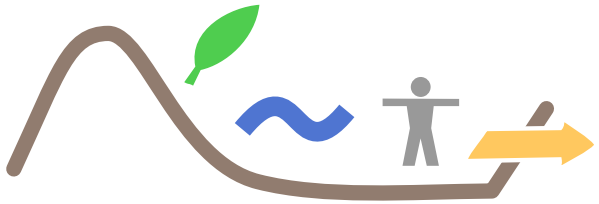


OpenFLUID

Software Environment
for Modelling Fluxes in Landscapes

Utilisation du package ROpenFLUID

Application au modèle MHYDAS

David Crevoisier, Jean-Christophe Fabre

LISAH
Laboratoire d'étude des Interactions
Sol-Agrosystème-Hydrosystème

Plan

Plan

Avant de commencer : le langage R

R est un logiciel libre de traitement des données et d'analyses statistiques :

- évolution du langage S (analyse, visualisation et manipulation de données - 1975),
- environnement statistique et langage de programmation autonome,
- programmation orientée objet, fonctionnelle ou procédurale,

Fonctionnement général :

- une base pour la statistique courante,
- des paquets (ou extensions) dédiés :
  - sur un théme (analyse de sensibilité, Kriegeage, approche bayésienne,...),
  - pour l'export (Latex, OpenDocument),
  - ou l'interfaçage (base de données, SIG).
- en ligne de commande ou interfaces graphiques conviviales : RStudio, RKWard,...

Introduction

ROpenFLUID est une extension du langage de programmation R permettant le pilotage de simulations OpenFLUID :

- le lancement,
- la définition et la modification de paramètres,
- la conversion des résultats en variables R.

Dans cette présentation, nous aborderons :

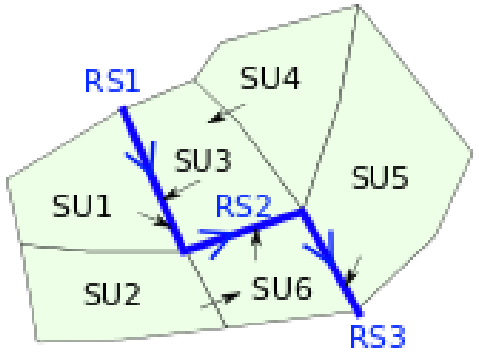
- les fonctionnalités de base du paquet ROpenFLUID
- une analyse statique de MHYDAS grâce au paquet R sensitivity
- une optimisation de paramètres grâce à la fonction R optim

Plan

La simulation à analyser

Le domaine :

- 6 parcelles (SU),
- 3 tronçons de fossés (RS)



Le modèle MHYDAS :

- production de pluie sur SU,
- partage infiltration et ruissellement (Morel-Seytoux),
- transfert du ruissellement sur SU (Hayami),
- transfert du ruissellement sur RS (Hayami).

Les sorties :

- infiltration / ruissellement (SU),
- débits et hauteurs d'eau (RS).

Plan

Lancement d'une simulation OpenFLUID

Lancer une simulation sous R :

- charger ROpenFLUID,
- définir le jeu de données d'entrée,
- définir le répertoire des fichiers résultats,
- définir d'un format des fichiers résultats adapté à R,
- exécuter la simulation

Lancement

```
library("ROpenFLUID")
OF.dirIn   = "/home/openfluid/.../IN"
OF.dirOut  = "/home/openfluid/.../OUT"
OF.simu    = OpenFLUID.openDataset(OF.dirIn)
OpenFLUID.setCurrentOutputDir(OF.dirOut)
OpenFLUID.addVariablesExportAsCSV(OF.simu,'RS')
OpenFLUID.runSimulation(OF.simu)
```

Exploitation des résultats

Données numériques

Chargement des résultats comme variables R

```
OF.Q.downstream = OpenFLUID.loadResult(OF.simu,"RS",3,"water.surf.Q.downstream-rs")
OF.H.level      = OpenFLUID.loadResult(OF.simu,"RS",3,"water.surf.H.level-rs")
```

