allocatable::outFiles(:) ! list of files where MAGE outputs (Qsim) are read

Contains
```

Figure 22. Création du module MAGE_model

- **Fonction MAGE_GetParNumber :** cette fonction a pour but de déterminer la quantité des paramètres à caler pour créer le vecteur paramètres à rendre afin de lancer la méthode.

  - **npar :** quantité de tronçons avec un coefficient de rugosité constant dans tout sa longueur, ça donnera la taille du vecteur paramètre. Il faut multiplier par 2 parce qu'il y en a pour le lit mineur et une autre pour le lit majeur.

```
subroutine MAGE_GetParNumber(npar,err,mess)
!^*****************************************************************
!^* Purpose: number of inferred parameters
!^*          = number of "xxxxxxxx" * 2 (main channel and floodplain)
!^*****************************************************************
!^* Programmer: Ben Renard and Felipe Mendez, INRAE Aix & INRAE Lyon
!^*****************************************************************
!^* Last modified:29/08/2022
!^*****************************************************************
!^* Comments:
!^*****************************************************************
!^* References:
!^*****************************************************************
!^* 2Do List:
!^*****************************************************************
!^* IN
!^*      nothing
!^* OUT
!^*      1. nparr, par. number
!^*      2.err, error code; <0:Warning, ==0:OK, >0: Error
!^*      3.mess, error message
!^*****************************************************************
integer(mik), intent(out)::npar,err
character(*),intent(out)::mess
! locals
character(250),parameter::procname='MAGE_GetParNumber'

err=0;mess=''
npar=size(RUGb)*2 ! For each reach, one parameter for main channel and one for floodplain

end subroutine MAGE_GetParNumber
```

Figure 23. Création de la fonction MAGE_GetParNumber

- **Fonction MAGE_Run :** l'idée est de trouver le chemin correct pour qu'il puisse lancer l'exécutable de MAGE à chaque itération, en recalculant les valeurs du vecteur paramètre pour estimer les débits à une section en particulière.

```
subroutine MAGE_Run(exeFile,projectDir,REPfile,theta,Y,feas,err,mess)
!^*****************************************************************************
!^* Purpose: Run MAGE
!^*****************************************************************************
!^* Programmer: Ben Renard and Felipe Mendez, INRAE Aix & INRAE Lyon
!^*****************************************************************************
!^* Last modified: 29/08/2022
!^*****************************************************************************
!^* Comments:
!^*****************************************************************************
!^* References:
!^*****************************************************************************
!^* 2Do List:
!^*****************************************************************************
!^* IN
!^*      1. exeFile, MAGE executable
!^*      2. projectDir, MAGE project directory
!^*      3. REPfile, MAGE .REP file
!^*      4. theta, parameter vector
!^* OUT
!^*      1. Y, discharge computed at requested points
!^*      3. feas, feasible?
!^*      4. err, error code; <0:Warning, ==0:OK, >0: Error
!^*      5. mess, error message
!^*****************************************************************************
use utilities_dmsl_kit,only:getSpareUnit
use DataRW_tools,only:DatRead
character(*), intent(in)::exeFile,projectDir,REPfile
real(mrk), intent(in)::theta(:)
real(mrk), intent(out)::Y(:,:)
logical, intent(out)::feas
integer(mik), intent(out)::err
character(*),intent(out)::mess
!locals
character(250),parameter::procname='MAGE_Run'
integer(mik),parameter::outFileNCOL=2
integer(mik)::i,n,unt
character(250)::forma,head(outFileNCOL)
character(len_uLongStr)::cmdString
real(mrk), pointer::foo(:,:)
```

Figure 24. Création de la fonction MAGE_Run

- La fonction aura besoin d'une autre fonction codé dans le repértoire « miniDMSL/utilities_dmsl_kit.f90 » appelée *getSpareUnit*, ainsi que la fonction *DatRead* répertorié dans le fichier « DataRW_tools » placé dans le dossier de « BMSL » ;

  Ensuite, il faut déclarer les variables à utiliser en entrée, en sortie et de type logique pour les conditions ;

- **exeFile :** chemin du répertoire où l'exécutable MAGE se trouve ;
- **projectDir :** chemin du répertoire du projet MAGE. Dossier avec tous les fichiers issus après lancement de l'exécutable ;
- **REPfile :** fichier qui sert à piloter et à trouver les fichiers dont l'outil a besoin afin de lancer MAGE, notamment le fichier « .RUG » qui a les valeurs des coefficients de rugosité ;
- **theta :** vecteur paramètre à caler via la méthode MCMC ;
- **Y :** valeurs du débit modélisé via MAGE sur une section définie ;
- **feas :** faisabilité du calcul. Variable logique ;

- **err :** détection d'une erreur sur le calcul, valeurs possibles -1, 0, 1 ;
- **mess :** message d'erreur qui s'affiche si err ≠0.

