REGSim tool documentation

Lakshmi E

Indian Institute of Technology Hyderabad

19-Mar-2021

Table of Contents

Indian Institute of Technology
Hyderabad

lakshhmi.17@gmail.com

1. Installation of external python libraries:

1.1 Installing the Platypus package for non-dominated sorting genetic algorithm (NSGA-II):

- To install using pip, run the following command,

```
pip install platypus-opt
```

- To install the Platypus package using anaconda,

```
conda config --add channels conda-forge
conda install platypus-opt
```

- For more details about the Platypus package,

https://platypus.readthedocs.io/en/latest/getting-started.html#

1.2 Installing pyDOE module package for Latin Hypercube sampling (LHS) [4]:

- To install the package using pip command,

```
pip install --upgrade pyDOE
```

- To install using anaconda,

```
conda install -c conda-forge pydoe
```

- To download and install manually,

https://pythonhosted.org/pyDOE/index.html

Note: REGSim is under progressive development, and you can download the latest version at https://github.com/LaksE91/REGSim.git

2. Introduction

The tutorial gives an application of the Recharge Estimation and Groundwater Simulation (REGSim) tool to simulate the groundwater level using a simple conceptual model(Box-1). This toolbox helps to understand groundwater behaviour at a regional scale to guide water management. The model works based on the water budget approach with inflow as recharge, lateral inflow, and outflow as pumping, lateral outflow, which influences groundwater storage. We also included geographic information system(GIS) tools in REGSim to automate lateral flow estimation based on the observed groundwater head.

The following section describes the process of the REGSim toolbox to run the framework in the python platform. The first step is to estimate the lateral flow fluxes, which are further used as input during the model's calibration period (Section 3.1). The second step is about the simulation and optimisation of the groundwater model. In the next level, validation of the model is performed based on the Pareto optimal solutions obtained during the calibration period (Section 3.2). The third part describes the uncertainty and sensitivity analysis used for the model (Section 3.3).

BOX-1:

The groundwater balance equation used in this framework is shown in equation (1) [2], and equation (2)

$$h_t = h_{t-1} + \frac{r * P_t}{S_y} - \frac{Q_{p_t}}{S_y * A} + \frac{Q_{in_t} - Q_{out_t}}{S_y * A} \quad (1)$$

$$h_t = h_{t-1} + \frac{r * (P_t - PE_t)}{S_y} - \frac{Q_{p_t}}{S_y * A} + \frac{Q_{in_t} - Q_{out_t}}{S_y * A} \quad (2)$$

where, h is the depth to water level [m], r is the recharge factor [-], P is the rainfall [m], PE is the potential evapotranspiration [m], S_y is the specific yield [-], $Q_{in/out}$ is the lateral inflow/outflow [m^3/month], Q_p is the pumping rate [m^3], A is the aquifer area [m^2], and subscript t denotes the current month.

3. Implementation of REGSim tool with an example dataset

The REGSim tool aims to model the time series of regional groundwater levels using a lumped conceptual groundwater model. The working process and methods are illustrated in detail with an application to the aquifer system of the urban agglomerate Hyderabad, India. Here, REGSim tool is incorporated with an example dataset to simulate the groundwater level. The dataset for optimisation and uncertainty analysis is supported by the comma-separated **(.csv)** file containing five inputs with monthly time steps includes rainfall, potential evapotranspiration, groundwater head, lateral inflow, and outflow.

The toolbox consists of four sets of modules, and each module works with the required input and methods for the simulation (Table T1). Step-2_Calibration_of_the_model.py, describes the calibration of the model with optimisation using NSGA-II, and the input used for the simulation are monthly depth to water level, rainfall and evapotranspiration. Step-3_Validation_of_the_model.py, predict the groundwater head using the optimal parameter sets (specific yield, pumping rate and recharge factor). Step-4a_Uncertainity_analysis.py and Step-4b_Sensitivity_analysis.py, used to predict the uncertainty and sensitivity of the input parameter using Generalised Likelihood Unceraintay estimation (GLUE) method. The input required for this process is monthly depth to water level, rainfall and potential evaporation, LHS parameter sets and the list of the function defined in these modules are explained in detail in table T2.

Table T1: Functionalities of the three main modules of REGSim.

Module	**Main Function**	**Input data requirement**	**Other inputs**
Step-2_Calibration_of_the_model.py Step-3_Validation_of_the_model.py	Optimisation using NSGA-II	Depth to water level time series, Rainfall time series, Potential evapotranspiration (monthly scale or coarser)	Feasible parameter ranges , number of function evaluations
Step-4a_Uncertainity_analysis.py Step-4b_Sensitivity_analysis.py	Uncertainty analysis using GLUE	Same as Optimisation	Number of random parameter sets created (LHS), the definition of the likelihood function (NSE)

Table T2: List of functions and their specifications used in REGSim.

Function	**User/pre-defined**	**Description**	**Operation**
`data_sep()`	User-defined function	It divides the data into training and testing period.	NSGA-II, GLUE
`sortinput()`		Sort the input data header generically.	
`gw_model()`		To solve the problem using the NSGA-II algorithm.	NSGA-II

`sinefunc()`
`linearfunc()`
`stepfunc()`
`trapzfunc()`
		Solve the seasonal pumping rate in the different distribution function	

`sim_mod()`		It invokes the groundwater model and returns the metrics	
`NSGAII()`	Per-defined class of Platypus	It calls the NSGA-II algorithm to perform optimisation	
`paretoplot()`		It shows the graphical representation of the pareto optimal set.	
`modelrun()`	User-defined function	It calls the groundwater model for the simulation.	Model validation
`valplot()`		It plots the simulated and observed head.	