Exploitation numérique

```
OF.deltaT = OpenFLUID.getDeltaT(OF.simu)  # get time step
cumOutlet = sum(OF.Q.downstream[,2])*OF.deltaT
maxLevel  = max(OF.H.level[,2])
```

Exploitation des résultats

Graphiques

Chargement des résultats comme variables R

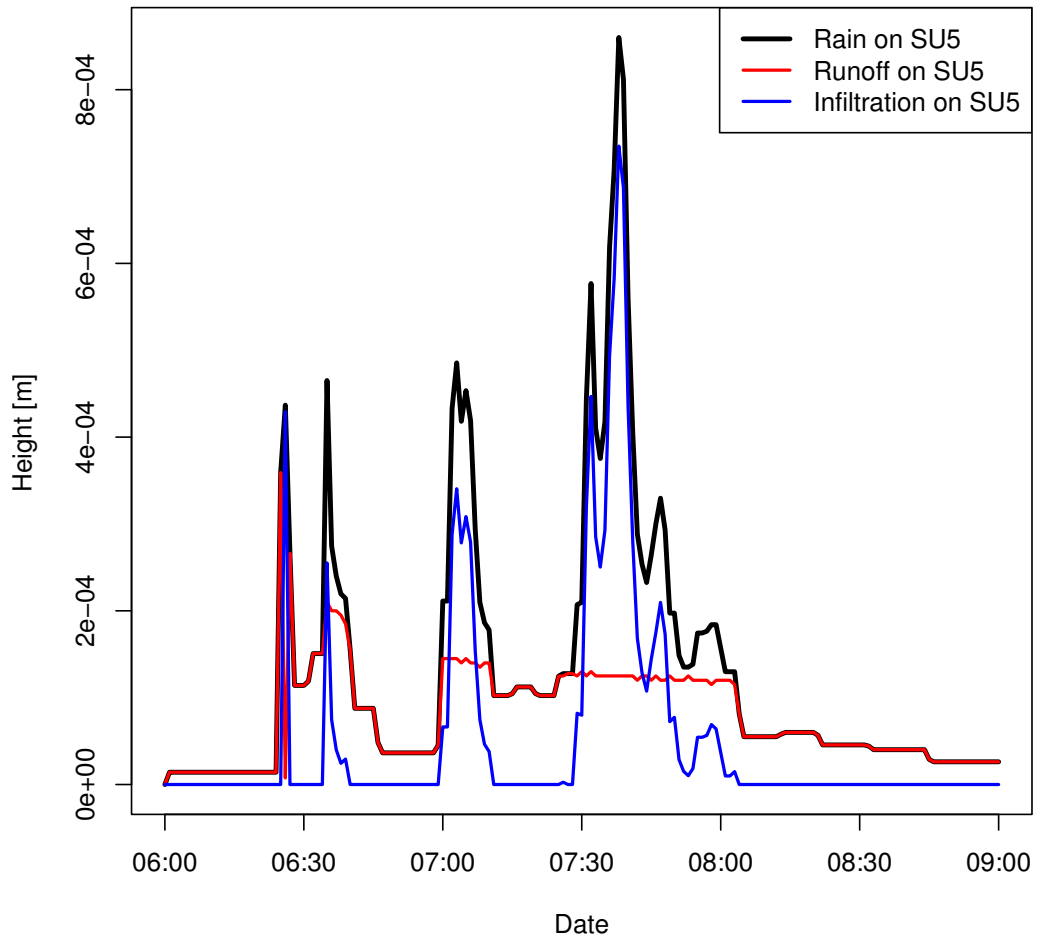
```
OF.H.rain          = OpenFLUID.loadResult(OF.simu,"SU",5,"water.atm-surf.H.rain")
OF.H.runoff        = OpenFLUID.loadResult(OF.simu,"SU",5,"water.surf.H.runoff")
OF.H.infiltration = OpenFLUID.loadResult(OF.simu,"SU",5,"water.surf.H.infiltration")
```

