**pyEMU**

Pyemu is a set of python modules for programmatically interacting with the PEST and PESTPP suites. It adopts much of the terminology from the PEST ecosystem, but implements the functionality in a pythonic, easy-to-use interface. pyEMU is available via github:

<https://github.com/jtwhite79/pyemu>

and can also be installed via PIP:

>>>pip install pyemu

pyEMU leverages heavily off of the scientific python ecosystem including numpy, pandas, and matplotlib. pyEMU has also made flopy an explicit dependency so that more direct linkages between pyEMU and flopy can be made, leading to some exciting capabilities (see below)

**rational for development**

The initial version of pyEMU was developed for prototyping and experimenting with linear analysis methods. While many PEST utilities existed (e.g. the PREDUNC and PREDVAR suites), the lead author was completely paralyzed by FORTRAN while trying to extend the capabilities of PREDVAR. Since that time, pyEMU have grown into an extensive library of python modules for dealing with many facets of setting up and post-processing PEST analyses, as well as support for many forms of linear analysis, Monte Carlo (and GLUE), as well as visualization of results from these analyses. In this way, pyEMU provides a reproducible, scripting interface for PEST and PESTPP analyses. pyEMU also serves as a “sand box” for prototyping new algorithms, a function that python excels at.

pyEMU has adopted the pythonic approach of making life easier for users: the functionality of pyEMU can be invoked with minimal user input by relying on default behavior wherever possible. For example, if a user does not pass a prior parameter covariance matrix to a linear analysis class, pyEMU will construct a diagonal prior parameter covariance matrix on-the-fly from the parameter bounds in the pest control file. In this way, users can begin using pyEMU quickly and, as needed, add more explicit and complex inputs.

**interoperability between pyEMU and the PEST and PESTPP suites**

pyEMU plays nicely with all of the programs of PEST and PESTPP suites; pyEMU can read and write most files from either of these suites including control files (including the new control file format with comments), binary and ASCII matrix files, PAR files, RES files just to name a few.

**Easy control file handling via the Pst class**

pyEMU has an extensively-tested class for handling pest control files. The pyEMU Pst class has attributes that are coincident with the control file sections. For example, the \* parameter data section is accessible as Pst.parameter\_data, while the \* control data section is accessible as Pst.control\_data. Furthermore, the python variable names in these sections are also commensurate with the control file quantity names.

Along with the Pst class, pyEMU offers several methods to create a new control file, ranging from a generic Pst instance with “dummy” parameter and observation names to creating control file using existing template and instruction files.

The Pst class also includes methods to write stand-alone LaTeX summary tables of the parameter and observations in the control file.

**easy linear algebra via the Matrix and Cov classes**

The Matrix and (derived) Cov classes in pyEMU are designed to make linear algebra easy. Through object oriented programming, the Matrix class overloads mathematical operators so that linear algebra equations can be quickly coded in pyEMU without users having to worry about row and column alignment­–the Matrix class with automatically and on-the-fly aligns matrices and vectors based on row and column names. For example, if one wanted to plot the singular spectrum of the normal matrix:

>>>pst = pyemu.Pst(“pest.pst”)

>>>jco = pyemu.Jco.from\_binary(“pest.jco”)

>>>oc = pyemu.Cov.from\_observation\_data(pst)

>>>xtqx = jco.T \* oc\*\*0.5 \* jco

>>>plt.plot(xtqx.s)

WAT?! How easy is that…this code will work for any PEST interface!

**easy linear analysis (FOSM) capabilities via the Schur and ErrVar classes**

pyEMU exposes two class for linear algebra: The Schur and ErrVar classes, which replicate the behavior of the PREDUNC and PREDVAR suites, respectively. The Schur class implements the Schur compliment equation for conditional covariance propagation (the equation of the PREDUNC suite). The Schur class includes functionality for parameter and forecast uncertainty estimation as well as methods for parameter and observation dataworth analyses. The ErrVar class implements subspace error variance analysis (similar to the PREDVAR suite).

**easy monte carlo and GLUE via the MonteCarlo, ParameterEnsemble and ObservationEnsemble classes**

The MonteCarlo and associated ParameterEnsemble and ObservationEnsemble classes in pyEMU facilitate preparing a Monte Carlo analysis within the PEST framework. The ParameterEnsemble class supports uniform and multivariate Gaussian draws and respects parameter transformation status and parameter bounds. The ParameterEnsemble and ObservationEnsemble classes play nicely with the PESTPP-SWP program so that users can quickly generate a parameter ensemble, evaluate the ensemble in parallel and then condition the resulting model output observation ensemble using GLUE-type measures.