```
! Init
err=0;mess='';feas=.true.;Y=undefRN
n=size(RUGb)
forma='(A1,I3,6x,2F10.0,2F10.2)'

! Write Theta as a .RUG file
call getSpareUnit(unt,err,mess)
if(err/=0) then;mess=trim(procname)//':'//trim(mess);return;endif
open(unit=unt,file=trim(projectDir)//trim(RUGfile),status='replace',iostat=err)
if(err/=0) then;mess=trim(procname)//':problem creating RUG file';return;endif
write(unt,'(A)') '**************** STRICKLERS **********************'
write(unt,'(A)') '*Bif        x deb     x fin     K min     K moy  '
write(unt,'(A)') '*---------=========----------=====================----------'
do i=1,n
    write(unt,forma) 'K',RUGb(i),RUGx(i),RUGx(i+1),theta(2*i-1),theta(2*i)
enddo
close(unt)
```

Figure 25. Ecriture des coefficients de rugosité

- Maintenant, il faut initialiser les variables « err », « mess », « feas » et « Y » ;
- **n :** variable qui permettra de dimensionner la taille du vecteur paramètre à caler ;
- **forma :** format à fournir sur fortran pour qu'il comprenne la façon d'écriture du fichier « .RUG », autrement dit des coefficients de rugosité qui sera le vecteur paramètre ;
- Ensuite, l'objectif est d'écrire le vecteur paramètre ou vecteur theta comme fichier « .RUG » et pour ce faire, il faudrait appeler la fonction *GetSpareUnit* qui contient l'information nécessaire de la variable à caler ;
- Il est souvent proposé des tests pour monitorer s'il y a eu une erreur entretemps afin d'arrêter le processus ;
- Une fois vérifié l'absence d'erreur, il se procède à ouvrir le fichier « .RUG » existant avec le commande de remplacer. C'est pour cela qu'il sera possible de modifier après avoir recalculé et ainsi de suite. Vu qu'il faut respecter le format, un en-tête a été proposé afin de suivre exactement le format résultat de MAGE ;
- Enfin, le fichier « .RUG » s'écrit en prenant en compte :
    - **RUGb :** bief concernant ;
    - **RUGx :** point kilométrique ;
    - **Theta :** vecteur paramètre.

```
! Call mage
cmdString='cd '//trim(projectDir)//'&&'//trim(exeFile)//' '//trim(REPfile)
call execute_command_line (trim(cmdString),wait=.true.,exitStat=err,cmdMsg=mess)
if(err/=0) then;mess=trim(procname)//':'//trim(mess);return;endif

! Read outputs into Y
do i=1,size(outFiles)
    call DatRead(file=outFiles(i),ncol=outFileNCOL,y=foo,headers=head,err=err,mess=mess)
    if(err/=0) then;mess=trim(procname)//':'//trim(mess);return;endif
    if( size(foo,dim=1) /= size(Y,dim=1) ) then ! MAGE didn't run properly
        feas=.false.;return
    endif
    Y(:,i)=foo(:,outFileNCOL)
enddo

end subroutine MAGE_Run
```

Figure 26. Lancement d'exécutable MAGE et calcul du débit après simulation

- Après avoir réécrit le fichier avec des coefficients, il faut lancer l'exécutable MAGE pour lancer la simulation avec la nouvelle configuration.
- Enfin, le but est de récupérer les débits simulés par le logiciel afin de comparer avec les observations et pouvoir réaliser le calage.

- **Fonction MAGE_XtraRead :** lecture d'information extra. Par exemple, variables d'état. Dans le présent cas, il n'y en pas d'information extra.