`rmse_metric()`
`mae_metric()`
`nse_metric()`
		The function used to invoke the performance metrics.	NSGA-II, GLUE

`lhs()`	Pre-defined class of pyDOE	To generate a uniform sample of the parameters.	
`rand()`		To call the LHS sample set for all the parameters considered for the three recharge cases.	
`uncertain()`		It invokes the GLUE method to estimate predictive uncertainty.	
`myglueplot()`	User-defined function	To plot the prediction intervals to capture the observed head.	GLUE
`obsv_inside()`		To determine the percentage of the observed head within the prediction interval.	
`sim_glue()`		It invokes the GLUE method to return the acceptable parameter set.	

`ecdf()`		To evaluate the empirical cumulative distribution function	
`eplt()`		The function to plot the CDF of parameter sets.	

The number of parameters (NPAR) required to simulate groundwater vary according to the recharge specification (e.g. 3, 4, or 5). The total number of simulation time periods (N) is divided into calibration (TCAL) and validation (TVAL) time steps. The user can also control the maximum allowable function evaluations (NFE) for NSGA-II, and the final pareto optimal set (POP) are obtained during optimisation. The minimum and maximum values of the parameter specified using PRANGE. The number of random parameter sets generated for GLUE is NLHS. The cumulative distribution function for each recharge scenarios (PCDF) is estimated. Table T3 details the inputs for each function.

Table T3: List of arguments implemented in the REGSim specified functions.

Arguments	**Function used**	**Description**	**Size**
input_data	`data_sep()`	Dataset of the model	Nx4 where N: number of simulation periods
input_para	`gw_model()`	Decision variables	NPAR
rech_case	`sim_mod()`, `paretoplot()`	Recharge scenarios (cases: 1,2,3)	1
area	`sim_mod()`	Area of the boundary	1
input_calib		Dataset during the calibration period	TCALx4
V, M	`NSGAII()`	V = the total number of decision variables for each case. M= the total number of objective functions considered. `problem.types` = it assigns the decision variables `problem.function` = defines the function (here,	1

		gw_model()) that call the model with a list of decision variables and list the objective values.	
Nsim		Number of iterations during simulations	NFE
df_opt	`paretoplot()`	Dataframe contains pareto optimal solutions.	POPxNPAR
optimal_set	`modelrun()`	List of optimal solutions for three cases.	NPAR
obsv_head	`valplot()`	Observed groundwater head data.	Nx1
gwhead		Simulated groundwater head.	Nx1
tcount		Duration of the model.	N
months		Variable to label the month/year in the graph.	1
mv		model variant scenario (0- for P and 1 for P-PE)	1
NPAR	`rand(),lhs()`	Number of parameters used to generate random sample sets	1
NLHS		Number of sample sets.	1
Qpmax		Maximum pumping range	PRANGE
Sy		Specific yield	PRANGE
r1 r11 r12 r21 r22 r23		Recharge factor for case-1 Recharge factor for non-monsoon, case-2 Recharge factor for monsoon, case-2 Recharge factor for winter, case-3	PRANGE

		Recharge factor for summer, case-3 Recharge factor for monsoon, case-3	
Qpmax Qpmin	`sinefunc()`, `linearfunc()`, `stepfunc()`, `trapzfunc()`	Maximum pumping rate Minimum pumping rate	PRANGE
samp_set	`uncertain()`, `sim_glue()`	Random sample parameter set	NLHS x NPAR
lb		The lower limit of the confidence interval	1
ub		The upper limit of the confidence interval	
cut_off1 cut_off2		The threshold for a behavioural set The threshold for a non-behavioural set	1
h_max		Maximum depth to water table within the study area (meter)	1
CI_bounds	`myglueplot()`, `obsv_inside()`	An input data frame of uncertainty prediction.	Nx5
evar_p	`eplt()`	Cumulative probability (0-1)	PCDF
evar_q		A sample set of each input parameter	

The input file required for the tutorial is provided in `Data/` folder, and the expected results of the groundwater model are added in `Example_results/` folder.

The execution of the scripts is supported by the **IDLE/Spyder/command prompt.**

3.1 Estimation of lateral flow:

a. **Code name:** *Step-1a_Estimation_of_slope.py*

Description:

Evaluation of slope along the boundary facilitated using the ArcGIS tools, and the manuscript addresses detailed methodology. The input data and the specifications required for this module are given in Table T4. The function and tools used to estimate the average slope are shown in table T5. The **Create Points on Lines** tool for creating a point on the lines is downloaded from http://ianbroad.com/arcgis-toolbox-create-points-polylines-arcpy/.

Table T4: Data and its specification for the module.

Input data	File format	File name format	Remarks
Groundwater elevation	Raster (.tif)	'YEAR_GWL_MONTH.tif'	'2004_GWL_Jan.tif'
Study area boundary	Vector (.shp)	'bound_XXXX.shp', 'bndin_XXXX.shp', 'bndout_XXXX.shp',	XXXX – study area name Make it as three copies

- Set the current directory where the data and codes are stored in the folder (Fig.T1). Given the user-defined buffer distance (meters) and the number of points, the average gradient along the study area boundary is calculated (Fig.T2).

```
# work in the current directory
env.workspace=(input("give the current directory:"))
dirpath = os.getcwd()

#assign the buffer distance
buffer_dist = input('Buffer distance between the study area (meters):')
num_pts     = input('no. of points considered across the boundary:')
```

Fig. T1: Screenshot of the script with user-defined inputs.

Table T5: Functions and tools used in REGSim to generate gradient across the model boundary.

Functions/Tools	Input dataset	Definitions
buffer()	bound (.shp)	Creates the buffer inside and outside using the reference boundary file.

ext_pts()	bound, boundin, boundout, bufin, bufout (.shp)	Create points across the reference boundary, buffer inside and buffer outside files
pts_value()	Raster (.tif) , list (list of shapefiles)	Extract groundwater elevation raster values to the points for three files such as reference boundary, buffer inside, and buffer outside.
avg_sl()	Raster	Estimate the average slope of the reference boundary.

