airGR

Suite des modèles hydrologiques de l'IRSTEA

(dans l'environnement « R »)

Aide-mémoire par L. Duffar (Rev. 1.08)

Utilisateur non affilié à l'IRSTEA

Téléchargez la dernière version de ce document au lien ci-dessous
https://zenodo.org/record/3256288

Sommaire

Voir le sommaire complet à la fin

1 Préambule

1.1 Introduction

« aiGR » est une « Librairie » qui implémente dans l'environnement du « langage R » les modèles hydrologiques suivants de l'IRSTEA:

- Modèles « Précipitations-Débit » aux pas journalier, horaire, annuel, mensuel et annuel (**GR4H**, **GR4J**, **GR5J**, **GR6J**, **GR2M**, **GR1A**)
- Modèle d'accumulation et fonte de neige **CemaNeige** au pas journalier (degree/day Snow accumulation and melt).

NB : Ni le modèle « GRP » ni l'utilitaire « Otamine » d'estimation de l'incertitude de prévision de début ne sont inclus dans « airGR ».

Cet aide-mémoire présente:

- les principes d'installation et d'utilisation des librairies « airRG » / « airGRTeaching » dans l'environnement du « langage R »,
- les bases du « langage R » nécessaires et suffisantes
- et quelques pistes de mise en œuvre des différents modèles, ainsi que les moyens de les implémenter dans l'environnement de la Librairie « airGR ».

Le présent document n'est pas une introduction au « langage R », dont la maitise fine n'est pas indispensable, du moins pour modéliser un bassin-versant unique (même si une programmation « R » plus avancée permet de construire des modélisations plus complexes, et de façon plus flexible qu'avec un logiciel à interface classique interactive ; Cf. § 1.2 ci-dessous). Voir § 1.3 ci-dessous pour les liens vers la documentation complète de « R » ainsi que vers des guides en français.

Ce document ne se substitue pas non plus à la documentation officielle de « airGR » dont il reprend les informations dans une présentation mettant en évidence la chronologie des opérations, tout en les complétant si besoin. Pour les dernières informations en date, la documentation IRSTEA reste la seule référence.

Par contre, le présent document présente les éléments de compréhension qui se sont avérés indispensables au rédacteur lors de sa propre prise en main de « airGR » et de l'environnement « R » (prise en main seulement élémentaire à ce jour).

L'utilisation de « airGR » ne s'improvise pas ! Cependant les informations fournies faciliteront le démarrage à condition de lire et respecter scrupuleusement CHAQUE INDICATION. De plus, il n'est pas exclu de rencontrer des incidents classiques d'environnement R, et l'utilisateur doit donc se préparer à les résoudre au cas par cas en consultant les forums.

1.2 Principe général des librairies « airGR » & « airGRTeaching »

Ces libraires créées par l'IRSTEA à partir du début des années 2010, conservent le cœur de calcul de ses différents modèles développés à l'origine en FORTRAN, en les intégrant dans une « Librairie R ». Ce portage permettait ainsi de profiter des avantages du « langage R », notamment :

- **Mise en œuvre flexible** des modèles GR (par programmation), au prix cependant d'une accessibilité moins immédiate qu'un logiciel classique à interface interactive
- De nombreux **utilitaires et librairies** existent en complément de « R » (installation libre « en ligne »), rendant disponibles des traitements puissants via des commandes simples (c'est le propre des langages de « Haut-niveau »). Par exemple, c'est la librairie R Open Source « Shiny » qui permet à l'utilisateur de « airGR » de manipuler et visualiser dans une page Web interactive la modélisation qu'il a construite, cela grâce à une unique commande de la librairie complémentaire « airGRTeaching » (en complément de la mise en œuvre par programmation « pure »).
- « R » offre le moyen d'**agglomérer des modules codés en différents langages** (en l'occurrence FORTRAN), comme Python autre langage de type « Glue » (Colle), ce qui permettait à la fois de préserver la vitesse de traitement et de limiter le développement (et le débogage) de nouvelles lignes de code.
- Enfin, l'**ergonomie par ligne de commande** permet un usage en programmation pure adaptée à des modélisations complexes, grâce à une flexibilité supérieure à celle d'un logiciel à interface interactive classique.

Ces caractéristiques apportent aux modèles hydrologiques GR à la fois plus d'ergonomie et plus de flexibilité que les outils FORTAN originaux, ce qui a d'ailleurs enfin permis leur diffusion officielle (La production par l'IRSTEA d'outils et méthodes opérationnelles a d'ailleurs explosée depuis le début du XXI^{ème} siècle). Par rapport au précédent portage sous Excel de certains des modèles IRSTEA au début des années 2000 (GR4J, GR2M, GR2A), « airGR » est bien plus robuste (en termes d'optimisation) et puissant (en termes de fonctionnalités), au prix cependant d'une courbe d'apprentissage de plusieurs heures (voire de plusieurs jours en l'absence d'expérience dans l'environnement R, ou pire de toute expérience de développement). De plus, il n'est pas exclu de renconter des incidents d'environnement logiciel dont la résolution nécessitera un minimum d'expertise informatique et de recherche sur les forums.

Les modèles implémentés dans « airGR » sont les suivants :

GR4H	Pluie-Débit Horaire 4 paramètres	Perrin et al., 2003
GR4J *	Pluie-Débit Journalier 4 paramètres	Mathevet, 2005
GR5J *	Pluie-Débit Journalier 5 paramètres	Le Moine, 2008
GR6J *	Pluie-Débit Journalier 6 paramètres	Pushpalatha et al., 2011
GR2M	Pluie-Débit mensuel 4 paramètres	Mouelhi, 2003; Mouelhi et al., 2006a
GR1A	Pluie-Débit Annuel 1 paramètre	Mouelhi, 2003; Mouelhi et al., 2006b
CemaNeige	Accumulation et fonte de neige Quotidienne (degré/jour) combiné avec GR4J, GR5J ou GR6J	Valéry et al., 2014

*: également dans l'interface « Shiny ».

1.3 Documentation en ligne

« R »

Documentation Officielle de R	Description complète de la syntaxe (En)
R pour les débutants	(fr)
Introduction au logiciel R (Statistique et Biologie)	(fr)
Manipuler des donnéees calendaires	(fr)
Saving Data into R data formats : RDS & RDATA	(En)
Gestion des données avec R	(fr)
R Data Import/Export	Doc officielle d'import/export de données

« airGR » et « airGRteaching »

La documentation en ligne est copieuse (en anglais) mais éclatée entre différents supports. Les éclaircissements ci-dessous portent sur la hiérarchisation des informations, tandis que les pas-à-pas (§ 4.2 ci-dessous et § 4.3 ci-dessous) reproduisent une grande partie du texte original en mettant en évidence la chronologie des opérations de construction d'un modèle:

- 2 documents **PDF officiels** décrivent la syntaxe de toutes les fonctions, et contiennent chacun un préambule de conseils généraux indispensable au démarrage, qui sont reproduits au début de chacun des 2 « pas-à-pas » du présent document (§ 4.2 ci-dessous et § 4.3 ci-dessous).
- 5 « **Vignettes** » **accessibles en ligne** décrivent des principes « méthodologiques ».
  2 « vignettes » représentent les guides de démarrage c'est pourquoi elles ont servi de base à chacun des 2 « pas-à-pas » du présent document (§ 4.2 ci-dessous et § 4.3 ci-dessous).
  Les 3 autres vignettes seront utiles pour certaines applications avancées (Par exemple programmer une fonction personnalisée d'optimisation ou utiliser les images satellites de couverture neigeuse pour l'utilisation de Cemaneige).
- 32 **pages web** exposent fonction par fonction la syntaxe complète (disponible par ailleurs dans les PDF de référence) sous une forme **accessible en ligne** (Le présent document contient aussi des liens url 🔗 qui renvoient vers ces pages)

Le tableau ci-dessous résume les sujets abordés dans chaque support, puis les 2 pas-à-pas de ce document en reprendront les informations dans une présentation mettant en évidence le déroulement des étapes de mise en œuvre (§ 4.2 ci-dessous et § 4.3 ci-dessous).

Support	Nom et lien	Description rapide
Documents PDF	airGR.pdf	Syntaxe de toutes les fonctions « airGR »
	airGRteaching.pdf	Syntaxe de toutes les fonctions « airGRTeaching »
5 « Vignettes » sur le site CRAN Principes méthodologiques	Get Started with airGR	Démarrer avec « airGR »
	Get Started with airGRTeaching	Démarrer avec « airGRTeaching »
	Parameter estimation within a Bayesian MCMC framework	Calage par tirage aléatoire « Monte Carlo Markov Chain »…. (algorithme d'optimisation non implémenté à ce jour)

Support	Nom et lien	Description rapide
	Plugging in new calibration algorithms in airGR	Conseils de programmation pour ajouter un algorithme d'optimisation personnel
	Using satellite snow cover area data for calibrating and improving CemaNeige	Amélioration du calage du modèle « Cemaneige » par utilisation des données satellite de superficie enneigée
32 pages web de MANuel Descriptions des fonctions « airGR » (identique aux informations du document « airGR.pdf », mais sans le paragraphe d'introduction)	https://rdrr.io/cran/airGR/	Page principale
	Description	Description globale de « airGR »
	BasinInfo:	Variable « BasinInfo » (type « List ») : caractéristiques du bassin versant (id, surface, hypsométrie)
	BasinObs:	Tableau « BasinObs » (type « Time Series ») des séries chronologiques de Pluie, ETP et Débit
	Calibration:	Fonction « Calibration »
	Calibration_Michel:	Fonction « Calibration Michel »
	CreateCalibOptions:	Fonction « CreateCalibOption » de création des objets « CalibOptions »
	CreateIniStates:	Fonction « CreateInitState » pour créer éventuellement l'argument « InitStates » de la fonction « CreateRunOptions »
	CreateInputsCrit	Creation of the InputsCrit object required to the ErrorCrit...
	CreateInputsModel	Creation of the InputsModel object required to the RunModel...
	CreateRunOptions	Creation of the RunOptions object required to the RunModel...
	DataAltiExtrapolation_	Altitudinal extrapolation of precipitation and temperature by Valery method...
	ErrorCrit ErrorCrit_KGE ErrorCrit_KGE2 ErrorCrit_NSE ErrorCrit_RMSE	Error criterion using : - provided function - KGE formula - KGE' formula - NSE formula - RMSE
	Param_Sets_GR4J	Calage de GR4J selon « Generalist parameters Sets » (2 algorithmes combinés Local & Global/Michel) pour **moins de 6 mois** **Au lieu de** l'Algorithme de Michel quand la période de calage couvre **plus de 6 mois**
	PEdaily_Oudin	Computation of daily series of potential evapotranspiration...
	plot.OutputsModel	Default preview of model outputs
	RunModel	Run with the provided hydrological model function
	RunModel_CemaNeige	Run with the CemaNeige snow module
	RunModel_CemaNeigeGR4J RunModel_CemaNeigeGR5J RunModel_CemaNeigeGR6J	Run the CemaNeige hydrological model for : - GR4J - GR5J - GR6J
	RunModel_GR1A	Run with the GR1A hydrological model
	RunModel_GR2M	Run with the GR2M hydrological model
	RunModel_GR4H	Run with the GR4H hydrological model
	RunModel_GR4J RunModel_GR5J RunModel_GR6J	Run with the GR4J hydrological model for : - GR4J - GR5J - GR6J
	SeriesAggreg	Conversion of time series to another time step (aggregation...

Support	Nom et lien	Description rapide
	TransfoParam	Transformation of the parameters using the provided function
Aide intégrée à l'interface web interactive ShinyGR	How to run the airGR hydrological models in only three simple steps with airGRteaching	Accessible uniquement depuis l'interface ShinyGR (Menu « Functionalities »)

2 Installation de « R » & « RStudio »

Le langage « R » est un langage de type « interprété » (tout comme « Python », autre langage Open Source de haut niveau datant de la fin du XXème siècle). Cette caractéristique rend nécessaire la présence de l'interpréteur « R » sur chaque machine d'exécution. Cette caractéristique n'est pas un inconvénient, ni en termes de temps de calcul (les modules de calcul sont « compilés »), ni en termes d'installation (l'interpréteur « R » est Open Source). De plus, l'environnement de développement « RStudio » (également Open Source) peut apporter une meilleure convivialité à l'interface de « R » de type « ligne de commande ».

Les 3 premiers paragraphes ci-après sont inspirée de : http://larmarange.github.io/analyse-R/installation-de-R-et-RStudio.html

Il est préférable de commencer par installer « R » avant « RStudio ».

2.1 Installer « R »

Sous Windows : Rendez-vous sur: http://cran.r-project.org/bin/windows/base/

- Cliquez sur le lien de téléchargement du programme d'installation.
- Lancez l'exécutable d'installation de « R » avec les options par défaut.

Sous Mac OS X : Rendez-vous sur http://cran.r-project.org/bin/macosx/

- Téléchargez le fichier d'installation
- Lancez l'exécutable d'installation de « R » avec les options par défaut.

Sous Linux : On peut normalement trouver « R » via le gestionnaire de paquets, mais cela peut dépendre de la distribution de Linux.

Alternative : R peut s'installer via la distribution « Anaconda » (Cf. § 2.4 ci-dessous), ce qui installe par la même occasion « Conda » qui rend possible en ligne les installations/mises-à-jour en ligne, notamment pour des outils et librairies supplémentaires (« RStudio » notamment).

2.2 Installer « RStudio »

Une fois « R » installé, l'installation de « RStudio » est recommandée bien qu'optionnelle (on peut se contenter de l'interface « Rgui » intégrée à R, mais ce serait dommage):

- Rendez-vous sur http://www.rstudio.com/products/rstudio/download/
- Téléchargez la dernière version stable de « RStudio » (Plus précisément, l'édition Open Source « RStudio Desktop », car il existe aussi une version serveur).
- Choisissez l'installateur correspondant au système d'exploitation de l'ordinateur
- Suivez les instructions du programme d'installation.

NB : La version de développement est destinée uniquement aux utilisateurs avancés (pour tester les dernières fonctionnalités de « RStudio »), car elle est moins stable.

Si on a choisi l'alternative d'installer R par le biais de « Anaconda », l'installation de « RStudio » se fait via le gestionnaire de Package « Conda », au choix …

- En ligne de commande (Fenêtre « Terminal » de l'OS) : `conda install rstudio`
- Ou via l'interface « Anaconda-Navigator » dans l'environnement voulu (de base ou utilisateur) : Onglet « Environnement » > Sélectionnez l'environnement > Onglet « Home » > Cliquez sur « Install » dans la tuile « RStudio »

2.3 Mettre à jour « R » sous Windows

Sauf cas particulier décrit plus bas, la mise à jour de « R » sous Windows, s'opère en téléchargeant et en lançant simplement la dernière version de l'exécutable d'installation. Cependant, la nouvelle version sera installée à côté de l'ancienne version, c'est pourquoi…

….Pour faire de la place sur le disque dur, il suffit de désinstaller l'ancienne version via l'utilitaire Windows de désinstallation de programme.

…Si plusieurs versions de « R » restent installées, « RStudio » exécute les commande avec la plus récente, mais il est possible de choisir la version exécutée via le menu « Tools > Global Options > General » de « RStudio ».

AVERTISSEMENT concernant les Librairies déjà installées avant la mise à jour de « R »

Les libraires » installées précédemment sous l'ancienne version de « R » ne seront plus disponibles pour la nouvelle version ! En effet les « Librairies » sont installées par défaut dans le répertoire « Utilisateur > Documents > R > win-library > x.y (x.y correspond au numéro de la version de R). Pour rendre à nouveau disponibles les Librairies déjà présentes :

- Recopiez le contenu de l'ancien dossier « x.y » dans le nouveau
- Lancez « RStudio » (s'il était déjà ouvert, fermez-le puis relancez-le)
- Mettez à jour l'ensemble des packages, soit avec la fonction, « update.packages » soit en cliquant sur « Update » dans l'onglet « Packages » du quadrant inférieur droit.

Cas particuliers

Ordinateur situé derrière un proxy

Il est préférable de choisir « Options de démarrage personnalisées » lorsque le programme d'installation le propose, puis « Internet2 » comme mode de connexion à Internet. Ainsi, « R » utilisera par défaut les paramètres de proxy du navigateur « Internet Explorer ».

Si « R » a été installé par le biais de la distribution « Anaconda » (Cf. § 2.4 ci-dessous)

La mise à jour de « R » se fait par une commande d'INSTALLATION via l'utilitaire « Conda » :

…. Dans la fenêtre « Terminal » de l'OS

- Tapez la commande suivante pour connaitre la version de R installée : `R --version`
- tapez la commande suivante en précisant si besoin la version voulue à installer :
  `conda install [-n[ame] nom_environnement] r==x.x.x`
  L'option ci-dessus permet de choisir si besoin l'environnement utilisateur contenant la version de « R » à mettre à jour (En alternative on peut activer au préalable l'environnemnent voulu par la commande `conda activate nom_environnement`)

2.4 Alternative : Installer la distribution « Anaconda » (R+Python+Conda)

Cette alternative est réservée à des développeurs avancés (Pour les autres, il est conseillé de se contenter de R et RStudio, car une expertise informatique est nécessaire pour la résolution des inévitables incidents d'environnement logiciel). Cette distribution apporte les avantages suivants :

- ✓ Installation simultanée 2 langages « R » et « Python » pouvant être complémentaires pour des usages très avancés de « airGR ».
- ✓ D'une façon plus générale un développeur trouvera des avantages décisifs dans les outils également installés avec la distribution « Anaconda » :
  - Outil « Conda » pour la « Gestion de Packages et d'environnements » (facilitant notamment l'installation et la mise à jour en ligne des librairies scientifiques diverses et des outils généralement nécessaires aux développeurs)
  - Outil « Anaconda-Navigator » apportant un environnement intégré interactif à l'ensemble des outils de développement R et Python (Conda, RStudio, Jupyter Notebook, Spyder).

2.4.1 Installer « Anaconda »

La distribution « Anaconda » installe simultanément les outils suivants, par le biais d'un unique fichier exécutable (PC, Mac ou Linux):

- ❖ Langages « Python » et « R »
- ❖ « Conda » : Gestionnaire de Packages et d'Environnements

Ce gestionnaire facilite l'installation et la mise à jour des diverses librairies python et R (depuis « Anaconda Cloud ») nécessaires pour le développement scientifique ou pour exploiter des jeux de données en accès libre.

❖ « Anaconda-Navigator » : Environnement graphique pour l'utilisation des divers outils (RStudio, Spyder, Jupyter Notebook etc…) permettant leur installation, leur lancement et leur mise à jour.

« Anaconda-Navigator » améliore également l'expérience d'utilisation de « Conda » (notamment il indique la liste et la version des Librairies installées, ainsi que la disponibilité des mises à jour !).

2.4.2 Installer « RStudio » en plus à l'aide de « Conda » (gestionnaire de « Package »)

Une fois la distribution « Anaconda » installée (si cette alternative est choisie), le gestionnaire « Conda » permet ensuite d'installer le Package « RStudio » en ligne depuis « Anaconda Cloud », à condition de disposer d'une connexion internet active (Les opérations de « Gestion de Package » représentent l'usage le plus évident, mais pas unique, de « Conda »).

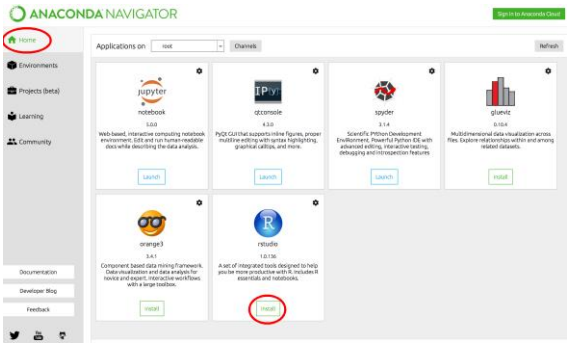
Au choix…

…Dans la fenêtre « Terminal » de Windows (Invite de commande), Mac ou Linux.

- Tapez `conda install rstudio[x.x]` (x.x optionnel pour spécifier la version)

…Ou dans l'utilitaire « Anaconda-Navigator » inclus dans la distribution « Anaconda ».

- Dans l'onglet « Home » > cliquez sur « Install » de la tuile « RStudio ».



2.4.3 Utiliser « Conda » en tant que « gestionnaire d'environnement »

En plus d'être un « gestionnaire de Package » très utile, « Conda » est également un « gestionnaire d'environnement » qui gagne à être utilisé par les ingénieurs et chercheurs utilisant des scripts de traitement attachés à des versions spécifiques de dépendances. Il permet en effet d'installer différentes versions des « Packages » (langages ou librairies) dans des environnements distincts, afin de pouvoir ensuite exécuter différents programmes nécessitant des versions spécifiques (en basculant d'un environnement à l'autre par une simple commande ou un clic).

Pour sélectionner l'environnement voulu avant l'installation ou l'exécution, on dispose des 2 moyens suivants au choix…

…Dans la fenêtre « Terminal » de l'OS :

- Tapez `conda install -n[ame] nom_environnement noms_package`
- Ou `conda activate -n[ame] nom_environnement`

…Ou via « Anaconda-Navigator » dans l'onglet « Environnement » :

- Créez un nouvel environnement en y installant les versions voulues de langage/librairies
- Ou sélectionnez à la souris le nom d'un environnement existant

2.4.4 Pour en savoir plus

- ✓ How to Download & Install R, RStudio, Anaconda
- ✓ Utiliser « Conda » pour installer un environnement de développement

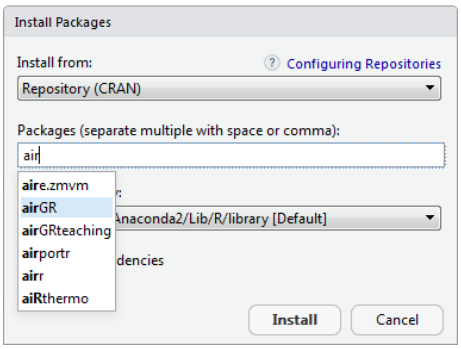
3 Installation de « airGR » & « airGRteaching »

Pour les dernières instructions en date de l'IRSTEA voir https://odelaigue.github.io/airGR/.