```
151 subroutine MAGE_XtraRead(file,xtra,err,mess)
152 !^**********************************************************************
153 !^* Purpose: Read Xtra information
154 !^**********************************************************************
155 !^* Programmer: Ben Renard and Felipe Mendez, INRAE Aix & INRAE Lyon
156 !^**********************************************************************
157 !^* Last modified: 29/08/2022
158 !^**********************************************************************
159 !^* Comments:
160 !^**********************************************************************
161 !^* References:
162 !^**********************************************************************
163 !^* 2Do List:
164 !^**********************************************************************
165 !^* IN
166 !^*     1. file, Xtra file
167 !^* OUT
168 !^*     1. xtra, xtra information
169 !^*     2. err, error code; <0:Warning, ==0:OK, >0: Error
170 !^*     3. mess, error message
171 !^**********************************************************************
172 use types_dmsl_kit, only:data_ricz_type
173 use utilities_dmsl_kit,only:getSpareUnit
174 character(*), intent(in)::file
175 type(data_ricz_type),intent(out):: xtra
176 integer(mik), intent(out)::err
177 character(*),intent(out)::mess
178 ! locals
179 character(250),parameter::procname='MAGE_XtraRead'
180 integer(mik)::unt
181
182 err=0;mess=''
183
184 call getSpareUnit(unt,err,mess)
185 if(err/=0) then;mess=trim(procname)//':'//trim(mess);return;endif
186 open(unit=unt,file=trim(file), status='old', iostat=err)
187 if(err>0) then;mess=trim(procname)//':problem opening file '//trim(file);return;endif
188 read(unt,*,iostat=err) xtra%cs1 ! 1. exe file
189 if(err/=0) then;mess=trim(procname)//':problem reading file '//trim(file);return;endif
190 read(unt,*,iostat=err) xtra%cs2 ! 2. Project Directory
191 if(err/=0) then;mess=trim(procname)//':problem reading file '//trim(file);return;endif
192 read(unt,*,iostat=err) xtra%cs3 ! 3. REP file
193 if(err/=0) then;mess=trim(procname)//':problem reading file '//trim(file);return;endif
194 close(unt)
195
196 end subroutine MAGE_XtraRead
```

Figure 27. Création de la fonction MAGE_XtraRead

- **Fonction MAGE_SetUp :** permet de repérer le répertoire du fichier « .REP » et ensuite, elle fait non seulement la connexion, mais aussi son interprétation.