Output:

```
give the current directory:'F:\CE15RESCH11013_LAKSHMI\Code\GWM\Code_process_instruct
\Step_1_Lateralflowestimation'
Buffer distance between the study area (meters):1000
no. of points considered across the boundary:1000
Creating buffer inside and outside the boundary area...
Converting polygon to line feature class...
Created points to the feature class...
bound_hmda.shp
bndin_hmda.shp
bndou_hmda.shp
buffin1000.shp
bufout1000.shp
Extracting the elevation data from the raster to the point featureclass...
2004_GWL_Jan.tif
buffin1000.shp
bndin_hmda.shp
bufout1000.shp
bndou_hmda.shp
Estimating slope in each point of the boundary area...
['bndin_Jan_extrpts.dbf', 'bndou_Jan_extrpts.dbf']

        Table:          bound1000_Jan_extrpts1000_04.dbf
        Type:           dBase III Plus
        Codepage:       ascii (plain ol' ascii)
        Status:         DbfStatus.CLOSED
        Last updated:   2020-04-09
        Record count:   1000
        Field count:    8
        Record length:  100
        --Fields--
          0) mem_point_ N(10,0)
          1) mem_point1 F(13,11)
          2) bound_hmda N(9,0)
          3) bound_hm_1 N(10,0)
          4) bound_hm_2 N(6,0)
          5) bound_hm_3 F(13,11)
          6) rastervalu F(19,11)
          7) slope F(19,11)

 Saving the output file
```

Fig. T2: Screenshot of the output obtained from Step-1a_Estimation_of_slope.py

b. Code name: *Step1b_Estimation_of_laterflow.py*

Description:

Lateral flow fluxes are estimated based on Darcy's law (Box-2). Input data required for the script is the **'.csv.'** file (*Note: Rearrange the file name month-wise, see the, e.g., figure, slope.csv*), which contains the file names of output (**'output.csv'**) from the previous step. Here, lateral flow divided into lateral inflow (flow enters into the study area boundary) and lateral outflow (flow leave the study area boundary).

```
slope - Notepad
File  Edit  Format  View  Help
bound1000_Jan_extrpts1000_04
bound1000_Feb_extrpts1000_04
bound1000_Mar_extrpts1000_04
bound1000_Apr_extrpts1000_04
bound1000_May_extrpts1000_04
bound1000_Jun_extrpts1000_04
bound1000_Jul_extrpts1000_04
bound1000_Aug_extrpts1000_04
bound1000_Nov_extrpts1000_04
bound1000_Dec_extrpts1000_04
```

Run the script and set the current directory where the data and codes are available. The average slope is generated as output with given user-defined input (Fig.T3).

BOX-2:

The lateral flow can be estimated using Darcy's law as follows (3):

$$Q_{in/out_t} = T * i * L \tag{3}$$

Where, $Q_{in/out}$ is the lateral flow (m^3/month), i is the hydraulic gradient (m/m), T is the transmissivity (m^2/month), l is the length of the study area boundary (m).

Output:

```
give the current directory:'F:\CE15RESCH11013_LAKSHMI\Code\GWM\Code_process_inst
ruct\Step_1_Lateralflowestimation'

iterating using zip
Transmissivity of the aquifer:(unit m2/day)144
Polyline study area boundary shapefile:'bound_hmda_line.shp'

iterating using zip
[2718909.7653732379] [2420892.2911623488]

 Lateral inflow and outflow are estimated
```

Fig. T3: Screenshot of the output obtained from the lateral flow estimation script.

The sample dataset to execute the lateral flow scripts (section 3.1 a, b) includes groundwater elevation raster (January 2004) and boundary shapefiles. User can automate the python script with the given monthly groundwater elevation raster and boundary shapefiles.

3.2 Calibration and validation of the model:

a. Code name: *Step_2_Calibration_of_the_model.py*

Description:

Non dominated sorting genetic algorithm II (NSGA-II) [3], multi-objective optimisation method used during the calibration of the model. The calibrated parameters such as specific yield, recharge factor, and maximum pumping rate and objective function as Root Mean Squared Error, Mean Absolute Error, and Nash-Sutcliffe model efficiency are considered during the optimisation process. Two different model variant condition is provided include, A: P and B: P-PE. The data required to calibrate the model are discussed in table T1 and simulated the model using the function `modelrun` (Fig. T4).

- We simulate the model under three recharge scenarios, such as constant recharge for all the months (Case 1), two recharge factors for monsoon and non-monsoon seasons (Case 2), and three recharge factors for winter, summer and monsoon seasons (Case 3).
- In the given example, the total number of months considered is 60 and the calibrated period is 48. Run the model with required recharge conditions.
- Set the parameters to range based on the characteristic of the aquifer considered for the analysis. The number of decision variables varies based on the test case is considered. E.g., Case 1 has three decision variables, such as pumping rate, specific yield, recharge factor, and three objective functions as default for all the cases. User can give their required iterations during the simulation.
- The model executed with the specified parameters, and the performance metrics are determined by fitting the observed and simulated groundwater head. Using the NSGA-II algorithm, the groundwater model is calibrated and computes the optimal pareto front. The best optimal solutions are selected based on user decisions. The optimal pareto solutions obtained during the simulation are stored as **'pareto_case{}_modvar{}.txt.'**
- The user can edit or add the objective functions in the script **'metrics.py'** to obtain the pareto optimal front (Fig. T5).

```
import numpy as np
import numpy.matlib
from metrics import *
```


lakshhmi.17@gmail.com

```python
from Utils import fillrech
from pumpfunc import *

## choice of selecting the return output
class choicedata():
    def __init__(self, rmse,mae,nse,gwhead):
        # you can put here some validation logic
        self.rmse     = rmse
        self.mae      = mae
        self.nse      = nse
        self.gwhead   = gwhead

#function to define the model
def modelrun(Pset,var,area,test_case,mv,pcase):

# assign the input to the variable
    pdata                    = np.array(var.P)  # rainfall
    gwdata                   = np.array(var.H)  # groundwater head
    pedata                   = np.array(var.PE) # potential evapotranspi
ration

# choose model variant (0: Recharge as the function of P; 1: Recharge
as the function of P and PE)
    def precp():
        er = pdata/1000
        return er

    def pevap():
        er                    = (pdata-pedata)/1000
        sort_er               = [i if i>0 else 0 for i in er]
        return sort_er

    switcher ={
            0: precp,
            1: pevap,
            }

# Switch case function to select the model variant condition
    def model_variant(argument):
```