3.1 Installer « airGR »

Au choix :

- ✓ Par la ligne de commande « R » : `install.packages("airGR")`
- ✓ OU sous « RStudio » par le menu « Tools > Install Package »…



…au choix :

- en ligne depuis le Repository CRAN (comme ci-contre),

- ou en local avec un fichier « Package » téléchargé au préalable (pour Windows choisir le fichier .zip, et non .gz ou .tgz)

3.2 Installer « airGRTeaching »

L'installation supplémentaire de « airGRTeaching », bien optionnelle, est très conseillée et rapide. La doc officielle conseille son utilisation aux débutants, mais tous les utilisateurs y trouveront y des avantages pour :

- ✓ Effectuer une modélisation en 4 instructions seulement (y compris la fonction de graphique, mais sans compter la création des objets du jeu de données qui reste à coder en R)
- ✓ Regrouper tous les résultats numériques/graphiques dans une interface interactive web, qui permet de plus de rejouer le calage en changeant de critière d'optimisation ou de modèle GR*J (via la simple commande « ShinyGR » placée dans une boucle ; Cf. § 4.3.2 ci-dessous).

La procédure d'installation est la même que plus haut, en remplaçant « airGR » par « airGRTeaching ».

4 Principes communs à tous les modèles contenus dans « airGR »

4.1 Comment exécuter les fonctions « aiGR »

L'exécution des fonctions peut se faire au choix…

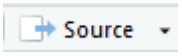
…Sur la ligne de commande de l'interpréteur « R » ou de la console « RStudio »:

- Tapez les commandes l'une après l'autre (si le nombre de commandes est limité, mais même dans ce cas il est avantageux d'en faire un script comme indiqué ci-dessous)

…Ou dans un fichier d'extension « .r » (script):

- Enregistrez les lignes de commande dans le fichier puis lancez le script.

Dans l'environnement « Rstudio » ces opérations sont facilitées par le menu :

- « File > New File » pour créer un fichier vide
- « File > Open File » (ou **CTRL**+**O**) pour ouvrir un fichier existant
- « Code > Source File » (ou **CTRL**+**O**) pour exécuter le code d'un fichier sans l'ouvrir
- « Code > Source »
- Cliquez sur  (ou **CTRL**+**SHIFT**+**P**) pour ré exécuter la dernière sélection de code
- Cliquez sur 

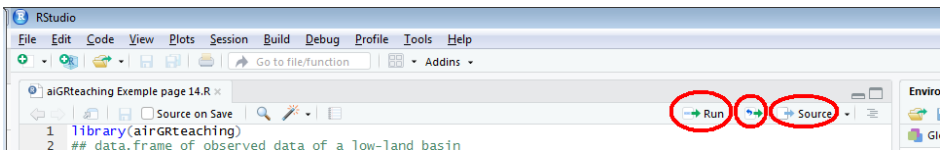
(ou **CTRL**+**SHIFT**+**S**) pour afficher le chemin du fichier courant

Le code d'un fichier ouvert à l'écran peut alors être exécuté de plusieurs façons :

- Cliquez sur 

(ou **CRTL** + **ENTER**) pour exécuter une ligne unique sur laquelle le pointeur est situé
- Mettez en surbrillance plusieurs lignes de code + Cliquez sur 

(ou **CTRL** + **ENTER**) pour exécuter tout le code sélectionné



4.2 Utilisation pas-à-pas de « airGR »

Ce guide pas-à-pas est notamment basé sur les informations de la documentation officielle (Cf. § 1.3 ci-dessus), avec quelques compléments indispensables (notamment des fonctions « R »). La présentation est prévue pour faciliter le démarrage :

- ✓ Les **fonctions « airGR »** sont décrites dans l'ordre chronologique d'exécution,
- ✓ et pour chaque `argument de fonction` demandant une préparation préalable (Objet complexes au lieu d'un simple paramètre), un bref renvoi indique où trouver les précisions nécessaires.
- ✓ Les étapes préparatoires éventuelles sont reportées après les **fonctions « airGR »** elles-mêmes
- ✓ Pour trouver des exemples de mise en œuvre de ces fonctions, voir le chapitre 5 ci-dessous consacré aux différents modèles.

Le pas-à pas ci-après a été élaboré en suivant notament les conseils suivants de l'IRSTEA.

Conseils généraux de IRSTEA (Tiré du PDF « airGR.pdf »)

Pour apprendre à utiliser les fonctions du package « airGR », l'IRSTEA recommande de suivre les étapes ci-dessous:

- ➢ Consultez l'aide de « RunModel_GR4J », puis lancez l'exemple de simulation fourni.
- ➢ Consultez l'aide de « CreateInputsModel » pour préparer les entrées d'un modèle.
- ➢ Consultez l'aide de « CreateRunOptions » pour fournir les options d'exécution d'un modèle
- ➢ Consultez l'aide de « ErrorCrit_NSE » et « CreateInputsCrit » pour choisir et comprendre le critère d'optimisation
- ➢ Consultez l'aide de « Calibration_Michel », exécutez l'exemple fourni, puis reportez-vous à l'aide de « CreateCalibOptions » pour la préparation du calage du modèle.

Résumé des étapes de modélisation

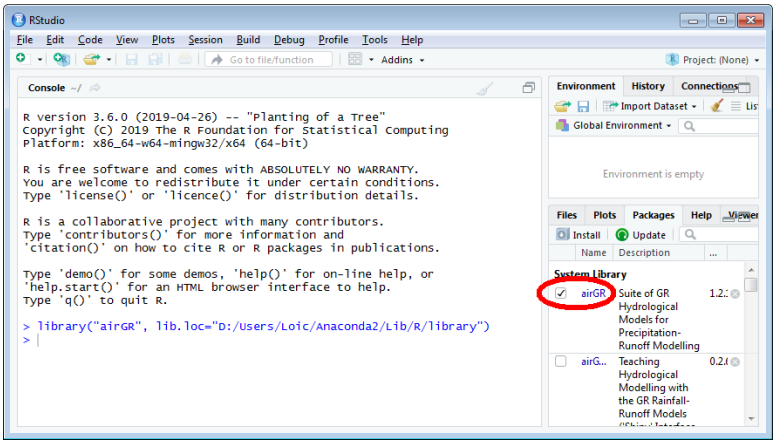
L'index ci-dessous des paragraphes constituent un résumé des étapes dans l'ordre chronologique de mise en oeuvre.

4.2.1 Charger les librairies « airGR » & « airGRTeaching »

Avant d'exécuter une commande « airGR » ou « airGRTeaching », il est nécessaire que la librairie correspondante soit chargée dans la session (une fois pour toutes par session), par les moyens suivant au choix…

- Dans la fenêtre « Terminal » tapez la commande « R » : `library(airgr)`
- Ou dans « RStudio » cochez les cases « airGR » et « airGRTeaching » dans l'onglet « Package » du panneau en bas à droite.

La procédure de chargement est la même pour « airGRTeaching ».



NB : Il peut arriver que le chargement d'une « Librairie » échoue avec un message d'erreur du type « `aucun package nommé 'xts' n'est trouvé` ».

- Dans ce cas installez la librairie manquante par le même moyen que ci-dessus, avant de retenter le chargement de « airGR » ou « airGRTeaching ».

4.2.2 Charger le jeu de données

Il faut créer deux objets comme expliqué ci-après.

AVERTISSEMENT : Contrairement aux exemples de la documentation qui chargent des jeux de donnée intégrés dans la Librairie, le jeu de données utilisateur doit être chargé par quelques lignes de code R dont aucun exemple n'est fourni dans la documentation (la fonction `data()` utilisée dans les exemples n'est pas pertinente pour un jeu de données utilisateur). De plus, alors que les objets s'appellent « BasinInfo » & « BasinObs » dans les exemples de la documentation, les noms sont en fait libres. Ce paragraphe fournit donc des informations supplémentaires indispensables pour coder en R le chargement du jeu de données, ainsi qu'un exemple complet (§ 4.2.2.3 ci-dessous).

Jeux de données intégrés à la librairie

NB : La librairie « airGR » inclut des jeux de données qui servent d'exemple et permettent d'exécuter les exemples fournis dans la documentation (Voir description détaillée au § 4.2.2.2 ci-dessous):

```
> data()

Data sets in package 'airGR':

BasinInfo (L0123001)    Data sample: characteristics of a different catchments
BasinInfo (L0123002)    Data sample: characteristics of a different catchments
BasinInfo (L0123003)    Data sample: characteristics of a different catchments
BasinInfo (X0310010)    Data sample: characteristics of a different catchments
BasinObs (L0123001)     Data sample: time series of observations of different catchments
BasinObs (L0123002)     Data sample: time series of observations of different catchments
BasinObs (L0123003)     Data sample: time series of observations of different catchments
BasinObs (X0310010)     Data sample: time series of observations of different catchments
Param_Sets_GR4J         Generalist parameter sets for the GR4J model
```


4.2.2.1 Caractéristiques du bassin-versant : Objet de type « List »

Tiré de la page web « BasinInfo »

Les caractéristiques du bassin versant (meta data) sont stockées dans un objet de type « List » (composé des éléments ci-après). Cependant, il semble (sous réserve de confirmation) que cet objet ne soit indispensable que pour stocker la courbe hypsométrique utilisée par le module « Cemaneige » (le nom du bassin n'est pas utilisé du tout et quant à la surface, elle permet seulement à l'utilisateur qui le souhaite de fournir la série de débit en l/s ou m3/s avant de la convertir en mm dans le code).

NB : Dans les exemples de la documentation l'objet intégré à la librairie s'appelle « BasinInfo », mais le nom est en fait libre, tout comme celui des attributs suivants:

- ✓ `BasinCode` Code bassin
- ✓ `BasinName` nom de la station de débit (type « String »)
- ✓ `BasinArea` Superficie du bassin versant en km^2 (Type « Float »)
- ✓ `HypsoData` Courbe hypsométrique du bassin versant (Type « vecteur ») : altitude (m) de 0 à 100% par pas de 1% (soit 101 valeurs croissantes)
  `if not defined a single elevation is used for CemaNeige`

```
str(BasinInfo)
List of 4
 $ BasinCode: chr "L0123001"
 $ BasinName: chr "Blue River at Nourlangie Rock"
 $ BasinArea: num 360
 $ HypsoData: num [1:101] 286 309 320 327 333 338 342 347 351 356 ...
```

Exemple de script pour créer l'objet des caractéristiques du bassin (List)

Voir l'exemple complet de création des 2 objets au § 4.2.2.3 ci-dessous.

```
library(airGR)
#! -------------- sources
#! source directory
pathSrc <- "../01_src"
#! hypsometry
pathHypsoRab <- file.path(pathSrc, "Rabioux-Chateauroux_hypso.csv")
```

```
#! ---------------------------- data loading and formating
#! -------------- metadata
metaRab <- List(Code = "Rabioux",
                Name = "Torrent du Rabioux a la station SCP",
                Area = 32.9)
#! -------------- hypsometry
#! reading
hypsoRab <- read.table(pathHypsoRab, sep = ";", header = FALSE, col.Names = "Alti")

#! vector of altitude quantiles
hypsoRab <- hypsoRab$Alti
```

Bassins versants intégrés à la librairie

La librairie inclut les objets suivants pour les besoins des exemples fournis dans la documentation (objet appelé « BasinInfo », alros que le nom de l'objet à définir par l'utilisateur est libre):

- ✓ L0123001, L0123002 & L0123003 : Bassins-versants fictifs
- ✓ X0310010 : Durance à Embrun [La Clapière] (05)

4.2.2.2 Séries chronologiques : objet de type « DataFrame »

Basé sur la vignette « Get Started with airGR », et complété par des notions de langage « R ».

Les séries hydrométéorologiques doivent être attribuées à un objet de type « DataFrame », (par exemple par lecture d'un fichier par un des moyens décrits plus bas) :

- o `DatesR` dates au format R POSIXt (Calendar Time ou Local Time, voir plus bas)
- o `P` Lame de bassin de précipitation [mm/jour]
- o `E` ETP de bassin [mm/jour]
- o `Qmm` Débit à l'exutoire du bassin en [mm/jour]
- o `T` Température de bassin [°C]

NB : on peut éventuellement fournir l'écoulement en unité de débit (Qls par exemple) mais il reste à éclaircir la façon d'effectuer la nécessaire conversion en [mm/jour] (conversion manuelle ?).

AVERTISSEMENTS :

- ✓ les lacunes (NA) de Précipitations et ETP ne sont pas autorisées !
- ✓ Dans les exemples de la documentation l'objet s'appelle « BasinObs » mais ce nom est en fait libre, tout comme les noms des vecteurs du DataFrame (librement attribuables, par exemple par les noms de colonnes du fichier chargé).
- ✓ Ordre des vecteurs date et séries chronologiques dans le DataFrame :
    - o Si l'ordre ci-dessus est respecté dans le DataFrame, celui-ci pourra être passé tel quel en argument de la fonction PrepGR() (et ShinyGR() le cas échéant)
    - o S'il n'est pas respecté, la fonction PrepGR() (et ShinyGR() le cas échéant), offre le moyen de passer individuellement les séries dans plusieurs arguments

  C'est pourquoi pour faciliter la transposition de l'exemple de script plus loin à un jeu de données personnel, on pourra par précaution respecter l'ordre ci-dessus au moment de la définition du DataFrame. Dans le cas contraire les pièges sont nombreux, notamment, si les données sont lues dans un fichier (la colonne de date en sortie de

`read.table()` est de type « facteur » ou « texte » (voir plus bas) et doit ensuite être convertie au format R en prenant soin de réaffecter la date convertie au vecteur date intial, sous peine que la date ne soit plus dans le premier vecteur du du DataFrame () !).

Détails sur les séries climatiques

- ✓ Faute de données ETP disponibles, la série peut être estimée avec la fonction PEdaily_Oudin() à partir de la température journalière et de la latitude (voir plus bas)
- ✓ Répartition selon l'altitude des Précipitations liquides/solides et de la Température:
  Le module « CemaNeige » utilise la courbe hypsométrique et l'altitude des données de Température (voir ci-dessous), pour répartir automatiquement selon les tranches d'altitude les précipitations pluie/neige, ainsi que déterminer les parts du stock neigeux qui fondent ou non à partir de la seule température moyenne sur le bassin
  ATTENTION : l'altitude de référence des données de Précipitation et Température est fournie en argument `ZInputs` de CreateInputsModel() § 4.2.3.1 ci-dessous (et non dans l'objet de caractéristiques du bassin comme on pourrait s'y attendre !)
- ✓ Par défaut, CemaNeige n'utilise pas l'option d'hystérésis qui permet a priori d'affiner la constitution du stock neigeux (à éclaircir cependant ; voir § 5.5.1 ci-dessous).

```
data(L0123001)
summary(BasinObs)
## DatesR P T E
## Min. :1984-01-01 Min. : 0.000 Min. :-18.700 Min. :0.000
## 1st Qu.:1991-04-02 1st Qu.: 0.000 1st Qu.: 4.100 1st Qu.:0.600
## Median :1998-07-02 Median : 0.300 Median : 9.100 Median :1.400
## Mean :1998-07-02 Mean : 2.915 Mean : 9.147 Mean :1.764
## 3rd Qu.:2005-10-01 3rd Qu.: 3.600 3rd Qu.: 14.500 3rd Qu.:2.900
## Max. :2012-12-31 Max. :66.800 Max. : 28.400 Max. :5.500
##
## Qls Qmm
## Min. : 70 Min. : 0.0168
## 1st Qu.: 1640 1st Qu.: 0.3936
## Median : 4050 Median : 0.9720
## Mean : 6130 Mean : 1.4713
## 3rd Qu.: 7850 3rd Qu.: 1.8840
## Max. :99500 Max. :23.8800
## NA's :772 NA's :772
```

Exemples de création de l'objet des séries chronologiques (DataFrame)

Exemple de script : Voir l'exemple complet de création des 2 objets au § 4.2.2.3 ci-dessous.

```
#! -------------- sources
#! source directory
pathSrc <- "../01_src"
#! time series
pathTsRab <- file.path(pathSrc, "Rabioux-Chateauroux_QMJ.csv")

#! ----------------------------- package loading
library(airGR)

#! ----------------------------- data loading and formating
#! -------------- time series
#! reading
tsRab <- read.table(pathTsRab, sep = ";", header = TRUE)

#! renaming columns
colnames(tsRab) <- c("Date", "Qls")
```

```
##### Missing data (average on the basin):
#####    - Precip : total precipitation (mm)
#####    - PET    : potential evapotranspiration (mm)
#####    - Temp   : temperature (Degre C)

#! conversion of Q from l/s to mm/d
tsRab$Qmm <- 0.0864 * tsRab$Qls / metaRab$Area

#!  conversion of dates from character to POSIXct format
tsRab$Date <- as.POSIXct(tsRab$Date, tz = "UTC", format = "%d/%m/%Y")
```

Echantillons d'objets Séries Chronologiques inclus dans « airGR »

NB : La librairie intègre les exemples d'objets suivants (appelés « BasinObs ») pour les besoins des exemples fournis dans la documentation:

- ✓ L0123001, L0123002 or L0123003 : Bassins versants ficitfs et données
- ✓ X0310010 : Durance à Embrun [La Clapière] (EDF)
  - Débits issu de la banque Hydro
  - Meteorological forcing (precipitation, temperature, potential evapotranspiration) derived from the SAFRAN reanalysis from Météo-France (Vidal et al., 2010).
  - MODIS snow cover area (SCA) data retrieved from the National Snow and Ice Data Center (NSIDC) repository (https://nsidc.org/). Five SCA time series are given, corresponding to 5 elevation bands of the CemaNeige model (default configuration). SCA data for days with important cloudiness (> 40 %) were set to missing values for the sake of data representativeness.

Times series for alls catchments (L0123001, L0123002 and X0310010) are at the daily time step for use with GR4J, GR5J, GR6J, CemaNeigeGR4J, CemaNeigeGR5J and CemaNeigeGR6J.

Times series for X0310010 are provided in order to test hysteresis version of CemaNeige (see CreateRunOptions (Riboust et al., 2019).

Times series for L0123003 are at the hourly time step for use with GR4H.

Fonction « airGR » utilisable pour estimer l'ETP

Pour calculer si besoin l'ETP à partir de la température journalière et de la latitude (ETP indispensable en argument `PotEvap` de `CreateInputsModel()` § 4.2.3.1 ci-dessous) :

`PEdaily_Oudin()` :

- `Julian days`
- `daily average air temperature`
- `latitude`

Grammaire « R » pour la création de l'objet DataFrame partir d'un fichier (txt, csv…)

Les 2 formats Date « POSIXt » de séries chronologiques : Tiré de la doc sur GitHub

- ✓ POSIXct : « Calendar Time » (En secondes depuis le 1er janvier 1970)
- ✓ POSIXlt : « Local Time » (`Jour Mois Année Heure min Sec`) dans toutes les syntaxes alphanumériques possibles (`May, 6 1985 ; 1990-9-1 ; 1/20/2012`)
  Ce type de format est a priori pratique (et plus universel que format « Calendar Time » dont l'origine de temps limitée à l'année 1970).

Lecture de fichier Texte : tiré de la doc R officielle

```
read.table(file, header = FALSE, sep = "", quote = "\"'",
           dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),
```

```
           row.names, col.names, as.is = !stringsAsFactors,
           na.strings = "NA", colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill = !blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#",
           allowEscapes = FALSE, flush = FALSE,
           stringsAsFactors = default.stringsAsFactors(),
           fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)
read.csv(file, header = TRUE, sep = ",", quote = "\"",
         dec = ".", fill = TRUE, comment.char = "", …)
read.csv2(file, header = TRUE, sep = ";", quote = "\"",
          dec = ",", fill = TRUE, comment.char = "", …)
read.delim(file, header = TRUE, sep = "\t", quote = "\"",
           dec = ".", fill = TRUE, comment.char = "", …)
read.delim2(file, header = TRUE, sep = "\t", quote = "\"",
            dec = ",", fill = TRUE, comment.char = "", …)
```

NB : En sortie de la fonction `read.table()`, la colonne de dates est en fait de type « facteur » ou « Texte » en fonction de l'argument `stringsAsFactors = TRUE ou FALSE`).

Sous l'environnement « RStudio » il est possible de charger manuellement un fichier en donnant un nom au dataset, ce qui peut être pratique pour une exécution unique d'un modèle (alors que le script de l'exemple plus haut est plus pratique pour des exécutions répétées car l'importation du fichier est automatisée une fois pour toutes).

- Menu « File > Import Data Set > From Text (readr)… » pour ouvrir une fenêtre de définition du format proche de celle d'Excel à l'ouverture d'un fichier Texte (ou « From Text (Base)… » pour charger sans aucun paramétrage un fichier reconnu directement)

  Cette opération du menu RStudio génère en fait le script suivant :

```
BasinObs <- read_delim("X:/../Ybourgues - BasinObs.csv",
    ";", escape_double = FALSE, trim_ws = TRUE,
    skip = 1)
View(BasinObs)
```

4.2.2.3 Exemple de script « R » pour charger un jeu complet de données personnelles

NB : Cet exemple complète utilement ceux de la documentation officielle qui ne portent pas sur les notions de R nécessaires à l'utilisation d'un jeu de données utilisateur (Script aimablement rédigé par l'auteur même de la documentation officielle !).

```
#! ----------------------------------------------------------------------------------
#! Description       : Run GR5J with Cemaneige on the Rabioux at Chateauroux
#! Authors           : Olivier Delaigue <olivier.delaigue@irstea.fr>
#! Date creation     : 2019-06-20 14:55:08
#! Date modification : 2019-06-20 15:53:36
#! Comments          :
#! ----------------------------------------------------------------------------------
#! ---------------------------- path definition
#! -------------- sources
#! source directory
pathSrc <- "../01_src"
#! time series
pathTsRab <- file.path(pathSrc, "Rabioux-Chateauroux_QMJ.csv")
#! hypsometry
pathHypsoRab <- file.path(pathSrc, "Rabioux-Chateauroux_hypso.csv")

#! -------------- results
#! results directory
pathRes <- "../02_res"

#! ---------------------------- package loading
library(airGR)
#! ---------------------------- data loading and formating
#! -------------- metadata
metaRab <- list(Code = "Rabioux",
                Name = "Torrent du Rabioux à station SCP",
                Area = 32.9)
#! -------------- time series
#! reading
tsRab <- read.table(pathTsRab, sep = ";", header = TRUE)

#! renaming columns
colnames(tsRab) <- c("Date", "Qls")
##### Missing data (average on the basin):
#####    - Precip : total precipitation (mm)
#####    - PET    : potential evapotranspiration (mm)
#####    - Temp   : temperature (°C)

#! conversion of Q from l/s to mm/d
tsRab$Qmm <- 0.0864 * tsRab$Qls / metaRab$Area

#!  conversion of dates from character to POSIXct format
tsRab$Date <- as.POSIXct(tsRab$Date, tz = "UTC", format = "%d/%m/%Y")

#! -------------- hypsometry
#! reading
hypsoRab <- read.table(pathHypsoRab, sep = ";", header = FALSE, col.names = "Alti")

#! vector of altitude quantiles
hypsoRab <- hypsoRab$Alti
```

4.2.3 Définir le modèle : 4 Fonctions « Create*() »

Tiré de la vignette « Get Started with airGR » et des différentes pages de manuel en ligne dont les liens url sont indiqués pour chaque fonction. La présentation met en évidence la succession chronologique des **fonctions « airGR »** (La description de la préparation préalable des arguments est reportée à la fin § 4.2.9 ci-dessous).

