1. Purpose

3-PG2Py is a pure Python version of 3-PG2 to facilitate its extension and application to broader communities. Except the single plot simulation same as the original 3-PG2, 3-PG2Py includes two global sensitivity analyses algorithms, i.e., the variance-based sensitivity analysis method and Fourier amplitude sensitivity test, and the state-parameter estimation using ensemble Kalman filter algorithm. Additionally, an interface for spatial simulation was also implemented. 3-PG2Py is compatible with Python2.7+. With 3-PG2Py, the users could adapt the model easily to more diversified applications, especially computationally intensive ones.

2. Usage

usage: python main.py -m <running mode> -a <end age the simulation>

Arguments:

-m: running mode and could be one of ['r', 'spatial', 'VB', 'FAST', 'EnKF'], in which 'r' means single plot simulation, 'spatial' means the spatial simulation, 'VB' means the variance-based GSA, 'FAST' means the FAST GSA, and 'EnKF' means the state-parameter estimation using ensemble Kalman Filter.
-a: end age of the simulation.

Optional arguments:

-h: help information.

Below we provide some examples to illustrate the usages of the main modules in 3-PG2Py.

2.1 Single plot simulation

The single plot simulation is essentially same as the original 3-PG2. The "3PGxl_Parameters.xls" file is kept and totally same as 3-PG2 to let user change parameter values, including the standard 3PG and site-specific parameters, and the meteorological data could be provided in the sheet named as "Silo metdata". The sites under simulation could be listed in sheet named "SiteSeries" in column order. Different with 3-PG2, the outputs of 3-PG2Py are written to a SQLite database, named as "Py3PG2Output.sqlite" (the default name could be changed in file "globalVariables.py"), in which each plot corresponds to one table where the values of state variables at each time step are stored.

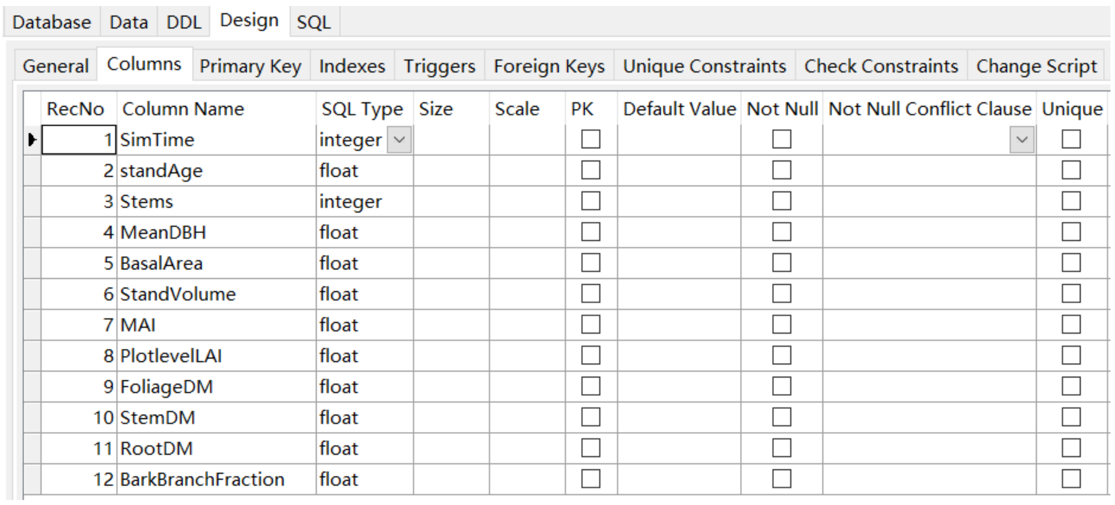
The output state variables of 3-PG2Py are defined in the "writeOutput()" procedure in file "main_module.py", and the user could modify the following codes to include/exclude particular state variables:

```
# create table
if dbWriteFlag == 0:
    sqlClause = "CREATE TABLE " + tableName + " (" + \
    "SimTime integer," + \
    "standAge float," + \
    "Stems integer," + \
    "MeanDBH float," + \
    "BasalArea float," + \
    "StandVolume float," + \
    "MAI float," + \
    "PlotlevelLAI float," + \
    "FoliageDM float," + \
    "StemDM float," + \
    "RootDM float," + \
    "BarkBranchFraction float"
    ");"

self.cursor.execute(sqlClause)

# insert records
if dbWriteFlag == 1:
    sqlClause = "INSERT INTO " + tableName + " values(" + \
    str(self.monthIncrease) + "," + \
    str(self.site.standAge) + "," + \
    str(self.site.StemNo) + "," + \
    str(self.site.avDBH) + "," + \
    str(self.site.BasArea) + "," + \
    str(self.site.StandVol) + "," + \
    str(self.site.MAI) + "," + \
    str(self.site.LAI) + "," + \
    str(self.site.WF) + "," + \
    str(self.site.WS) + "," + \
    str(self.site.WR) + "," + \
    str(self.site.fracBB) + \
    ");"
```

