statista Documentation

Mostafa Farrag

Feb 17, 2023

CONTENTS

CHAPTER

ONE

STATISTA - STATISTICS PACKAGE

statista is a statistics package.

1.1 Main Features

-
-

1.1.1 Installation

Stable release

Please install `statista` in a Virtual environment so that its requirements don't tamper with your system's python.

conda

the easiest way to install `statista` is using `conda` package manager. `statista` is available in the conda-forge channel. To install you can use the following command:

- `conda install -c conda-forge statista`

If this works it will install Hapi with all dependencies including Python and gdal, and you skip the rest of the installation instructions.

Installing Python and gdal dependencies

The main dependencies for statista are an installation of Python 3.9+, and gdal

Installing Python

For Python we recommend using the Anaconda Distribution for Python 3, which is available for download from https://www.anaconda.com/download/. The installer gives the option to add `python` to your `PATH` environment variable. We will assume in the instructions below that it is available in the path, such that `python`, `pip`, and `conda` are all available from the command line.

Note that there is no hard requirement specifically for Anaconda's Python, but often it makes installation of required dependencies easier using the conda package manager.

Install as a conda environment

The easiest and most robust way to install Hapi is by installing it in a separate conda environment. In the root repository directory there is an `environment.yml` file. This file lists all dependencies. Either use the `environment.yml` file from the master branch (please note that the master branch can change rapidly and break functionality without warning), or from one of the releases {release}.

Run this command to start installing all Hapi dependencies:

- `conda env create -f environment.yml`

This creates a new environment with the name `statista`. To activate this environment in a session, run:

- `conda activate statista`

For the installation of Hapi there are two options (from the Python Package Index (PyPI) or from Github). To install a release of Hapi from the PyPI (available from release 2018.1):

- `pip install statista==={release}`

From sources

The sources for HapiSM can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/MAfarrag/statista
```

Or download the tarball:

```
$ curl -OJL https://github.com/MAfarrag/statista/tarball/main
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

To install directly from GitHub (from the HEAD of the master branch):

- `pip install git+https://github.com/MAfarrag/statista.git`

or from Github from a specific release:

- `pip install git+https://github.com/MAfarrag/statista.git@{release}`

Now you should be able to start this environment's Python with `python`, try `import statista` to see if the package is installed.

More details on how to work with conda environments can be found here: https://conda.io/docs/user-guide/tasks/manage-environments.html

If you are planning to make changes and contribute to the development of Hapi, it is best to make a git clone of the repository, and do a editable install in the location of you clone. This will not move a copy to your Python installation directory, but instead create a link in your Python installation pointing to the folder you installed it from, such that any changes you make there are directly reflected in your install.

- `git clone https://github.com/MAfarrag/statista.git`
- `cd statista`
- `activate statista`
- `pip install -e .`

Alternatively, if you want to avoid using `git` and simply want to test the latest version from the `master` branch, you can replace the first line with downloading a zip archive from GitHub: https://github.com/MAfarrag/statista/archive/master.zip libraries.io.

Install using pip

Besides the recommended conda environment setup described above, you can also install Hapi with `pip`. For the more difficult to install Python dependencies, it is best to use the conda package manager:

- `conda install numpy scipy gdal netcdf4 pyproj`

you can check libraries.io. to check versions of the libraries

Then install a release {release} of statista (available from release 2018.1) with pip:

- `pip install statista==`{release}

Check if the installation is successful

To check it the install is successful, go to the examples directory and run the following command:

- `python -m statista.*******`

This should run without errors.

