# Comment implémenter un nouveau modèle dans *BaM!*

BR, EP, JLC, 30/01/2019

Environnement de développement :

CodeBlocks (cf. Matteo), IVF – Microsoft Visual Studio (cf. Ben), “à l’ancienne” avec éditeur de texte + makefile (voir exemple dans BaM\trunk\core\linux)

Aspirer BaM et HydroStat (côte à côte) depuis la forge.

Il y a 4 familles de modules Fortran à inclure dans le projet :

1. Dans BaM/trunk/core/src/MiniDMSL : **MiniDMSL** : utilitaires numériques et statistiques (Dimitri’s Mathematical and Statistical Library). Normalement, pas besoin d’y toucher.
2. Dans HydroStat/trunk/Fortran/src: **HydroStat** : autres utilitaires numériques et statistiques (ceux de Ben). Normalement, pas besoin d’y toucher. La liste des fichiers nécessaires à la compilation de BaM est la suivante (voir aussi le makefile dans BaM/trunk/core/linux) :
   * Aggregation\_tools.f90
   * BayesianEstimation\_tools.f90
   * DataRW\_tools.f90
   * Dates\_tools.f90
   * Distribution\_tools.f90
   * EmpiricalStats\_tools.f90
   * MCMC\_tools.f90
   * MCMCStrategy\_tools\_NoRFortran.f90
   * TimeSeries\_tools.f90
3. Dans BaM/trunk/core/src : **Models** : un module par modèle. Pour créer un nouveau modèle, il faut donc créer le nouveau module correspondant. Normalement, pas besoin de toucher aux fichiers déjà existants.
4. Dans BaM/trunk/core/src : **BaM!** : les fichiers spécifiques à BaM :
   * ModelLibrary\_tools.f90, le module qui fait l’interface entre les modèles du point 3. et le formalisme de BaM. Il faut le modifier pour « brancher » chaque nouveau modèle.
   * BaM\_tools.f90, outils statistiques et de configuration. Normalement, pas besoin d’y toucher.
   * BaM\_main.f90, le programme principal. Normalement, pas besoin d’y toucher.

Pour ajouter un nouveau modèle dans *BaM!*, il faut ajouter :

1. Un nouveau module XXX\_model.f90 dans BaM/trunk/core/src/Models.
2. Le brancher en modifiant ModelLibrary\_tools.f90 dans BaM/trunk/core/src.
3. Ne toucher à aucun autre fichier! *BaM!* est fait pour être indépendant des modèles ; seul ModelLibrary\_tools est conscient à la fois de *BaM!* et des modèles.

## Création de SuspendedLoad\_model.f90

On copie un modèle existant (Linear\_model.f90)

Renommer le module comme le fichier (conseillé), au début et à la fin du code

Commentaires à mettre à jour

Pour faciliter le branchement, mieux vaut avoir 2 fonctions : une qui retourne le nb de paramètres du modèle, une autre qui applique le modèle. Eventuellement une 3ème pour gérer les options éventuelles du modèle.

Public :: les 2 fonctions

Contains : définition des deux fonctions

Définir entrées, sorties, paramètres, erreur, message

/ !\ Il faut écrire les modèles sous forme matricielle, pas pour une seule valeur de h, mais un vecteur h(:). Ceci pour permettre les modèles dynamiques.

*N=size(h)*

*do i=1,n*

*Qs(i) = theta(1) \*(h(i)- theta(2))\*\* theta(3) \*(h(i)- theta(4))\*\* theta(5)*

*enddo*

Il faut ajouter les contraintes sur les valeurs d’entrée et les paramètres, et les messages d’erreur.

Tests paramètres : à mettre avant la boucle

Tests sur h(i), éventuellement dans la boucle.

feas pour faisabilité : vecteur de logicals de même taille que la sortie

/ !\ spécifique au modèle SuspendedLoad :

* + - Ré-écriture en quotient : theta(5) > 0
    - Ré-écriture avec Q et Cs en sorties, et Qs=Q\*Cs en variable d’état

## Branchement du modèle dans ModelLibrary\_tools.f90

Annoncer qu’un nouveau modèle est disponible :

*use SuspendedLoad\_model*

Déclarer le nom du modèle :

*MDL\_SuspendedLoad="MDL\_SuspendedLoad"*

Ajouter un « case » pour appeler (call) la fonction SuspendedLoad\_GetParNumber qu’on vient de coder.

Ajouter un « case » pour appeler (call) la fonction SuspendedLoad\_Apply qu’on vient de coder.

Ajouter un « case » pour appeler (call) la fonction SuspendedLoad\_XtraRead qu’on codera plus tard (on ne fait rien pour le moment).

Ajouter un « case » pour définir le nombre et le nom des paramètres dérivés et variables d’état (pour le moment, il y en a zéro de chaque…)

## Compiler BaM\_miniDMSL.exe…

Voir comment Matteo fait avec CodeBlocks.

Fini : le modèle est prêt à être utilisé dans *BaM!* !!!