```
200 subroutine MAGE_setUp(projectDir,REPfile,err,mess)
201 !^**********************************************************************
202 !^* Purpose: Finalize MAGE setup after having read Xtra information. In
203 !^*          particular this sub defines module variables (RUG stuff and outFiles)
204 !^**********************************************************************
205 !^* Programmer: Ben Renard and Felipe Mendez, INRAE Aix & INRAE Lyon
206 !^**********************************************************************
207 !^* Last modified: 29/08/2022
208 !^**********************************************************************
209 !^* Comments:
210 !^**********************************************************************
211 !^* References:
212 !^**********************************************************************
213 !^* 2Do List:
214 !^**********************************************************************
215 !^* IN
216 !^*     1. projectDir, MAGE project directory
217 !^*     2. REPfile, MAGE .REP file
218 !^* OUT
219 !^*     1.err, error code; <0:Warning, ==0:OK, >0: Error
220 !^*     2.mess, error message
221 !^**********************************************************************
222 use utilities_dmsl_kit,only:getNumItemsInFile,getSpareUnit
223 character(*), intent(in)::projectDir,REPfile
224 integer(mik), intent(out)::err
225 character(*),intent(out)::mess
226 !locals
227 character(250),parameter::procname='MAGE_setUp'
228 integer(mik)::unt,i,n,k
229 character(250)::foo,prefix,var,ib,pk,mode,dt0,forma,bar
230 character(len_uLongStr)::line,fname
231 character(len_uLongStr),allocatable::fnames(:)
232 real(mrk)::c1,c2
```

Figure 28. Création de la fonction MAGE_setUp

- Afin de lancer la fonction, il faut utiliser la fonction *getNumItemsInFile* et *getSpareUnit* disponibles dans le dossier « miniDMSL/utilities_dmsl_kit.f90 ».
- **projectDir :** chemin d'accès au dossier résultat de MAGE après simulation ;
- **REPfile :** nom du fichier « .REP » qui sert à guider à *BaM* pour repérer les fichiers qu'il faut lire pour le calage automatique.

Figure 29. Ouverture et interprétation du fichier REP

- En premier lieu, il faut que *BaM* réussit à ouvrir le fichier REP, cela est réalisée dans la première partie du code ;
- Ensuite, il faut l'interpréter en fonction des besoins. Pour le calage du modèle hydrodynamique, il a besoin de lire le fichier qui associe les coefficients de rugosité (RUG) et les variables de sortie du modèle après simulation (CSV) autrement dit le débit, la hauteur, entre autres ;
- Enfin, il faut spécifier la variable à récupérer depuis MAGE en respectant le format propre pour l'exportation des fichiers .csv.

Figure 30. Lecture du fichier RUG

- La dernière partie de la sous-routine a pour but de lire le fichier qui lie les coefficients de rugosité, parce qu'il faut le lire à chaque itération puisqu'il va changer tout le temps.

2.2.2. Paramétrage sur CodeBlocks

Même s'il y a un projet déjà sur le GitHub, il est aussi possible de le créer directement sur le *integrated development environment (IDE)*, notamment CodeBlocks. Pour ce faire, il faut prendre en compte la procédure suivante pour bien monter le projet en prenant comme référent une guide élaborée par Matteo Darienzo.

1. Installer CodeBlocks, la version avec gfortran incorporé sur windows ;
2. Ouvrir CodeBlocks et créer un nouveau projet ;
3. Sélectionner Fortran application ;
4. Donner un nom au projet ;
5. Sélectionner le compilateur (par exemple : GNU fortran compiler)
6. Le projet est créé, ensuite, il faut supprimer le fichier Fortran Sources/main.f90 créé par défaut.
7. Ajouter les fichiers suivants de manière récursive (click droite sur workspace et ensuite, choisir l'option « *add files recursively* ». Une fenêtre s'ouvre on vous demandant les fichiers à télécharger. Décocher tous les fichiers qui ne sont pas utilisés:
    a. Charge le dossier « MiniDMSL » (depuis miniDMSL/src)
    Cocher les fichiers suivants:
        - DynamicsModelLibrary. f90
        - Kinds_dmsl_kit. f90
        - Linalg_dmsl_kit.f90

- Mt19937-64.f95
- Numerix_dmsl_kit.f90
- Types_dmsl_kit.f90
- Uniran1_minidmsl_mod.f90
- Utilities_dmsl_kit.f90

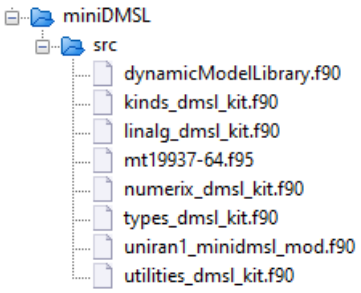


Figure 31. Fichiers du dossier MiniDMSL à charger

b. Charge le dossier « BMSL » (depuis BMSL/src)
Cocher les fichiers suivants:
   - Aggregation_tools.f90
   - BayesianEstimation_tools.f90
   - DataRW_tools.f90
   - Dates_tools.f90
   - EspiricalStats_tools.f90
   - Geodesy_tools.f90
   - MCMC_tools.f90
   - MCMCStrategy_tools.f90
   - TimeSeries_tools.f90

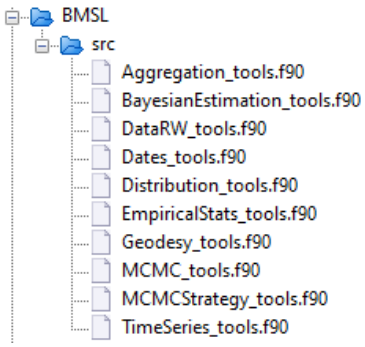


Figure 32. Fichiers du dossier BMSL à charger

c. Charger dossier Models (from BaM/src/Models) :
Charger tous les fichiers disponibles.

d. Charger BaM main files et éviter de charger le dossier « *old* »
   - BaM_main.f90
   - BaM_tools.f90
   - ModelLibrary_tools.f90

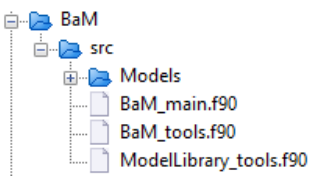


Figure 33. Fichiers du dossier BaM à charger

Il a été montré la manière de monter son propre modèle sur CodeBlocks. Pourtant, il est conseillé de travailler avec le projet disponible sur le GitHub afin de prendre le paramétrage exact de la version du serveur.

3. RBaM (R package)

Ce package a été développé par Benjamin Renard avec le but d'assister non seulement à l'écriture des fichiers de configuration, mais aussi au lancement de l'exécutable BaM.exe. J'ai adapté le script proposé dans le GitHub de *BaM* afin de mettre en place la modification automatique des fichiers pour le cas de MAGE. Je présenterai au fur et à mesure les différents morceaux du script avec son explication respective. Le cas d'exemple utilisé et un modèle avec quatre observations notamment une campagne de jaugeages et plusieurs enregistrements de lignes d'eau.

Avant de lancer le script, il faut forcément respecter une arborescence des dossiers. Autrement dit, la routine R doit être sauvegardée au même niveau du dossier « BaM_MAGE » (*workspace*). En descendant dans l'arborescence, il faut que les trois dossiers affichés dans la figure ci-après soient disponibles.

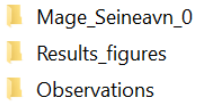


Figure 34. Dossiers de base dans l'espace du travail

Le premier dossier contient les fichiers issus de MAGE. Ensuite, le suivant sert à sauvegarder les différents plots générés par la routine et enfin, le dossier « Observations » dispose de l'information de toutes les variables de sortie qui peuvent servir à l'étape du calage.

Après avoir pris en compte ces considérations, il est possible de se lancer sur le script. En premier lieu, il faut appeler les librairies dont on aura besoin pour exécuter des fonctions qui permettent non seulement le traitement de données, mais aussi la représentation graphique des résultats.