4.2.3.1 CreateInputsModel()

CreateInputsModel(FUN_MOD = RunModel_GR4J, DatesR = BasinObs$DatesR, Precip = BasinObs$P,
PotEvap = BasinObs$E)

- FUN_MOD : Nom de la fonction RunModel_GR* correspondant au modèle
- DatesR : Vecteur Date
- Precip : Vecteur Précipitations
- PotEvap : Vecteur Évapotranspiration [calculable si besoin à partir de la température journalière et de la latitude avec la fonction PEdaily_Oudin() ; voir § 4.2.2.2 ci-dessus]
- TempMean (optional) [numeric] time series of mean air temperature [°C], required to create the CemaNeige module inputs
- TempMin(optional) [numeric] time series of min air temperature [°C], possibly used to create the CemaNeige module inputs
- TempMax(optional) [numeric] time series of max air temperature [°C], possibly used to create the CemaNeige module inputs
- ZInputs(optional) [numeric] real giving the mean elevation of the Precip and Temp series (before extrapolation) [m], possibly used to create the CemaNeige module inputs
- HypsoData (optional) [numeric] vector of 101 reals: min, q01 to q99 and max of catchment elevation distribution [m], if not defined a single elevation is used for CemaNeige
- NLayers(optional) [numeric] integer giving the number of elevation layers requested [-], required to create CemaNeige module inputs, default=5
- verbose(optional) [boolean] boolean indicating if the function is run in verbose mode or not, default = TRUE

Renvoie un objet « **InputsModel** » (Liste des valeurs suivantes) à passer en argument à la fonction **CreateRunOptions**() § 4.2.3.3 ci-dessous.

- ✓ $DatesR [POSIXlt] vector of dates
- ✓ $Precip [numeric] time series of total precipitation (catchment average) [mm/time step]
- ✓ $PotEvap [numeric] time series of potential evapotranspiration (catchment average) [mm/time step], defined if FUN_MOD includes GR4H, GR4J, GR5J, GR6J, GR2M or GR1A
- ✓ $LayerPrecip [list] list of time series of precipitation (layer average) [mm/time step], defined if FUN_MOD includes CemaNeige
- ✓ $LayerTempMean [list] list of time series of mean air temperature (layer average) [°C], defined if FUN_MOD includes CemaNeige
- ✓ $LayerFracSolidPrecip [list] list of time series of solid precipitation fraction (layer average) [-], defined if FUN_MOD includes CemaNeige

Details

Users wanting to use « FUN_MOD » functions that are not included in the package must create their own « InputsModel » object accordingly.

Please note that if « CemaNeige » is used, and « ZInputs » is different than « HypsoData », then precipitation and temperature are interpolated with the « `DataAltiExtrapolation_Valery` » function.

Exemple avec « InputsModel () » : Simulation simple (sans calage)

```
library(airGR)

## loading catchment data
data(L0123001)

## preparation of the InputsModel object
InputsModel <- CreateInputsModel(FUN_MOD = RunModel_GR4J, DatesR = BasinObs$DatesR,
                                 Precip = BasinObs$P, PotEvap = BasinObs$E)
## run period selection
Ind_Run <- seq(which(format(BasinObs$DatesR, format = "%Y-%m-%d")=="1990-01-01"),
               which(format(BasinObs$DatesR, format = "%Y-%m-%d")=="1999-12-31"))
## preparation of the RunOptions object
RunOptions <- CreateRunOptions(FUN_MOD = RunModel_GR4J,
                               InputsModel = InputsModel, IndPeriod_Run = Ind_Run)
## simulation
Param <- c(X1 = 734.568, X2 = -0.840, X3 = 109.809, X4 = 1.971)
OutputsModel <- RunModel(InputsModel = InputsModel, RunOptions = RunOptions, Param = Param,
                         FUN_MOD = RunModel_GR4J)
## results preview
plot(OutputsModel, Qobs = BasinObs$Qmm[Ind_Run])

## efficiency criterion: Nash-Sutcliffe Efficiency
InputsCrit  <- CreateInputsCrit(FUN_CRIT = ErrorCrit_NSE, InputsModel = InputsModel,
                                RunOptions = RunOptions, Obs = BasinObs$Qmm[Ind_Run])
OutputsCrit <- ErrorCrit_NSE(InputsCrit = InputsCrit, OutputsModel = OutputsModel)
```

4.2.3.2 CreateIniStates()

C'est la seule fonction optionnelle dans cette phase (Les 4 autres fonctions Create*() » sont par contre obligatoires). Elle permet de définir un état initial personalisé, à transmettre en argument de la fonction CreateRunOptions() § 4.2.3.3 ci-dessous (Dans le cas contraire).

```
CreateIniStates(FUN_MOD, InputsModel, IsHyst = FALSE, ProdStore = 350,
       RoutStore = 90, ExpStore = NULL, UH1 = NULL, UH2 = NULL,
       GCemaNeigeLayers = NULL, eTGCemaNeigeLayers = NULL,
       GthrCemaNeigeLayers = NULL, GlocmaxCemaNeigeLayers = NULL,
       verbose = TRUE)
```

- FUN_MOD [function] Nom de la fonction RunModel_GR* correspondant au modèle choisi
- InputsModel [object of class InputsModel] see CreateInputsModel for details
- IsHys [boolean] boolean indicating if the hysteresis version of CemaNeige is used. See details
- ProdStore [numeric] production store level [mm]
- RoutStore [numeric] routing store level [mm]
- ExpStore (optional) [numeric] series of exponential store level (negative) [mm] for the GR6J model
- UH1 (optional) [numeric] unit hydrograph 1 levels [mm]
- UH2 (optional) [numeric] unit hydrograph 2 levels [mm]

- GCemaNeigeLayers (optional) [numeric] snow pack [mm], possibly used to create the CemaNeige model initial state
- eTGCemaNeigeLayers (optional) [numeric] snow pack thermal state [°C], possibly used to create the CemaNeige model initial state
- GthrCemaNeigeLayers (optional) [numeric] melt threshold [mm], possibly used to create the CemaNeige model initial state in case the Linear Hysteresis version is used
- GlocmaxCemaNeigeLayers (optional) [numeric] local melt threshold for hysteresis [mm], possibly used to create the CemaNeige model initial state in case the Linear Hysteresis version is used
- verbose (optional) [boolean] boolean indicating if the function is run in verbose mode or not, default = TRUE

Retourne un objet « **initStat** » éventuellement nécessaire pour CreateRunOptions() § 4.2.3.3 ci-dessous

- ✓ $Store [numeric] list of store levels ($Prod, $Rout and $Exp)
- ✓ $UH [numeric] list of unit hydrographs levels ($UH1 and $UH2)
- ✓ $CemaNeigeLayers [numeric] list of CemaNeige variables ($G and $eTG)

Détails

20 numeric values are required for « UH1 » and 40 numeric values are required for « UH2 » if GR4J, GR5J or GR6J are used (respectively 20*24 and 40*24 for the hourly model GR4H).

Nota : Depending on the « X4 » parameter value that will be provided when running the model, not all the values may be used (only the first int(X4)+1 values are used for « UH1 » and the first 2*int(X4)+1 for UH2).

« GCemaNeigeLayers » and « eTGCemaNeigeLayers » require each numeric values as many as given in « CreateInputsModel » with the « NLayers » argument. « eTGCemaNeigeLayers » values can be negatives.

The structure of the object of class « IniStates » returned is always exactly the same for all models (except for the unit hydrographs levels that contain more values with GR4H), even if some states do nt exist (e.g. UHUH1 for GR2M).
If CemaNeige is not used, « $CemaNeigeLayers$G » and « $CemaNeigeLayers$eTG » are set to NA.
Nota: the « StateEnd objects » from the outputs of « RunModel* » functions already respect the format given by the « CreateIniStates function ».

Exemple avec « CreateIniStates () » : Simulation simple (sans calage)

```
library(airGR)

## loading catchment data
data(L0123001)

## preparation of the InputsModel object
InputsModel <- CreateInputsModel(FUN_MOD = RunModel_GR4J, DatesR = BasinObs$DatesR,
                                 Precip = BasinObs$P, PotEvap = BasinObs$E)
## run period selection
Ind_Run <- seq(which(format(BasinObs$DatesR, format = "%Y-%m-%d")=="1990-01-01"),
               which(format(BasinObs$DatesR, format = "%Y-%m-%d")=="1999-12-31"))

### preparation of the IniStates object with low values of ProdStore and RoutStore
IniStates <- CreateIniStates(FUN_MOD = RunModel_GR4J, InputsModel = InputsModel,
                             ProdStore = 0, RoutStore = 0, ExpStore = NULL,
                             UH1 = c(0.52, 0.54, 0.15, rep(0, 17)),
```

```
                            UH2 = c(0.057, 0.042, 0.015, 0.005, rep(0, 36)),
                            GCemaNeigeLayers = NULL, eTGCemaNeigeLayers = NULL,
                            GthrCemaNeigeLayers = NULL, GlocmaxCemaNeigeLayers = NULL)
str(IniStates)

## preparation of the RunOptions object
RunOptions <- CreateRunOptions(FUN_MOD = RunModel_GR4J, InputsModel = InputsModel,
                               IndPeriod_WarmUp = 0L,
                               IndPeriod_Run = Ind_Run, IniStates = IniStates)
## simulation
Param <- c(X1 = 257.238, X2 = 1.012, X3 = 88.235, X4 = 2.208)
OutputsModel <- RunModel_GR4J(InputsModel = InputsModel,
                              RunOptions = RunOptions, Param = Param)
## results preview
plot(OutputsModel, Qobs = BasinObs$Qmm[Ind_Run])

### preparation of the IniStates object with high values of ProdStore and RoutStore
IniStates <- CreateIniStates(FUN_MOD = RunModel_GR4J, InputsModel = InputsModel,
                            ProdStore = 450, RoutStore = 100, ExpStore = NULL,
                            UH1 = c(0.52, 0.54, 0.15, rep(0, 17)),
                            UH2 = c(0.057, 0.042, 0.015, 0.005, rep(0, 36)),
                            GCemaNeigeLayers = NULL, eTGCemaNeigeLayers = NULL,
                            GthrCemaNeigeLayers = NULL, GlocmaxCemaNeigeLayers = NULL)
str(IniStates)

## preparation of the RunOptions object
RunOptions <- CreateRunOptions(FUN_MOD = RunModel_GR4J, InputsModel = InputsModel,
                               IndPeriod_WarmUp = 0L,
                               IndPeriod_Run = Ind_Run, IniStates = IniStates)
## simulation
Param <- c(X1 = 257.238, X2 = 1.012, X3 = 88.235, X4 = 2.208)
OutputsModel <- RunModel_GR4J(InputsModel = InputsModel,
                              RunOptions = RunOptions, Param = Param)
## results preview
plot(OutputsModel, Qobs = BasinObs$Qmm[Ind_Run])
```

4.2.3.3 *CreateRunOptions ()*

CreateRunOptions(FUN_MOD = RunModel_GR4J,InputsModel = InputsModel,
IndPeriod_Run = Ind_Run, IniStates = NULL, IniResLevels = NULL,
IndPeriod_WarmUp = NULL)

- FUN_MOD : Nom de fonction RunModel_GR* correspondant au modèle GR choisi
- InputsModel : Objet « **InputsModel** » créé par la fonction précédente
- IndPeriod_Run : Période de simulation définie par fonction R **seq()** § 4.2.9 ci-dessous
- IniStates : États initiaux des 2 Hydrogrammes Unitaires (20 + 40 = 60 units) Voir plus bas la valeur par défaut
- IniResLevels : Niveaux initiaux des réservoirs de Production et Transfert Voir plus bas la valeur par défaut
- IndPeriod_WarmUp : Période de mise en route (même format que « IndPeriod_Run ») Voir plus bas la valeur par défaut
- Outputs_Cal (optional) [character] vector giving the outputs needed for the calibration (e.g. c("Qsim")), the fewer outputs the faster the calibration
- Outputs_Sim (optional) [character] vector giving the requested outputs (e.g. c("DatesR", "Qsim", "SnowPack")), default = "all"
- RunSnowModule (deprecated) [boolean] option indicating whether CemaNeige should be activated. Please adapt FUN_MOD instead
- MeanAnSolidPrecip (optional) [numeric] vector giving the annual mean of average solid precipitation for each layer (computed from InputsModel if not defined) [mm/y]
- IsHyst [boolean] boolean indicating if the hysteresis version of CemaNeige is used. See details
- warnings (optional) [boolean] boolean indicating if the warning messages are shown, default = TRUE

- `verbose(optional)` [boolean] boolean indicating if the function is run in verbose mode or not, default = TRUE

`NULL` à la place d'un argument indique le choix d'une des valeurs par défaut ci-dessous, et un message de warning le signale comme dans l'exemple ci-après:

- ✓ « IniStates » & « IniResLevels »=0
- ✓ « IniResLevels »=30% & 50% des niveaux des réservoirs de Production et Transfert
- ✓ « IndPeriod_WarmUp »=1 an de mise en route

Renvoie un objet « **RunOptions** » (de type List » avec éléments ci-dessous) pour préparer les options nécessaires à l'exécution d'une des 10 fonctions `RunModel_*()` correspondant au modèle choisi (Cf. § 4.2.6 ci-dessous).

- ✓ IndPeriod_WarmUp [numeric] index of period to be used for the model warm-up [-]
- ✓ IndPeriod_Run [numeric] index of period to be used for the model run [-]
- ✓ IniStates [numeric] vector of initial model states [mm and °C]
- ✓ IniResLevels [numeric] vector of initial filling rates for production and routing stores [-] and level for the exponential store for GR6J [mm]
- ✓ Outputs_Cal [character] character vector giving only the outputs needed for the calibration
- ✓ Outputs_Sim [character] character vector giving the requested outputs
- ✓ MeanAnSolidPrecip [numeric] vector giving the annual mean of average solid precipitation for each layer [mm/day]

Détails

Objet « RunOptions » personnalisé

Users wanting to use `FUN_MOD` functions that are not included in the package must create their own « RunOptions » object accordingly.

Choix de la variante de « CemaNeige »

- ✓ IsHyst = FALSE : Variante originale de CemaNeige de Valéry et al. (2014).
- ✓ IsHyst = TRUE : Variante de Riboust et al. (2019).

Voir la différence entre les 2 variantes § 5.5.1 ci-dessous

Initialisation options :

The model initialisation options can either be set to a default configuration or be defined by the user.

This is done via three vectors: « IndPeriod_WarmUp », « IniStates », « IniResLevels ».

A default configuration is used for initialisation if these vectors are not defined.

(1) Default initialisation options:

« IndPeriod_WarmUp » default setting ensures a one-year warm-up using the time steps preceding the « IndPeriod_Run ». The actual length of this warm-up might be shorter depending on data availability (no missing value of climate inputs being allowed in model input series).

« IniStates » and « IniResLevels » are automatically set to initialise all the model states at 0, except for the production and routing stores which are respectively initialised at

30 % and 50 % of their capacity. In case GR6J is used, the exponential store is initialised by default with 0 mm. This initialisation is made at the very beginning of the model call (i.e. at the beginning of « IndPeriod_WarmUp » or at the beginning of « IndPeriod_Run » if the warm-up period is disabled).

(2) Customisation of initialisation options:

✓ « IndPeriod_WarmUp » can be used to specify the indices of the warm-up period (within the time series prepared in InputsModel).

remark 1: for most common cases, indices corresponding to one or several years preceding « IndPeriod_Run » are used (e.g. « IndPeriod_WarmUp » = 1000:1365 and « IndPeriod_Run » = 1366:5000).
However, it is also possible to perform a long-term initialisation if other indices than the warm-up ones are set in « IndPeriod_WarmUp » (e.g. « IndPeriod_WarmUp » = c(1:5000, 1:5000, 1:5000, 1000:1365)).

remark 2: it is also possible to completely disable the warm-up period when using « IndPeriod_WarmUp = 0L ». This is necessary if you want « IniStates » and/or « IniResLevels » to be the actual initial values of the model variables from your simulation (e.g. to perform a forecast form a given initial state).

✓ « IniStates » and « IniResLevels » can be used to specify the initial model states.

remark 1: « IniStates » and « IniResLevels » can not be used with GR1A.
remark 2: if « IniStates » is used, two possibilities are offered:
- « IniStates » can be set to the « $StateEnd » output of a previous « RunModel » call, as « $StateEnd » already respects the correct format;
- « IniStates » can be created with the « CreateIniStates » function.

remark 3: in addition to « IniStates », « IniResLevels » allows to set the filling rate of the production and routing stores for the GR models. For instance :

- for GR4J and GR5J: IniResLevels = c(0.3, 0.5) should be used to obtain initial fillings of 30 % and 50 % for the production and routing stores, respectively.
- For GR6J, IniResLevels = c(0.3, 0.5, 0) should be use to obtain initial fillings of 30 % and 50 % for the production, routing stores and 0 mm for the exponential store, respectively.

« IniResLevels » is optional and can only be used if « IniStates » is also defined (the state values corresponding to these two other stores in « IniStates » are not used in such case).

Exemple avec « CreateRunOptions () » : Simulation simple (sans calage)

Idem exemple de CreateInputsModel() § 4.2.3.1 ci-dessus.

4.2.3.4 CreateInputsCrit ()

```
CreateInputsCrit(FUN_CRIT = ErrorCrit_NSE, InputsModel = InputsModel,
                 RunOptions = RunOptions, Obs = BasinObs$Qmm[Ind_Run])
```

- FUN_CRIT : Fonction critère (voir plus loin les choix possibles)
- InputsModel : Objet « **InputsModel** » créé par la fonction ci-dessus CreateInputsModel()
- RunOptions : Objet « **RunOptions** » créé par la fonction ci-dessus CreateRunOptions()
- Obs : (deprecated) Liste des débits observés [mm/time step] ([mm/time step] pour débit ou SWE, [-] pour SCA)
- VarObs (optional) Liste des noms de variables observées ("Q" par défaut, "SCA", "SWE")
- BoolCrit (optional) Liste de booléens (Même longueur que Obs) pour le choix des différents pas de temps des calcul de critères (TOUS les pas de temps par défaut)
- transfo (optional) Liste de noms de transformation (e.g. "", "sqrt", "log", "inv", "sort")
- Weights (optional) Liste des poids pour le calcul d'un critère composite (même longueur que FUN_CRIT) pour les fonctions FUN_CRIT [-]. See details
- Ind_zeroes (deprecated) [numeric] indices of the time steps where zeroes are observed
- Epsilon (optional) [numeric (atomic or list)] small value to add to all observations and simulations when "log" or "inv" transformations are used [same unit as Obs]. See details
- warnings (optional) [boolean] boolean indicating if the warning messages are shown, default = TRUE
- verbose (deprecated) [boolean] boolean indicating if the function is run in verbose mode or not, default = TRUE

Renvoie un objet « **InputsCrit** » (de type liste avec les éléments ci-dessous) à passer en argument aux fonctions ErrorCrit_*() (§ 4.2.4 ci-dessous).

- ✓ $FUN_CRIT [function] error criterion function (e.g. ErrorCrit_RMSE, ErrorCrit_NSE)
- ✓ $Obs [numeric] series of observed variable(s) ([mm/time step] for discharge or SWE, [-] for SCA)
- ✓ $VarObs [character] names of the observed variable(s)
- ✓ $BoolCrit [boolean] boolean giving the time steps considered in the computation
- ✓ $transfo [character] name of the transformation (e.g. "", "sqrt", "log", "inv", "sort")
- ✓ $epsilon [numeric] small value to add to all observations and simulations when "log" or "inv" transformations are used [same unit as Obs]
- ✓ $Weights [numeric] vector (same length as VarObs) giving the weights to use for elements of FUN_CRIT [-]

- ❖ When Weights = NULL, « CreateInputsCrit » returns an object of class « Single » that is a list such as the one described above.
- ❖ When « Weights » contains at least one NULL value and « Obs » contains a list of observations, « CreateInputsCrit » returns an object of class « Multi » that is a list of lists such as the one described above. The « ErrorCrit » function will then compute the different criteria prepared by « CreateInputsCrit ».
- ❖ When « Weights » is a list of at least 2 numerical values, « CreateInputsCrit » returns an object of class « Compo » that is a list of lists such as the one described above. This object will be useful to compute composite criterion with the « ErrorCrit » function.

❖ To calculate composite or multiple criteria, it is necessary to use the « ErrorCrit function ». The other « ErrorCrit_* » functions (e.g. « ErrorCrit_RMSE », « ErrorCrit_NSE ») can only use objects of class « Single » (and not Multi or Compo).

Détails

Users wanting to use « FUN_CRIT » functions that are not included in the package must create their own « InputsCrit » object accordingly.

The epsilon value is useful when "log" or "inv" transformations are used (to avoid calculation of the inverse or of the logarithm of zero). The impact of this value and a recommendation about the epsilon value to use (usually one hundredth of average observation) are discussed in Pushpalatha et al. (2012) for NSE and in Santos et al. (2018) for KGE and KGE'.

We do not advise computing KGE or KGE' with log-transformation as it might be wrongly influenced by discharge values close to 0 or 1 and the criterion value is dependent on the discharge unit. See Santos et al. (2018) for more details and alternative solutions (see the references list below).

Users can set the following arguments as atomic or list: FUN_CRIT, Obs, VarObs, BoolCrit, transfo, Weights.
If the list format is chosen, all the lists must have the same length.