Visualisation des données dans une graphique

```
plot(OF.H.rain[,1],OF.H.rain[,2],type='l',xlab='Date',ylab='Flow [m3.s-1]')
lines(OF.H.infiltration[,1],OF.H.infiltration[,2],col='red')
lines(OF.H.runoff[,1],OF.H.runoff[,2],type='l',col='blue')
legend("topright",c('Rain on SU5','Runoff on SU5','Infiltration on SU5'))
```

Exploitation des résultats

Graphiques



Plan

Facteurs et sorties considérés

Influence de 3 facteurs :

- C_{RS} : la célérité moyenne sur les RS (paramètre du simulateur transferts sur les fossés)
- $\theta_{I\ SU5}$: l'humidité initiale de la parcelle 5 (attribut de la SU5)
- L_{RS3} : la largeur du fossé 3 (attribut de la RS3)

Sur 2 sorties :

- Q_{RS3} : le débit cumulé à l'exutoire
- H_{RS3} : la hauteur d'eau maximale à l'exutoire

Fonction R multi-simulation

Définir une fonction R qui prend en argument :

- une matrice X contenant N lignes (C_{RS},$\theta_{I\ SU5}$,L_{RS3})
- un type de variable de sortie ("H" ou "Q")

Et rend en sortie :

- un vecteur contenant les débits ou hauteurs simulés des N jeux de facteurs

Intérêt :

- une seule commande pour N simulations
- format nécessaire pour l'usage du paquet R "sensitivity"

Deux nouvelles commandes :

Modification des paramètres de simulateurs et des attributs

```
OpenFLUID.setSimulatorParam(OF.simu,simid,paramname,paramval)
OpenFLUID.setAttribute(OF.simu,unitclass,unitid,attrname,attrval)
```

Fonction R multi-simulation

Multi-simulation

```
OF.multiRun <- function(X,varOut) {

 for (i in seq(1:nrow(X))) { # loop on the lines of the factors vector

  C = X[i,1]; H = X[i,2]; W = X[i,3] # redefine factors
  OpenFLUID.setSimulatorParam(OF.simu,"water.surf.transfer-rs.hayami","meancel",C)
  OpenFLUID.setAttribute(OF.simu,"SU",5,"thetaini",H)
  OpenFLUID.setAttribute(OF.simu,"RS",3,"width",W)

  OpenFLUID.addVariablesExportAsCSV(OF.simu,'RS')
  OpenFLUID.runSimulation(OF.simu) # run simulation with new data set

  if (varOut=="Q") {  # choice of output variables
   outOF  = OpenFLUID.loadResult(OF.simu,"RS",3,"water.surf.Q.downstream-rs")
   out[i] = sum(outOF[,2])*OpenFLUID.getDeltaT(OF.simu)
  }
  else if (varOut=="H") {
   outOF  = OpenFLUID.loadResult(OF.simu,"RS",3,"water.surf.H.level-rs")
   out[i] = max(outOF[,2])
  }
 }
 return(out)
}
```

Propagation d'incertitudes et intervalles de confiance

Préparation de l'étude

Définition de 3 types de distribution pour le jeu de facteurs étudié

	C_{RS}	$\theta_{I\ SU5}$	L_{RS3}
distribution 1	0.6 ± 67%	0.3 ± 16%	0.5 ± 30%
distribution 2	0.6 ± 16%	0.3 ± 16%	0.5 ± 30%
distribution 3	0.6 ± 67%	0.3 ± 16%	0.5 ± 10%