Indian Institute of Technology
Hyderabad
lakshhmi.17@gmail.com

```
        # Get the function from switcher dictionary
        func = switcher.get(argument)
        if func is None:
            raise ValueError("test case not found")
        # Execute the function
        return func()

# Check the lateral flow inclusion
    try:
        qin                  = np.array(var.Qin) # Lateral inflow
        qout                 = np.array(var.Qin) # lateral outflow
    except:
        qin                  = np.zeros(len(gwdata))
        qout                 = np.zeros(len(gwdata))

# Parameterisation
    Sy                       = float(Pset[0]) #specific yield
    Qpmax                    = float(Pset[1]) # pumping discharge

#get the number of months of avialable data
    nummonths                = len(gwdata)
    numyears                 = nummonths/12

# func call to generate the recharge factor
    rechargetimes= fillrech(test_case,var,Pset,summer=6, winter=10, mo
nsoon=None)
# max pumping with 50% less in monsoon and 100% in nonmonsoon season
    if pcase ==1:
        tm                        = (np.arange(1,13,1))
        Qp                        = np.array(sinefunc(tm,Qpmax,0.5*Qpmax
))
        #Qp                       = np.array([1,1,1,1,1,1,0.5,0.5,0.5,0.
5,1,1])
        pumping                   = Qp*10**6
    if pcase==2:
        tm                        = (np.arange(1,13,1))
        Qp                        = np.array(linearfunc(tm,Qpmax,0.5*Qpm
ax))
        pumping                   = Qp*10**6
```

```
    if pcase ==3:
        tm                              = (np.arange(1,13,1))
        Qp                              = np.array(stepfunc(tm,Qpmax,0.5*Qpmax
))
        pumping                         = Qp*10**6
    if pcase==4:
        tm                              = (np.arange(1,13,1))
        Qp                              = np.array(trapzfunc(tm,Qpmax))
        pumping                         = Qp*10**6

# repeat the data for the respective years
    pumptimes                       = numpy.matlib.repmat(pumping,1,numyears)
    pumptimes                       = pumptimes.reshape(nummonths)

#assign Constant and initial variables as an input to the model
    gwhead                          = np.zeros(nummonths)
    gwhead[0]                       = gwdata[0]             # initial head
    effectiveer                     = model_variant(mv)     # converting millime
ter to meter

# iteration of the model starts
    for m in range(1,nummonths):
        rh                          = (effectiveer[m]*rechargetimes[m])/Sy
        ph                          = (pumptimes[m]/(Sy*area))
        lh                          = ((qin[m]-qout[m])/(Sy*area))
        gwhead[m]                   = (gwhead[m-1] - rh + ph -lh)

# calculate the metrics
    rmse                            = rmse_metric(gwdata,gwhead) # minimize
    mae                             = mae_metric(gwdata,gwhead)  # minimize
    nse                             = -nse_metric(gwdata,gwhead) # maximize
    return rmse,mae,nse,gwhead

# Calbration process for the model
def sim_mod(Pset,var,area,test_case,mv,pcase):
    choice_data= modelrun(Pset,var,area,test_case,mv,pcase)
    return choice_data[0], choice_data[1],choice_data[2]
```

Fig. T4: Commented REGSim code to simulate the groundwater model.

Indian Institute of Technology
Hyderabad

lakshhmi.17@gmail.com

```
# Calculate Root Mean Squared Error
def rmse_metric(obs,sim):
    rmse    = np.sqrt((np.mean((obs - sim)**2)))
    return rmse

# Calculate Mean Absolute Error
def mae_metric(obs,sim):
    mae = np.mean(np.abs((obs - sim)))
    return mae

# Calculate Nash Sutcliff Efficiency
def nse_metric(obs,sim):
    nse     = 1 - sum((sim-obs)**2)/sum((obs-np.mean(obs))**2)
    return nse
```

Fig. T5: Implementation of performance metrics used in REGSim framework (metric.py).

Output:

The function `paretoplot()` is invoked to perform the specific task, and the output is generated using the python code **'visualplot.py'** (Fig.T6) .

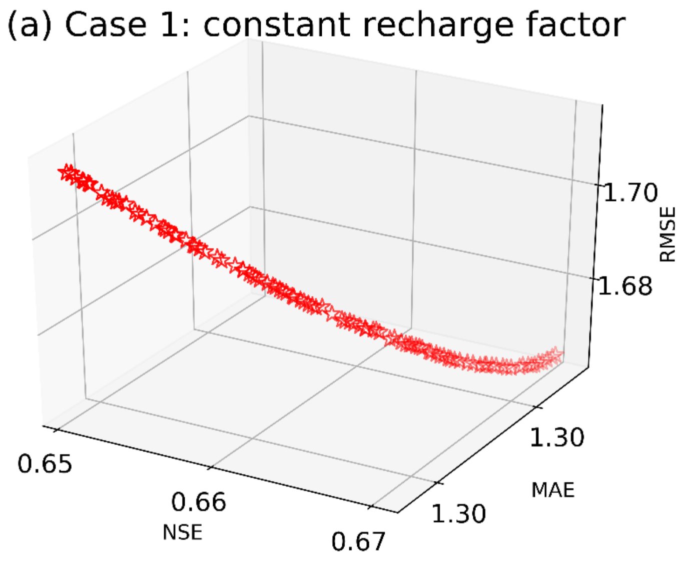


Fig. T6: Pareto optimal front obtained for the model variant B: P-PE as a recharge function for constant recharge factor.

