Performance Monitoring using Pecos

Release 0.1

Apr 19, 2016

Contents

1 Overview

Pecos is an open-source python package designed to monitor performance of time series data, subject to a series of quality control tests. The software includes methods to run quality control tests defined by the user and generate reports which include performance metrics, test results, and graphics. The software can be customized for specific applications. Some high-level features include:

- Pecos uses Pandas DataFrames *[McKinney2013]* for time series analysis. This dependency facilitates a wide range of analysis options and date-time functionality.
- Data columns can be easily reassigned to common names through the use of a translation dictionary. Translation dictionaries also allow data columns to be grouped for analysis.
- Time filters can be used to eliminate data at specific times from quality control tests (i.e. early evening and late afternoon).
- Application specific models can be incorporated into performance monitoring tests.
- General and custom performance metrics can be saved to keep a running history of system health.
- Analysis can be setup to run on an automated schedule (i.e. Pecos can be run each day to analyze data collected on the previous day).
- HTML formatted reports can be sent via email or hosted on a website.

2 Installation

Pecos requires Python 2.7 along with several python package dependencies. Information on installing and using python can be found at https://www.python.org/. Python distributions, such as Python(x,y) and Anaconda, can also be used to manage the Python interface. These distributions include the Python packages needed for Pecos.

To install Pecos using pip:

```
pip install pecos
```

To build Pecos from source using git:

```
git clone https://github.com/kaklise/pecos
cd pecos
python setup.py install
```

Python package dependencies include:

- Pandas *[McKinney2013]*: analyze and store time series data, http://pandas.pydata.org/
- Numpy *[vanderWalt2011]*: support large, multi-dimensional arrays and matrices, http://www.numpy.org/
- Matplotlib *[Hunter2007]*: produce figures, http://matplotlib.org/

Optional python packages include:

- pyyaml: store configuration options in human readable data format, http://pyyaml.org/
- win32com: send email

All other dependencies are part of the Python Standard Library.

To use Pecos, import the package from within a python console:

```
import pecos
```

3 Simple example

A simple example is included in the examples/simple directory. This example uses data from an excel file, **simple.xlsx**, which contains 4 columns of data (A through D).

- A = elapsed time in days
- B = uniform random number between 0 and 1
- C = sin(10*A)
- D = C+(B-0.5)/2

The data includes missing timestamps, duplicate timestamps, non-monotonic timestamps, corrupt data, data out of expected range, data that doesn't change, and data that changes abruptly.

- Missing timestamp at 5:00
- Duplicate timestamp 17:00
- Non-monotonic timestamp 19:30
- Column A has the same value (0.5) from 12:00 until 14:30
- Column B is below the expected lower bound of 0 at 6:30 and above the expected upper bound of 1 at 15:30
- Column C has corrupt data (-999) between 7:30 and 9:30
- Column C does not follow the expected sine function from 13:00 until 16:15. The change is abrupt and gradually corrected.
- Column D is missing data from 17:45 until 18:15

The script, **simple_example.py** (shown below), is used to run quality control analysis using Pecos. The script performs the following steps:

- Define input for quality control tests, including
  - Expected frequency of the timestamp
  - Time filter to exclude data points early and late in the day
  - Corrupt data values
  - Upper and lower bounds for data range and data increments
  - sine wave model to compute measurement error
- Load time series data from an excel file
- Run quality control tests
- Generate an HTML report, test results CSV file, and performance metrics CSV file

```
"""
In this example, simple time series data is used to demonstrate basic functions
in pecos.
* Data is loaded from an excel file which contains four columns of values that
  follow linear, random, and sine models.
* A translation dictionary is defined to map and group the raw data into
  common names for analysis
* A time filter is established to screen out data between 3 AM and9 PM
* The data is loaded into a pecos PerformanceMonitoring class and a series of
  quality control tests are run, including range tests and increment tests
* The results are printed to csv and html reports
"""
```

```
import pecos
import pandas as pd
import matplotlib.pyplot as plt
import os
import numpy as np

# Initialize logger
pecos.logger.initialize()

# Input
system_name = 'Simple'
data_file = 'simple.xlsx'
translation_dictonary = {
    'Linear': ['A'],
    'Random': ['B'],
    'Wave': ['C','D']}
expected_frequency = 900
time_filter_min = 3*3600
time_filter_max = 21*3600
corrupt_values = [-999]
range_bounds = {
    'Random': [0, 1],
    'Wave': [-1, 1],
    'Wave Absolute Error': [None, 0.25]}
increment_bounds = {
    'Linear': [0.0001, None],
    'Random': [0.0001, None],
    'Wave': [0.0001, 0.5]}

 # Define output files and directories
results_directory = 'Results'
if not os.path.exists(results_directory):
    os.makedirs(results_directory)
results_subdirectory = os.path.join(results_directory, system_name + '_2015_01_01')
if not os.path.exists(results_subdirectory):
    os.makedirs(results_subdirectory)
metrics_file = os.path.join(results_directory, system_name + '_metrics.csv')
test_results_file = os.path.join(results_subdirectory, system_name + '_test_results.csv')
report_file =  os.path.join(results_subdirectory, system_name + '.html')

# Create an PerformanceMonitoring instance
pm = pecos.monitoring.PerformanceMonitoring()

# Populate the PerformanceMonitoring instance
df = pd.read_excel(data_file)
pm.add_dataframe(df, system_name)
pm.add_translation_dictonary(translation_dictonary, system_name)

# Check timestamp
pm.check_timestamp(expected_frequency)

# Generate time filter
clock_time = pm.get_clock_time()
time_filter = (clock_time > time_filter_min) & (clock_time < time_filter_max)
pm.add_time_filter(time_filter)

# Check missing
pm.check_missing()
```

```
# Check corrupt
pm.check_corrupt(corrupt_values)

# Add composite signals
elapsed_time= pm.get_elapsed_time()
wave_model = np.sin(10*(elapsed_time/86400))
wave_model.columns=['Wave Model']
pm.add_signal('Wave Model', wave_model)
wave_mode_abs_error = np.abs(np.subtract(pm.df[pm.trans['Wave']], wave_model))
wave_mode_abs_error.columns=['Wave Absolute Error C', 'Wave Absolute Error D']
pm.add_signal('Wave Absolute Error', wave_mode_abs_error)

# Check range
for key,value in range_bounds.items():
    pm.check_range(value, key)

# Check increment
for key,value in increment_bounds.items():
    pm.check_increment(value, key)

# Compute metrics
mask = pm.get_test_results_mask()
QCI = pecos.metrics.qci(mask, pm.tfilter)

# Create a custom graphic
plt.figure(figsize = (7.0,3.5))
ax = plt.gca()
df.plot(ax=ax, ylim=[-1.5,1.5])
plt.savefig(os.path.join(results_subdirectory, system_name+'_custom_1.jpg'))

# Write metrics, test results, and report files
pecos.io.write_metrics(metrics_file, QCI)
pecos.io.write_test_results(test_results_file, pm.test_results)
pecos.io.write_monitoring_report(report_file, results_subdirectory, pm, QCI)
```

Results are saved in examples/simple/Results. Results include:

- HTML report, **Simple_2015_01_01/Simple.html** (shown below), includes summary tables and graphics
- Test results CSV file, **Simple_2015_01_01/Simple_test_results.csv**, includes information from the summary tables
- Performance metric CSV file, **Simple_metrics.csv**, includes a quality control index based on the analysis.

4 Time series data

Pandas DataFrames store 2D data with labeled columns. Pecos uses Pandas DataFrames to store and analyze data indexed by time. Pandas includes a wide range of time series analysis and date-time functionality. To import pandas:

```
import pandas as pd
```

Pandas includes many built-in functions to read data from csv, excel, sql, etc. For example, data can be loaded from an excel file using:

```
df = pd.read_excel('data.xlsx')
```

The PerformanceMonitoring class is the base class used by Pecos to define performance monitoring analysis. To get started, an instance of the PerformanceMonitoring class is created:

```
pm = pecos.monitoring.PerformanceMonitoring()
```

The DataFrame can then be added to the PerformanceMonitoring object as follows:

```
pm.add_dataframe(df, system_name)
```

Multiple DataFrames can be added to the PerformanceMonitoring object. The 'system_name' is used to distinquish DataFrames.