Lancement de la multi-simulation, puis analyse statistique

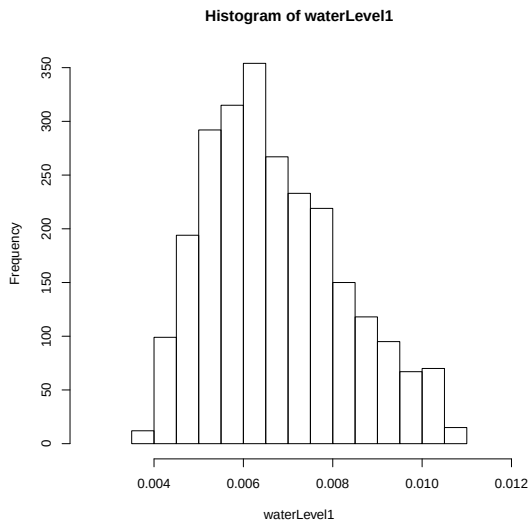
Analyse statistique des résultats

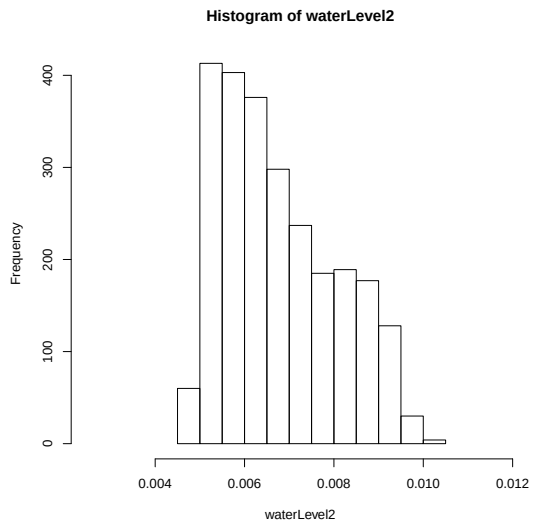
```
waterLevel = OF.multiRun(v,"H") # launch of the multi-simulation
quantile(waterLevel, probs = c(0.05, 0.5, 0.95)) # statistical uncertainty analysis
hist(waterLevel) # plot of results distribution
```

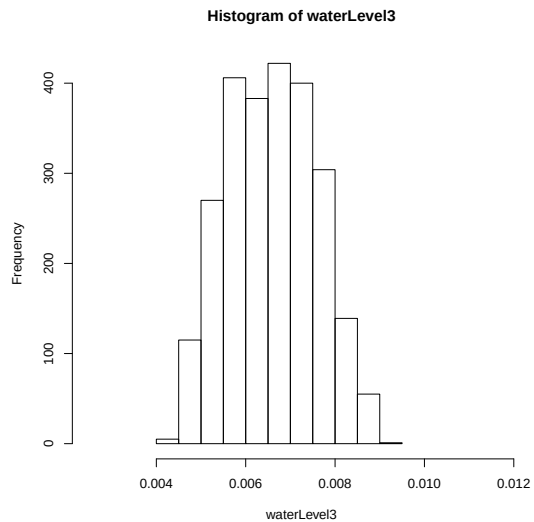
Propagation d'incertitudes et intervalles de confiance

Données numériques et graphiques

	médiane [m]	**intervalle de confiance 90%** [m]	**incertitude**
distribution 1	0.0107	de 0.00912 à 0.01296	± 17%
distribution 2	0.0110	de 0.00935 à 0.01311	± 16%
distribution 3	0.0108	de 0.01019 à 0.01145	± 6%







Analyse de sensibilité d'un modèle

Principes

Principes de l'analyse de Morris :

