BOMBS Functions
Version 0.1.0

DAVID GOMEZ-CABEZA

David.Gomez@ed.ac.uk

January 13, 2021

1 Model Generation

1.1 defModStruct()

Inputs: None
Function: This function allows the user to obtain the dictionary structure with all the keys for the model generation section so only the values for each key have to be filled.
Outputs: Dictionary with keys NameF, nStat, nPar, nInp, stName, parName, inpName, eqns, Y0eqs, Y0ON, tols and solver. Values for each key are empty lists [].

1.2 checkStruct(model_def)

Inputs: Dictionary with the keys specified in defModStruct() and filled values for each one.
Function: This function checks the contents of the dictionary introduced (structure from defModStruct()). If everything is correct, then some processing of the values will be done (mostly indexing and extraction of elements to ease further indexing of structures). If the value for some key has a wrong structure or content, the function will break (dictionary not returned) and a message will be printed to help the user identify where the issue is (mostly information about the key that had not passed the check).
Outputs: Same dictionary introduced with necessary modifications of some fields to aid it's use in further sections. If the value for any key is wrong, nothing will be returned.

1.3 GenerateModel(model_def)

Inputs: Dictionary with the keys specified in defModStruct() and filled values for each one.
Function: This function takes the information given by the user about the model and generates a Julia script with the necessary function to simulate the model.
The 4 functions contained in the script are:

- nameODE!(du,u,p,t): Function containing the ODEs of your model. For more information check `https://diffeq.sciml.ai/v2.0/`. du indicates the system of ODEs, u the value of the states, p parameters and t time.
- nameSteadyState(p,I): Function containing information about the steady state. This can be a set of equations or an empty function that returns the same y0 that you introduce. p is the parameter values and I the y0 values for each state (experimental values that need to be specified with the name of the state starting with exp).
- name_solvecoupledODE(ts, p, sp, inputs, ivss, pre=[]): This is the function that will allow you to solve the ODE system with different events (external inputs that change across the experiment, so no fixed parameters). ts is the time vector (from 0 to end time every 1), p is the vector of parameters, sp is the switching times for the external input, inputs is a matrix with the values for each external inducer for each step, ivss is the initial values for the model (y0) and pre is the concentration of the inducers for the steady state done before the start of the experiment (if any).
- name_SolveAll(ts, pD, sp, inputs, ivss, samps, pre=[]): This function allows you to simulate your ODEs using multiple instances for the parameters automatically. The inputs for the function are the same with the exception of pD (in this case this is the parameter matrix with all your theta samples to be used for the simulations) and samps (sampling time vector for the experiment, which might have a different resolution than 1).

Where name is the name you have given to the model in the key NameF.
Outputs: Same dictionary introduced with the extra key modelpath containing the path to the file generated.

2 Model Simulation

2.1 defSimulStruct()

Inputs: None
Function: This function allows the user to obtain the dictionary structure with all the keys for the model simulation section so only the values for each key have to be filled.
Outputs: Dictionary with keys Nexp, finalTime, switchT, y0, preInd, uInd, theta, tsamps, plot and flag. Values for each key are empty lists [].

2.2 checkStructSimul(model_def, simul_def)

Inputs: Dictionary with the keys specified in defSimulStruct() and filled values for each one.
Function: This function checks the contents of the dictionary introduced (structure from defSimulStruct()). If everything is correct, then some processing of the values will be done (mostly indexing and extraction of elements to ease further indexing of structures). If the value for some key has a wrong structure or content, the function will break (dictionary not returned) and a message will be printed to help the user identify where the issue is (mostly information about the key that had not passed the check).
Outputs: Same dictionary introduced with necessary modifications of some fields to aid it's use in further sections. If the value for any key is wrong, nothing will be returned.

2.3 fileStructInfo()

Inputs: None
Function: This function prints in the console information about the structure the CSV files need to have in order to be given to BOMBS so experimental details are extracted from it.
Outputs: None

2.4 defSimulStructFiles()

Inputs: None
Function: This function allows the user to obtain the dictionary structure with all the keys for the model simulation section (if experimental details are given with CSV files) so only the values for each key have to be filled.