Note: This documentation was generated on Feb 17, 2023

Documentation for the development version: https://statista.readthedocs.org/en/latest/

Documentation for the stable version: https://statista.readthedocs.org/en/stable/

1.1.2 Distributions

Generalized extreme value distribution (GEV)

- **The generalised extreme value (or generalized extreme value) distribution characterises the behaviour of 'block**
  maxima'

probability density function (pdf)

$$f(x) = \frac{1}{\sigma} * Q(x)^{\xi+1} * e^{-Q(x)}$$

- **where**
  - **math**
    sigma is the scale parameter
  - **math**
    mu is the location parameter
  - **math**
    delta is the scale parameter

Cumulative distribution function (cdf)

$$F(x) = e^{-Q(x)}$$

Gumbel Distribution

- **The Gumbel distribution is a special case of the *Generalized extreme value distribution (GEV)* when the shape**
  parameter :math: *sigma* equals zero.

probability density function (pdf)

$$f(x) = \frac{1}{\sigma} * e^{-(\frac{x-\mu}{\delta})-e^{-(\frac{x-\mu}{\delta})}}$$

Cumulative distribution function (cdf)

$$F(x) = e^{-e^{-(\frac{x-\mu}{\delta})}}$$

1.1.3 Sensetivity Analysis (OAT)

OAT sensitivity analysis is a tool that is based

One of the simplest and most common approaches of sensitivity analysis is that of changing one-factor-at-a-time (OAT), to see what effect this produces on the output.

OAT customarily involves

- moving one parameter, keeping others at their baseline (nominal) values, then,
- returning the parameter to its nominal value, then repeating for each of the other parameters in the same way.

Sensitivity may then be measured by monitoring changes in the output. This appears a logical approach as any change observed in the output will unambiguously be due to the single parameter changed. Furthermore, by changing one parameter at a time, one can keep all other parameters fixed to their central or baseline values. This increases the comparability of the results (all 'effects' are computed with reference to the same central point in space)

If we want to check the sensitivity of the HBV hydrological model performance to predict stream flow to each parameter, the One-At-a Time sensitivity analysis is agreat meathod that helps in this area OAT fixes the value of all parameters and change the value of one parameter within boundaries each time to check the result of the given function based on different values of one of the inputs

First of all to run the HBV lumped model which we need to test its performance (based on RMSE error) based on a defined range for each parameter

Steps:

- Run the model with the baseline parameter *Run the model*

- Define wrapper function and type *Define wrapper function and type*
- Instantiate the SensitivityAnalysis object *Instantiate the SensitivityAnalysis object*
- Run the OAT method *Run the OAT method*
- Display the result with the SOBOL plot *Display the result with the SOBOL plot*

Run the model

```
import pandas as pd

import Hapi.rrm.hbv_bergestrom92 as HBVLumped
from Hapi.run import Run
from Hapi.catchment import Catchment
from Hapi.rrm.routing import Routing
import Hapi.statistics.performancecriteria as PC
from Hapi.statistics.sensitivityanalysis import SensitivityAnalysis as SA

Parameterpath = "/data/Lumped/Coello_Lumped2021-03-08_muskingum.txt"

Path = "/data/Lumped/"

### meteorological data
start = "2009-01-01"
end = "2011-12-31"
name = "Coello"
Coello = Catchment(name, start, end)
Coello.ReadLumpedInputs(Path + "meteo_data-MSWEP.csv")

### Basic_inputs
# catchment area
CatArea = 1530
# temporal resolution
# [Snow pack, Soil moisture, Upper zone, Lower Zone, Water content]
InitialCond = [0,10,10,10,0]

Coello.ReadLumpedModel(HBVLumped, CatArea, InitialCond)

### parameters
 # no snow subroutine
Snow = 0
# if routing using Maxbas True, if Muskingum False
Maxbas = False
Coello.ReadParameters(Parameterpath, Snow, Maxbas=Maxbas)

parameters = pd.read_csv(Parameterpath, index_col = 0, header = None)
parameters.rename(columns={1:'value'}, inplace=True)

UB = pd.read_csv(Path + "/UB-1-Muskinguk.txt", index_col = 0, header = None)
parnames = UB.index
UB = UB[1].tolist()
LB = pd.read_csv(Path + "/LB-1-Muskinguk.txt", index_col = 0, header = None)
LB = LB[1].tolist()
```

(continues on next page)

(continued from previous page)

```
Coello.ReadParametersBounds(UB, LB, Snow)

# observed flow
Coello.ReadDischargeGauges(Path + "Qout_c.csv", fmt="%Y-%m-%d")
### Routing
Route=1
# RoutingFn=Routing.TriangularRouting2
RoutingFn = Routing.Muskingum

### run the model
Run.RunLumped(Coello, Route, RoutingFn)
```