```
# Libraries:
library(RBaM)
library("xlsx")
library("readxl")
library(stringr)
library(ggplot2)
```

Figure 35. Fichiers du dossier BaM à charger

Ensuite, certaines variables doivent être déclarées, car elles serviront à stocker informations dans des listes. La plupart du temps, il existe des commentaires à côté des variables pour expliquer plus en détail leur rôle.

```
# Creation of variable to stock information:
plot_obs_pred <- c()          # list of plots which compare the data of simulation and prediction
station_name_df <- c()        # data frame with stations' names
pred_var_name <- c()          # list with names of each variable to propagate (q,z) for each station
pred_confi_file_name <- c()   # vector with setting prediction file name (Config_Pred_"type of prediction")
doStructural_logical <- c()   # logical vector with setting in function of type of prediction
doParametric_logical <- c()   # logical vector with setting in function of type of prediction
spag_fig_cum <- c()           # variable to accumulate different spaghetti and then, create a plot
priorNsim_int <- c()          # list of number of simulation for propagation
pred_list <- c()              #list with information about all predictions
```

Figure 36. Déclaration des variables

Puis, il faut affecter certaines variables avec l'information initiale en fonction du cas d'étude en analyse. Dans ce cas-là, l'objectif est de réaliser l'étude en mars 2017 en transférant la même information d'un modèle existant d'autre logiciel appelé Mascaret.

De même, étant donné qu'il y a quatre variables de sortie ou observations (une campagne de jaugeages et trois limnigrammes), il faut affecter une incertitude aux données, car elles n'en ont pas, elle a été affectée de 8%.

En ce qui concerne les mesures de hauteur d'eau, la fréquence d'enregistrement de données est très élevée. C'est pourquoi, un

filtrage est proposé afin de dégrader l'observation pour empêcher que les limnigramme prennent beaucoup de poids dans la rechercher du vecteur optimal via l'approche Bayésienne.

```
# Initialization of variables:

# Initial date of simulation set by Mascaret model
intial_date_time_mascaret = as.POSIXct(strptime(as.character("2017-03-28 00:10:00"),
                                                format="%Y-%m-%d %H:%M:%S",
                                                tz = "GMT"))
slim_tide_gauge <- 10    # Slim factor: only one iteration every slim_tide_gauge is kept.
uncertainty <- 8         # Relative uncertainty associated to observations
nX=1                     # Number of input variables
nY=4                     # Number of output variables
```

Figure 37. Initialisation des quelques variables d'entrée

Ensuite, il faut paramètre quelques variables concernées à la prédiction. A ce stade, quatre types de prédiction sont supportés comme la figure ci-dessous illustre. Le script peut être lancé automatiquement si au moins les trois derniers types de prédiction sont activés, sinon il faudrait modifier le script à la main quand il fera le processus.

```
# Prediction to calculate. Only these options are supported: "Prior","ParamU","Maxpost","TotalU"
# Be careful: we must to estimate at least the last three prediction to run the script.
# Only Prior prediction could be neglect
prediction_file <- c("Prior","ParamU","Maxpost","TotalU")
n_prediction <- length(prediction_file)
nsim_prior <- 100  # number of simulation for propagation (def : 100 for priori prediction)
```

Figure 38. Initialisation des variables concernées à la prédiction

Enfin, il se propose une option pour que l'usager lance l'étape de calibration et/ou l'étape de prédiction. De plus, il faut affecter le paramétrage du brûlage des données après calibration, ainsi que l'affinage. La dernière variable montrée à la fin de la suivante figure illustre un vecteur qui a seulement pour but de sauvegarder l'information de la période de simulation.