The corresponding table structure in SQLite is as follows (as shown by SQLite Expert Personal 5.3):



Database | Data | DDL | Design | SQL

General | Columns | Primary Key | Indexes | Triggers | Foreign Keys | Unique Constraints | Check Constraints | Change Script

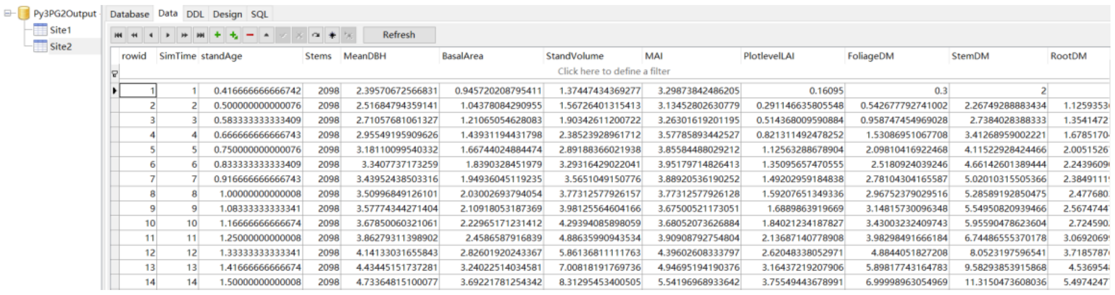
RecNo	Column Name	SQL Type	Size	Scale	PK	Default Value	Not Null	Not Null Conflict Clause	Unique
1	SimTime	integer			☐		☐		☐
2	standAge	float			☐		☐		☐
3	Stems	integer			☐		☐		☐
4	MeanDBH	float			☐		☐		☐
5	BasalArea	float			☐		☐		☐
6	StandVolume	float			☐		☐		☐
7	MAI	float			☐		☐		☐
8	PlotlevelLAI	float			☐		☐		☐
9	FoliageDM	float			☐		☐		☐
10	StemDM	float			☐		☐		☐
11	RootDM	float			☐		☐		☐
12	BarkBranchFraction	float			☐		☐		☐

The user could refer Sands (2004) Almeida et al. (2007) for more information about the model structure and parameter definitions.

To start a single plot simulation, the user could run the model as follows (for example we would like to simulate the growth of one plot till 50 years old):

```
python main.py -m r -a 50
```

The simulations for all the plots will be executed sequentially, and the users could then check the output SQLite database to see the simulated state variables' values (note that in this case two plots are simulated as listed in the left panel):