- ✓ Calculation of a single criterion (e.g. NSE computed on discharge) is prepared by providing to « CreateInputsCrit » arguments atomics only.
- ✓ Calculation of multiple criteria (e.g. NSE computed on discharge and RMSE computed on discharge) is prepared by providing to « CreateInputsCrit » arguments lists except for « Weights » that must be set as NULL.
- ✓ Calculation of a composite criterion (e.g. the average between NSE computed on dscharge and NSE computed on log of discharge) is prepared by providing to « CreateInputsCrit » arguments lists including Weights.

« ErrorCrit_RMSE » cannot be used in a composite criterion since it is not a unitless value.

Exemple avec « CreateInputsCrit () » : Simulation simple (sans calage)

```
library(airGR)

## loading catchment data
data(L0123001)

## preparation of the InputsModel object
InputsModel <- CreateInputsModel(FUN_MOD = RunModel_GR4J, DatesR = BasinObs$DatesR,
                                 Precip = BasinObs$P, PotEvap = BasinObs$E)
## calibration period selection
Ind_Run <- seq(which(format(BasinObs$DatesR, format = "%Y-%m-%d")=="1990-01-01"),
               which(format(BasinObs$DatesR, format = "%Y-%m-%d")=="1999-12-31"))
## preparation of RunOptions object
RunOptions <- CreateRunOptions(FUN_MOD = RunModel_GR4J, InputsModel = InputsModel,
                               IndPeriod_Run = Ind_Run)
## simulation
Param <- c(X1 = 257.238, X2 = 1.012, X3 = 88.235, X4 = 2.208)
OutputsModel <- RunModel_GR4J(InputsModel = InputsModel, RunOptions = RunOptions, Param =
Param)
```

```
## single efficiency criterion: Nash-Sutcliffe Efficiency
InputsCritSingle <- CreateInputsCrit(FUN_CRIT = ErrorCrit_NSE,
                                     InputsModel = InputsModel, RunOptions = RunOptions,
                                     Obs = list(BasinObs$Qmm[Ind_Run]),
                                     VarObs = "Q",
                                     transfo = "",
                                     Weights = NULL)
str(InputsCritSingle)
invisible(ErrorCrit(InputsCrit = InputsCritSingle, OutputsModel = OutputsModel))

## 2 efficiency criteria: RMSE and Nash-Sutcliffe Efficiency
InputsCritMulti <- CreateInputsCrit(FUN_CRIT = list(ErrorCrit_RMSE, ErrorCrit_NSE),
                                    InputsModel = InputsModel, RunOptions = RunOptions,
                                    Obs = list(BasinObs$Qmm[Ind_Run],
                                    BasinObs$Qmm[Ind_Run]),
                                    VarObs = list("Q", "Q"),
                                    transfo = list("", "sqrt"),
                                    Weights = NULL)
str(InputsCritMulti)
invisible(ErrorCrit(InputsCrit = InputsCritMulti, OutputsModel = OutputsModel))

## efficiency composite criterion: Nash-Sutcliffe Efficiency mixing
##                                 both raw and log-transformed flows
InputsCritCompo <- CreateInputsCrit(FUN_CRIT = list(ErrorCrit_NSE, ErrorCrit_NSE),
                                    InputsModel = InputsModel, RunOptions = RunOptions,
                                    Obs = list(BasinObs$Qmm[Ind_Run],
                                    BasinObs$Qmm[Ind_Run]),
                                    VarObs = list("Q", "Q"),
                                    transfo = list("", "log"),
                                    Weights = list(0.4, 0.6))
str(InputsCritCompo)
invisible(ErrorCrit(InputsCrit = InputsCritCompo, OutputsModel = OutputsModel))
```

4.2.3.5 *CreateCalibOptions ()*

CreateCalibOptions(FUN_MOD, FUN_CALIB = Calibration_Michel, FUN_TRANSFO = NULL,

IsHyst = FALSE, FixedParam = NULL, SearchRanges = NULL,

StartParamList = NULL, StartParamDistrib = NULL)

- FUN_MOD : Nom de la fonction correspondant au modèle
- FUN_CALIB: Nom de la fonction de calage (Calibration_Michel est le seul algorthme implémenté, mais l'utilisateur peut fournir une fonction personnalisée)
- FUN_TRANSFO (optional) [function] model parameters transformation function, if the FUN_MOD used is native in the package, FUN_TRANSFO is automatically defined
- IsHyst [boolean] boolean indicating if the hysteresis version of CemaNeige is used. See details
- FixedParam (optional) [numeric] vector giving the values set for the non-optimised parameter values (NParam columns, 1 line)
  Example:
  ```
  NA    NA    3.34   ...   NA
  ```
- SearchRanges (optional) [numeric] matrix giving the ranges of real parameters (NParam columns, 2 lines)
  Example:
  ```
         [X1]  [X2]  [X3]  [...]  [Xi]
  [1,]   0     -1    0     ...    0.0
  [2,]   3000  +1    100   ...    3.0
  ```
- StartParamList(optional) [numeric] matrix of parameter sets used for grid-screening calibration procedure (values in columns, sets in line)
  Example:
  ```
         [X1]  [X2]  [X3]  [...]  [Xi]
  [set1] 800   -0.7  25    ...    1.0
  [set2] 1000  -0.5  22    ...    1.1
  ```

```
[...]  ...    ...    ...    ...    ...
[set n]200    -0.3   17     ...    1.0
```

- StartParamDistrib (optional) [numeric] matrix of parameter values used for grid-screening calibration procedure (values in columns, percentiles in line)

```
Example:
             [X1]   [X2]   [X3]   [...]  [Xi]
[value1]     800    -0.7   25     ...    1.0
[value2]     1000   NA     50     ...    1.2
[value3]     1200   NA     NA     ...    1.6
```

Renvoie un objet « **CalibOptions** » (Liste des valeurs suivantes) nécessaire au calage (§ 4.2.5 ci-dessous)

- ✓ $FixedParam [numeric] vector giving the values to allocate to non-optimised parameter values
- ✓ $SearchRanges [numeric] matrix giving the ranges of raw parameters
- ✓ $StartParamList [numeric] matrix of parameter sets used for grid-screening calibration procedure
- ✓ $StartParamDistrib [numeric] matrix of parameter values used for grid-screening calibration procedure

Détails

Fonctions personnalisées « FUN_MOD », « FUN_CALIB » et « FUN_TRANSFO »
Users wanting to use functions that are not included in the package must create their own « CalibOptions » object accordingly.

Variantes de « CemaNeige »
If « IsHyst » = FALSE, the original CemaNeige version from Valéry et al. (2014) is used.
If « IsHyst » = TRUE, the CemaNeige version from Riboust et al. (2019) is used.
Voir les différences des 2 variantes § 5.5.1 ci-dessous

Exemple avec « CreateCalibOptions () » : Simulation simple (sans calage)

```
library(airGR)

## loading catchment data
data(L0123001)

## preparation of InputsModel object
InputsModel <- CreateInputsModel(FUN_MOD = RunModel_GR4J, DatesR = BasinObs$DatesR,
                                 Precip = BasinObs$P, PotEvap = BasinObs$E)
## calibration period selection
Ind_Run <- seq(which(format(BasinObs$DatesR, format = "%Y-%m-%d")=="1990-01-01"),
               which(format(BasinObs$DatesR, format = "%Y-%m-%d")=="1999-12-31"))
## preparation of RunOptions object
RunOptions <- CreateRunOptions(FUN_MOD = RunModel_GR4J,
                               InputsModel = InputsModel, IndPeriod_Run = Ind_Run)
## calibration criterion: preparation of the InputsCrit object
InputsCrit <- CreateInputsCrit(FUN_CRIT = ErrorCrit_NSE, InputsModel = InputsModel,
                               RunOptions = RunOptions, Obs = BasinObs$Qmm[Ind_Run])
## preparation of CalibOptions object
CalibOptions <- CreateCalibOptions(FUN_MOD = RunModel_GR4J, FUN_CALIB = Calibration_Michel)

## calibration
OutputsCalib <- Calibration(InputsModel = InputsModel, RunOptions = RunOptions,
                            InputsCrit = InputsCrit, CalibOptions = CalibOptions,
                            FUN_MOD = RunModel_GR4J, FUN_CALIB = Calibration_Michel)
## simulation
Param <- OutputsCalib$ParamFinalR
```

```
OutputsModel <- RunModel(InputsModel = InputsModel, RunOptions = RunOptions,
                         Param = Param, FUN = RunModel_GR4J)
## results preview
plot(OutputsModel, Qobs = BasinObs$Qmm[Ind_Run])

## efficiency criterion: Nash-Sutcliffe Efficiency
InputsCrit  <- CreateInputsCrit(FUN_CRIT = ErrorCrit_NSE, InputsModel = InputsModel,
                                RunOptions = RunOptions, Obs = BasinObs$Qmm[Ind_Run])
OutputsCrit <- ErrorCrit_NSE(InputsCrit = InputsCrit, OutputsModel = OutputsModel)

## efficiency criterion: Kling-Gupta Efficiency
InputsCrit  <- CreateInputsCrit(FUN_CRIT = ErrorCrit_KGE, InputsModel = InputsModel,
                                RunOptions = RunOptions, Obs = BasinObs$Qmm[Ind_Run])
OutputsCrit <- ErrorCrit_KGE(InputsCrit = InputsCrit, OutputsModel = OutputsModel)
```

4.2.4 Choisir le critère d'évaluation à calculer : « ErrorCrit_*() »

Tiré de la vignette « Get Started with airGR » et complété.

Le critère est à choisir parmi les suivants (ou à définir par l'utilisateur dans une fonction « R » personnalisée) :

- ✓ `ErrorCrit_RMSE()`: Root mean square error (RMSE)
- ✓ `ErrorCrit_NSE()`: Nash-Sutcliffe model efficiency coefficient (NSE)
- ✓ `ErrorCrit_KGE()`: Kling-Gupta efficiency criterion (KGE)
- ✓ `ErrorCrit_KGE2()`: modified Kling-Gupta efficiency criterion (KGE')

NB: Le critère KGE (Kling-Gupta) est préférable au critère NSE classique (Nash) pour optimiser le calage.

```
ErrorCrit_KGE(InputsCrit, OutputsModel, warnings = TRUE, verbose = TRUE)
```

- o InputsCrit [object of class InputsCrit] see CreateInputsCrit for details
- o OutputsModel [object of class OutputsModel] see RunModel_GR4J() § 5.3 ci-dessous or RunModel_CemaNeigeGR4J() § 5.5.3 ci-dessous
- o warnings (optional) [boolean] boolean indicating if the warning messages are shown, default = TRUE
- o verbose (optional) [boolean] boolean indicating if the function is run in verbose mode or not, default = TRUE

Retourne la **Liste des valeurs suivantes**, commençant par la **valeur du critère choisi** et comprenant aussi un entier (-1 ou +1) à multiplier par la valeur du critère pour le calage du modèle avec la fonction d'optimisation § 4.2.5 ci-dessous.

- ✓ $CritValue [numeric] value of the criterion
- ✓ $CritName [character] name of the criterion
- ✓ $CritBestValue [numeric] theoretical best criterion value
- ✓ $Multiplier [numeric] integer indicating whether the criterion is indeed an error (+1) or an efficiency (-1)
- ✓ $Ind_notcomputed [numeric] indices of the time steps where InputsCrit$BoolCrit = FALSE or no data is available

Exemple avec « ErrorCrit_* » : Simulation simple (sans calage)

```
library(airGR)

## loading catchment data
data(L0123001)
```

```
## preparation of the InputsModel object
InputsModel <- CreateInputsModel(FUN_MOD = RunModel_GR4J, DatesR = BasinObs$DatesR,
                                 Precip = BasinObs$P, PotEvap = BasinObs$E)
## run period selection
Ind_Run <- seq(which(format(BasinObs$DatesR, format = "%Y-%m-%d")=="1990-01-01"),
               which(format(BasinObs$DatesR, format = "%Y-%m-%d")=="1999-12-31"))
## preparation of the RunOptions object
RunOptions <- CreateRunOptions(FUN_MOD = RunModel_GR4J,
                               InputsModel = InputsModel, IndPeriod_Run = Ind_Run)
## simulation
Param <- c(X1 = 734.568, X2 = -0.840, X3 = 109.809, X4 = 1.971)
OutputsModel <- RunModel(InputsModel = InputsModel, RunOptions = RunOptions,
                         Param = Param, FUN = RunModel_GR4J)
## efficiency criterion: Nash-Sutcliffe Efficiency
InputsCrit <- CreateInputsCrit(FUN_CRIT = ErrorCrit_NSE, InputsModel = InputsModel,
                               RunOptions = RunOptions, Obs = BasinObs$Qmm[Ind_Run])
OutputsCrit <- ErrorCrit_NSE(InputsCrit = InputsCrit, OutputsModel = OutputsModel)

## efficiency criterion: Nash-Sutcliffe Efficiency on log-transformed flows
transfo <- "log"
InputsCrit <- CreateInputsCrit(FUN_CRIT = ErrorCrit_NSE, InputsModel = InputsModel,
                               RunOptions = RunOptions, Obs = BasinObs$Qmm[Ind_Run],
                               transfo = transfo)
OutputsCrit <- ErrorCrit_NSE(InputsCrit = InputsCrit, OutputsModel = OutputsModel)

## efficiency criterion: Kling-Gupta Efficiency above a threshold (quant. 75 %)
BoolCrit <- BasinObs$Qmm[Ind_Run] >= quantile(BasinObs$Qmm[Ind_Run], 0.75, na.rm = TRUE)
InputsCrit <- CreateInputsCrit(FUN_CRIT = ErrorCrit_NSE, InputsModel = InputsModel,
                               RunOptions = RunOptions, Obs = BasinObs$Qmm[Ind_Run],
                               BoolCrit = BoolCrit)
OutputsCrit <- ErrorCrit_NSE(InputsCrit = InputsCrit, OutputsModel = OutputsModel)
```

4.2.5 Caler le modèle : « Calibration_Michel() »

Tiré de la vignette « Get Started with airGR » avec une présentation mettant en évidence le déroulement du processus (les fonctions « R » nécessaires à la définition des arguments sont reportées à la fin), et complété.

NB : Cette étape peut être omise dans le cadre d'une simple simulation (notamment pour le contrôle d'un modèle déjà calé ; 4.2.8 ci-dessous) ; si les paramètres du modèle sont imposés par l'utilisateur (comme dans TOUS les exemples précédents de ce pas-à-pas).

Cette fonction implémente le seul algorithme implémenté à ce jour, mais l'utilisateur peut programmer sa propre fonction d'optimisation, comme expliqué aux pages suivantes de la doc en ligne:

- Fonction Calibration() cette fonction remplace alors Calibration_Michel()
- Plugging in new calibration algorithms in airGR
- Parameter estimation within a Bayesian MCMC framework (Algorithme par tirage aléatoire de « Monte Carlo Markov Chain » non implémenté à ce jour)

```
Calibration_Michel(InputsModel = InputsModel, RunOptions = RunOptions,
                   InputsCrit = InputsCrit, CalibOptions = CalibOptions,
                   FUN_MOD = RunModel_GR4J, FUN_CRIT = ErrorCrit_NSE,
                   FUN_TRANSFO = NULL, verbose = TRUE)
```

- InputsModel [object of class InputsModel] Cf. § 4.2.3.1 ci-dessus
- RunOptions [object of class RunOptions] Cf. § 4.2.3.3 ci-dessus
- InputsCrit [object of class InputsCrit] Cf. § 4.2.3.4 ci-dessus

- CalibOptions [object of class CalibOptions] Cf. § 4.2.3.5 ci-dessus
- FUN_MOD [function] hydrological model function (e.g. RunModel_GR4J, RunModel_CemaNeigeGR4J)
- FUN_CRIT (deprecated)[function] error criterion function (e.g. ErrorCrit_RMSE, ErrorCrit_NSE)
- FUN_TRANSFO (optional) [function] model parameters transformation function, if the FUN_MOD used is native in the package FUN_TRANSFO is automatically defined
- verbose (optional) [boolean] boolean indicating if the function is run in verbose mode or not, default = TRUE

Retourne un objet « **OutputsCalib** » composé des éléments ci-dessous, notamment les valeurs du critère d'optimisation choisi (`OutputsCalib$CritFinal`), ainsi que les paramètres optimisés (`OutputsCalib$ParamFinalR`) à passer en argument de `RunModel_GR4J()` pour lancer une simulation (Cf. § 4.2.6 ci-dessous).

- ✓ $ParamFinalR [numeric] parameter set obtained at the end of the calibration
- ✓ $CritFinal [numeric] error criterion selected as objective function obtained at the end of the calibration
- ✓ $NIter [numeric] number of iterations during the calibration
- ✓ $NRuns [numeric] number of model runs done during the calibration
- ✓ $HistParamR [numeric] table showing the progression steps in the search for optimal set: parameter values
- ✓ $HistCrit [numeric] table showing the progression steps in the search for optimal set: criterion values
- ✓ $MatBoolCrit [boolean] table giving the requested and actual time steps over which the model is calibrated
- ✓ $CritName [character] name of the calibration criterion used as objective function
- ✓ $CritBestValue [numeric] theoretical best criterion value

Détails

A screening is first performed either based on a rough predefined grid (considering various initial values for each parameter) or from a list of initial parameter sets.
The best set identified in this screening is then used as a starting point for the steepest descent local search algorithm.
For this search, since the ranges of parameter values can be quite different, simple mathematical transformations are applied to parameters to make them vary in a similar range and get a similar sensitivity to a predefined search step. This is done using the TransfoParam functions.
During the steepest descent method, at each iteration, we start from a parameter set of NParam values (NParam being the number of free parameters of the chosen hydrological model) and we determine the 2*NParam-1 new candidates by changing one by one the different parameters (+/- search step).
All these candidates are tested and the best one kept to be the starting point for the next iteration. At the end of each iteration, the the search step is either increased or decreased to adapt the progression speed. A composite step can occasionally be done.
The calibration algorithm stops when the search step becomes smaller than a predefined threshold.

Exemple avec « Calibration_Michel() » : Seul exemple avec calage de la doc

```
library(airGR)

## loading catchment data
data(L0123001)
```

```
## preparation of InputsModel object
InputsModel <- CreateInputsModel(FUN_MOD = RunModel_GR4J, DatesR = BasinObs$DatesR,
                                 Precip = BasinObs$P, PotEvap = BasinObs$E)
## calibration period selection
Ind_Run <- seq(which(format(BasinObs$DatesR, format = "%Y-%m-%d")=="1990-01-01"),
               which(format(BasinObs$DatesR, format = "%Y-%m-%d")=="1999-12-31"))
## preparation of RunOptions object
RunOptions <- CreateRunOptions(FUN_MOD = RunModel_GR4J, InputsModel = InputsModel,
                               IndPeriod_Run = Ind_Run)
## calibration criterion: preparation of the InputsCrit object
InputsCrit <- CreateInputsCrit(FUN_CRIT = ErrorCrit_NSE, InputsModel = InputsModel,
                               RunOptions = RunOptions, Obs = BasinObs$Qmm[Ind_Run])
## preparation of CalibOptions object
CalibOptions <- CreateCalibOptions(FUN_MOD = RunModel_GR4J, FUN_CALIB = Calibration_Michel)

## calibration
OutputsCalib <- Calibration_Michel(InputsModel = InputsModel, RunOptions = RunOptions,
                                   InputsCrit = InputsCrit, CalibOptions = CalibOptions,
                                   FUN_MOD = RunModel_GR4J)
## simulation
Param <- OutputsCalib$ParamFinalR
OutputsModel <- RunModel_GR4J(InputsModel = InputsModel,
                              RunOptions = RunOptions, Param = Param)
## results preview
plot(OutputsModel, Qobs = BasinObs$Qmm[Ind_Run])

## efficiency criterion: Nash-Sutcliffe Efficiency
InputsCrit  <- CreateInputsCrit(FUN_CRIT = ErrorCrit_NSE, InputsModel = InputsModel,
                                RunOptions = RunOptions, Obs = BasinObs$Qmm[Ind_Run])
OutputsCrit <- ErrorCrit_NSE(InputsCrit = InputsCrit, OutputsModel = OutputsModel)

## efficiency criterion: Kling-Gupta Efficiency
InputsCrit  <- CreateInputsCrit(FUN_CRIT = ErrorCrit_KGE, InputsModel = InputsModel,
                                RunOptions = RunOptions, Obs = BasinObs$Qmm[Ind_Run])
OutputsCrit <- ErrorCrit_KGE(InputsCrit = InputsCrit, OutputsModel = OutputsModel)
```

4.2.6 Lancer le modèle : « RunModel_*() »

Une fois définis la modélisation et les paramètres du modèle (Par calage ou non, comme indiqué aux paragraphes précédents de ce pas-à-pas), on peut effectuer la simulation par l'une des 10 fonctions correspondant au modèle choisi, décrites en détails aux liens ci-dessous :

- ✓ RunModel_GR1A() § 5.1 ci-dessous
- ✓ RunModel_GR2M() § 5.2 ci-dessous
- RunModel_GR4J() § 5.3 ci-dessous
- ✓ RunModel_GR5J() § 5.3 ci-dessous
- ✓ RunModel_GR6J() § 5.3 ci-dessous
- ✓ RunModel_GR4H() § 5.4 ci-dessous
- ✓ RunModel_CemaNeige() § 5.5.1 ci-dessous
- ✓ RunModel_CemaNeigeGR4J() § 5.5.3 ci-dessous
- ✓ RunModel_CemaNeigeGR5J() § 5.5.3 ci-dessous
- ✓ RunModel_CemaNeigeGR6J() § 5.5.3 ci-dessous

```
RunModel_*(InputsModel, RunOptions, Param)
```

- InputsModel : [object of class InputsModel] Voir CreateInputsModel §
- RunOptions : [object of class RunOptions] Voir CreateRunOptions
- Param : [numeric] vector of 4 parameters
  - GR4J X1 production store capacity [mm]
  - GR4J X2 intercatchment exchange coefficient [mm/d]
  - GR4J X3 routing store capacity [mm]
  - GR4J X4 unit hydrograph time constant [d]
  - GR5J X5 intercatchment exchange threshold [-]
  - GR6J X6 coefficient for emptying exponential store [mm]
  - Cemaneige X1 weighting coefficient for snow pack thermal state [-]
  - Cemaneige X2 degree-day melt coefficient [mm/°C/d]
  - Cemaneige X3 (optional) accumulation threshold [mm] (needed if IsHyst = TRUE)
  - Cemaneige X4 (optional) percentage (between 0 and 1) of annual snowfall defining the melt threshold [-] (needed if IsHyst = TRUE)