```
# Activate / disactivate Calibration or prediction step
run_option_calibration = F # logical value to run calibration or not
run_option_prediction = F  # logical value to run prediction or not

# Settings of parameters for mcmc cooking
nCycles = 100 # Number of cycles
burn=0.5      # Percentage of data burned
nSlim=10      # Slim factor: after burning, only one iteration every Nslim is kept.

################
# Don't change!!!
# Variable to stock simulation period
sim_time <- rep(0,2)  # creation a vector with start and end simulation time.
```

Figure 39. Variables d'entrée à affecter par l'utilisateur

Maintenant, il faut créer un vecteur avec le point kilométrique (PK) et les noms des stations de contrôle. En d'autres termes, les stations qui disposent d'information qui peut servir comme des variables de sortie ou des observations.

```
# Control station : information about tide gauge or gauging measurement
#                   (must be updated if user wants the results in another KP)
KP_station=c(0,16823,27439,41568,51186,58878,65055,78482,85578,
             97091,111227,116862,123562,130623,137983)
name_station=c("Poses","Elbeuf","Oissel","Rouen", "Petit Couronne",
               "La Bouille", "Val_des_leux","Duclair","Mesnil_sous_Jumieges",
               "Heurteauville","Caudebec","Vatteville","Aizier","Saint_Leornard",
               "Tancarville")
```

Figure 40. Vecteur avec les stations qui servent comme des observations

A ce stade, il est possible de commencer à écrire l'information des fichiers dont *BaM* a besoin pour être lancé. C'est pour cela que le premier fichier à écrire sera les coefficients de rugosité. Pour ce faire, l'écriture de ce fichier sera réalisée à partir de la fonction « parameter » disponible sur RBaM.

Cette fonction contient entre trois et quatre arguments qui sont :

- name : nom de la variable
- init : valeur initiale pour démarrer le calcul
- prior.dist : distribution de probabilité de l'a priori. Dans ce cas-là, si le paramètre ne varie pas, il faut affecter comme « FIX » pour que *BaM* comprenne
- prior.par : valeurs initiales des paramètres de la fonction de probabilité choisie

```
# File with parameters' information

# Reading the number of Strickler coefficients in MAGE model
Proj_dir=paste0(workspace,'/Mage_Seineavn_0/')
rug_file = read.table(file=paste0(Proj_dir,"Seineav1.RUG"),comment.char = "*", header = F)

n_reach=nrow(rug_file)
Ks=vector(mode='list',length=2*n_reach)

for (i in 1:n_reach){
  Kmin=parameter(name=paste0('Kmin',i),init=37,prior.dist='LogNormal',prior.par=c(3.7,0.1))
  Kmaj=parameter(name=paste0('Kmaj',i),init=10,prior.dist='FIX')
  Ks[[2*i-1]]=Kmin
  Ks[[2*i]]=Kmaj
}
```

Figure 41. Ecriture de l'information des paramètres

Maintenant, l'objectif est d'écrire le fichier « Config_setup.txt » et enfin, on aura toute l'information nécessaire pour monter seulement le modèle MAGE sur BaM.

```
# use xtraModelInfo to pass the names of the inputs and the formulas
MAGE_executable='C:/Users/famendezrios/Documents/BaM_MAGE/Mage_BaM_maregraphe_jau/MAGE/mage'
Repfile='Seineav1.REP'

xtra=xtraModelInfo(fname="Config_setup.txt",object=c(MAGE_executable,Proj_dir,Repfile))

# model
mod=model(ID='MAGE',nX=nX,nY=nY,par=Ks,xtra=xtra)
```

Figure 30. Ecriture du fichier du « Config_setup.txt » et création du modèle à fournir dans BaM

Ensuite, on va extraire d'information des stations à analyser depuis le fichier « .REP ». Effectivement, il contient les fichiers csv qu'on a sollicité que MAGE exportait. Grâce au format défini par Fortran, il est possible d'identifier si les résultats de la modélisation ont été estimés en instantanée ou une moyenne de temps. Il faut conserver l'extraction des données au pas de temps instantané pour éviter des erreurs.

```
# Lecture station with tide gauge and gauging(.REP)
file_station_modify = readLines(paste0(Proj_dir,Repfile), warn=F)
station_observation_temp = file_station_modify[16:length(file_station_modify)]
station_REPfile_temp = str_split(station_observation_temp, " ",simplify=T)

# Verify if the results of the simulations are instataneous -> condition!

if (station_REPfile_temp[1,6]!=1){
  warning("WARNING : the results of simulations are mean of time instead of instantaneous!
          It could be a mimsinterpretation of results")
}

station_REPfile = station_REPfile_temp [,c(3,5)]
```

Figure 30. Lecture des stations avec l'information à croiser

Afin d'éviter de mettre à la main le temps de simulation, il est possible d'obtenir l'information via le fichier « .NUM » qui fournit le démarrage et le final de la simulation. Puis, on estime le temps total de simulation en secondes.