Indian Institute of Technology
Hyderabad
lakshhmi.17@gmail.com

b. **Code name:** *Step_3_Validation_of_the_model.py*

Description:

The optimal solutions obtained from the pareto front is further used to validate the model for three recharge scenarios. For the given example, the optimal value of calibrated parameters chosen for the Case 1 recharge scenario.

Output:

The given example plot is generated based on the `matplotlib` module used in the code. The user can modify the code x-axis range based on the time and month of the graph in **'Step 3 Validation_of_the_model.py'** and also the other specification such as annotations, text properties (***visualplot.py***) concerning the requirement (Fig. T7).

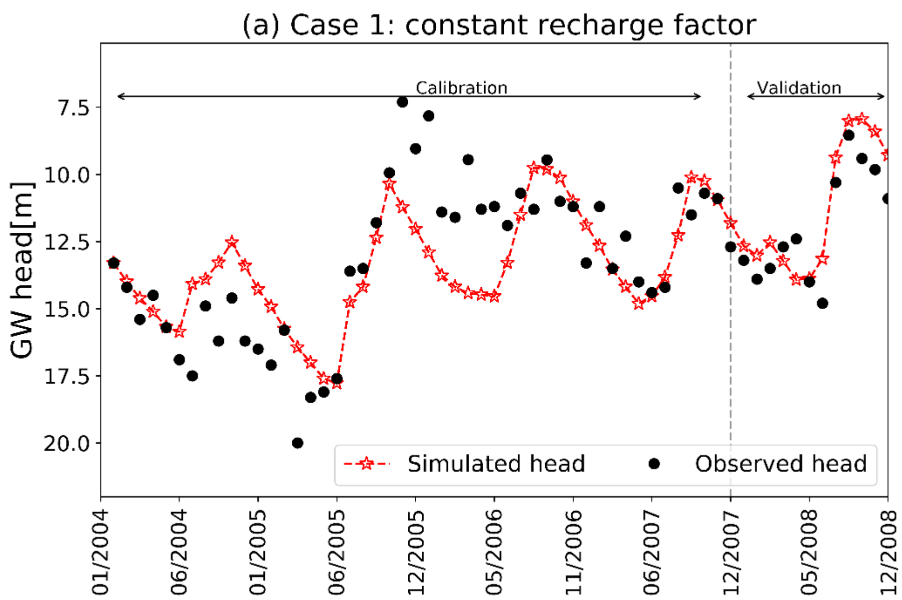


Fig. T7: Validation of the groundwater model for the constant recharge factor (model variant A).

3.3 Uncertainty and sensitivity analysis:

a. **Code name:** *Step_4_Uncertainity_analysis.py*

Description:

- Generalised likelihood uncertainty estimation (GLUE) proposed by [1] is employed to predict the uncertainty in the groundwater model (Please refer to the author's paper for the detailed methodology) (Fig.T9). To assess the uncertainty, the model assigns a plausible range of each parameter. Here, random parameter samples obtained using the Latin hypercube sampling method (LHS) (Fig.T8).

- Run the script and give the parameter range for all three cases to generate the random sample sets, as shown in the figure below. The histogram shows the LHS sampling for all the parameter sets.

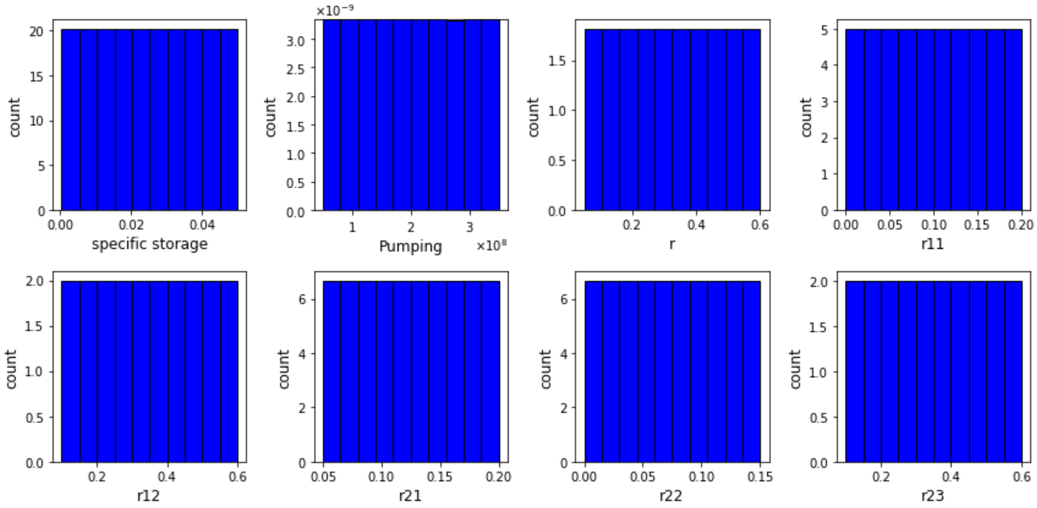


Fig. T8 Screenshot of the generation random samples using LHS method

- Assign the confidence interval limit to predict the uncertainty interval. For example, 90% confidence interval is used, where 5% is the lower limit, and 95% is the upper limit. Also, assign the percentage of the acceptable threshold (behavioural set), say 5% here. Assume the maximum groundwater level (h_max) is feasible in your study area. Here, we considered 25m as the maximum groundwater level to avoid negative values during the simulation process

- The output from `rand` generates a random sample for the parameter set with a size of NLHS x NPAR matrix. The underlying GLUE method is defined in the `gluerun` module, and the uncertainty analysis is shown in Fig. T9. The result of the uncertainty function is visualised with the help of `glueplot` function. The sensitivity of the parameters in the model is mapped using the `ecdfplot` function, which works based on the cumulative distribution function (CDF).

```
import numpy as np
import numpy.matlib
import pandas as pd
from metrics import *
```