- Measure the performance of the baseline parameters

```
Metrics = dict()
Qobs = Coello.QGauges[Coello.QGauges.columns[0]]

Metrics['RMSE'] = PC.RMSE(Qobs, Coello.Qsim['q'])
Metrics['NSE'] = PC.NSE(Qobs, Coello.Qsim['q'])
Metrics['NSEhf'] = PC.NSEHF(Qobs, Coello.Qsim['q'])
Metrics['KGE'] = PC.KGE(Qobs, Coello.Qsim['q'])
Metrics['WB'] = PC.WB(Qobs, Coello.Qsim['q'])

print("RMSE= " + str(round(Metrics['RMSE'],2)))
print("NSE= " + str(round(Metrics['NSE'],2)))
print("NSEhf= " + str(round(Metrics['NSEhf'],2)))
print("KGE= " + str(round(Metrics['KGE'],2)))
print("WB= " + str(round(Metrics['WB'],2)))
```

Define wrapper function and type

Define the wrapper function to the OAT method and put the parameters argument at the first position, and then list all the other arguments required for your function

the following defined function contains two inner function that calculates discharge for lumped HBV model and calculates the RMSE of the calculated discharge.

the first function *RUN.RunLumped* takes some arguments we need to pass it through the *OAT* method [ConceptualModel,data,p2,init_st,snow,Routing, RoutingFn] with the same order in the defined function "wrapper"

the second function is RMSE takes the calculated discharge from the first function and measured discharge array

to define the argument of the "wrapper" function 1- the random parameters valiable i=of the first function should be the first argument "wrapper(Randpar)" 2- the first function arguments with the same order (except that the parameter argument is taken out and placed at the first potition step-1) 3- list the argument of the second function with the same order that the second function takes them

There are two types of wrappers - The first one returns one value (performance metric)

```
# For Type 1
def WrapperType1(Randpar,Route, RoutingFn, Qobs):
    Coello.Parameters = Randpar

    Run.RunLumped(Coello, Route, RoutingFn)
```

(continues on next page)

(continued from previous page)

```
    rmse = PC.RMSE(Qobs, Coello.Qsim['q'])
    return rmse
```

Instantiate the SensitivityAnalysis object

```
fn = WrapperType2

Positions = [10]

Sen = SA(parameters,Coello.LB, Coello.UB, fn, Positions, 5, Type=Type)
```