```
# Lecture time of simulation from MAGE model (.NUM) start run and end run

for (i in 1:2){
  sim_time_REP = read.table(file=paste0(Proj_dir,"Seineav1.NUM"),
                            sep = "",
                            header = F,
                            skip= i,
                            nrows = 1)
  sim_time_vector = as.numeric(str_split((rev(sim_time_REP)[1]),":",simplify=T))

  for (j in 1:length(sim_time_vector)){
    sim_time_temp= switch (
      j,
      "1" = sim_time_vector[j]*86400,
      "2" = sim_time_vector[j]*3600,
      "3" = sim_time_vector[j]*60,
      "4" = sim_time_vector[j]
    )
    sim_time [i] <- sim_time [i] +as.numeric(sim_time_temp)
  }
}

start_run <- sim_time[1]
end_run <- sim_time[2]
```

Figure 30. Lecture des stations avec l'information à croiser

Après avoir défini la période de simulation, cette information permettra d'établir la dimension du vecteur/matrice des variables d'entrée et variable de sortie. Le script facilite l'extraction de cette information parce qu'il la récupère directement depuis les fichiers fournis par MAGE.

```
start_date_sim = intial_date_time_mascaret+start_run
end_date_sim = intial_date_time_mascaret+end_run

# Define period of time to analyse
year <- format(intial_date_time_mascaret, format="%Y")

# Enter variable vector
file_time_step = read.table(file=paste0(Proj_dir,"Seineav1.NUM"),sep = "", header = F, skip= 3, nrows = 1)
time_step = as.numeric(rev(file_time_step)[2])

n_row = (end_run-start_run) /time_step+1

X=data.frame(time_hours=seq(from=0,by=time_step/3600,length.out = n_row))

# Size of observation data
X_seg <- as.integer(round(X[,]*3600))
Y_temps <- cbind(X_seg/3600)
colnames(Y_temps) <- c("time")
```

Figure 30. Dimension du vecteur avec les variables d'entrée et sortie

A ce stade, il nous manque encore charger les données des variables de sortie (campagne de jaugeages et limnigrammes). C'est pour cela que l'idée est de parcourir le fichier .REP pour récupérer l'information des stations qui serviront comme observations.

```
#Load tide gauge and gauging at stations in the order provided by .REP file
for (i in 1 : nrow(station_REPfile)){

  station_name = obs_stations[match(station_REPfile[i,2],obs_stations[,1]),2]
  station_name_df = rbind(station_name_df,station_name)
  observation_vector_temp <- data.frame(rep(NA,n_row))
```

Figure 30. Extraction des noms des stations qui serviront comme variables de sortie

Ensuite, une routine est développée afin d'affecter la donnée observée, le débit et la hauteur d'eau sont seulement les deux cas supportés par le script, au pas du temps concordant. Pour ce faire, les dates des deux variables (d'entrée et de sortie) sont comparées afin d'identifier le plus proche et ensuite, affecter la mesure.

La même procédure s'applique à la lecture d'une campagne de jaugeages et d'un limnigramme. La seule différence est la mise en place d'un filtre du limnigramme afin de dégrader l'observation afin d'empêcher qu'elle prenne beaucoup de poids à moment de chercher la solution optimale via l'approche Bayésienne. En effet, on est confronté à ce problème à cause du manque de mesures tout au long de la période de simulation, ce qui semble cohérent pour la difficulté de prendre la mesure en continue.

```
if (station_REPfile[i,1] == 'q'){

  gauging_file = paste0(folder_observation,
                        "/jaugeage_",
                        station_name,
                        "_",
                        months(intial_date_time_mascaret),
                        "_",
                        year,
                        ".txt")

  gauging_data = switch(file.exists(gauging_file),
                        "TRUE" = as.data.frame(read.table(file=gauging_file,header=T,sep="\t")),
                        "FALSE" = warning ('No file  of gauging has been read')
                        )

  gauging_data_time_sim <- gauging_data[which((gauging_data$Temps_mascaret>=start_run) &
                                              (gauging_data$Temps_mascaret<=end_run)),
                                        match(c("Q","Temps_mascaret"),colnames(gauging_data))]

  for (j in 1:nrow(gauging_data_time_sim)){
    delta_input_outputs <- abs(X_seg-gauging_data_time_sim$Temps_mascaret[j])
    location <- which(delta_input_outputs==min(abs(delta_input_outputs)))[1]
    flow_obs <- gauging_data_time_sim[j,1]
    observation_vector_temp [location,] <-  flow_obs
  }
  colnames(observation_vector_temp) = paste0("Q_",station_name)
  Y_temps <- cbind(Y_temps,observation_vector_temp)
```