DataFrames are accessed using:

```
pm.df
```

5 Translation dictionary

A translation dictionary maps database column names into common names. The translation dictionary can also be used to group columns with similar properties into a single variable.

Each entry in a translation dictionary is a key:value pair where 'key' is the common name of the data and 'value' is a list of original column names in the database. For example, {temp: [temp1,temp2]} means that columns named 'temp1' and 'temp2' in the database file are assigned to the common name 'temp' in Pecos. In the simple example, the following translation dictionary is used to rename column 'A' to 'Linear', 'B' to 'Random', and group columns 'C' and 'D' to 'Wave':

```
trans = {
  Linear: [A],
  Random: [B],
  Wave: [C,D]}
```

The translation dictionary can then be added to the PerformanceMonitoring object as follows:

```
pm.add_translation_dictonary(trans, system_name)
```

If no translation is desired (i.e. raw column names are used), a 1:1 map can be generated using the following code:

```
trans = dict(zip(df.columns, [[col] for col in df.columns]))
pm.add_translation_dictonary(trans, system_name)
```

As with DataFrames, multiple translation dictionaries can be added to the PerformanceMonitoring object, distinguished by the 'system_name'.

Keys defined in the translation dictionary can be used in quality control tests, for example:

```
pm.check_range([-1,1], 'Wave')
```

Inside Pecos, the translation dictionary is used to index into the DataFrame, for example:

```
pm.df[pm.trans['Wave']]
```

returns columns C and D from the DataFrame.

6 Time filter

A time filter is a Boolean time series that indicates if specific timestamps should be used in quality control tests. The time filter can be defined using elapsed time, clock time, or other custom algorithms. Pecos includes methods to get the elapsed and clock time of the DataFrame (in seconds).

The following example defines a time filter between 3 AM and 9 PM:

```
clock_time = pm.get_clock_time()
time_filter = (clock_time > 3*3600) & (clock_time < 21*3600)
```

The time filter can also be defined based on sun position, see **pv_example.py** in the examples/pv directory.

The time filter can then be added to the PerformanceMonitoring object as follows:

```
pm.add_time_filter(time_filter)
```

7 Quality control tests

Pecos includes several quality control tests. When a test fails, information is stored in a summary table. This information is used to create the final report. For each test, the minimum number of consecutive failures can be specified for reporting. Quality controls tests fall into five categories.

7.1 Timestamp test

The **check_timestamp** method can be used to check the time index for missing, duplicate, and non-monotonic indexes. By using Pandas DataFrames, Pecos is able to take advantage of a wide range of timestamp strings, including UTC offset. If a duplicate timestamp is found, Pecos keeps the first occurrence. If timestamps are not monotonic, the timestamps are reordered. For this reason, the timestamp should be corrected before other quality control tests are run. Input includes:

- Expected frequency of the time series in seconds
- Minimum number of consecutive failures for reporting (default = 1 timestamp)

For example:

```
pm.check_timestamp(60)
```

checks for missing, duplicate, and non-monotonic indexes assuming an expected frequency of 60 seconds.

7.2 Missing data test

The **check_missing** method can be used to check for missing values. Unlike missing timestamps, missing data only impacts a subset of data columns. NaN is included as missing. Input includes:

- Data column (default = all columns)
- Minimum number of consecutive failures for reporting (default = 1 timestamp)

For example:

```
pm.check_missing('Wave', min_failures=5)
```

checks for missing data in the columns associated with the key ‘Wave’. Warnings are only reported if there are 5 consecutive failures.