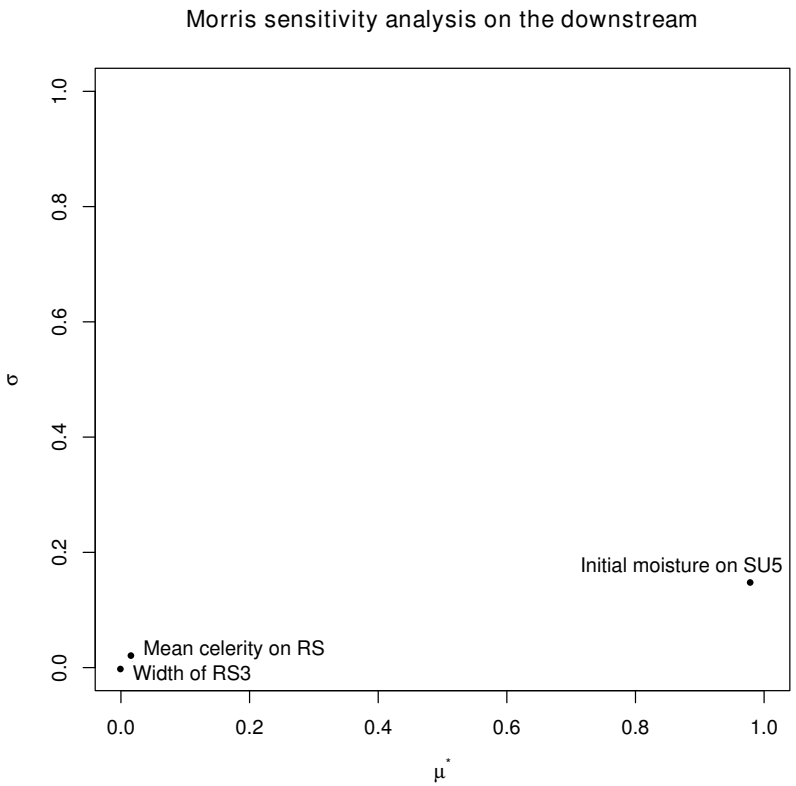
- *n* répétitions du modèle
- perturbation d'un facteur par répétition
- étude de la moyenne et la variance des effets élémentaires (perturbation de la sortie)

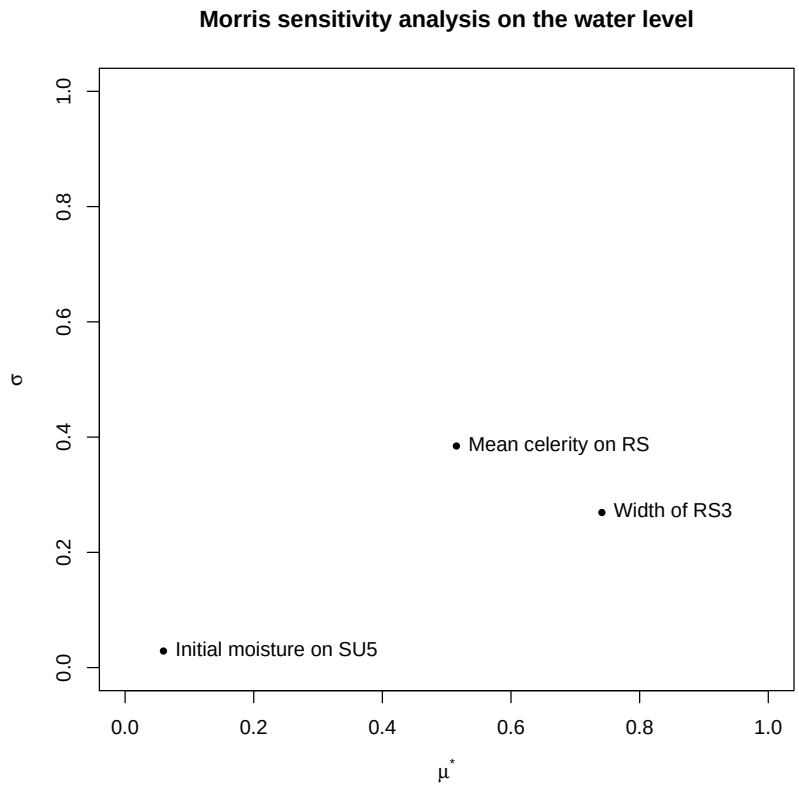
Analyse de Morris

```
library("sensitivity") # load of sensitivity package
OF.morris.Q <- morris(model=OF.multiRun,r=500,
                      design=list(type="oat", levels=25, grid.jump=12),
                      binf=c(0.1,0.05,0.1),bsup=c(1.0,0.35,1.0),varOut="Q")
plot(OF.morris.Q,xlim=c(0,1),ylim=c(0,1))
```