Retourne un objet « **OutputModel** » composée des **listes et séries chronologiques** suivantes qui pourront être passés en argument de la fonction `plot()` § 4.2.7 ci-dessous :

- ✓ $DatesR[POSIXlt] series of dates
- ✓ $PotEvap [numeric] series of input potential evapotranspiration [mm/d]
- ✓ $Precip[numeric] series of input total precipitation [mm/d]
- ✓ $Prod [numeric] series of production store level [mm]
- ✓ $Pn [numeric] series of net rainfall [mm/d]
- ✓ $Ps [numeric] series of the part of Pn filling the production store [mm/d]
- ✓ $AE [numeric] series of actual evapotranspiration [mm/d]
- ✓ $Perc [numeric] series of percolation (PERC) [mm/d]
- ✓ $PR [numeric] series of PR=Pn-Ps+Perc [mm/d]
- ✓ $Q9 [numeric] series of UH1 outflow (Q9) [mm/d]
- ✓ $Q1 [numeric] series of UH2 outflow (Q1) [mm/d]
- ✓ $Rout [numeric] series of routing store level [mm]
- ✓ $Exch [numeric] series of potential semi-exchange between catchments [mm/d]
- ✓ $AExch1[numeric] series of actual exchange between catchments for branch 1 [mm/d]
- ✓ $AExch2[numeric] series of actual exchange between catchments for branch 2 [mm/d]
- ✓ $AExch [numeric] series of actual exchange between catchments (1+2) [mm/d]
- ✓ $QR [numeric] series of routing store outflow (QR) [mm/d]
- ✓ $QD [numeric] series of direct flow from UH2 after exchange (QD) [mm/d]
- ✓ $Qsim [numeric] series of simulated discharge [mm/d]

- ✓ $CemaNeigeLayers [list] list of CemaNeige outputs (1 list per layer)
  - $CemaNeigeLayers[[iLayer]]$Pliq [numeric] series of liquid precip. [mm/d]
  - $CemaNeigeLayers[[iLayer]]$Psol [numeric] series of solid precip. [mm/d]
  - $CemaNeigeLayers[[iLayer]]$SnowPack [numeric] series of snow pack [mm]
  - $CemaNeigeLayers[[iLayer]]$ThermalState [numeric] series of snow pack thermal state [°C]
  - $CemaNeigeLayers[[iLayer]]$Gratio [numeric] series of Gratio [0-1]
  - $CemaNeigeLayers[[iLayer]]$PotMelt [numeric] series of potential snow melt [mm/d]
  - $CemaNeigeLayers[[iLayer]]$Melt [numeric] series of actual snow melt [mm/d]
  - $CemaNeigeLayers[[iLayer]]$PliqAndMelt [numeric] series of liquid precip. + actual snow melt [mm/d]
  - $CemaNeigeLayers[[iLayer]]$Temp [numeric] series of air temperature [°C]
  - $CemaNeigeLayers[[iLayer]]$Gthreshold [numeric] series of melt threshold [mm]
  - $CemaNeigeLayers[[iLayer]]$Glocalmax [numeric] series of local melt threshold for hysteresis [mm]
- ✓ $StateEnd [numeric] states at the end of the run: store & unit hydrographs levels [mm], CemaNeige states [mm & °C], see CreateIniStates()§ 4.2.3 ci-dessus

Exemple avec « RunModel_GR4J »

```
library(airGR)
## loading catchment data
data(L0123001)
## preparation of the InputsModel object
InputsModel <- CreateInputsModel(FUN_MOD = RunModel_GR4J, DatesR = BasinObs$DatesR,
               Precip = BasinObs$P, PotEvap = BasinObs$E)
## run period selection
Ind_Run <- seq(which(format(BasinObs$DatesR, format = "%Y-%m-%d")=="1990-01-01"),
               which(format(BasinObs$DatesR, format = "%Y-%m-%d")=="1999-12-31"))
## preparation of the RunOptions object
RunOptions <- CreateRunOptions(FUN_MOD = RunModel_GR4J,
                               InputsModel = InputsModel, IndPeriod_Run = Ind_Run)
## simulation
Param <- c(X1 = 257.238, X2 = 1.012, X3 = 88.235, X4 = 2.208)
OutputsModel <- RunModel_GR4J(InputsModel = InputsModel,
                              RunOptions = RunOptions, Param = Param)
## results preview
plot(OutputsModel, Qobs = BasinObs$Qmm[Ind_Run])

## efficiency criterion: Nash-Sutcliffe Efficiency
InputsCrit  <- CreateInputsCrit(FUN_CRIT = ErrorCrit_NSE, InputsModel = InputsModel,
                                RunOptions = RunOptions, Obs = BasinObs$Qmm[Ind_Run])
OutputsCrit <- ErrorCrit_NSE(InputsCrit = InputsCrit, OutputsModel = OutputsModel)
```

4.2.7 Visualiser graphiquement les résultats : « plot() »

```
plot(x, Qobs = NULL, IndPeriod_Plot = NULL, BasinArea = NULL, which = "all",
     log_scale = FALSE,  cex.axis = 1, cex.lab = 0.9, cex.leg = 0.9,
     lwd = 1, verbose = TRUE, ...)
```

- x [object of class OutputsModel] list of model outputs (which must at least include DatesR, Precip and Qsim) [POSIXlt, mm, mm]
- Qobs (optional) [numeric] time series of observed flow (for the same time steps than simulated) [mm/time step]
- IndPeriod_Plot(optional) [numeric] Période (indices du vecteur date) à représenter (among the OutputsModel series)
- BasinArea (optional) [numeric] basin area [km2], used to plot flow axes in m3/s
- which (optional) [character] choice of plots (e.g. c("Precip", "Temp", "SnowPack", "Flows", "Regime", "CumFreq", "CorQQ")), default = "all"
- log_scale (optional) [boolean] indicating if the flow axis is to be logarithmic, default = FALSE
- log_scale (optional) [numeric] the magnification to be used for axis annotation relative to the current setting of cex
- cex.leg (optional) [numeric] the magnification to be used for x and y labels relative to the current setting of cex
- cex.leg (optional) [numeric] the magnification to be used for the legend labels relative to the current setting of cex
- lwd (optional) [numeric] the line width (a positive number)
- verbose (optional) [boolean] indicating if the function is run in verbose mode or not, default = TRUE
- ... other parameters

Affiche une **fenêtre avec les graphiques** suivants selon le modèle:

```
(1) time series of total precipitation
(2) time series of temperature (plotted only if CemaNeige is used)
(3) time series of snow pack (plotted only if CemaNeige is used)
(4) time series of simulated flows (and observed flows if provided)
(5) interannual median monthly simulated flow (and observed flows if provided)
(6) correlation plot between simulated and observed flows (if observed flows provided)
(7) cumulative frequency plot for simulated flows (and observed flows if provided)
```

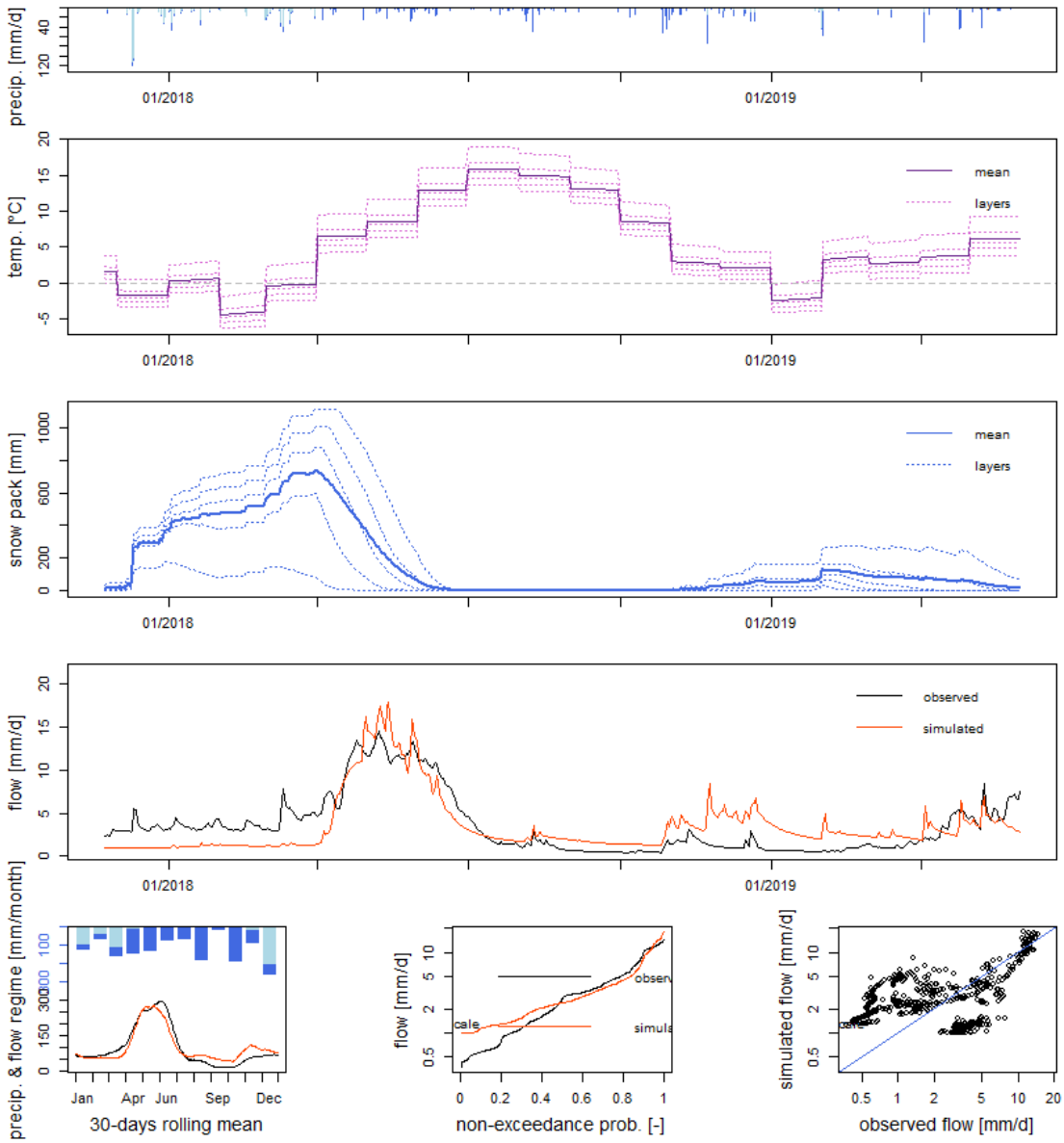
Exemple avec « plot() »

Tous les exemples fournis aux autres paragraphes comprennent au moins une fonction `plot()` !

En cas d'échec de la fonction « plot() » dans « RStudio »

Sous « RStudio » si le graphique ne s'affiche avec le message d'erreur suivant (« `Error in plot.new() : figure margins too large` »), il suffit d'élargir le volet « RStudio » dans lequel le graphique s'affiche (en bas à droite de la fenêtre « Files Plots Packages Help Viewer »). Le volet peut ensuite être rétécit à nouveau jusqu'à la prochaine exécution.

NB : Le nom réel `plot.OutputsModel()` de la fonction est abrégé en `plot()` (grâce à la fonctionnalité « S3 method » de « R » qui permet d'abréger par un nom générique unique les méthodes plot() ou print() spéciques à différentes classes).



Exemple pour CemaNeige (Températures mensuelles uniquement, ce qui explique la performance médiocre du modèle)

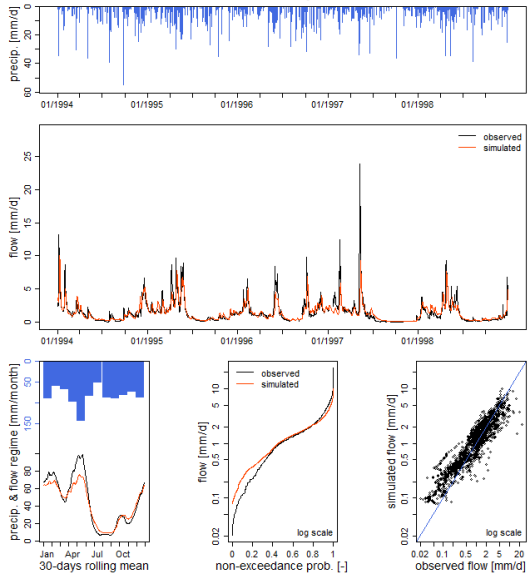
4.2.8 Contrôler la modélisation

Traduit librement de la vignette « Get Started with airGR ».

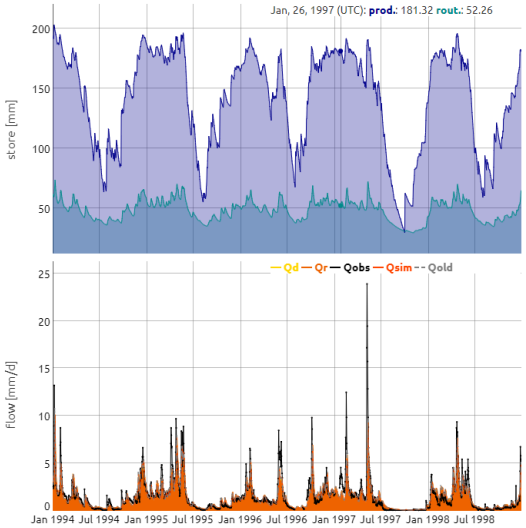
Cette opération consiste à évaluer la robustesse du modèle, c'est-à-dire sa capacité à conserver des performances stables en dehors des conditions de calage. La méthode classique de contrôle consiste effectuer une simulation sur une période des données n'ayant pas servie au calage. Autant que possible, cette période de contrôle devrait correspondre à une situation climatique différente de celles de la période de calage afin de mettre en évidence les qualités et les faiblesses du modèle.

Avec « airGR » le contrôle s'effectue en effectuant une simulation comme expliqué aux paragraphes précédents jusqu'à § 4.2.6 ci-dessus (mais sans l'étape de calage § 4.2.5), puis à constater si les critères de performance sont stables ou non par rapport à la période de calage.

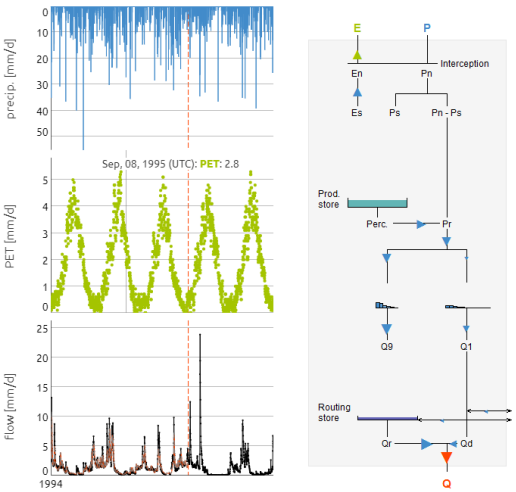
Model Performance



State variables



Model Diagram



4.2.9 Fonctions « R » utiles

Pour connaitre la version installée de R : `version`

Pour les besoins de la fonction `seq()` ci-dessous :

```
which(x, arr.ind = FALSE, useNames = TRUE)
```

- X — Condition (ou vector/array de conditions avec valeurs manquantes (NA) autorisées et traitées comme FALSE)
- arr.ind — booléen indiquant si which renvoie la position des éléments quand X est de type « array »
- useNames — logical indicating if the value of arrayInd() should have (non-null) dimnames at all

Retourne une **liste** avec la position des éléments (ligne, colonne, index de tableau) pour lesquel la condition donnée est vraie

Exemple de seq() et wich() :

```
seq(which(format(BasinObs$DatesR, format = "%d/%m/%Y")=="01/01/1990"),
    which(format(BasinObs$DatesR, format = "%d/%m/%Y")=="31/12/1999"))
```

- BasinObs$DatesR : Vecteur Date (comme en argument de CreateInputsModel() § 4.2.3.1 ci-dessus)

Retourne un **objet R de type « séquence »** (Période de simulation) à passer à l'argument de différentes fonctions (par exemple `IndPeriod_Run` de la fonction CreateRunOptions() ci-dessus).

```
str(Ind_Run)
## int [1:3652] 2193 2194 2195 2196 2197 2198 2199 2200 2201 2202 ...
```

Pour créer une séquence pour le besoin de certains arguments de différentes fonctions:

```
seq(from=, to=, by=, length.out=, along.whith=)
```

- from, to — the starting and (maximal) end values of the sequence. Of length 1 unless just from is supplied as an unnamed argument.
- by — number: increment of the sequence.
- length.out — desired length of the sequence. A non-negative number, which for seq and seq.int will be rounded up if fractional.
- along.whith — take the length from the length of this argument

Retourne une **séquence de nombres**.

Pour créer une Liste de plusieurs valeurs/objets pour le besoin de certains arguments de différentes fonctions:

```
c(…, recursive = FALSE, use.names = TRUE)
```

- … — Objets à combiner
- recursive — logical. If recursive = TRUE, the function recursively descends through lists (and pairlists) combining all their elements into a vector.
- use.names — logical indicating if names should be preserved.

Retourne un **objet « List »** composé des objets combinés

Pour rendre un objet invisible à l'affichage tant qu'il n'est pas assigné (mais assignable)

```
invisible(x)
```

Retourne l'objet intiale quelconque, sous une forme assignable mais pas affichable tant qu'il n'est pas lui-même assigné (Cf. § 4.2.3.4 ci-dessus)

Pour enregistrer dans un fichier des résultats de modélisation, notamment les débits simulés :

```
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ", eol = "\n", na = "NA",
            dec = ".", row.names = TRUE, col.names = TRUE, qmethod = c("escape","double"),
           fileEncoding = "")
```

- `col.names` vecteur de noms de colonnes ou FALSE/TRUE (utilise les noms de variables)
-

Voir aussi « Gestion des données avec R » pour une description de la structure des données dans « R ».

4.3 Utilisation de « airGRteaching »

4.3.1 Présentation

Librement traduit du PDF « airGRteaching.pdf ».

« airGRteaching » est un package pédagogique qui vient en complément du package « airGR » pour en simplifier l'utilisation (Modèles journaliers UNIQUEMENT).

Il permet d'utiliser les modèles avec des compétences en programmation très faibles. Ce package fournit également un utilitaire graphique pour aider les étudiants à explorer les données et les résultats de la modélisation.

Les noms des fonctions et leurs arguments ont été choisis pour faciliter l'utilisation des fonctionnalités « airGR ».

Le package est basé sur 3 familles de fonctions:

- ✓ les fonctions permettant de réaliser très simplement un exercice de modélisation hydrologique;
- ✓ des fonctions de graphiques pour aider les étudiants à explorer les données observées et à interpréter les résultats du calage et de la simulation des modèles GR
- ✓ une fonction qui exécute une interface graphique « Shiny » qui permet d'afficher en temps réel les paramètres du modèle sur les hydrogrammes

Fonctions de modélisation

Trois fonctions permettent de réaliser très simplement une modélisation hydrologique:

- ✓ préparation des données: **PrepGR** ()
- ✓ calage des modèles: **CalGR** ()
- ✓ Simulation: **SimGR** ()

Fonctions de graphiques

Deux types de fonctions permettent de produire des graphiques statiques (**plot** ()) ou dynamiques (**dyplot**() ; événements de souris et graphiques interactifs).
Ces fonctions permettent ainsi d'explorer les données observées et d'interpréter les résultats de calage et de simulation des modèles.

Interface « Shyny »

La fonction **ShinyGR**() fournit une interface web interactive (Shiny) permettant (Modèle journalier UNIQUEMENT):

- ✓ des simulations avec modification interactives de paramètres;
- ✓ une calibration automatique;
- ✓ l'affichage de l'évolution des variables internes;
- ✓ la sélection de la période affichée.

Les modèles

Les 6 modèles Pluie-Débit de « airGR » ainsi que le modèle de fonte et d'accumulation de neige sont également disponibles dans « airGRteaching », par appel des noms de modèle suivants:

- ✓ GR4H
- ✓ GR4J *
- ✓ GR5J *
- ✓ GR6J *
- ✓ GR2M
- ✓ GR1A
- ✓ CemaNeige: combiné avec GR4J, GR5J ou GR6J

*: disponible dans l'interface Shiny.

4.3.2 Utilisation pas-à-pas de « airGRteaching »

Tiré de la description intégrée à l'interface web « ShinyGR » (Accès par le menu « Functionalities » 

) mais non disponible par ailleurs ni en ligne ni en PDF.

Ce pas-à pas présenté dans l'ordre chronologique d'exécution des fonctions, est complété par une description détaillée de chaque fonction au paragraphe § 4.3.3 ci-dessous.

On rappelle que la librairie « airGRteaching » fournit des fonctions supplémentaires pour :

- ✓ Effectuer une modélisation en 4 instructions seulement (y compris la fonction de graphique, mais sans compter la création des objets de jeu de données qui reste à coder en R)
- ✓ Regrouper tous les résultats numériques/graphiques dans une interface interactive web, qui permet d'effecter une simulation en sélectionnant le modèle GR*J/Cemaneige, et ou un calage en changeant de critière d'optimisation (via la simple commande « ShinyGR » placée dans une boucle ; Cf. § 4.3.2).