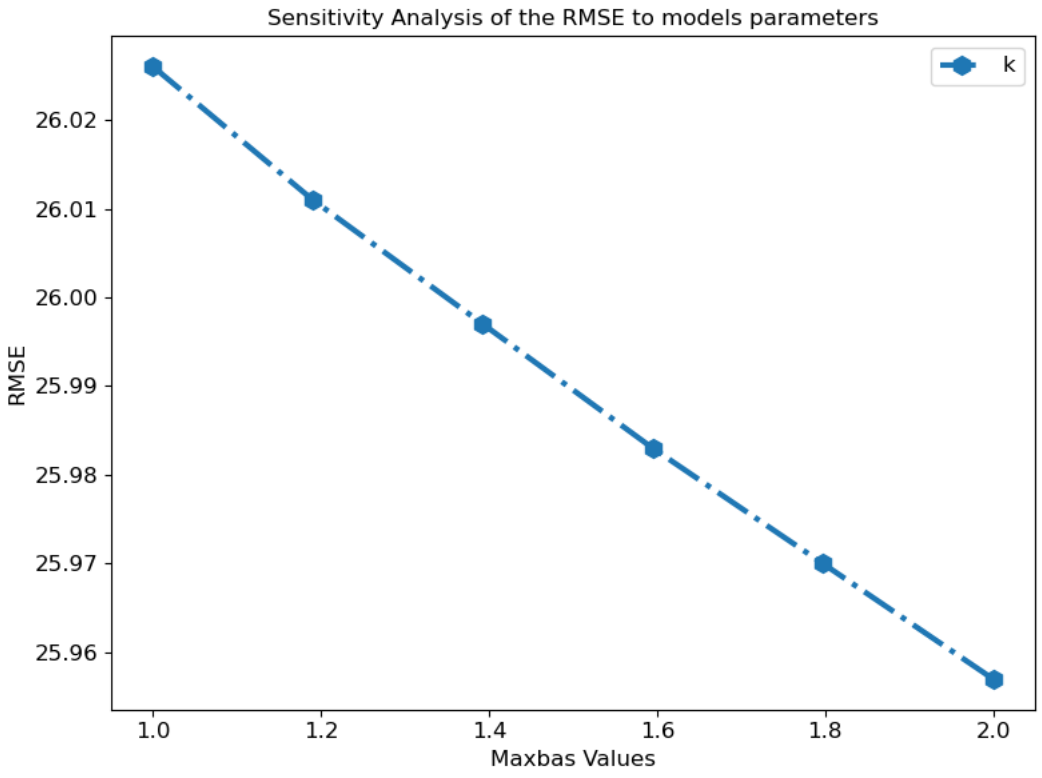
Run the OAT method

Display the result with the SOBOL plot

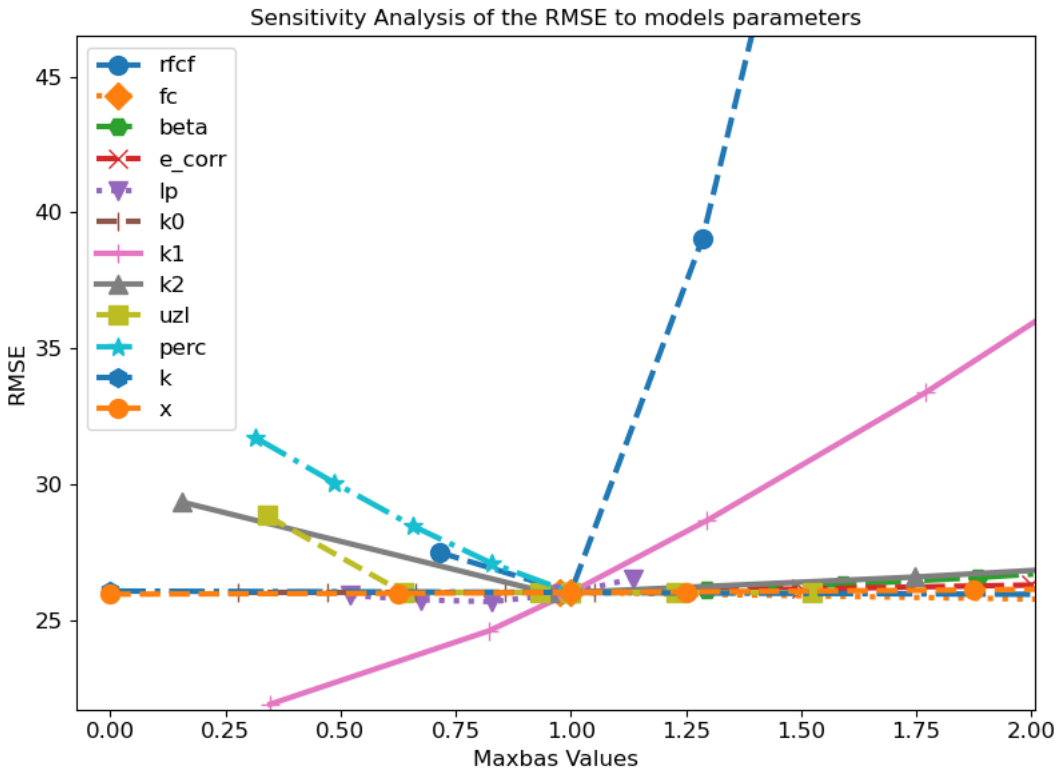
```
From = ''
To = ''

    fig, ax1 = Sen.Sobol(RealValues=False, Title="Sensitivity Analysis of the RMSE to models parameters",
                xlabel = "Maxbas Values", ylabel="RMSE", From=From, To=To,xlabel2='Time',
                ylabel2='Discharge m3/s', spaces=[None,None,None,None,None,None])
```