Below is an example of how one could use the posterior parameter distribution (best-fit parameter values and posterior covariance matrix from Schur class) to draw 10,000 parameter realizations and save those realizations to a PESTPP-SWP compatible CSV file:

>>>pst = pyemu.Pst(“pest.pst”)

>>>pst.parrep(“pest.par”)

>>>sc = pyemu.Schur(jco=”pest.jco”)

>>> mc = pyemu.MonteCarlo(pst=pst,parcov=sc.posterior\_parameter)

>>>mc.draw(10000)

>>>mc.parensemble.to\_csv(“sweep\_in.csv”)

WAT?! How easy is that…this code will work for any PEST interface!

**easy visualization via the pyemu.plot module**

The plot module of the pyEMU provides functionality to make many common plots associated with parameter estimation and uncertainty analysis, including:

Deterministic and stochastic line-of-equality plots

Multi-component objective function pie charts

Multiple ensemble histogram plots

Linear-analysis-implied gaussian distributions

Objective function progress by iteration

Obs vs sim time series plots

Stacked identifiability bar charts

Prior Gaussian distributions implied by parameter bounds

As with other parts of pyEMU, these plotting routines are designed to work with minimal user input but offer fine-grained control through optional key-word arguments.

**geostats module** is a pure-python implementation of geostatistical methods include ordinary kriging and covariance matrix construction. The geostats module supports nested geostatistical structures and can read and write PEST-compatible structure files. The geostats module can replicate the functionality of PPk2FAC and FAC2REAL and includes functionality for reading and writing GSLIB and SGEMS file formats.

**gw\_utils module** has functionality for setting up template and instruction files related to the MODFLOW family of codes (including MT3D-USGS). Much of this functionality is used within the PstFromFlopyHelper class.

**pp\_utils module** helps users deal with setting up pilot points.

**from zero to hero: the PstFromFlopyHelper class**

Much of the recent work in pyEMU has been focused on decreasing the amount of time required to construct a PEST interface for an existing model. In many cases, the effort required to construct the interface results in practitioners waiting until the model is “ready” before undertaking the interface construction (where “ready” is when the project is late and out of money). The PstFromFlopyHelper class combines the functionality of flopy and pyEMU to rapidly setup a sophisticated PEST interface for any flopy-compatible MODFLOW family model. In this scheme, all parameters are setup as multipliers against the base model input arrays, allowing users to construct complex mixtures of zones, pilot points, karhuenen-loeve covariance eigen multipliers and grid-scale parameter for any given array-based model input. This allows users to easily and reproducibly invest parameterization decisions. The helper also includes functionality to setup spatial and temporal list-based input multipliers­–all modflow list-based boundary condition datasets are supported. The helper includes functionality for setting up observations from a variety of binary and ASCII MODFLOW output files. Most importantly, the PstFromFlopyHelper class writes the forward run script, the prior parameter covariance matrix (using geostatistic where appropriate) and the PEST control file, so that following execution, users can immediately commence with parameter estimation or Monte Carlo type uncertainty analysis.

The true power of the PstFromFlopyHelper can be realized when it is used in conjunction with PESTPP-IES. The helper can be used to quickly generate a PEST interface with hundreds of thousands of parameters, allowing users to forego the agonizing decision related to parameterization density. The resulting PEST interface is ideally suited for use with PESTPP-IES. In this situation, users can generate a high-dimensional parameter ensemble via memory efficient PstFromFlopyHelper.draw() method.

**Topics:**

Linear uncertainty analysis – basic parameter and forecast uncertainty estimation

Linear – dataworth via observation testing

Linear – dataworth via parameter testing

Linear – 2-term Error variance

Linear – 3-term error variance

Nonlinear uncertainty analysis – prior (unconstrained) monte carlo

Nonlinear – rejection sampling and GLUE

Nonlinear – preconditioning with linear posterior parameter covariance

Matrix handling: matrix I/O, slicing and dicing, Jacobian testing

Pst handling: pst I/O, slicing and dicing

Pst handling: creating a pst from scratch

Geostats: basic objects and interactions

Geostats: building covariance matrices

Geostats: factor calculation via OK

Flopy interaction: interface setup like a boss

Parameterization: pilot points setup and use

Parameterization: KL setup and use

Pest and Pest++ interaction

Helper functions

Plotting functions