4.3.2.1 Preparation des données observées : PrepGR()

L'étape de préparation des données observées est assurée par la fontion `PrepGR()`, qui se substitue donc à la fonction standards `CreateInputsModel()`, avec en plus le choix du type de modèle. En effet, les arguments sont les suivants (Voir arguments en détail § 4.3.3.1 ci-dessous) :

- `ObsDF`: data.frame of hydrometeorological observations time series
- Les arguments suivants peuvent être utilisés à la place de `ObsDF` (`DatesR`, `Precip`, `PotEvap`, `Qobs`, `TempMean`)
  If one variable is provided in « ObsDF » and also separately, then only the data included in « ObsDF » are used.
- `HydroModel`: the name of the hydrological model you want to run (GR1A, GR2M, GR4J, GR5J, GR6J or GR4H)
- `CemaNeige`: TRUE if you want to use the snowmelt and accumulation model

Les autres arguments ci-dessous sont nécessaires uniquement pour utiliser Cemaneige:

- Vecteur Température dans le dataFrame de l'argument `ObsDF` ou dans l'argument `TempMean`
- `HypsoData`: a vector of 101 reals: min, quantiles (1 % to 99 %) and max of catchment elevation distribution [m]; if not defined a single elevation layer is used for CemaNeige
- `NLayers`: nombre de couches d'altitude the number of elevation layers requested [-], required to create CemaNeige module inputs (if used)

Séries chronologiques (au pas journalier UNIQUEMENT)

Les séries chronologiques suivantes sont à fournir soit sous forme d'un DataFrame unique dans l'argument `ObsDF`, soit de vecteurs séparés dans les 5 arguments décrits ci-dessus:

- ✓ DatesR : dates — POSIXt format
- ✓ P : average precipitation — [mm/time step]
- ✓ T : catchment average air temperature — [°C] [OPTIONAL]
- ✓ E : catchment average potential evapotranspiration — [mm/time step]
- ✓ Qmm : outlet discharge — [mm/time step]

```
head(BasinObs)
##       DatesR    P   E    Qmm    T
## 1 1984-01-01  4.1 0.2 0.6336  0.5
## 2 1984-01-02 15.9 0.2 0.8256  0.2
## 3 1984-01-03  0.8 0.3 2.9280  0.9
## 4 1984-01-04  0.0 0.3 1.8240  0.5
## 5 1984-01-05  0.0 0.1 1.5000 -1.6
## 6 1984-01-06  0.0 0.3 1.3560  0.9
```

Exemple

```
CalGR(PrepGR, CalCrit = c("NSE", "KGE", "KGE2", "RMSE"), WupPer = NULL, CalPer,
      transfo = c("", "sqrt", "log", "inv", "sort"), verbose = TRUE)
```

4.3.2.2 Calage : CalGR()

L'étape de calage est assurée par la fonction `CalGR()`. Cette fonction se substitue à la fonction `Calibration_Michel()` de « airGR » (§ 4.2.5 ci-dessus), mais sans les arguments optionnels, et sans répéter les arguments déjà passé à la fonction `PrepGR()` à l'étape précédente (contrairement à « airGR » qui nécessite de répéter les mêmes arguments dans les fonctions successives). Les arguments nécessaires sont les suivants (Voir arguments en détail § 4.3.3.2 ci-dessous):

- PrepGR: the object returned by the PrepGR() function
- CalPer: a vector of 2 dates to define the calibration period
- transfo (optional) [character] name of the transformation transformation applied to discharge for calculating the objective function (must be one of "", "sqrt", "log", "inv" or "sort")
- verbose (optional) [boolean] logical value indicating if the function is run in verbose mode or not

Si les arguments suivants sont omis, le calage utilise le critère « Nash–Sutcliffe » (NSE), et la période de mise en route est automatiquement fixée (en fonction du modèle utilisé):

- CalCrit: name of the objective function ("NSE", "KGE", "KGE2", "RMSE")
- WupPer: a vector of 2 dates to define the warm-up period

Exemple

```
CAL <- CalGR(PrepGR = PREP, CalCrit = "KGE2",
             WupPer = NULL, CalPer = c("1990-01-01", "1993-12-31"))

## Grid-Screening in progress (0% 20% 40% 60% 80% 100%)
##   Screening completed (243 runs)
##       Param =  175.915 ,   -0.110 ,   83.931 ,    1.857 ,    0.467
##       Crit KGE'[Q]      = 0.8300
## Steepest-descent local search in progress
##   Calibration completed (18 iterations, 406 runs)
##       Param =  188.670 ,    1.456 ,   83.931 ,    1.779 ,    0.493
##       Crit KGE'[Q]      = 0.8787
```

4.3.2.3 Simulation : SimGR()

L'étape de simulation, assurée par la fonction `SimGR()` qui se substitue à une des fonctions « airGR » de type `RunModel_*()` (§ 4.2.6 ci-dessus), à la différence que le type de modèle est choisi au préalable dans la fonction PrepGR(). Les arguments nécessaires sont les suivants (Voir arguments en détail § 4.3.3.3 ci-dessous) :

- PrepGR et WupPer : idem fonction CalGR()
- EffCrit : critère de performance à calculer
- SimPer : période de simulation

Exemple

```
SIM <- SimGR(PrepGR = PREP, CalGR = CAL, EffCrit = "KGE2", WupPer = NULL,
             SimPer = c("1994-01-01", "1998-12-31"))
## Crit. KGE'[Q] = 0.8545
##  SubCrit. KGE'[Q] cor(sim, obs, "pearson") = 0.9008
##  SubCrit. KGE'[Q] cv(sim)/cv(obs)          = 0.8969
##  SubCrit. KGE'[Q] mean(sim)/mean(obs)      = 0.9737
```

4.3.2.4 Fonctions de graphiques prédéfinis : plot() & dyplot()

La fonction dyplot() de « airGRteaching » permet de créer un graphique dynamique des séries observées et simulées. De plus des graphiques complémentaires peuvent être obtenus avec la fonction plot() standard appliquée aux objets PREP, CAL ou SIM de « airGRteaching ».

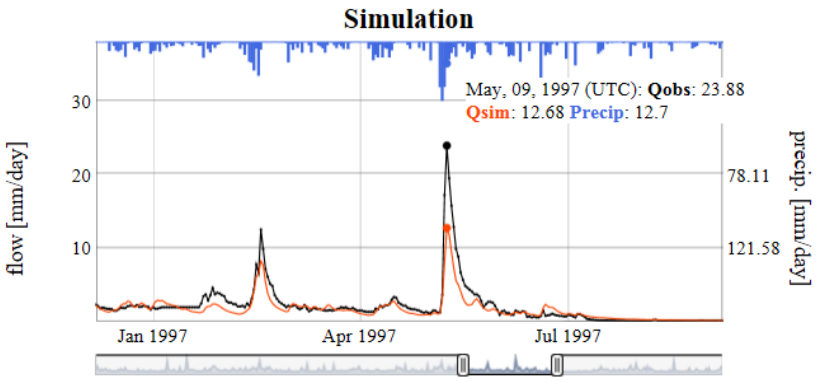
4.3.2.4.1 Dynamic plots : dyplot()

Cette fonction permet de générer un graphique intéractif pour représenter les séries chronologiques de type « PrepGR », « CalGR » ou « SimGR ». Voir les arguments en détail au § 4.3.3.4 ci-dessous.

The user can zoom on the plot device and can read the exact values. With this function, users can easily explore the data time series and also explore and interpret the possible problems of the calibration or simulation steps.

Exemple : dyplot() + SIM

```
dyplot(SIM, main = "Simulation")
```



4.3.2.4.2 Static plots : plot()

En complément du graphique Précipitations-Débit de dyplot(), la fonction plot() standard de « airGR » peut être utilisée avec les objets PREP et CAL de « airGRteaching » pour produire les graphiques standards de « airGR » (Snowpack, Temperature, Régime, CumFreqs, CorrelQQ), ainsi que des graphiques supplémentaires (évolution des paramètres au cours de l'optimisation).

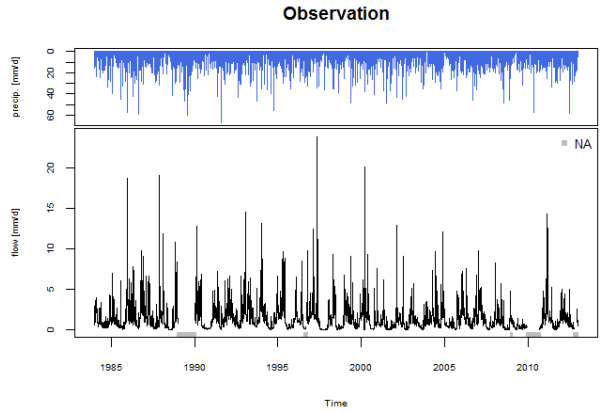
PREP

With a « PrepGR » object), the call of the plot() function draws the observed precipitation and discharge time series (avec un 2ème axe des débits gradué en m3/s)

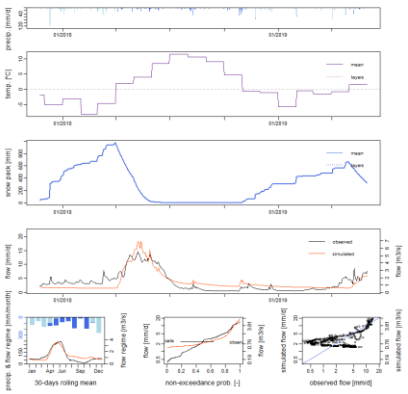
CAL + which="perf"

With the « CalGR » object, if the argument `which = "perf"` (by default), the call of the plot() function draws the classical airGR plot diagnostics (observed and simulated time series together with diagnostic plot)

```
plot(PREP, main = "Observation")
```




```
plot(CAL, which = "perf")
```

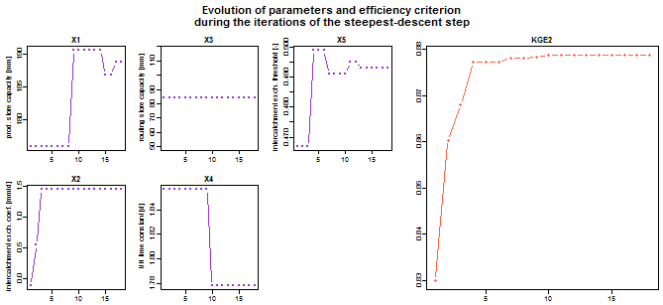


Idem graphiques standards de plot()
(avec bug sur NLayers et légende à ce jour)

CAL + which="iter"

With the « CalGR » object, if the argument `which` is set to "iter", the plot() function draws the evolution of the parameters and the values of the objective function during the second step of the calibration (steepest descent local search algorithm):

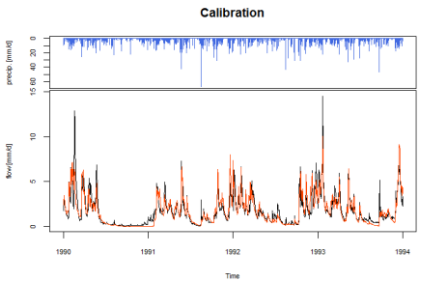
```
plot(CAL, which = "iter")
```



CAL + which="ts"

With the « CalGR » object, if the argument `which` is set to "ts", the plot() function simply draws the time series of the observed precipitation, and the observed and simulated flows :

```
plot(CAL, which = "ts", main = "Calibration")
```



SIM

With a « SimGR » object the plot() function draws the classical airGR plot diagnostics.

```
plot(SIM)
```

Idem graphiques standards de plot()

4.3.2.4.3 Interface web interactive: ShinyGR()

En alternative aux 2 fonctions de graphique ci-dessus, « airGRteaching » permet de générer une interface interactive web grâce à la fonction « ShinyGR() » décrites au paragraphe § 4.3.3.5 ci-dessous.

4.3.3 Fonctions de « airGRteaching » en détail

La présentation est prévue pour accélérer l'assimilation du processus. Pour cela, les étapes préparatoires éventuelles sont reportées après les **fonctions « airGR »** elles-mêmes :

- ✓ Les **fonctions « airGR »** sont décrites en premier dans l'ordre chronologique de lancement,
- ✓ et pour chaque `argument de fonction` demandant une préparation préalable (Objet complexes au lieu d'un simple paramètre), un bref renvoi indique où trouver les instructions nécessaires.

Pour trouver des exemples de mise en œuvre de ces fonctions, voir le chapitre 5 ci-dessous consacré aux différents modèles.

4.3.3.1 Préparation des entrées : PrepGR()

Tiré de « airGRteaching.pdf »

Cette fonction permet de créer les entrées nécessaires à « CalGR » et « SimGR ».

```
PrepGR(ObsDF = NULL,
       DatesR = NULL, Precip = NULL, PotEvap = NULL, Qobs = NULL, TempMean = NULL,
       ZInputs = NULL, HypsoData = NULL, NLayers = 5,
       HydroModel= "GR4J", CemaNeige = FALSE)
```

- ObsDF (optional): data.frame (or a list of data.frame) avec les vecteurs impérativement dans l'ordre suivant (date, precipitation, evapotranspiration, discharge, temperature only if CemaNeige is used) see below for the units
  Si l'argument « ObsDF » est utilisé, les arguments suivants ne sont plus nécessaires (DatesR, Precip, PotEvap, Qobs, TempMean), et vice-versa
  If one variable is provided in « ObsDF » and also separately, then only the data included in « ObsDF » are used.
- DatesR (optional) [POSIXt] vector of dates required to create the GR and CemaNeige (if used) models inputs. Time zone must be defined as "UTC"
- DatesR (optional) [numeric] time series of total precipitation (catchment average) [mm/time step], required to create the GR and CemaNeige (if used) models inputs
- PotEvap (optional) [numeric] time series of potential evapotranspiration (catchment average) [mm/time step], required to create the GR model inputs
- Qobs (optional) [numeric] time series of observed discharges [mm/time step]
- TempMean (optional) [numeric] time series of mean air temperature [°C], required to create the CemaNeige model inputs

- ZInputs (optional) [numeric] real giving the mean elevation of the Precip and Temp-Mean series (before extrapolation) [m], possibly used to create the CemaNeige (if used) model inputs
- HypsoData (optional) [numeric] vector of 101 reals: min, q01 to q99 and max of catchment elevation distribution [m]; if not defined a single elevation is used for CemaNeige (if used)
- NLayers (optional) [numeric] integer giving the number of elevation layers requested [-], required to create CemaNeige model inputs (if used)
- HydroModel [character] name of the hydrological model (must be one of "GR1A", "GR2M", "GR4J", "GR5J", "GR6J" or "GR4H")
- CemaNeige [boolean] option indicating whether CemaNeige should be activated (only available when HydroModel is equal to any of "GR4J", "GR5J" or "GR6J")

Retourne un objet « **PrepGr** » (Liste des résultats ci-dessous) contenant les résultats requis pour évaluer les sorties du modèle.

- ✓ InputsModel [list] object of class InputsModel containing the data required to evaluate the model outputs (Mêmes sorties que la fonction CreateInputsModel() § 4.2.3.1 ci-dessus)
- ✓ Qobs [numeric] time series of observed discharges [mm/time step]
- ✓ HydroModel [character] name of the function of the hydrological model used

Exemple de code avec « PrepGR »

```
library(airGRteaching)

## data.frame of observed data
data(L0123001)
BasinObs2 <- BasinObs[, c("DatesR", "P", "E", "Qmm", "T")]

## Preparation of observed data for modelling
PREP <- PrepGR(ObsDF = BasinObs2, HydroModel = "GR4J", CemaNeige = FALSE)
str(PREP)
```

4.3.3.2 Calage: CalGR()

Tiré de « airGRteaching.pdf »

Calibration algorithm that optimises the error criterion selected as objective function using the Irstea-HBAN procedure described by C. Michel.

```
CalGR(PrepGR, CalCrit = c("NSE", "KGE", "KGE2", "RMSE"),
      WupPer = NULL, CalPer,
      transfo = c("", "sqrt", "log", "inv", "sort"), verbose = TRUE)
```

- PrepGR [object of class PrepGR] see PrepGR for details
- CalCrit [character] name of the objective function (must be one of "NSE", "KGE", "KGE2" or "RMSE")
- WupPer (optional) [character] vector of 2 values to define the beginning and end of the warm-up period ["YYYY-mm-dd" or "YYYY-mm-dd HH:MM:SS"]
- CalPer [character] vector of 2 values to define the beginning and end of the calibration period ["YYYY-mm-dd" or "YYYY-mm-dd HH:MM:SS"]
- transfo (optional) [character] name of the transformation transformation applied to discharge for calculating the objective function (must be one of "", "sqrt", "log", "inv" or "sort")
- verbose (optional) [boolean] logical value indicating if the function is run in verbose mode or not

Retourne un objet « **CalGR** » (de type « List » avec les éléments ci-dessous).

- ✓ OptionsCalib [list] object of class RunOptions (see: CreateRunOptions)
- ✓ Qobs [numeric] series of observed discharges [mm/time step]
- ✓ OutputsCalib [list] object of class OutputsCalib (see: Calibration)
- ✓ OutputsModel [list] object of class OutputsModel (see: RunModel)
- ✓ TypeModel [character] name of the function of the hydrological model used
- ✓ CalCrit [character] name of the function that computes the error criterion during the calibration step
- ✓ PeriodModel [list] $WarmUp: vector of 2 POSIXct values defining the beginning and end of the warm-up period, $Run: vector of 2 POSIXct values defining the beginning and end of the calibration period

Exemple de code avec « CalGR »

```
library(airGRteaching)

## data.frame of observed data
data(L0123001)
BasinObs2 <- BasinObs[, c("DatesR", "P", "E", "Qmm", "T")]

## Preparation of observed data for modelling
PREP <- PrepGR(ObsDF = BasinObs2, HydroModel = "GR4J", CemaNeige = TRUE)

## Calibration step
CAL <- CalGR(PrepGR = PREP, CalCrit = "KGE2", WupPer = NULL, CalPer = c("1990-01-01", "1993-
12-31"))
str(CAL)
```

4.3.3.3 Simulation : SimGR()

Tiré de « airGRteaching.pdf »

Cette fonction permet d'exécuter une simulation.

```
SimGR(PrepGR, CalGR = NULL, Param = NULL, EffCrit = c("NSE", "KGE", "KGE2", "RMSE"),
      WupPer = NULL, SimPer,
      transfo = c("", "sqrt", "log", "inv", "sort"), verbose = TRUE)
```

- o PrepGR [object of class PrepGR] see PrepGR for details
- o CalGR (optional) [object of class CalGR] see CalGR and below for details
- o Param (optional) [numeric] vector of parameters (the length of the vector depends on the model used), see below for details
- o EffCrit [character] name of the efficiency criterion (must be one of "NSE", "KGE", "KGE2" or "RMSE")
- o WupPer (optional) [character] vector of 2 values to define the beginning and end of the warm-up period ["YYYY-mm-dd" or "YYYY-mm-dd HH:MM:SS"]
- o SimPer [character] vector of 2 values to define the beginning and end of the simulation period ["YYYY-mm-dd" or "YYYY-mm-dd HH:MM:SS"]
- o transfo (optional) [character] name of the transformation applied to discharge for calculating the error criterion (must be one of "", "sqrt", "log", "inv" or "sort")
- o verbose (optional) [boolean] logical value indicating if the function is run in verbose mode or not

Retourne un objet « **SimGr** » (Liste des résultats ci-dessous).

- ✓ OptionsSimul [list] object of class RunOptions (see: § 4.2.3.3 ci-dessus)
- ✓ OptionsCrit [list] object of class InputsCrit (see: 4.2.3.4 ci-dessus)
- ✓ OutputsModel [list] object of class OutputsModel (see: RunModel § 4.2.6 ci-dessus)
- ✓ Qobs [numeric] series of observed discharges [mm/time step]
- ✓ TypeModel [character] name of the function of the hydrological model used

- ✓ CalCrit [character] name of the function that computes the error criterion during the calibration step
- ✓ EffCrit [list] name of the function that computes the error criterion during the simulation step
- ✓ PeriodModel [list] $WarmUp: vector of 2 POSIXct values defining the beginning and end of the warm-up period; $Run: vector of 2 POSIXct values defining the beginning and end of the calibration period

Détails

The user can customize the parameters with the « Param » argument. The user can also use the parameters resulting from a calibration. In this case, it is necessary to use the « CalGR » argument.

Exemple de code avec « SimGR »

```
library(airGRteaching)

## data.frame of observed data
data(L0123001)
BasinObs2 <- BasinObs[, c("DatesR", "P", "E", "Qmm", "T")]

## Preparation of observed data for modelling
PREP <- PrepGR(ObsDF = BasinObs2, HydroModel = "GR4J", CemaNeige = FALSE)

## Calibration step
CAL <- CalGR(PrepGR = PREP, CalCrit = "KGE2", WupPer = NULL, CalPer = c("1990-01-01", "1993-
12-31"))

## Simulation step using the result of the automatic calibration method to set the model
parameters
SIM <- SimGR(PrepGR = PREP, CalGR = CAL, EffCrit = "KGE2", WupPer = NULL, SimPer = c("1994-
01-01", "1998-12-31"))

## Simulation step using model parameters set by the user
SIM <- SimGR(PrepGR = PREP, Param = c(270.426, 0.984, 108.853, 2.149), EffCrit = "KGE2",
WupPer = NULL, SimPer = c("1994-01-01", "1998-12-31"))
str(SIM)
```

4.3.3.4 Graphique interactif de série chronologiques : dyplot()

Tiré de « airGRteaching.pdf »

Cette fonction permet de générer un graphique intéractif pour représenter les séries chronologiques de type « PrepGR », « CalGR » ou « SimGR ». Les différences avec le graphique de la fonction plot() standard sont précisées plus bas.

