ENEN90031 Quantitative Environmental Modelling

Overview of functions used in Modelling assignment 2

The following provides an overview of each of the MatLab functions provided along with Modelling Assignment 2 to help explain how the various MatLab functions fit together and can be used for each component of the assignment.

For the assignment you will need to finish the scripts for each part. You do not need to modify these functions, except:

- SCE_calibration.m: you need to set an appropriate ‘ngs’ value in part 1

Some functions are used as the main functions for particular assignment parts. Others are sub-functions used by the main ones. In the following there is a series of diagrams showing the functions called by the main function used in each section. Table 1 shows the functions, a brief summary of their purposes, and, where a function is the main function used for a particular part of the assignment, this is also pointed out. More detail is provided further below.

Each of the key functions has documentation in the first part of the function file which can be accessed either through the editor or by using;

```
>>help  functionName
```


Table 1: Summary of MatLab functions

Function	Purpose	Assignment part
cceua	A sub-function of Shuffled Complex Evolution code to do the evolution in a simplex	Part 1. Called by ‘sceua’
convertDailyToMonthly	Adds daily flows up to monthly totals	N/A
getParameterValues	Finds the values of a subset of parameters from the parameter structure	Wherever you need to extract parameter values from the parameter structure.
GLUE_existingRuns	Does a GLUE analysis using an existing set of model simulations	Potentially part 3 if you want to do GLUE without rerunning lots of Monte Carlo runs
HBV_noSnow	The actual HBV model	N/A (called from other functions)
HBV_GLUE_newRuns	Does GLUE analysis including running a new set of model simulations for new parameters	Potentially part 3 if you want to run GLUE independently of RSA
HBV_MonteCarlo	HBV_MonteCarlo	Part 4. Also called by GLUE and RSA codes to actually do Monte Carlo runs of the model
HBV_RegionalSensivity Analysis	Undertakes RSA and (optionally) GLUE analyses	Part 2 and potentially part 3 if you do RSA and GLUE together.
hydroDesign	Divides a capital budget between water storage and other infrastructure	Part 4
hydroPower	Simulates reservoir water storage and hydropower production	Part 4
parameterMatrices	Sets up matrices of randomised parameters for RSA and GLUE codes	N/A (Called by RSA and GLUE codes)
SCE_calibration	undertakes a shuffled complex evolution calibration of the wind energy model.	Part 1
sceua	The main function of the shuffled complex evolution algorithm	Part 1. Called by ‘SCE_calibration’

setParameterValues	Copies the values of a subset of parameters from a vector into the parameter structure	Part 2, 3 and 4. called by 'pumpedHydro_MonteCarlo'

Important MatLAb Codes

Functions for each part of the assignment

Functions to help with Parameter structures

Two general functions (getParameterValues, setParameterValues) are provided to interact with the parameter structures that are provided. These can be used to either find or modify the values of parameters in the structures. You can get or set anything from one to all the parameter values.

getParameterValues

```
>> parameterVector = getParameterValues( parameterNames, ...
     parameterStructure)
```

This function is used to get vector of parameter values from a structure variable. When this function is called values of the parameters mentioned in parameterNames will be retrieved as a vector from parameterStructure variable. Note that the order of names in the cell array (i.e. parameterNames) and the parameter value vector are the same. Note also the curly braces used to create the cell array.

e.g.

```
>>parameterNames = {'fc'; 'beta'};
>> parameterVector = getParameterValues( parameterNames, ...
     Params_UpperBound)
```


% output will be


```
>> parameterVector =

        700
          7
```

setParameterValues

This function takes a vector of parameter values and copies them into a parameter structure.

E.g. if you want to change Upper limit values of INSC and SMSC to 30 and 1500 respectively, then,

```
>> parameterNames = {'fc'; 'beta'};
>> parameterUpperlimit = [ 350; 2.1];
>> Params_UpperBound = setParameterValues(parameterUpperlimit, ...
     parameterNames, Params_UpperBound);
```

% the output well be

```
>> Params_UpperBound =

       fc: 350
     beta: 2.1
```

```
       pwp: 1
         l: 50
        k0: 0.5000
        k1: 0.5000
        kp: 0.9000
        k2: 0.9000
```

The model itself: HBV_noSnow

HBV_noSnow runs the daily rainfall runoff model HBV without any routing and returns streamflow.