Indian Institute of Technology
Hyderabad
lakshhmi.17@gmail.com

```
from Utils import fillrech
from pumpfunc import *


# function defintion for GLUE simulation
def sim_glue(test_case,data,calib,Psets,A,lb,ub,c,h_max,mv,tb):
###################################################################
######################
    #create a dataframe for metrics
    inf_liklhd                         = pd.DataFrame()
    inf_liklhd["nse"]                  = np.zeros_like(Psets.p)

    #Initalize inputs to the model
    ns                                 = len(Psets)        # no of
simulation
    nm                                 = len(calib)        # no of
months
    numyears                           = nm/12             # no of
years
    h_int                              = np.zeros((nm,ns)) #Gw head
    h_int[0,:]                         = calib.H[0]        # initia
l head
###################################################################
############################
    # assign the input to the variable
    pdata                              = calib.P   # rainfall
    pedata                             = calib.PE  # potential evap
otranspiration

    # choose model variant (0: Recharge as the function of P; 1:
Recharge as the function of P and PE)
    def precp():
        er = pdata/1000
        return er

    def pevap():
        er                      = (pdata-pedata)/1000
        sort_er             = [i if i>0 else 0 for i in er]
        return sort_er
```

```
    switcher ={
            0: precp,
            1: pevap,
            }

# Switch case function to select the model variant condition
    def model_variant(argument):
        # Get the function from switcher dictionary
        func = switcher.get(argument)
        if func is None:
            raise ValueError("test case not found")
        # Execute the function
        return func()

# Check the lateral flow inclusion
    try:
        qin                = np.array(calib.Qin) # Lateral inflo
w
        qout               = np.array(calib.Qin) # lateral outfl
ow
    except:
        qin                = np.zeros(len(calib.H))
        qout               = np.zeros(len(calib.H))
################################################################
##########################
    effectiveer            = model_variant(mv)   # unit of rainf
all mm to m

    if  test_case==3:
        #case3
        r1                             = Psets.r21
        r2                             = Psets.r22
        r3                             = Psets.r23
        rechargeratio                  = np.array([r1,r1,r2,r2,
r2,r2,r3,r3,r3,r3,r1,r1])
    if  test_case==2:
        #case2
        r1                             = Psets.r11
```


Indian Institute of Technology
Hyderabad

lakshhmi.17@gmail.com

```
        r2                              = Psets.r12
        rechargeratio                   = np.array([r1,r1,r1,r1,
r1,r1,r2,r2,r2,r2,r1,r1])
    if  test_case==1:
        #case1
        r                               = Psets.r
        rechargeratio                   = np.array([r,r,r,r,r,r,
r,r,r,r,r,r]) #[0,0,0,0,0,1,1,1,1,1,0,0]

    # repeat the data for the respective years
    rechargetimes                   = np.zeros((nm,ns))
    for i in range(ns):
        rechargetimes[:,i]          = np.matlib.repmat(recharger
atio[:,i],1,numyears)

    # max pumping with 50% less in monsoon and 100% in nonmonsoo
n season (unit MCM)
    pumping                         = np.array([1,1,1,1,1,1,0.5,
0.5,0.5,0.5,1,1])*10**6
    pumptimes                       = np.matlib.repmat(pumping,1
,numyears)
    pumptimes                       = pumptimes.reshape(nm)

    for i in range(ns):
        for j in range(nm-1):
            rh                      = (effectiveer[j+1]*recharge
times[j+1,i])/(Psets.s[i])
            ph                      = (pumptimes[j+1]*Psets.p[i]
)/(Psets.s[i]*A)
            #lh                      = ((qin[j+1]-qout[j+1])/(Ps
ets.s[i]*A))
            h_int[j+1,i]            = (h_int[j,i] - (rh-ph))#+lh
))

    # calculate the metrics
        inf_liklhd.nse[i]           = nse_metric(calib.H,h_int[:
,i])

    ####### behavorial set######
```

```
    cutoff                              = c # assigning 100% as beha
varial
    numBehav                            = cutoff*len(Psets)

    if tb ==1:
        metrics                         = inf_liklhd.sort_values('ns
e', ascending=False)
        index                           = metrics.index.values
        # defining the likelihood
        #index is the reference of the sorted of nse
        behav_index                     = index[0:int(numBehav)]
        behav_rank                      = np.arange(numBehav,0,-1)
        behav_rank                      = behav_rank/1.0
        posterior                       = behav_rank / sum(behav_ran
k)
    #here posterior = likelihood(nse) as it is uniform distirubu
tion

    if tb ==2:
        numnonBehav                     = cutoff*len(Psets)
        metrics                         = inf_liklhd.sort_values('ns
e', ascending=True)
        index                           = metrics.index.values
        # defining the likelihood
        behav_index                     = index[0:int(numnonBehav)]
#index is the reference of the sorted of nse
        nbehav_rank                     = np.arange(numnonBehav,0,-1
)
        posterior                       = nbehav_rank / sum(nbehav_r
ank)

    if test_case ==1:
        df_case1                        = pd.DataFrame()
        df_case1["p"]                   = list(Psets.p[behav_index])
        df_case1["s"]                   = list(Psets.s[behav_index])
        df_case1["r"]                   = list(Psets.r[behav_index])
        db                              = df_case1
    if test_case==2:
        df_case2                        = pd.DataFrame()
```

```
        df_case2["p"]                    = list(Psets.p[behav_index])
        df_case2["s"]                    = list(Psets.s[behav_index])
        df_case2["r11"]                  = list(Psets.r11[behav_index
])
        df_case2["r12"]                  = list(Psets.r12[behav_index
])
        db                               = df_case2
    if test_case==3:
        df_case3                         = pd.DataFrame()
        df_case3["p"]                    = list(Psets.p[behav_index])
        df_case3["s"]                    = list(Psets.s[behav_index])
        df_case3["r21"]                  = list(Psets.r21[behav_index
])
        df_case3["r22"]                  = list(Psets.r22[behav_index
])
        df_case3["r23"]                  = list(Psets.r23[behav_index
])
        db                               = df_case3

     # assign the input to the variable
    pdata                                = data.P  # total rainfall
    pedata                               = data.PE  # total potential
evapotranspiration

    nsims                                = int(numBehav)  # no of sim
ulation
    nmon                                 = len(data)       # no of mo
nths
    nyrs                                 = nmon/12         # no of yea
rs

    gwhead_pred                          = np.zeros((nmon,nsims))
    gwhead_pred[0,:]                     = data.H[0]       # initial
head
    effrech                              = model_variant(mv)   # unit
of rainfall from mm to m
    nse_val                              = np.zeros(nsims)

    if test_case==3:
```