Analyse de sensibilité d'un modèle

Résultats





les points du graphique représentent la variance en fonction de la moyenne des effets élémentaires de *n* répétitions du modèle (la variation de la sortie lorsqu'on fait varier le paramètre considéré).

Analyse de sensibilité d'un modèle

Illustration

Perturbations des variables de sortie en fonction des perturbations des facteurs les plus sensibles :

variables de sortie	$\theta_{I\ SU5}$ - %25	L_{RS3} - %25
Q	-12%	0%
H	-6%	+34%

Plan

Optimisation de paramètres

Calcul de la fonction coût à optimiser

Définir une fonction R qui prend en argument :

- un vecteur X contenant (C_{RS},L_{RS3})
- la solution référence vers laquelle doit tendre la simulation

Et rend en sortie :

- la fonction coût (RMSE entre mesuré et simulé)

L'optimisation vise à faire tendre la fonction coût vers 0.
Nous utiliserons des fonctions R dédiées à l'optimisation.

Optimisation de paramètres

Calcul de la fonction coût

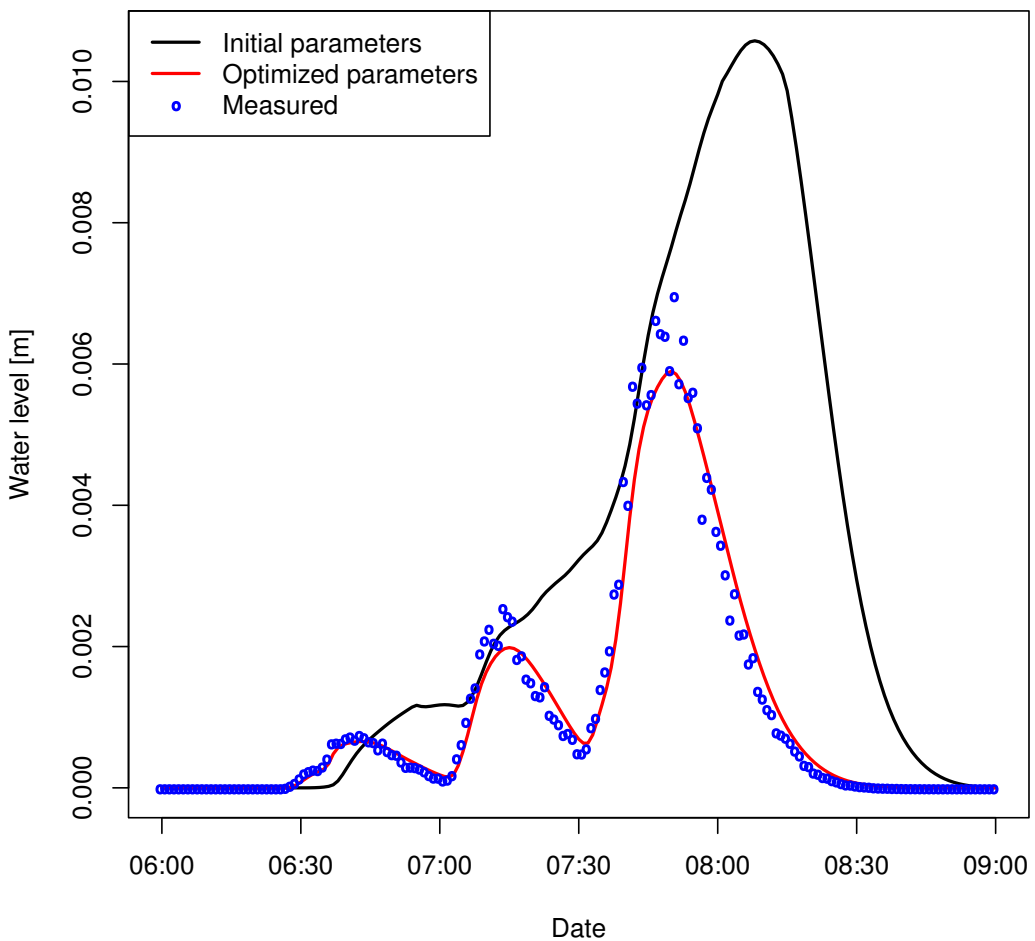
```
OF.singleRun <- function(X,ref) {

    # redefine simulator parameters and input data according to vector X
    OpenFLUID.setSimulatorParam(OF.simu,"water.surf.transfer-rs.hayami","meancel",X
        [1])
    OpenFLUID.setAttribute(OF.simu,"RS",3,"width",X[2])

    OpenFLUID.addVariablesExportAsCSV(OF.simu,'RS')
    OpenFLUID.runSimulation(OF.simu) # run simulation with new data set

    outOF = OpenFLUID.loadResult(OF.simu,"RS",3,"water.surf.H.level-rs")
    out   = sqrt(sum((outOF[,2]-ref[,2])**2)) # compute objective function
    return(out)
}
```