- Type 1 with one parameter



- Type 1 with all parameters



The second type

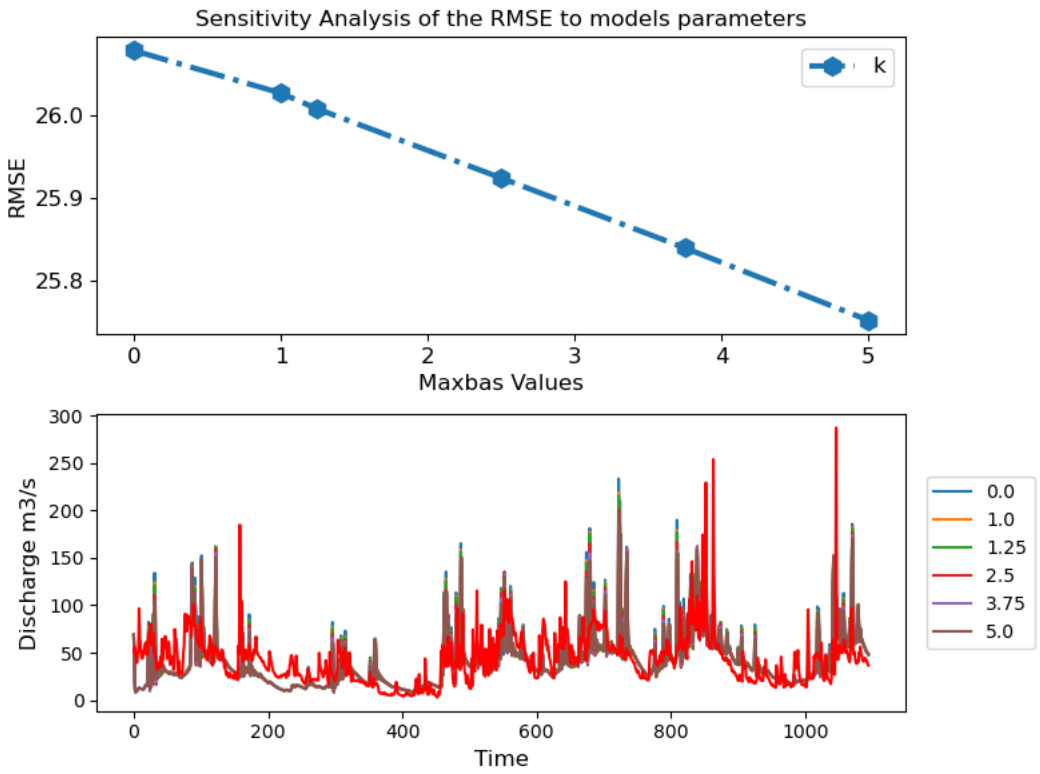
- The second wrapper returns two values (the performance metric and the calculated output from the model)

```
# For Type 2
def WrapperType2(Randpar,Route, RoutingFn, Qobs):
    Coello.Parameters = Randpar

    Run.RunLumped(Coello, Route, RoutingFn)
    rmse = PC.RMSE(Qobs, Coello.Qsim['q'])
    return rmse, Coello.Qsim['q']


    fig, (ax1,ax2) = Sen.Sobol(RealValues=False, Title="Sensitivity Analysis of the RMSE␣
↪to models parameters",
          xlabel = "Maxbas Values", ylabel="RMSE", From=From, To=To,xlabel2='Time',
          ylabel2='Discharge m3/s', spaces=[None,None,None,None,None,None])
    From = 0
    To = len(Qobs.values)
    ax2.plot(Qobs.values[From:To], label='Observed', color='red')
```

- Type 2


Sensitivity Analysis of the RMSE to models parameters
k
RMSE
26.0
25.9
25.8
0
1
2
3
4
5
Maxbas Values
Discharge m3/s
300
250
200
150
100
50
0
0
200
400
600
800
1000
Time
0.0
1.0
1.25
2.5
3.75
5.0