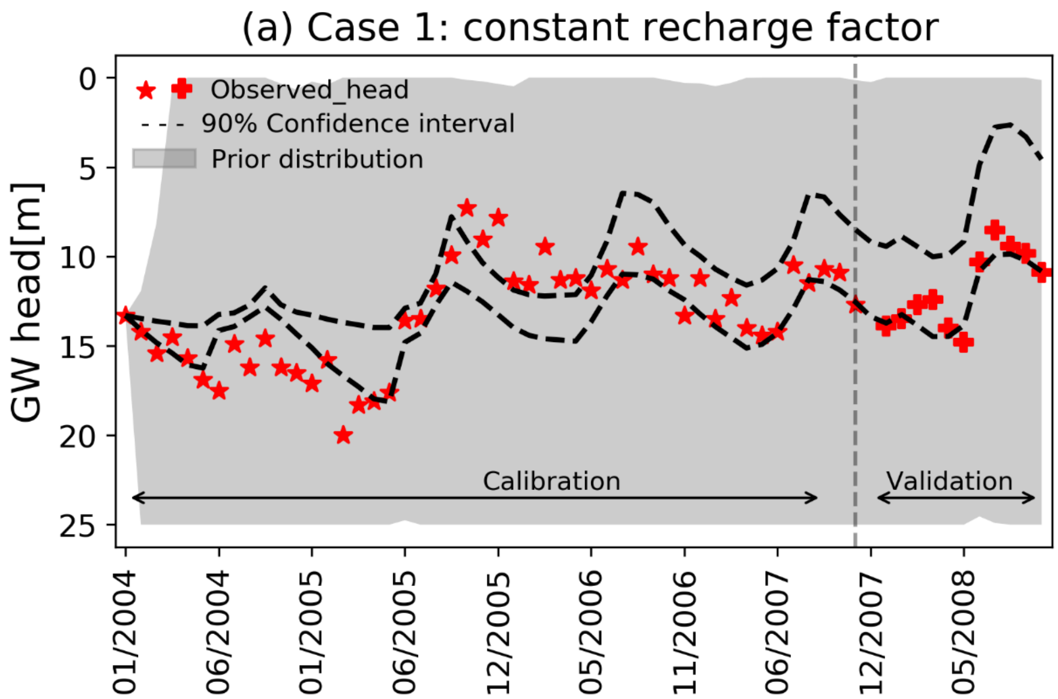
```
    # repeat the data for the respective years
        #case3
        r1                           = db.r21
        r2                           = db.r22
        r3                           = db.r23
        rechargeratio1               = np.array([r1,r1,r2,r2,r2,r
2,r3,r3,r3,r3,r1,r1])
    if test_case==2:
        #case2
        r1                           = db.r11
        r2                           = db.r12
        rechargeratio1               = np.array([r1,r1,r1,r1,r1,r
1,r2,r2,r2,r2,r1,r1])
    if test_case==1:
        #case1
        r                            = db.r
        rechargeratio1               = np.array([r,r,r,r,r,r,r,r,
r,r,r,r])

    rtimes                           = np.zeros((nmon,nsims))
    for i in range(nsims):
        rtimes[:,i]                  = np.matlib.repmat(recharger
atio1[:,i],1,nyrs)

    ptimes                           = np.matlib.repmat(pumping,1
,nyrs)
    ptimes                           = ptimes.reshape(nmon)

    for ii in range(nsims):
        for jj in range(nmon-1):
            rh1                      = (effrech[jj+1]*rtimes[jj+1
,ii])/(db.s[ii])
            ph1                      = (ptimes[jj+1]*db.p[ii])/(d
b.s[ii]*A)
            #lh                       = ((data.Qin[jj+1]-data.Qin
[jj+1])/(db.s[ii]*A))
            head_1                   = (gwhead_pred[jj,ii] - (rh1
-ph1))#+lh))
            if head_1 > h_max:
```

```
                gwhead_pred[jj+1,ii] =h_max
            elif head_1 < 0:
                gwhead_pred[jj+1,ii] = 0
            else:
                gwhead_pred[jj+1,ii] = head_1

        # calculate the metrics
        nse_val[ii]                 = nse_metric(data.H,gwhead_p
red[:,ii])

    lower                           = []
    upper                           = []

    # updating the likelihood at every time step
    for k in range(nmon):
        newgwhead                    = np.sort(gwhead_pred[k,:])
        #print(newgwhead)
        indx                         = np.argsort(newgwhead)
        #print(indx)
        newpost                      = np.cumsum(posterior[indx]
)

        lower.append(np.interp(lb,newpost,newgwhead))
        upper.append(np.interp(ub,newpost,newgwhead))

    return lower,upper,db

def uncertain(test_case,data,calib,Psets,A,lb,ub,c,h_max,mv,tb):
   Choice_data=sim_glue(test_case,data,calib,Psets,A,lb,ub,c,h_m
ax,mv,tb)
   return Choice_data[0], Choice_data[1]
```

Fig. T9: GLUE, uncertainty method used in the REGSim

Output:

In the example, we run the model for the case-1 scenarios with a 90% prediction interval, as shown in the Fig.T10. The grey portion interval is the total range of the parameter set considered. In contrast, the black dotted line is the 90% confidence interval (User can modify the plotting code, ***glueplot.py*** based on their requirement).




```
percentage of observation within Confidence interval:50.0%
```

Fig. T10: 90% prediction interval obtained from GLUE method for the constant recharge factor (model variant A).

b. Code name: <u>*Step_4b_Senstivity_analysis.py*</u>

Description:

- Empirical cumulative distribution (ECDF) function is used to plot the distribution of the datasets to identify the sensitivity of the input parameters. The ranges of the input parameters are based on the wide range values considered during the uncertainty analysis using the LHS method.
- ECDF curve for each input parameter is plotted based on the user-defined input variables such as the recharge cases, confidence interval, and behavioural and non-behavioural (Fig. T11). The behavioural set is the acceptable threshold of the performance metrics (say, top 5% of NSE), whereas the non-behavioural set is the remaining dataset of the performance metric (say, 1- 0.05 = 0.95).