For calculating monthly streamflow

```
>>flow_daily_mm=HBV_noSnow(Params_initialModel, ...
    Scotts_Creek_climate, true);
```

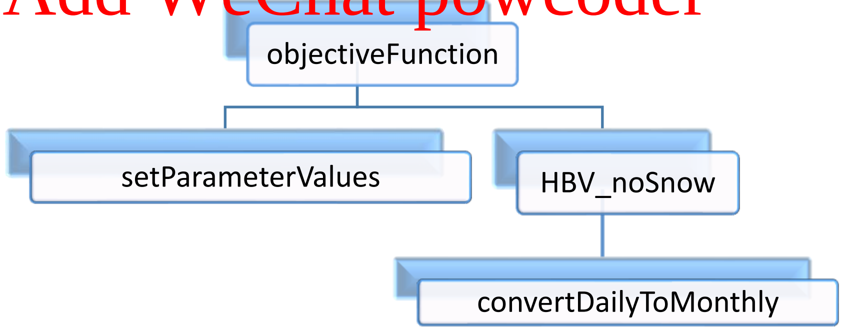
For calculating monthly streamflow

```
>>flow_monthly_mm=HBV_noSnow(Params_initialModel, ...
    Scotts_Creek_climate, false);
```

objectiveFunction

ObjectiveFunction.m is used in various places and is also a useful general function. This sets the model parameters to those supplied, runs the model for the specified period and calculates an objective function value plus model residuals.


```
>>[obj2,residuals]=objectiveFunction(bestParameters, ...
    parameterNames,parameterStructure,timeStart, timeEnd, ...
   Scotts_Creek_climate, Scotts_Creek_mm_day, false);
```

Functions for each part of the assignment

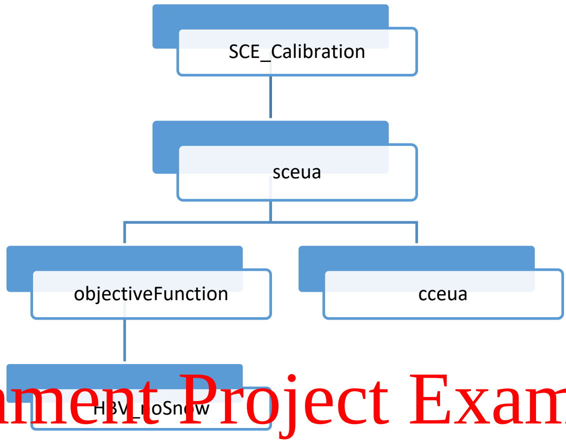
Note: Parts 1-3 should be undertaken with data from 2005-2018.

Part 1 – SCE Calibration

For undertaking the SCE calibration you can use SCE_calibration. The aim here is to find the best parameter set for the given data set. The calibration scheme parameters (*kstop* etc) can be found at lines 58 to 64 of the code

```
% Set Optimisation Settings
optimisationSettings.maxn = 200000;
optimisationSettings.kstop = 5;
optimisationSettings.pcento = 0.00001;
optimisationSettings.peps = 1e-6;
optimisationSettings.ngs = 4;

>> [Params_best,bestObjFn,Observed_Flow_trimmed,ClimateData_trimmed] = ...
      SCE_calibration( Params_initialModel, Params_LowerBound, ...
      Params_UpperBound, timeStart, timeEnd, Scotts_Creek_climate, …
      Scotts_Creek_mm_day, doMonthly, optimisationSettings);
```