Py3PG2Output
Site1
Site2

Database | Data | DDL | Design | SQL

Refresh

Click here to define a filter

rowid	SimTime	standAge	Stems	MeanDBH	BasalArea	StandVolume	MAI	PlotlevelLAI	FoliageDM	StemDM	RootDM
1	1	0.416666666666742	2098	2.39570672566831	0.945720208795411	1.37447434369277	3.29873842486205	0.16095	0.3	2	
2	2	0.500000000000076	2098	2.51684794359141	1.04378084290955	1.56726401315413	3.13452802630779	0.291146635805548	0.542677792741002	2.26749288883434	1.1259353
3	3	0.583333333333409	2098	2.71057681061327	1.21065054628083	1.90342611200722	3.26301619201195	0.514368009590884	0.958747454969028	2.7384028388333	1.3541472
4	4	0.666666666666743	2098	2.95549195909626	1.43931194431798	2.38523928961712	3.57785893442527	0.821311492478252	1.53086951067708	3.41268959002221	1.6785170
5	5	0.750000000000076	2098	3.18110099540332	1.66744024884474	2.89188366021938	3.85584488029212	1.12563288678904	2.09810416922468	4.11522928424466	2.0051526
6	6	0.833333333333409	2098	3.3407737173259	1.8390328451979	3.29316429022041	3.95179714826413	1.35095657470555	2.5180924039246	4.66142601389444	2.2439609
7	7	0.916666666666743	2098	3.43952438503316	1.94936045119235	3.5651049150776	3.88920536190252	1.49202959184838	2.78104304165587	5.02010315505366	2.3849111
8	8	1.00000000000008	2098	3.50996849126101	2.03002693794054	3.77312577926157	3.77312577926128	1.59207651349336	2.96752379029516	5.28589192850475	2.477680
9	9	1.08333333333341	2098	3.57774344271404	2.10918053187369	3.98125564604166	3.67500521173051	1.6889863919669	3.14815730096348	5.54950820939466	2.5674744
10	10	1.16666666666674	2098	3.67850060321061	2.22965171231412	4.29394085898059	3.68052073626884	1.84021234187827	3.43003232409743	5.95590478623604	2.724590
11	11	1.25000000000008	2098	3.86279311398902	2.4586587916839	4.88635990943534	3.90908792754804	2.13687140778908	3.98298491666184	6.74486555370178	3.0692069
12	12	1.33333333333341	2098	4.14133031655843	2.82601920243367	5.86136811111763	4.39602608333797	2.62048338052971	4.8844051827208	8.0523197596541	3.7185787
13	13	1.41666666666674	2098	4.43445151737281	3.24022514034581	7.00818191769736	4.94695194190376	3.16437219207906	5.89817743164783	9.58293853915868	4.536954
14	14	1.50000000000008	2098	4.73364815100077	3.69221781254342	8.31295453400505	5.54196968933642	3.75549443678991	6.99998963054969	11.3150473608036	5.4974247

2.2 Global sensitivity analysis

Two popular GSA methods, i.e. the Fourier amplitude sensitivity analysis test (FAST) and variance-based (VB) method (Saltelli et al., 2010; Xu et al., 2011; Song et al., 2012; Song et al., 2013), are implemented in 3-PG2Py. The user could start VB or FAST by running (50 means, for example, the end age of the simulation):

```
python main.py -m VB -a 50
```

or

```
python main.py -m FAST -a 50
```

In VB, the algorithm exports the first-order effect and total effect (which includes the interactions with other parameters) on the variance of a specific model state variable in files “si_SiteName.txt” and “sti_SiteName.txt” (“SiteName” refers to the plot name given in file “3PGxl_Parameters.xls” and will be replaced during model running), respectively.

To calculate si and sti, three intermediate variables, i.e., yA, yB, and yC to store the values of a specific state variable, e.g., WS or LAI, need to be calculated. The user could modify the state variable under sensitivity analysis in file “main.py” in Lines 294 (yA), 297 (yB) , and 307 (yC), respectively.

Line	Note: The state variable could be avDBH, BasArea, LAI, StandVol, WR, WF, WS, ET, Transp, fASW, etc.
294	yA[index_N] = plot.site.WS
297	yB[index_N] = plot.site.WS
307	yCi[index_N] = plot.site.WS

The parameters list under sensitivity analysis could be modified by changing the variable “sa_parameters” in procedure “varianceBasedSA()” in file “main.py”. For example:

```
sa_parameters = ['alphaCx', 'MaxCond']
```

Similar to VB, the parameter list and state variable could to be modified by the user if necessary in the “FAST()” procedure in file “main.py”. The parameter list is defined by the variable “input_parameters”:

```
input_parameters = ['SLA1', 'nWS']
```

The state variable, e.g. LAI, under analysis is stored in the variable “Y”:

```
Y.append(plot.site.LAI)
```

The output file of FAST is named, for example, for a simulation ended at 30 years old, as “fast_30.txt”. In this file, the sensitivity indices of each parameter as listed in the variable “input_parameters” are stored by column order. For example:

Id	**nWS**	**alphaCx**	**MaxCond**
1	0.158483	0.075613	0.122098
2	0.168321	0.104246	0.118617
3	0.183828	0.085072	0.098257
4	0.147493	0.074513	0.118551
5	0.209482	0.097323	0.101655
6	0.177605	0.093194	0.100129
7	0.18335	0.075078	0.107193
…	…	…	…

FAST could be executed by multiple times to get the uncertainty of the sensitivity indices, and the user could adjust it by modifying the following code in file “main.py”:

Line	Code
489	for times in range(0, **30**): …

2.3 Ensemble Kalman filter (EnKF)

The EnKF algorithm, proposed by Moradkhani et al. (2005), implemented in 3-PG2Py is for data assimilation, including parameter calibration, purposes. The user could start EnKF by running (for example we would like to do the data assimilation till 50 years old of a specific plot):

```
python main.py -m EnKF -a 50
```

The code of EnKF is distributed in files “main.py”, “EnKF.py”, and “main_module_EnKF.py”. Since EnKF involves data exchange between multiple instances, the Observer design pattern is applied, and the same framework is applied in the spatial simulation module. To be specific, the instance of class “dualStateEnKF” in file “main.py” contains reference to each ensemble (by the “dualStateEnKF” class instance in file “EnKF.py”), and can issue events, for example, changing/collecting parameter value or running the simulation one step further in each ensemble. The file “main_module_EnKF.py” is almost a copy of file “main_module.py” by adding a procedure called “run3PG2_EnsembleVersion()” to fit ensemble simulation mode.

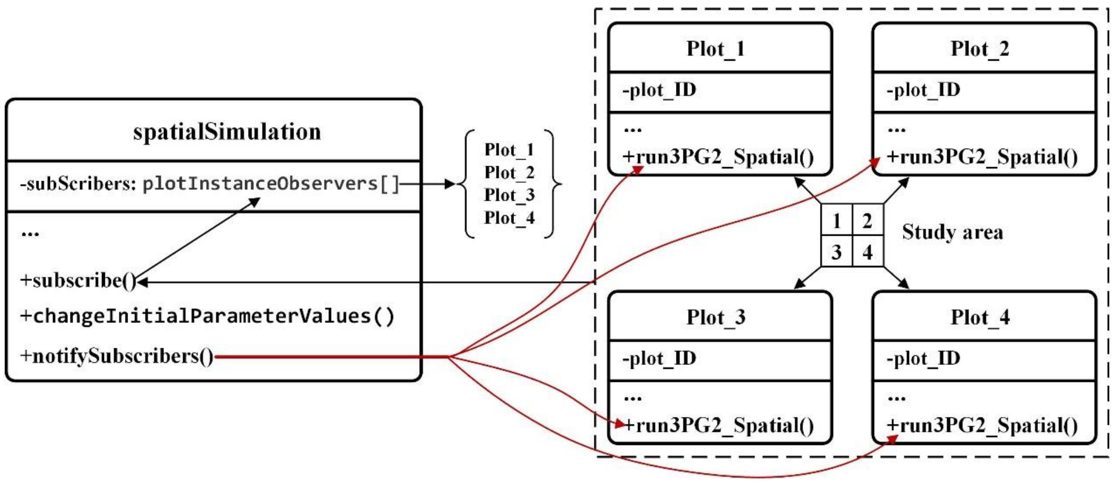
The user could change the ensemble size (default is 30), i.e. the variable “ensembleSize”, in file “main.py”. The other important variables, e.g., the parameters, forcing data factors, state variables, and observed variables (target to be assimilated), are defined by variables “parameters_enkf”, “forcingFactors_enkf”, “stateVariables_enkf”, and “observedStateVariables_enkf” respectively in class “dualStateEnKF” in file “main.py”, and the user could change them if necessary.

3-PG2Py will generate a subdirectory named as “EnKF” under the “Output” directory, if not existed previously, to store the outputs of EnKF algorithm. The outputs are a series text files named in numerical order; and the total number of output text files is defined by the variable “parameterSpaceSamplingNum” in file “main.py”, which means how many starting points are sampled in the parameter space (see Moradkhani et al., 2005). The output text file will look like the following table, in which the values of the parameters and state variables estimated by the EnKF, and the observed data are listed by column order (note that the frequency of the observed data is every half a year). By assuming the estimated parameters and state variables following a normal distribution, the uncertainty of the estimated parameters and state variables could be obtained by using all the output data (the file number is determined by the variable “parameterSpaceSamplingNum”).