Output:

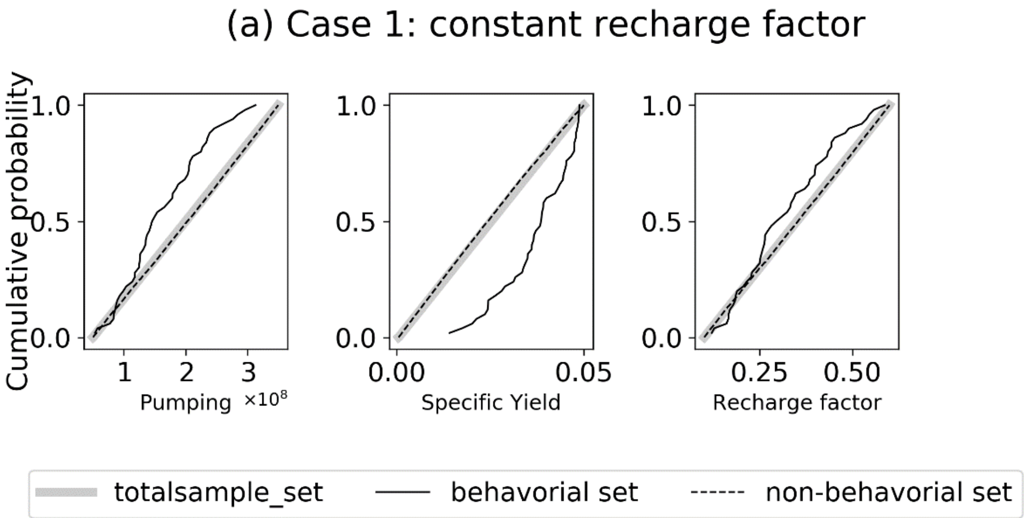


Fig. T11: The cumulative probability (y-axis) of each parameter as a function of parameter value (x-axis) for behavioural (solid black), non-behavioural (solid dashed), and prior distributions (grey solid) for case-1 (model variant A).

4. Norms of the aquifer properties:

The aquifer properties, such as transmissivity, specific yield, and recharge, can be used for the groundwater assessment based on the report published by the groundwater resource estimation committee (GEC). The following tables (T6, T7, T8) are the recommended values of the aquifer properties and utilised in the area with a lack of sufficient data and information available in the field (Source: http://cgwb.gov.in/documents/gec97.pdf).

Table T6: Transmissivity for different hydrogeological condition

Type of Aquifer	**Transmissivity range (m^2/day)**
POROUS ROCK FORMATIONS • Unconsolidated formations • Semi-consolidated formations	 250 to 4000 100 to 2300
HARD ROCK FORMATIONS	10 to 500

• Igneous and metamorphic rocks excluding volcanic and carbonate rocks • Volcanic rocks	25 to 100

Table T7: Specific yield for different hydrogeological condition

S.No	Formation	Recommended value (%)	Minimum value (%)	Maximum value(%)
1.	***Alluvial areas***			
	Sandy	16	12	20
	Silty	10	8	12
	Clayey	6	4	8
2.	***Hard rock areas***			
	Weathered granites, gneiss, schist with low clay content	3	2	4
	Weathered granites, gneiss, schist with significant clay content	1.5	1	2
	Weathered or vesicular, jointed basalt	2	1	3
	Laterite	2.5	2	3
	Sandstone	3	1	5
	Quartzite	1.5	1	2
	Limestone	2	1	3
	Karstified limestone	8	5	15
	Phyllites, shales	1.5	1	2
	Massive poorly fractured rock	0.3	0.2	0.5

Table T8: Recharge due to rainfall for different hydrogeological condition

S.No	Formation	Recommended value (%)	Minimum value (%)	Maximum value(%)
1.	***Alluvial areas***			
	Indo-Gangetic and inland areas	22	20	25
	East coast	16	14	18
	West coast	10	8	12
2.	***Hard rock areas***			
	Weathered granites, gneiss, schist with low clay content	11	10	12
	Weathered granites, gneiss, schist with significant clay content	8	5	9
	Granulite facies like charnockite etc.	5	4	6
	Vesicular and jointed basalt	13	12	14
	Weathered basalt	7	6	8
	Laterite	7	6	8
	Semi-consolidated sandstone	12	10	14
	Consolidated sandstone, quartzite, limestone (expect cavernous limestone)	6	5	7
	Phyllites, shales	4	3	5
	Massive poorly fractured rock	1	1	3

References:

[1] Beven, K., & Binley, A. (1992). The future of distributed models: model calibration and uncertainty prediction. *Hydrological processes*, *6*(3), 279-298, https://doi.org/10.1002/hyp.3360060305.

[2] Bredenkamp, D. B., Botha, L. J., Van Tonder, G. J., & Van Rensburg, H. J. (1995). Manual on quantitative estimation of groundwater recharge and aquifer storativity: based on practical hydro-logical methods. Water Research Commission.

[3] Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., & Fast, A. (2002). Nsga-ii. *IEEE transactions on evolutionary computation*, *6*(2), 182-197, https://doi.org/10.1109/4235.996017 .

[4] Lee, A. D. (2018). pyDOE: Design of experiments for Python. Python package version 0.3