```
}else if (station_REPfile[i,1] == 'z'){
  tide_gauge_file = paste0(folder_observation,
                           "/maregraphes_mesures_",
                           station_name,
                           ".xlsx")

  tide_gauge_data = switch(file.exists(tide_gauge_file),
                           "TRUE" = as.data.frame(read_excel(paste0(folder_observation,
                                                                    "/maregraphes_mesures_",
                                                                    station_name,
                                                                    ".xlsx"),
                                                             sheet = 1,
                                                             range = cell_cols("F:H")
                                                             )
                                                  ),
                           "FALSE" = warning ('No file  of tide gauge has been read')
  )

  colnames(tide_gauge_data) = c("time_hours","Date_sh","Stage")

  tide_gauge_data_time_sim = tide_gauge_data[which((tide_gauge_data$time_hours>=start_run) &
                                                   (tide_gauge_data$time_hours<=end_run)),]

  # Filtering of the measurement to avoid to give a lot of decision weight for the calibration step

  tide_gauge_data_time_sim_sample = tide_gauge_data_time_sim[seq(1,nrow(tide_gauge_data_time_sim),by=slim_tide_gauge),]

  tide_gauge_data_time_sim_sample$time_hours <- as.integer(round(tide_gauge_data_time_sim_sample$time_hours))

  for (j in 1:nrow(tide_gauge_data_time_sim_sample)){
    delta_input_outputs <- abs(X_seg-tide_gauge_data_time_sim_sample$time_hours[j])
    location <- which(delta_input_outputs==min(abs(delta_input_outputs)))[1]
    stage_obs <- tide_gauge_data_time_sim_sample[j,3]
    observation_vector_temp [location,] <-  stage_obs
  }
  colnames(observation_vector_temp) = paste0("z_",station_name)
  Y_temps <- cbind(Y_temps,observation_vector_temp)
}else{

  warning ('No variable is supported as an observation. Set gauging or tide gauge as observations')
}
```

Figure 30. Affectation de la donnée observée (débit ou ligne d'eau) au pas de temps plus proche de la variable d'entrée (temps)

Ensuite, les incertitudes sont associées aux observations et puis on aura toute l'information nécessaire pour créer le fichier qui contient les variables d'entrée et de sortie.

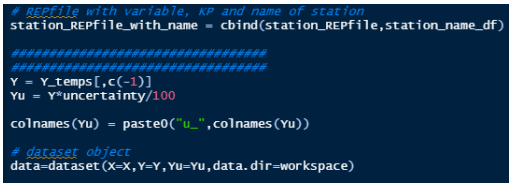


Figure 30. Affectation de l'incertitude aux observations et création de l'objet « dataset »

Le modèle et les observations sont déjà chargés à ce stade, on poursuit avec le paramétrage interne de la méthode MCMC. En d'autres termes, affecter le nombre de cycles, la portion à brûler et l'affinage défini dès début du script.

Seulement le paramétrage du modèle d'erreur manque à régler. Il faut définir un modèle erreur par variable de sortie. En revanche, si le modèle d'erreur ne change pas de configuration, il est possible de sauvegarder le paramétrage dans un même fichier comme la figure ci-après illustre. Effectivement, la configuration du modèle d'erreur hétéroscédastique est sauvegardée dans le fichier par défaut alors que le paramétrage du modèle d'erreur homoscédastique est sauvegardé dans « Config_RemnantSigma_var2.txt ».

Figure 30. Configuration des options de la méthode MCMC et modèles d'erreurs des observations

Enfin, il est possible de lancer la fonction qui appelle l'exécutable BaM depuis le package RBaM. Il faut seulement donner le modèle, l'information des variables d'entrée, de sortie, le paramétrage du MCMC et les modèles d'erreur.

Figure 30. Fonction sur RBaM pour lancer l'exécutable BaM.exe

Centre (Lyon – Grenoble – Auvergne–Rhône-Alpes)
5 Rue de la Doua
69100 Villeurbanne
Tél. : +33 04 72 20 87 87

Rejoignez-nous sur :
site internet national ou site internet du centre

Institut national de recherche pour l'agriculture, l'alimentation et l'environnement

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

INRAE