Time	MaxCond	nWS	alphaCx	WF	WS	WR	StandVol	forestSW	avDBH	Observed
0	0.03	3.00	0.08	2.17	4.27	1.03	2.95	1257.55	3.91	3.69
1	0.04	3.62	0.05	3.77	6.56	1.06	4.56	1236.51	4.58	-
2	0.03	3.25	0.09	5.11	8.65	1.09	6.04	1264.12	5.11	-
3	0.03	1.56	0.10	6.59	11.09	1.12	7.79	1244.00	5.64	-
4	0.02	2.30	0.14	9.93	16.88	1.21	11.92	1205.20	6.66	-
5	0.02	3.43	0.15	18.07	32.17	1.48	22.85	1157.18	8.60	-
6	0.01	2.08	0.03	20.84	38.20	1.60	27.27	1130.17	9.25	8.63
7	0.04	4.00	0.08	28.61	55.79	2.10	40.02	1077.33	10.66	-
8	0.04	3.95	0.06	31.82	63.88	2.50	46.06	1035.14	11.25	-
9	0.02	4.57	0.09	34.94	72.05	3.21	52.20	1013.84	11.80	-
10	0.02	3.83	0.01	35.02	72.83	3.26	53.01	998.64	11.85	-
11	0.02	3.58	0.04	35.45	74.52	3.48	54.50	981.23	11.95	-
12	0.02	3.63	0.10	35.94	76.38	3.83	56.11	967.36	12.08	11.85

2.4 Spatial simulation

3-PG2Py is a stand level model and does not support spatial simulation at current stage. An interface suitable for spatial simulation is implemented in current version of 3-PG2Py and could be used and extended for those who interested in spatial simulation. The Observer design pattern, a typical solution to propagate events to all members, e.g. pixels, in an ensemble in object-oriented programming, is applied in the module (same as EnKF), in which the "spatialSimulation" object contains reference to each plot/pixel (see figure below). The "spatialSimulation" object could issue events, for example, changing parameter value ("changeInitialParameterValues()") or running the simulation one step further by calling "notifySubscribers()", to all plots/pixels. The spatial scale of a single pixel is up to the user and could be determined by both of application requirement and data availability.



The user could start spatial simulation by running (for example we would like to do the data assimilation till 50 years old of a specific plot):

```
python main.py -m spatial -a 50
```

The code of the spatial simulation is distributed in files “main.py”, “Spatial.py”, and “main_module_spatial.py”. In file “main.py”, the class “spatialSimulationDemo” is defined. By using the class “spatialSimulation” defined in “Spatial.py”, the user could register, i.e. subscribing, any number of plots/pixels by using the “subscribe()” method in class “spatialSimulation”. The plots/pixels number is defined by the variable “pixelNumber” in the class “spatialSimulationDemo”. The values of the parameters, including the forcing data, of each plot/pixel could be changed by calling the “changeInitialParameterValues()” procedure in class “spatialSimulation”, and this is the place where the actual users of spatial simulation should extend. The simulation of each plot/pixel is performed by the procedure “run3PG2_Spatial()” in file “main_module_spatial.py”.

3. Installation

3-PG2Py is compatible with Python 2.7+.

4. Author and contact

Yu Song (songyu@hznu.edu.cn), Xiaodong Song (xdsongy@gmail.com)

5. Citation

Yu Song (2021). Introducing 3-PG2Py, an open-source forest growth model in Python. Environmental Modelling and Software (in review).

6. References

Almeida, A. C., Paul, K. I., Siggins, A., Sands, P. J., Polglase, P., Marcar, N. E., Jovanovic, T., Theiveyanathan, S., Crawford, D. F., England, J. R., Falkiner, R., C., H., White, D., 2007. Development, calibration and validation of the forest growth model 3-PG with an improved water balance. CSIRO, Client Report No. 1786.

Moradkhani, H., Sorooshian, S., Gupta, H. V., Houser, P. R., 2005. Dual state-parameter estimation of hydrological models using ensemble Kalman filter. Advances in Water Resources 28(2), 135-147.

Saltelli, A., Annoni, P., Azzini, I., Campolongo, F., Ratto, M., Tarantola, S., 2010. Variance based sensitivity analysis of model output. Design and estimator for the total sensitivity index. Computer physics communications 181(2), 259-270.

Sands P J. Adaptation of 3-PG to novel species: guidelines for data collection and parameter assignment[J]. CRC Sustainable Production Forestry, Hobart, 2004, 34.

Song, X., Bryan, B. A., Almeida, A. C., Paul, K. I., Zhao, G., Ren, Y., 2013. Timedependent sensitivity of a process-based ecological model. Ecological Modelling 265, 114-123.

Song, X., Bryan, B. A., Paul, K. I., Zhao, G., 2012. Variance-based sensitivity analysis of a forest growth model. Ecological Modelling.

Xu, C., Gertner, G., 2011. Understanding and comparisons of different sampling approaches for the Fourier Amplitudes Sensitivity Test (FAST). Computational statistics & data analysis 55(1), 184-198.