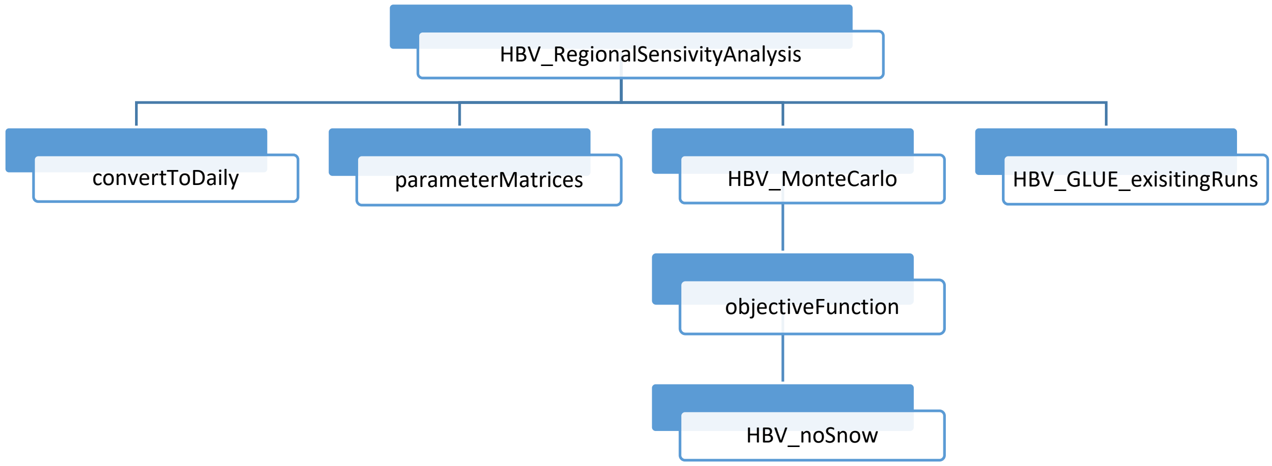
Part 2 – Regional Sensitivity Analysis

You have a choice of undertaking Regional Sensitivity Analysis and GLUE together, which the following command would do (because GLUEthreshold is supplied). Alternatively you can undertake the two separately (leave out: `'GLUEthreshold', 0.5, 'GLUEbins', false, 'effectiveZero', 1e-3`). We suggest you start this analysis with a relatively small number of realisations – say 10,000. There are various options (related to what type of level spacing and how many levels you use) that we have covered in lectures and tutorials and that you will need to make choices about.

```
[MCparameters, performance, ~, ~, GLUE] = ...
    HBV_RegionalSensivityAnalysis(Params_best, Params_LowerBound, ...
    Params_UpperBound, sevensCreekClimate, 1980, 2015, ...
    doMonthly, realisations, sevensCreek_mm_day, parameterNames, 8, ...
    'groupingType','logarithmic', 'GLUEthreshold', 0.5, 'GLUEbins', ...
    false, 'keepTimeseries', false, 'effectiveZero', 1e-3);
```

Note that daily flow observations should always be used here – they are converted to monthly inside the code if need be.

Note that the output GLUE includes all the outputs from the GLUE code, including the behavioural runs.



Part 3 - Parameter uncertainty analysis

The following code does a GLUE analysis on pre-existing model runs, for example, as generated by RSA. For inputs you need the parameters names (same as for analysis the generated the original runs), the matrix of parameter values from the original runs, the performance vector from the original runs, the simulated flows from the original runs, a threshold value that you can vary to see its effect and the observed flows. Note if the GLUE output from part 2 is available, it contains all these inputs except *threshold* and *observed*. If you set a generous threshold in part 2, you can check the impact of lower threshold values quickly using this code. This is a stand-alone function.


```
[ medianPrediction, predictionCIs, parameterDistributions, ...
    behaviouralParams, behaviouralPredictions, likelihood, ...
    pcntObsAboveUpperCI, pcntObsBelowLowerCI ] = ...
    GLUE_existingRuns( ParamNames4Glue, parameterValues, ...
    performance, simulated, threshold, observed, true, 1e-3)
```

You can also do a complete GLUE analysis including running HBV to produce an analysis. To do so, use the following:

```
[ medianPrediction, predictionCIs, parameterDistributions, ...
    behaviouralParams, behaviouralPredictions, likelihood, ...
    pcntObsAboveUpperCI, pcntObsBelowLowerCI, MCparameters, ...
    performance, simulated, outDates, paramsForSimulated, ...
    sseSimulated] = HBV_GLUE_newRuns( start_year, end_year, ...
    parameters, parametersLowerBound, parametersUpperBound, ...
    parameterNames, sevensCreekClimate, sevensCreek_mm_day, ...
    doMonthly, realisations, threshold, true, 1e-3)
```

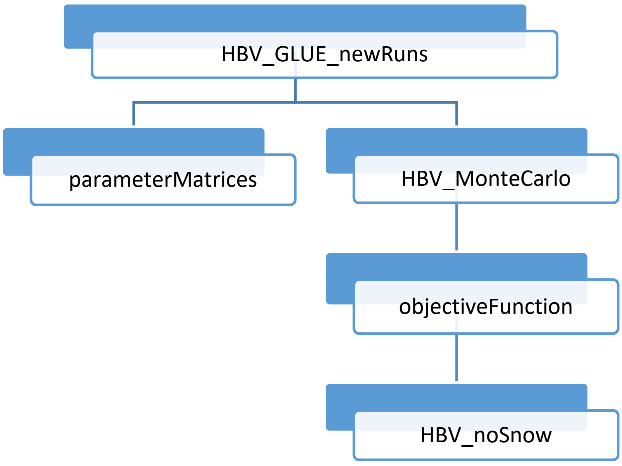
Again, these outputs could be used in GLUE_existingRuns with other threshold values. To do so the equivalent variable names are:

parameterNames === parameterNames4GLUE

behaviouralParams === parameters

performance === performance

simulated === behaviouralPredictions



Part 4 - Model Predictions

The following code can generate Monte Carlo realisations from HBV by using an ensemble of model parameters plus climate data. The output from HBV_MonteCarlo can then be input to hydroDesign to divide a capital budget between water storage and other hydroelectric infrastructure. HBV_MonteCarlo output can also be input to hydropower directly to find reliability.


```
[~, simulated, outputDates ] = HBV_MonteCarlo(behaviouralParams, ...
    Params_best(bestRun), timeStart, timeEnd, Scotts_Creek_climate, ...
    Scotts_Creek_climate(:,1:4), doMonthly, true, true);

[bestReliability, bestHydroCapacity,bestDamCapacity] = ...
        hydroDesign(budget, outputDates, simulated, demandData, 300, 361);

[~, ~, CC_reliability, ~] = hydroPower(outputDates, ...
        CC_simulated, demandData, head, catchmentArea, bestDamCapacity, ...
        bestHydroCapacity, bestDamCapacity);
```