Outputs: Dictionary with keys ObservablesFile, EventInputsFile, theta, MainDir, plot, flag. Values for each key are empty lists [].

2.5 extractSimulCSV(model_def, simul_def)

Inputs: Dictionary with the keys specified in defSimulStructFiles() and filled values for each one.
Function: This function checks that the CSV files introduced exist and if so, proceeds to extract all the necessary information to populate the values of the dictionary structure defined in defSimulStruct.
Outputs: Dictionary with the structure defined in defSimulStruct where all the values for each key have been extracted from the CSV files.

2.6 plotSimsODE(simuls,model_def,simul_def)

Inputs: Simulation results, model definition and simulation definition dictionaries.
Function: This function generates and saves (in the results directory) the plots for the simulations done by the user. A separate plot for each experiment will be generated, where there will be a subplot for each state and each inducer of the system.
Outputs: None

2.7 simulateODEs(model_def, simul_def)

Inputs: Model definition and simulation definition dictionaries.
Function: This is the main function of the section, which takes all the information from the model and simulation structures, simulates the ODEs system, saves the simulation results in the results folder and generates the plots if the user has selected the option.
Outputs: Simulation results dictionary plus model definition and simulation definition dictionaries with the savepath key pointing to the saved files. The simulation definition dictionary has the path and file name splint in savepath and savename.

3 Pseudo-Data Generation

3.1 defPseudoDatStruct()

Inputs: None
Function:
Outputs:

3.2 checkStructPseudoDat(model_def, pseudo_def)

Inputs:
Function:
Outputs:

3.3 defPseudoDatStructFiles()

Inputs: None
Function:
Outputs:

3.4 extractPseudoDatCSV(model_def, pseudo_def)

Inputs:
Function:
Outputs:

3.5 PDatCSVGen(pseudo_res,model_def,pseudo_def)

Inputs:
Function:
Outputs:

3.6 plotPseudoDatODE(pseudo_res,model_def,pseudo_def)

Inputs:
Function:
Outputs:

3.7 GenPseudoDat(model_def, pseudo_def)

Inputs:
Function:
Outputs:

4 Maximum Likelihood Estimation

4.1 defMLEStruct()

Inputs: None
Function:
Outputs:

4.2 SimToMle(mle_def, simul_def)

Inputs:
Function:
Outputs:

4.3 checkStructMLE(model_def, mle_def)

Inputs:
Function:
Outputs:

4.4 selectObsSim_te(simul, Obs, stName)

Inputs:
Function:
Outputs:

4.5 restructInputs_te(model_def, mle_def, expp)

Inputs:
Function:
Outputs:

4.6 UVloglike(dats, mes, errs)

Inputs:
Function:
Outputs:

4.7 MVloglike(dats, mes, errs)

Inputs:
Function:
Outputs:

4.8 plotMLEResults(mle_res,model_def,mle_def)

Inputs:
Function:
Outputs:

4.9 defCrossValMLEStruct()

Inputs: None
Function:
Outputs:

4.10 checkStructCrossValMLE(model_def, cvmle_def)

Inputs:
Function:
Outputs:

4.11 plotCrossValMLEResults(cvmle_res,model_def,cvmle_def)

Inputs:
Function:
Outputs:

4.12 CrossValMLE(model_def, cvmle_def)

Inputs:
Function:
Outputs:

4.13 finishMLEres(mle_res, model_def, mle_def)

Inputs:
Function:
Outputs:

4.14 MLEtheta(model_def, mle_def)

Inputs:
Function:
Outputs:

5 Stan Inference of Parameters

5.1 defBayInfStruct()

Inputs: None
Function:
Outputs:

5.2 defBayInfDataStruct()

Inputs: None
Function:
Outputs:

5.3 defBayInfDataFromFilesStruct()

Inputs: None
Function:
Outputs:

5.4 defBasicStanSettingsStruct()

Inputs: None
Function:
Outputs:

5.5 convertBoundTo2(x, bo, up)

Inputs:
Function:
Outputs:

5.6 fitPriorSamps(priorsamps, model_def)

Inputs:
Function:
Outputs:

5.7 fitPriorSampsMultiNorm(priorsamps, model_def)