Les exe

```
## Default S3 method:
dyplot(x, Qsup = NULL, Qsup.name = "Qsup", col.Precip = c("royalblue", "lightblue"),
           col.Q = c("black", "orangered", "grey"), col.na = "lightgrey",
           ylab = NULL, main = NULL, plot.na = TRUE, RangeSelector = TRUE,
           Roller = FALSE, LegendShow = c("follow", "auto", "always", "onmouseover",
       "never"), ...)
```

- x [object of class PrepGR, CalGR or SimGR] see PrepGR, CalGR, SimGR for details
- Qsup (optional) [numeric] additional time series of flows (at the same time step than
- argument x) [mm/time step]

- Qsup.name (optional) [character] a label for the legend of Qsup
- col.Precip (optional) [character] vector of 1 (total precip.) or 2 (liquid and solid precip. with CemaNeige) color codes or names for precipitation (these can be of the form "#AABBCC" or "rgb(255, 100, 200)" or "yellow"), see par and rgb
- col.Q (optional) [character] vector of up to 3 color codes or names for observed (first value), simulated (second value, if provided) and additional (last value, if provided) flows, respectively (these can be of the form "#AABBCC" or "rgb(255, 100, 200)" or "yellow"), see par and rgb
- col.na (optional) [character] color code or name for missing values(these can be of the form "#AABBCC" or "rgb(255, 100, 200)" or "yellow"), see par and rgb
- ylab (optional) [character] a label for the y-axis (flow and precipitation)
- main (optional) [character] a main title for the plot
- plot.na [boolean] indicating if the missing values are plotted on the x-axis
- RangeSelector (optional) [boolean] add a range selector to the bottom of the chart that allows users to pan and zoom to various date ranges (see dyRangeSelector)
- Roller (optional) [numeric] number of time scale units (e.g. days, months, years) to average values over (see dyRoller)
- LegendShow (optional)[character] when to display the legend. Specify "always" to always show the legend. Specify "onmouseover" to only display it when a user mouses over the chart. Specify "follow" (default) to have the legend show as overlay to the chart which follows the mouse. See dyLegend
- ... other parameters to be passed through to plotting functions

Exemple de code avec « dyplot() »

```
library(airGRteaching)

## data.frame of observed data
data(L0123001, package = "airGR")
BasinObs2 <- BasinObs[, c("DatesR", "P", "E", "Qmm", "T")]

## Preparation of observed data for modelling
PREP <- PrepGR(ObsDF = BasinObs2, HydroModel = "GR4J", CemaNeige = FALSE)
dyplot(PREP, main = "Observation")

## Calibration step
CAL <- CalGR(PrepGR = PREP, CalCrit = "KGE2", WupPer = NULL, CalPer = c("1990-01-01", "1993-
12-31"))
dyplot(CAL, main = "Calibration")

## Simulation
SIM <- SimGR(PrepGR = PREP, CalGR = CAL, EffCrit = "KGE2", WupPer = NULL, SimPer = c("1994-
01-01", "1998-12-31"))

dyplot(SIM, main = "Simulation")
```

Avantages

Outre le fait de ne nécessiter que 3 fonctions préalables (Cf. paragraphes précédants), ce graphique présente les avantages suivant par rapport à la fonction plot() standard :

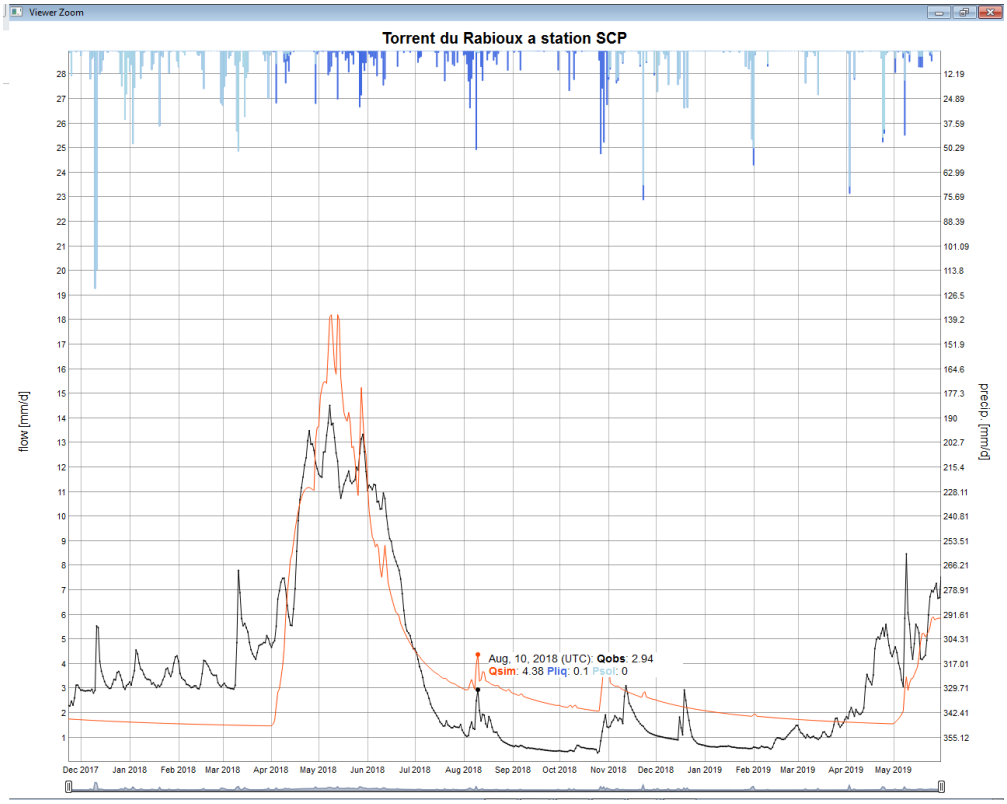
- ✓ Même intéractivité que le graphique de l'interface ShinyGr (Affichage des coordonnées des séries pour la date survolée par le curseur de souris, et période réglable)
- ✓ Option de légende permanente (mais occulte en partie les précipitations)
- ✓ Graduation plus précise de l'axe des temps (à condition de faire une simple copie d'écran comme ci-dessous)
- ✓ Possibilité de tracer une série de débit supplémentaire (par exemple, tout autre simulation au choix)
- ✓ Ajout possible d'un titre

Inconvénients

Plusieurs fonctionnalités de la fonction plot() standard ne sont pas disponibles avec cette variante dyplot():

- ✓ Impossible d'ajouter un 2ème axe des débits en m3/s (uniquement en mm)
- ✓ Représente uniquement les séries chronologiques Précipitations et débit (Aucun des graphiques supplémentaires de la fonction plot() standard : Snowpack, Température, Régime, CumFreqs, Corrélation QQ etc…)

  Cependant la fonction plot() standard de « airGR » appliquée aux objets de « airGRteaching », permet de générer tous ces graphiques habituels, ainsi que des graphiques supplémentaires ! (Cf. § 4.3.2.4.2 ci-dessus)



Exemple pour CemaNeige (Températures mensuelles uniquement, ce qui explique la performance médiocre du modèle)

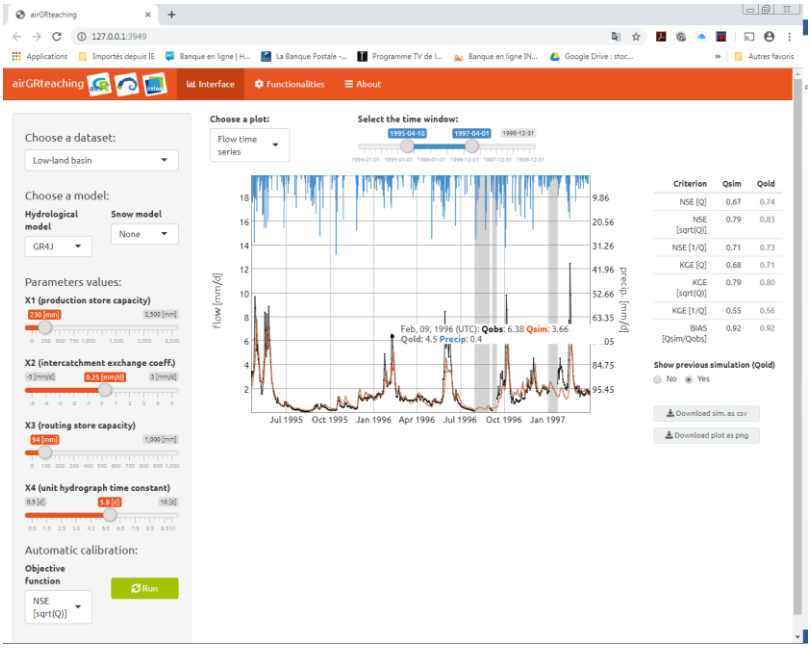
4.3.3.5 Interface interactive : ShinyGR()

Cette interface web interactive (Cf. copie d'écran ci-dessous) est accessible grâce à la fonction « ShinyGR » de la librairie « airGRTeaching » (UNIQUEMENT pour GRJ et Cemaneige). Elle permet les opérations suivantes (voir plus loin ce qu'elle ne permet pas) :

- ✓ Sélectionner l'un des modèles « GR*J », et activer/désactiver « Cemaneige »
- ✓ Choisir le critère d'optimisation du calage, ou forcer interactivement les paramètres du modèle (et recommencer à volonté)
- ✓ visualiser les résultats numériques/Graphiques regroupés dans une page web interactive.

  La copie d'écran ci-dessous illustre le graphique « Flow Time Series » (le graphique affiche les coordonnées du point survolé par le pointeur de souris). Les 3 autres graphiques sélectionnables au moyen du menu déroulant « Choose a plot » sont illustrés plus bas :
  - Model Performance,
  - State variables (Visualiser l'évolution temporelle des variables internes)
  - Model Diagram.
- ✓ Comparer les valeurs de tous les critères de qualité ainsi que les débits simulés, entre 2 calages successifs du modèle
- ✓ Modifier la période de représentation de l'hydrogramme, ce qui change également la période sur laquelle le calage peut être relancé
- ✓ De plus le programmeur a la possibilité de définir plusieurs jeux de paramètres de modélisation de la neige, afin que l'utilisateur puisse les sélectionner successivement dans l'interface, afin de comparer les différents hydrogrammes simulés, ainsi que les valeurs de critères correspondantes.

  Un jeu d'hypothèses est constitué des arguments de la fonction ShinyGR : DataFrame des séries hydroclimatiques, altitude de référence de ces données, et courbe hypsométrique du bassin



Syntaxe

```
ShinyGR(ObsDF = BasinObs,
        DatesR = NULL, Precip = NULL, PotEvap = NULL, Qobs = NULL, TempMean = NULL,
```

```
ZInputs = NULL, HypsoData = NULL, NLayers = 5, SimPer = ,
NamesObsBV = NULL, theme = "RStudio")
```

- ObsDF: data.frame (or a list of data.frame) avec les vecteurs impérativement dans l'ordre suivant (date, precipitation, evapotranspiration, discharge, temperature only if CemaNeige is used) see below for the units
  If one variable is provided in « ObsDF » and also separately, then only the data included in « ObsDF » are used.
  Les arguments ci-après peuvent être utilisés à la place pour fournir les vecteurs individuellement (« DatesR », « Precip », « PotEvap », « Qobs » and « TempMean »), si jamais les vecteurs dans « ObsDF » ne sont pas dans l'ordre requis
- DatesR (optional) [POSIXt] vector of dates required to create the GR and CemaNeige models inputs. Time zone must be defined as "UTC"
- Precip (optional) [numeric] time series of total precipitation (catchment average) [mm/time step], required to create the GR and CemaNeige models inputs
- PotEvap (optional) [numeric] time series of potential evapotranspiration (catchment average) [mm/time step], required to create the GR model inputs
- TempMean (optional) [numeric] time series of mean air temperature [°C], required to create the CemaNeige model inputs (if used)
- Qobs (optional) [numeric] time series of observed discharge [mm/time step]
- ZInputs (optional) [numeric or list of numerics] real giving the mean elevation of the Precip and TempMean series (before extrapolation) [m], used to create the CemaNeige model inputs (if used)
- HypsoData (optional) [numeric or list of numerics] vector of 101 reals: min, q01 to q99 and max of catchment elevation distribution [m]; if not defined a single elevation is used for CemaNeige (if used)
- NLayers (optional) [numeric or list of numerics] integer giving the number of elevation layers requested [-], required to create CemaNeige model inputs (if used)
- SimPer [character or list of characters] vector of 2 values to define the beginning and the end of the simulation period ["YYYY-mm-dd" or "YYYY-mm-dd HH:MM:SS"], see below for details
- NamesObsBV (optional) [character] vector of values to define the data inputs name(s) (if the ObsDF list is not already named)
- theme (optional) [character] alternative stylesheet ["RStudio" (default), "Cerulean", "Cyborg", "Flatly", "United" or "Yeti"]

Détails

- ✓ La période de mise en route est déduite automatiquemnet de la période de Simulation (SimPer) : Elle débute à la première date du jeu de données et finit avant la période de simulation.
- ✓ Les séries chronologiques sont fournies au choix dans le DataFrame `ObsDF`, ou dans des arguments séparés `DatesR Precip PotEvap Qobs TempMean` (cette dernière solution présentant l'avantage de s'affranchir de l'ordre imparatif des vecteurs dans le DataFrame)

Exemple de code avec « shinyGr () »

Tiré du PDF « airGRteaching.pdf » page 13

```
library(airGRteaching)

## data.frame of observed data of a low-land basin
data(L0123001, package = "airGR")
BV_L0123001 <- BasinObs[0001:6000, c("DatesR", "P", "E", "Qmm", "T")]
BI_L0123001 <- BasinInfo

## data.frame of observed data of a mountainous basin
data(L0123002, package = "airGR")
BV_L0123002 <- BasinObs[5000:9999, c("DatesR", "P", "E", "Qmm", "T")]
BI_L0123002 <- BasinInfo
```

```
## Interactive simulation step using default parameters
if (interactive()) {
   ShinyGR(ObsDF = list("Low-land basin" = BV_L0123001, "Mountainous basin" = BV_L0123002),
       ZInputs = list(NULL, median(BI_L0123002$HypsoData)),
       HypsoData = list(NULL, BI_L0123002$HypsoData),
       NLayers = list(5, 5),
       SimPer = list(c("1994-01-01", "1998-12-31"), c("2004-01-01", "2006-12-31")),
       theme = "United")
}
```

Avantages

Outre son intérêt pédagogique, cette l'interface ShinyGr apporte un réel confort à l'utilisateur :

- ✓ Affichage des coordonnées des séries pour la date survolée par le curseur de souris,
- ✓ Période réglable
- ✓ Ajout possible d'un titre

Inconvénients

Plusieurs fonctionnalités de « airGR » ne sont pas disponibles dans l'interface « ShinyGr » :

- ✓ la période de mise en route est déduite de la période de simulation (Impossible de la choisir comme avec `CreateRunOptions()`)
- ✓ Impossible de forcer l'état initial du modèle avant l'optimisation/simulation (ce que permet `CreateIniStates()`)
- ✓ Critère KGE2 non disponible
- ✓ Représente uniquement les séries chronologiques Précipitations et débit (Aucun des graphiques supplémentaires de la fonction plot() standard : Snowpack, Température, Régime, CumFreqs, Corrélation QQ etc…)
- ✓ Impossible d'ajouter un 2ème axe des débits en m3/s (uniquement en mm)

5 Principes spécifiques aux différents modèles GR*

5.1 GR1A

Il existe une variante de la fonction `RunModel_GR*()` pour chaque modèle GR.

```
RunModel_GR1A(InputsModel, RunOptions, Param)
```

- InputsModel [object of class InputsModel] see CreateInputsModel()§ 4.2.3 ci-dessus
- RunOptions [object of class RunOptions] see CreateRunOptions() § 4.2.3 ci-dessus
- Param [numeric] vector of 1 parameters
  - X1 model parameter [-]

Retourne une **liste des séries chronologiques** suivantes :

- ✓ $DatesR[POSIXlt] series of dates
- ✓ $PotEvap [numeric] series of input potential evapotranspiration [mm/y]
- ✓ $Precip[numeric] series of input total precipitation [mm/y]
- ✓ $Qsim [numeric] series of simulated discharge [mm/y]
- ✓ $StateEnd [numeric] states at the end of the run (NULL) [-]

Exemple GR1A: Simulation simple (sans calage)

```
library(airGR)

## loading catchment data
data(L0123001)

## conversion of example data from daily to yearly time step
TabSeries             <-  data.frame(BasinObs$DatesR,  BasinObs$P,  BasinObs$E,  BasinObs$T,
BasinObs$Qmm)
TimeFormat      <- "daily"
NewTimeFormat   <- "yearly"
ConvertFun      <- c("sum", "sum", "mean", "sum")
YearFirstMonth  <- 09;
NewTabSeries    <- SeriesAggreg(TabSeries = TabSeries, TimeFormat = TimeFormat,
                                NewTimeFormat = NewTimeFormat, ConvertFun = ConvertFun,
                                YearFirstMonth = YearFirstMonth)
BasinObs        <- NewTabSeries
names(BasinObs) <- c("DatesR", "P", "E", "T", "Qmm")

## preparation of the InputsModel object
InputsModel <- CreateInputsModel(FUN_MOD = RunModel_GR1A, DatesR = BasinObs$DatesR,
                                 Precip = BasinObs$P, PotEvap = BasinObs$E)
## run period selection
Ind_Run <- seq(which(format(BasinObs$DatesR, format = "%Y")=="1990"),
               which(format(BasinObs$DatesR, format = "%Y")=="1999"))
## preparation of the RunOptions object
RunOptions <- CreateRunOptions(FUN_MOD = RunModel_GR1A,
                               InputsModel = InputsModel, IndPeriod_Run = Ind_Run)
## simulation
Param <- c(X1 = 0.840)
OutputsModel <- RunModel_GR1A(InputsModel = InputsModel, RunOptions = RunOptions, Param =
Param)
```

```
## results preview
plot(OutputsModel, Qobs = BasinObs$Qmm[Ind_Run])

## efficiency criterion: Nash-Sutcliffe Efficiency
InputsCrit  <- CreateInputsCrit(FUN_CRIT = ErrorCrit_NSE, InputsModel = InputsModel,
                                RunOptions = RunOptions, Obs = BasinObs$Qmm[Ind_Run])
OutputsCrit <- ErrorCrit_NSE(InputsCrit = InputsCrit, OutputsModel = OutputsModel)
```

5.2 GR2M

Une variante spécifique de la fonction `RunModel_GR*()` est requise pour chacun des modèles GR (les arguments décrits ci-dessous pour GR2M sont cependant identiques pour tous les modèles).

RunModel_GR2M(InputsModel, RunOptions, Param)

- InputsModel [object of class InputsModel] see CreateInputsModel()§ 4.2.3 ci-dessus
- RunOptions [object of class RunOptions] see CreateRunOptions() § 4.2.3 ci-dessus
- Param [numeric] vector of 1 parameters
  - X1 production store capacity [mm]
  - X2 groundwater exchange coefficient [-]

Retourne une **liste des séries chronologiques** suivantes :

- ✓ $DatesR[POSIXlt] series of dates
- ✓ $PotEvap [numeric] series of input potential evapotranspiration [mm/month]
- ✓ $Precip[numeric] series of input total precipitation [mm/month]
- ✓ $AE [numeric] series of actual evapotranspiration [mm/month]
- ✓ $Pn [numeric] series of net rainfall (P1) [mm/month]
- ✓ $Perc [numeric] series of percolation (P2) [mm/month]
- ✓ $PR [numeric] series of PR=Pn+Perc (P3) [mm/month]
- ✓ $Exch [numeric] series of potential exchange between catchments [mm/month]
- ✓ $Prod [numeric] series of production store level [mm]
- ✓ $Rout [numeric] series of routing store level [mm]
- ✓ $Qsim [numeric] series of simulated discharge [mm/month]
- ✓ $StateEnd [numeric] states at the end of the run (production store level and routing store level) [mm], see CreateIniStates() § 4.2.3 ci-dessus

Exemple avec GR2M: Simulation simple (sans calage)

```
library(airGR)

## loading catchment data
data(L0123001)

## conversion of example data from daily to monthly time step
TabSeries             <-  data.frame(BasinObs$DatesR,  BasinObs$P,  BasinObs$E,  BasinObs$T,
BasinObs$Qmm)
TimeFormat      <- "daily"
NewTimeFormat   <- "monthly"
ConvertFun      <- c("sum", "sum", "mean", "sum")
NewTabSeries    <- SeriesAggreg(TabSeries = TabSeries, TimeFormat = TimeFormat,
                                NewTimeFormat = NewTimeFormat, ConvertFun = ConvertFun)
BasinObs        <- NewTabSeries
names(BasinObs) <- c("DatesR", "P", "E", "T", "Qmm")

## preparation of the InputsModel object
InputsModel <- CreateInputsModel(FUN_MOD = RunModel_GR2M, DatesR = BasinObs$DatesR,
                                 Precip = BasinObs$P, PotEvap = BasinObs$E)
```

```
## run period selection
Ind_Run <- seq(which(format(BasinObs$DatesR, format = "%Y-%m")=="1990-01"),
               which(format(BasinObs$DatesR, format = "%Y-%m")=="1999-12"))

## preparation of the RunOptions object
RunOptions <- CreateRunOptions(FUN_MOD = RunModel_GR2M,
                               InputsModel = InputsModel, IndPeriod_Run = Ind_Run)
## simulation
Param <- c(X1 = 265.072, X2 = 1.040)
OutputsModel <- RunModel_GR2M(InputsModel = InputsModel, RunOptions = RunOptions, Param =
Param)

## results preview
plot(OutputsModel, Qobs = BasinObs$Qmm[Ind_Run])

## efficiency criterion: Nash-Sutcliffe Efficiency
InputsCrit  <- CreateInputsCrit(FUN_CRIT = ErrorCrit_NSE, InputsModel = InputsModel,
                                RunOptions = RunOptions, Obs = BasinObs$Qmm[Ind_Run])
OutputsCrit <- ErrorCrit_NSE(InputsCrit = InputsCrit, OutputsModel = OutputsModel)
```

5.3 GR4J, GR5J, GR6J

Chaque modèle GR requiert une variante spécifique de la fonction `RunModel_GR*()`.

```
RunModel_GR4J(InputsModel, RunOptions, Param)
RunModel_GR5J(InputsModel, RunOptions, Param)
RunModel_GR6J(InputsModel, RunOptions, Param)
```

- InputsModel [object of class InputsModel] see CreateInputsModel()§ 4.2.3 ci-dessus
- RunOptions [object of class RunOptions] see CreateRunOptions() § 4.2.3 ci-dessus
- Param [numeric] vector of 4 parameters
  - ✓ GR4J X1 production store capacity [mm]
  - ✓ GR4J X2 intercatchment exchange coefficient [mm/d]
  - ✓ GR4J X3 routing store capacity [mm]
  - ✓ GR4J X4unit hydrograph time constant [d]

Retourne une **liste des séries chronologiques** suivantes :

- ✓ $DatesR[POSIXlt] series of dates
- ✓ $PotEvap [numeric] series of input potential evapotranspiration [mm/d]
- ✓ $Precip[numeric] series of input total precipitation [mm/d]
- ✓ $Prod [numeric] series of production store level [mm]
- ✓ $Pn [numeric] series of net rainfall [mm/d]
- ✓ $Ps [numeric] series of the part of Pn filling the production store [mm/d]
- ✓ $AE [numeric] series of actual evapotranspiration [mm/d]
- ✓ $Perc [numeric] series of percolation (PERC) [mm/d]
- ✓ $PR [numeric] series of PR=Pn-Ps+Perc [mm/d]
- ✓ $Q9 [numeric] series of UH1 outflow (Q9) [mm/d]
- ✓ $Q1 [numeric] series of UH2 outflow (Q1) [mm/d]
- ✓ $Rout [numeric] series of routing store level [mm]
- ✓ $Exch [numeric] series of potential semi-exchange between catchments [mm/d]
- ✓ $AExch1[numeric] series of actual exchange between catchments for branch1
  - [mm/d]
- ✓ $AExch2[numeric] series of actual exchange between catchments for branch2
  - [mm/d]
- ✓ $AExch [numeric] series of actual exchange between catchments (1+2) [mm/d]
- ✓ $QR [numeric] series of routing store outflow (QR) [mm/d]
- ✓ $QD [numeric] series of direct flow from UH2 after exchange (QD) [mm/d]
- ✓ $Qsim [numeric] series of simulated discharge [mm/d]
- ✓ $StateEnd [numeric] states at the end of the run (res. levels, UH1 levels, UH2 levels) [mm],
- ✓ see CreateIni3States5() for more details §

Exemple pour GR4J : Simulation simple (sans calage)

```
library(airGR)
## loading catchment data
data(L0123001)
## preparation of the InputsModel object
InputsModel <- CreateInputsModel(FUN_MOD = RunModel_GR4J, DatesR = BasinObs$DatesR,
               Precip = BasinObs$P, PotEvap = BasinObs$E)
## run period selection
Ind_Run <- seq(which(format(BasinObs$DatesR, format = "%Y-%m-%d")=="1990-01-01"),
               which(format(BasinObs$DatesR, format = "%Y-%m-%d")=="1999-12-31"))
## preparation of the RunOptions object
RunOptions <- CreateRunOptions(FUN_MOD = RunModel_GR4J,
                               InputsModel = InputsModel, IndPeriod_Run = Ind_Run)
## simulation
Param <- c(X1 = 257.238, X2 = 1.012, X3 = 88.235, X4 = 2.208)
OutputsModel <- RunModel_GR4J(InputsModel = InputsModel,
                              RunOptions = RunOptions, Param = Param)
## results preview
plot(OutputsModel, Qobs = BasinObs$Qmm[Ind_Run])

## efficiency criterion: Nash-Sutcliffe Efficiency
InputsCrit  <- CreateInputsCrit(FUN_CRIT = ErrorCrit_NSE, InputsModel = InputsModel,
                                RunOptions = RunOptions, Obs = BasinObs$Qmm[Ind_Run])
OutputsCrit <- ErrorCrit_NSE(InputsCrit = InputsCrit, OutputsModel = OutputsModel)
```

Exemple pour les autres modèles GR*J

Remplacez « `RunModel_GR4J()` » par « `RunModel_GR5J()` » ou « `RunModel_GR6J()` »

Et remplacez l'instruction d'affectation des paramètres par :

```
GR5J : Param <- c(X1 = 245.918, X2 = 1.027, X3 = 90.017, X4 = 2.198, X5 = 0.434)

GR6J : Param <- c(X1 = 242.257, X2 = 0.637, X3 = 53.517, X4 = 2.218, X5 = 0.424, X6 = 4.759)
```

5.4 GR4H

Chaque modèle GR requiert une variatne spécifique de la fonction `RunModel_GR*()`.