Optimisation de paramètres

```
OF.factorsOptim=optim(par=c(0.01,0.2),fn=OF.singleRun,method='L-BFGS-B',
                      lower=0.1*c(0.01,0.2),upper=5.0*c(0.01,0.2),ref=OF.refOptim)
```

Optimisation de paramètres

Résultats



Paramètres	C_{RS}	L_{RS3}
initiaux	0.01	0.2
optimisés	0.05	0.489

Liste des commandes ROpenFLUID

```
OpenFLUID.addExtraSimulatorsPaths(paths)
OpenFLUID.addVariablesExportAsCSV(ofblob,unitclass)
OpenFLUID.createAttribute(ofblob,unitclass,attrname,attrval)
OpenFLUID.getDeltaT(ofblob)
OpenFLUID.getExtraSimulatorsPaths()
OpenFLUID.getGeneratorParam(ofblob,unitclass,varname,paramname)
OpenFLUID.getAttribute(ofblob,unitclass,unitid,attrname)
OpenFLUID.getModelGlobalParam(ofblob,paramname)
OpenFLUID.getPeriodBeginDate(ofblob)
OpenFLUID.getPeriodEndDate(ofblob)
OpenFLUID.getSimulatorParam(ofblob,simid,paramname)
OpenFLUID.getUnitsClasses(ofblob)
OpenFLUID.getUnitsIDs(ofblob,unitclass)
OpenFLUID.getVersion()
OpenFLUID.loadResult(ofblob,unitclass,unitid,varname)
OpenFLUID.loadResultFile(filepath)
OpenFLUID.openDataset(path)
OpenFLUID.openProject(path)
OpenFLUID.printSimulationInfo(ofblob)
OpenFLUID.runProject(path)
OpenFLUID.runSimulation(ofblob)
OpenFLUID.setCurrentOutputDir(path)
OpenFLUID.setDeltaT(ofblob,deltat)
OpenFLUID.setGeneratorParam(ofblob,unitclass,varname,paramname,paramval)
OpenFLUID.setAttribute(ofblob,unitclass,unitid,attrname,attrval)
OpenFLUID.setModelGlobalParam(ofblob,paramname,paramval)
OpenFLUID.setPeriodBeginDate(ofblob,begindate)
OpenFLUID.setPeriodEndDate(ofblob,enddate)
OpenFLUID.setSimulatorParam(ofblob,simid,paramname,paramval)
```