Inputs:
Function:
Outputs:

5.8 checkStructBayInf(model_def, bayinf_def)

Inputs:
Function:
Outputs:

5.9 checkStructBayInfData(model_def, data_def)

Inputs:
Function:
Outputs:

5.10 checkStructBayInfDataFiles(model_def, data_def)

Inputs:
Function:
Outputs:

5.11 checkStructBayInfStanSettings(model_def, stan_def)

Inputs:
Function:
Outputs:

5.12 genStanInitDict(samps, names, chains)

Inputs:
Function:
Outputs:

5.13 reparamDictStan(standict, bayinf_def)

Inputs:
Function:
Outputs:

5.14 genStanModel(model_def, bayinf_def)

Inputs:
Function:
Outputs:

5.15 restructureDataInference(model_def, bayinf_def)

Inputs:
Function:
Outputs:

5.16 getStanInferenceElements(model_def, bayinf_def)

Inputs:
Function:
Outputs:

5.17 saveStanResults(rc, chns, cnames, model_def, bayinf_def)

Inputs:
Function:
Outputs:

5.18 runStanInference(model_def, bayinf_def)

Inputs:
Function:
Outputs:

5.19 plotStanResults(staninf_res, model_def, bayinf_def)

Inputs:
Function:
Outputs:

5.20 StanInfer(model_def, bayinf_def)

Inputs:
Function:
Outputs:

6 Entropy Approximation

6.1 genSamplesPrior(model_def, bayinf_def, nsamps, mu,coo)

Inputs:
Function:
Outputs:

6.2 H_Upper(w,E)

Inputs:
Function:
Outputs:

6.3 mvGauss(x, MU, E)

Inputs:
Function:
Outputs:

6.4 H_Lower(w, E, MU)

Inputs:
Function:
Outputs:

6.5 GaussMix(x, MU, E, w)

Inputs:
Function:
Outputs:

6.6 ZOTSE(MU, E, w)

Inputs:
Function:
Outputs:

6.7 GaussMix2(x)

Inputs:
Function:
Outputs:

6.8 FMix(x, MU, E, w)

Inputs:
Function:
Outputs:

6.9 SOTSE(MU, E, w)

Inputs:
Function:
Outputs:

6.10 computeH(sampl, model_def, tag)

Inputs:
Function:
Outputs:

6.11 computeHgain(prior, posterior, model_def, tag)

Inputs:
Function:
Outputs:

7 Optimal Experimental Design for Model Selection

7.1 defODEModelSelectStruct()

Inputs: None
Function:
Outputs:

7.2 checkStructOEDMS(oedms_def)

Inputs:
Function:
Outputs:

7.3 BhattacharyyaDist(mu1, mu2, sd1, sd2)

Inputs:
Function:
Outputs:

7.4 EuclideanDist(sm1, sm2)

Inputs:
Function:
Outputs:

7.5 genOptimMSFuncts(oedms_def)

Inputs:
Function:
Outputs:

7.6 plotOEDMSResults(oedms_res, oedms_def)

Inputs:
Function:
Outputs:

7.7 settingsBayesOpt(oedms_def)

Inputs:
Function:
Outputs:

7.8 mainOEDMS(oedms_def)

Inputs:
Function:
Outputs:

8 Optimal Experimental Design for Model Calibration

8.1 defODEModelCalibrStruct()

Inputs: None
Function:
Outputs:

8.2 checkStructOEDMC(oedmc_def)

Inputs:
Function:
Outputs:

8.3 genOptimMCFuncts(oedmc_def)

Inputs:
Function:
Outputs:

8.4 plotOEDMCResults(oedmc_res, oedmc_def)

Inputs:
Function:
Outputs:

8.5 settingsBayesOptMC(oedmc_def)

Inputs:
Function:
Outputs:

8.6 mainOEDMC(oedmc_def)

Inputs:
Function:
Outputs:

9 Others

9.1 infoAll(woo)

Inputs:
Function:
Outputs: None

9.2 printLogo()

Inputs: None
Function:
Outputs: None

9.3 versionBOMBS()

Inputs: None
Function:
Outputs: None