`RunModel_GR4H``(InputsModel, RunOptions, Param)`

- InputsModel [object of class InputsModel] see `CreateInputsModel()` § 4.2.3 ci-dessus
- RunOptions [object of class RunOptions] see `CreateRunOptions()` § 4.2.3 ci-dessus
- Param [numeric] vector of 4 parameters
  - ✓ GR4J X1 production store capacity [mm]
  - ✓ GR4J X2 groundwater exchange coefficient [mm/h]
  - ✓ GR4J X3 routing store capacity [mm]
  - ✓ GR4J X4 unit hydrograph time constant [h]

Retourne une **liste des séries chronologiques** suivantes :

- ✓ $DatesR[POSIXlt] series of dates
- ✓ $PotEvap [numeric series of input potential evapotranspiration [mm/h]
- ✓ $Precip[numeric] series of input total precipitation [mm/h]
- ✓ $Prod [numeric] series of production store level [mm]
- ✓ $AE [numeric] series of actual evapotranspiration [mm/h]
- ✓ $Perc [numeric] series of percolation (PERC) [mm/h]
- ✓ $PR [numeric] series of PR=Pn-Ps+Perc [mm/h]
- ✓ $Q9 [numeric] series of UH1 outflow (Q9) [mm/h]
- ✓ $Q1 [numeric] series of UH2 outflow (Q1) [mm/h]
- ✓ $Rout [numeric] series of routing store level [mm]
- ✓ $Exch [numeric] series of potential semi-exchange between catchments [mm/h]
- ✓ $AExch [numeric] series of actual exchange between catchments (1+2) [mm/h]
- ✓ $QR [numeric] series of routing store outflow (QR) [mm/h]
- ✓ $QD [numeric] series of direct flow from UH2 after exchange (QD) [mm/h]
- ✓ $Qsim [numeric] series of simulated discharge [mm/h]
- ✓ $StateEnd [numeric] states at the end of the run (res. levels, UH1 levels, UH2 levels) [mm], see `CreateIniStates()` § 4.2.3 ci-dessus

Exemple GR4H: Simulation simple (sans calage)

```
library(airGR)

## load of catchment data
data(L0123003)

## preparation of the InputsModel object
InputsModel <- CreateInputsModel(FUN_MOD = RunModel_GR4H, DatesR = BasinObs$DatesR,
                                 Precip = BasinObs$P, PotEvap = BasinObs$E)
## run period selection
Ind_Run <- seq(which(format(BasinObs$DatesR, format = "%Y-%m-%d %H:%M")=="2004-03-01 00:00"),
               which(format(BasinObs$DatesR, format = "%Y-%m-%d %H:%M")=="2008-12-31 23:00"))

## preparation of the RunOptions object
RunOptions <- CreateRunOptions(FUN_MOD = RunModel_GR4H,
                               InputsModel = InputsModel, IndPeriod_Run = Ind_Run)
## simulation
Param <- c(X1 = 521.113, X2 = -2.918, X3 = 218.009, X4 = 4.124)
OutputsModel <- RunModel_GR4H(InputsModel = InputsModel, RunOptions = RunOptions,
                              Param = Param)
## results preview
plot(OutputsModel, Qobs = BasinObs$Qmm[Ind_Run])

## efficiency criterion: Nash-Sutcliffe Efficiency
InputsCrit  <- CreateInputsCrit(FUN_CRIT = ErrorCrit_NSE, InputsModel = InputsModel,
                                RunOptions = RunOptions, Obs = BasinObs$Qmm[Ind_Run])
OutputsCrit <- ErrorCrit_NSE(InputsCrit = InputsCrit, OutputsModel = OutputsModel)
## run period selection
```

5.5 Cemaneige

Le choix de la variante de CemaNeige est expliqué à `CreateRunOptions()` § 4.2.3 ci-dessus.

5.5.1 Variantes de CemaNeige : Valery & Riboust

L'argument « If IsHyst » de la fonction `CreateRunOptions()` (§ 4.2.3.3 ci-dessus) permet de choisir entre 2 variantes de Cemaneige :

- ✓ If IsHyst = FALSE : Variante originale de CemaNeige de Valéry et al. (2014).
- ✓ If IsHyst = TRUE : Variante de Riboust et al. (2019).

La version « Riboust » présente les différences suivantes par rapport à la version originale « Valéry »:

- ✓ Nécessite 2 paramètres supplémentaires.
- ✓ Inclut une représentation de la relation d'hystérésis entre la zone de couverture de neige (SCA) et l'équivalent en eau de neige (SWE) dans le bassin versant.
  L'hystérésis incluse dans airGR est l' « Hystérésis linéaire modifiée (LH *) »; Elle est représenté sur le panneau b) de la figure 3 dans Riboust et al. (2019). Riboust et al. (2019) conseillent d'utiliser la version LH * de CemaNeige avec des paramètres calibrés à l'aide d'une fonction objectif combinant 75% de KGE calculée sur le débit simulé à partir d'un modèle pluie-ruissellement par rapport au débit observé et 5% de KGE calculée sur SCA sur 5 degrés CemaNeige SCA par satellite (par exemple MODIS) (voir l'équation (18), le tableau 3 et la figure 6). Riboust et al. (2019) ont été réalisés avec GR4J comme modèle de pluie et de ruissellement choisi.

5.5.2 « CemaNeige() »

Cette fonction lance une exécution du modèle journalier « Cemaneige » (sans modèle GRJ.

```
RunModel_CemaNeige(InputsModel, RunOptions, Param)
   ○ InputsModel   [object of class InputsModel] see CreateInputsModel()§ 4.2.3 ci-dessus
   ○ RunOptions    [object of class RunOptions] see CreateRunOptions() § 4.2.3 ci-dessus
   ○ Param         [numeric] vector of 4 parameters
        ✓ X1    weighting coefficient for snow pack thermal state        [-]
        ✓ X2    degree-day melt coefficient                              [mm/°C/d]
        ✓ X3    (optional) accumulation threshold (needed if IsHyst = TRUE)[mm]
        ✓ X4    (optional) percentage (between 0 and 1) of annual snowfall defining the
          melt threshold (needed if IsHyst = TRUE)                       [-]
```

Retourne un objet **« OutputModel »** composé des **listes et séries chronologiques** suivantes :

- ✓ `$DatesR[POSIXlt]` series of dates
- ✓ `$CemaNeigeLayers [list]` list of CemaNeige outputs (1 list per layer)

- ✓ $CemaNeigeLayers[[iLayer]]$Pliq [numeric] series of liquid precip. [mm/d]
- ✓ $CemaNeigeLayers[[iLayer]]$Psol [numeric] series of solid precip.[mm/d]
- ✓ $CemaNeigeLayers[[iLayer]]$SnowPack [numeric] series of snow pack [mm]
- ✓ $CemaNeigeLayers[[iLayer]]$ThermalState [numeric] series of snow pack thermal state
  - [°C]
- ✓ $CemaNeigeLayers[[iLayer]]$Gratio [numeric] series of Gratio [0-1]
- ✓ $CemaNeigeLayers[[iLayer]]$PotMelt [numeric] series of potential snow melt
  - [mm/d]
- ✓ $CemaNeigeLayers[[iLayer]]$Melt [numeric] series of actual snow melt [mm/d]
- ✓ $CemaNeigeLayers[[iLayer]]$PliqAndMelt [numeric] series of liquid precip. + actual snow melt [mm/d]
- ✓ $CemaNeigeLayers[[iLayer]]$Temp [numeric] series of air temperature [°C]
- ✓ $CemaNeigeLayers[[iLayer]]$Gthreshold [numeric] series of melt threshold [mm]
- ✓ $CemaNeigeLayers[[iLayer]]$Glocalmax [numeric] series of local melt threshold for hysteresis [mm]
- ✓ $StateEnd [numeric] states at the end of the run: CemaNeige states [mm & °C], see CreateIniStates() § 4.2.3 ci-dessus

Exemple CemaNeige: Simulation simple (sans calage)

```
library(airGR)
## load of catchment data
data(L0123002)

## preparation of the InputsModel object
InputsModel <- CreateInputsModel(FUN_MOD = RunModel_CemaNeige, DatesR = BasinObs$DatesR,
                                 Precip = BasinObs$P,TempMean = BasinObs$T,
                                 ZInputs = BasinInfo$HypsoData[51],
                                 HypsoData=BasinInfo$HypsoData, NLayers = 5)
## run period selection
Ind_Run <- seq(which(format(BasinObs$DatesR, format = "%Y-%m-%d")=="1990-01-01"),
               which(format(BasinObs$DatesR, format = "%Y-%m-%d")=="1999-12-31"))

## ---- original version of CemaNeige

## preparation of the RunOptions object
RunOptions <- CreateRunOptions(FUN_MOD = RunModel_CemaNeige, InputsModel = InputsModel,
                               IndPeriod_Run = Ind_Run)
## simulation
Param <- c(CNX1 = 0.962, CNX2 = 2.249)
OutputsModel <- RunModel_CemaNeige(InputsModel = InputsModel,
                                   RunOptions = RunOptions, Param = Param)
## results preview
plot(OutputsModel)

## ---- version of CemaNeige with the Linear Hysteresis
## preparation of the RunOptions object
RunOptions <- CreateRunOptions(FUN_MOD = RunModel_CemaNeige, InputsModel = InputsModel,
                               IndPeriod_Run = Ind_Run, IsHyst = TRUE)
## simulation
Param <- c(CNX1 = 0.962, CNX2 = 2.249, CNX3 = 100, CNX4 = 0.4)
OutputsModel <- RunModel_CemaNeige(InputsModel = InputsModel,
                                   RunOptions = RunOptions, Param = Param)
## results preview
plot(OutputsModel)
```

5.5.3 « CemaNeige_GR*J »

Ces fonctions lancent une exécution de « Cemaneige » couplée avec l'un des modèles « GRJ » journaliers.

```
RunModel_CemaNeigeGR4J(InputsModel, RunOptions, Param)
RunModel_CemaNeigeGR5J(InputsModel, RunOptions, Param)
RunModel_CemaNeigeGR6J(InputsModel, RunOptions, Param)
```

- InputsModel [object of class InputsModel] see CreateInputsModel()§ 4.2.3 ci-dessus
- RunOptions [object of class RunOptions] see CreateRunOptions() § 4.2.3 ci-dessus
- Param [numeric] vector of 8 to 10 parameters
  - ✓ GR5J X1 production store capacity [mm]
  - ✓ GR5J X2 intercatchment exchange coefficient [mm/d]
  - ✓ GR5J X3 routing store capacity [mm]
  - ✓ GR5J X4 unit hydrograph time constant [d]
  - ✓ GR5J X5 intercatchment exchange threshold [-]
  - ✓ GR5J X6 coefficient for emptying exponential store [-]
  - ✓ Cemaneige X1 weighting coefficient for snow pack thermal state [-]
  - ✓ Cemaneige X2 degree-day melt coefficient [mm/°C/d]
  - ✓ Cemaneige X3 (optional) accumulation threshold [mm] (needed if IsHyst = TRUE)
  - ✓ Cemaneige X4 (optional) percentage (between 0 and 1) of annual snowfall defining the melt threshold [-] (needed if IsHyst = TRUE)

Retourne un objet **« OutputModel »** composé des **listes et séries chronologiques** suivantes :

- ✓ $DatesR[POSIXlt] series of dates
- ✓ $PotEvap [numeric] series of input potential evapotranspiration [mm/d]
- ✓ $Precip[numeric] series of input total precipitation [mm/d]
- ✓ $Prod [numeric] series of production store level [mm]
- ✓ $Pn [numeric] series of net rainfall [mm/d]
- ✓ $Ps [numeric] series of the part of Pn filling the production store [mm/d]
- ✓ $AE [numeric] series of actual evapotranspiration [mm/d]
- ✓ $Perc [numeric] series of percolation (PERC) [mm/d]
- ✓ $PR [numeric] series of PR=Pn-Ps+Perc [mm/d]
- ✓ $Q9 [numeric] series of UH1 outflow (Q9) [mm/d]
- ✓ $Q1 [numeric] series of UH2 outflow (Q1) [mm/d]
- ✓ $Rout [numeric] series of routing store level [mm]
- ✓ $Exch [numeric] series of potential semi-exchange between catchments [mm/d]
- ✓ $AExch1[numeric] series of actual exchange between catchments for branch 1 [mm/d]
- ✓ $AExch2[numeric] series of actual exchange between catchments for branch 2 [mm/d]
- ✓ $AExch [numeric] series of actual exchange between catchments (1+2) [mm/d]
- ✓ $QR [numeric] series of routing store outflow (QR) [mm/d]
- ✓ $QD [numeric] series of direct flow from UH2 after exchange (QD) [mm/d]
- ✓ $Qsim [numeric] series of simulated discharge [mm/d]
- ✓ $CemaNeigeLayers [list] list of CemaNeige outputs (1 list per layer)
  - $CemaNeigeLayers[[iLayer]]$Pliq [numeric] series of liquid precip. [mm/d]
  - $CemaNeigeLayers[[iLayer]]$Psol [numeric] series of solid precip. [mm/d]
  - $CemaNeigeLayers[[iLayer]]$SnowPack [numeric] series of snow pack [mm]
  - $CemaNeigeLayers[[iLayer]]$ThermalState [numeric] series of snow pack thermal state [°C]
  - $CemaNeigeLayers[[iLayer]]$Gratio [numeric] series of Gratio [0-1]
  - $CemaNeigeLayers[[iLayer]]$PotMelt [numeric] series of potential snow melt [mm/d]
  - $CemaNeigeLayers[[iLayer]]$Melt [numeric] series of actual snow melt [mm/d]
  - $CemaNeigeLayers[[iLayer]]$PliqAndMelt [numeric] series of liquid precip. + actual snow melt [mm/d]
  - $CemaNeigeLayers[[iLayer]]$Temp [numeric] series of air temperature [°C]
  - $CemaNeigeLayers[[iLayer]]$Gthreshold [numeric] series of melt threshold [mm]
  - $CemaNeigeLayers[[iLayer]]$Glocalmax [numeric] series of local melt threshold for hysteresis [mm]

- ✓ $StateEnd [numeric] states at the end of the run:
  store & unit hydrographs levels [mm], CemaNeige states [mm & °C],
  see CreateIniStates()§ 4.2.3 ci-dessus

Exemple CemaNeige_GR5J: Simulation simple (sans calage)

```
library(airGR)

## loading catchment data
data(L0123002)

## preparation of the InputsModel object
InputsModel <- CreateInputsModel(FUN_MOD = RunModel_CemaNeigeGR5J, DatesR = BasinObs$DatesR,
                                 Precip = BasinObs$P, PotEvap = BasinObs$E,
                                 TempMean = BasinObs$T,
                                 ZInputs = median(BasinInfo$HypsoData),
                                 HypsoData = BasinInfo$HypsoData, NLayers = 5)
## run period selection
Ind_Run <- seq(which(format(BasinObs$DatesR, format = "%Y-%m-%d")=="1990-01-01"),
               which(format(BasinObs$DatesR, format = "%Y-%m-%d")=="1999-12-31"))
## preparation of the RunOptions object
RunOptions <- CreateRunOptions(FUN_MOD = RunModel_CemaNeigeGR5J, InputsModel = InputsModel,
                               IndPeriod_Run = Ind_Run)
## simulation
Param <- c(X1 = 179.139, X2 = -0.100, X3 = 203.815, X4 = 1.174, X5 = 2.478,
           CNX1 = 0.977, CNX2 = 2.774)
OutputsModel <- RunModel_CemaNeigeGR5J(InputsModel = InputsModel,
                                       RunOptions = RunOptions, Param = Param)
## results preview
plot(OutputsModel, Qobs = BasinObs$Qmm[Ind_Run])

## efficiency criterion: Nash-Sutcliffe Efficiency
InputsCrit  <- CreateInputsCrit(FUN_CRIT = ErrorCrit_NSE, InputsModel = InputsModel,
                                RunOptions = RunOptions, Obs = BasinObs$Qmm[Ind_Run])
OutputsCrit <- ErrorCrit_NSE(InputsCrit = InputsCrit, OutputsModel = OutputsModel)

## simulation with the Linear Hysteresis
## preparation of the RunOptions object
RunOptions <- CreateRunOptions(FUN_MOD = RunModel_CemaNeigeGR5J, InputsModel = InputsModel,
                               IndPeriod_Run = Ind_Run, IsHyst = TRUE)
Param <- c(179.139, -0.100, 203.815, 1.174, 2.478, 0.977, 2.774, 100, 0.4)
OutputsModel <- RunModel_CemaNeigeGR5J(InputsModel = InputsModel,
                                       RunOptions = RunOptions, Param = Param)
## results preview
plot(OutputsModel, Qobs = BasinObs$Qmm[Ind_Run])

## efficiency criterion: Nash-Sutcliffe Efficiency
InputsCrit  <- CreateInputsCrit(FUN_CRIT = ErrorCrit_NSE, InputsModel = InputsModel,
                                RunOptions = RunOptions, Obs = BasinObs$Qmm[Ind_Run])
OutputsCrit <- ErrorCrit_NSE(InputsCrit = InputsCrit, OutputsModel = OutputsModel)
```

Exemple pour d'autres CemaNeige_GR*J

Remplacez simplement GR5J par GR4J ou GR6J.

5.5.4 « DataAltiExtrapolation_Valery() »

If « ZInputs » is different than « HypsoData », cette fonction permet d'extrapoler les séries de Précipitation et Températures à différentes couches d'altitudes par la méthode de Valéry (2010). Application à préciser (voir `CreateInputsModel()` § 4.2.3.1 ci-dessus).

```
DataAltiExtrapolation_Valery(DatesR, Precip, PrecipScale = TRUE,
                             TempMean, TempMin = NULL, TempMax = NULL,
                             ZInputs, HypsoData, NLayers, verbose = TRUE)
```

- DatesR [POSIXt] vector of dates
- Precip [numeric] time series of daily total precipitation (catchment average) [mm/d]
- PrecipScale (optional) [boolean] indicating if the mean of the precipitation interpolated on the elevation layers must be kept or not, required to create CemaNeige module inputs, default = TRUE (the mean of the precipitation is kept to the original value)
- TempMean [numeric] time series of daily mean air temperature [°C]
- TempMin (optional) [numeric] time series of daily min air temperature [°C]
- TempMax (optional) [numeric] time series of daily max air temperature [°C]
- ZInputs [numeric] real giving the mean elevation of the Precip and Temp series (before extrapolation) [m]
- HypsoData [numeric] vector of 101 reals: min, q01 to q99 and max of catchment elevation distribution [m]
- NLayers [numeric] integer giving the number of elevation layers requested, required for CemaNeige model inputs (if used) [-]
- verbose (optional) [boolean] boolean indicating if the function is run in verbose mode or not, default = TRUE

Retourne une **liste des séries chronologiques** (Précipitations et Température) extapolées pour les différentes tranches d'altitude, à transmettre en argument de `CreateInputsModel()` § 4.2.3.1 ci-dessus:

- ✓ $LayerPrecip [list] list of time series of daily precipitation (layer average) [mm/d]
- ✓ $LayerTempMean [list] list of time series of daily mean air temperature (layer average) [°C]
- ✓ $LayerTempMin [list] list of time series of daily min air temperature (layer average) [°C]
- ✓ $LayerTempMax [list] list of time series of daily max air temperature (layer average) [°C]
- ✓ $LayerFracSolidPrecip [list] list of time series of daily solid precip. fract. (layer average) [-]
- ✓ $ZLayers [numeric] vector of median elevation for each layer

Détails

Elevation layers of equal surface are created the 101 elevation quantiles (HypsoData) and the number requested elevation layers (NLayers).
Forcing data (precipitation and air temperature) are extrapolated using gradients from Valery (2010). (e.g. gradP = 0.0004 [m-1] for France and gradT = 0.434 [°C/100m] for January, 1st).

Sommaire complet