7.3 Corrupt data test

The **check_corrupt** method can be used to check for corrupt values. Input includes:

- List of corrupt values
- Data column (default = all columns)
- Minimum number of consecutive failures for reporting (default = 1 timestamp)

For example:

```
pm.check_corrupt([-999, 999])
```

checks for data with values -999 or 999 in the entire DataFrame.

7.4 Range test

The **check_range** method can be used to check that data is within an expected range. Range tests are very flexible. The test can be also be used to compare modeled vs. measured values (i.e. absolute error or relative error) or relationships between data columns (i.e. column A divided by column B). An upper bound, lower bound or both can be specified. Additionally, the data can be smoothed using a rolling mean. Input includes:

- Upper and lower bound
- Data column (default = all columns)
- Rolling mean window (default = 1 timestamp, which indicates no rolling mean)
- Minimum number of consecutive failures for reporting (default = 1 timestamp)

For example:

```
pm.check_range([None,1], 'A', rolling_window=2)
```

checks for values greater than 1 in the columns associated with the key 'A', using a rolling average of 2 time steps.

7.5 Increment test

The **check_increment** method can be used to check that the difference between consecutive data values (or other specified increment) is within an expected range. This method can be used to test if data is not changing or if the data has an abrupt change. Like the check_range method, the user can specify if the data should be smoothed using a rolling mean. Input includes:

- Upper and lower bound
- Data column (default = all columns)
- Increment used for difference calculation (default = 1 timestamp)
- Flag indicating if the absolute value is taken (default = True)
- Rolling mean window (default = 1 timestamp, which indicates no rolling mean)
- Minimum number of consecutive failures for reporting (default = 1 timestamp)

For example:

```
pm.check_increment([None, 0.000001], min_failure=60)
```

checks if value increments are greater than 0.000001 for 60 consecutive time steps:

```
pm.check_increment([-800, None], absolute_value = False)
```

checks if value increments decrease by more than -800 in a single time step.

8 Performance metrics

Pecos can be used to track performance of time series data over time. The quality control index (QCI) is a general metric which indicates the percent of the data points that passed quality control tests. Duplicate and non-monotonic indexes are not counted as failed tests (duplicates are removed and non-monotonic indexes are reordered). QCI is defined as:

$$QCI = \frac{\sum_{d\in D}\sum_{t\in T} X_{dt}}{|DT|}$$

where D is the set of data columns and T is the set of timestamps in the analysis. X_{dt} is a data point for column d time t' that passed all quality control test. $|DT|$ is the number of data points in the analysis.

A value of 1 indicates that all data passed all tests. For example, if the data consists of 10 columns and 720 times that are used in the analysis, then $|DT|$ = 7200. If 7000 data points pass all quality control tests, then the QCI is 0.972.

To compute QCI:

```
QCI = pecos.metrics.qci(pm)
```

Additional metrics can be added to the QCI dataframe and saved to a file:

```
pecos.io.write_metrics(metrics_filename, QCI)
```

If 'metrics_filename' already exists, the metrics will be appended to the file.

9 Composite signals

Composite signals are used to generate new data columns based on existing data. Composite signals can be used to add modeled data values or relationships between data columns. Data created from composite signals can be used in the quality control tests.

The following example adds 'Wave Model' data to the PerformanceMonitor object:

```
elapsed_time= pm.get_elapsed_time()
wave_model = np.sin(10*(elapsed_time/86400))
wave_model.columns=['Wave Model']
pm.add_signal('Wave Model', wave_model)
```

10 Configuration file

A configuration file can be used to store information about data and quality control tests. The configuration file is not used directly within Pecos, therefore there are no specific formatting requirements. Configuration files can be useful when using the same python script to analyze several systems that have slight differences.

The simple example includes a configuration file that defines system specifications, translation dictionary, composite signals, corrupt values, and bounds for range and increment tests.

```
Specifications:
  Frequency: 900
  Multiplier: 10

Translation:
  Linear: [A]
  Random: [B]
  Wave: [C,D]

Composite Signals:
- Wave Model: "np.sin({Multiplier}*{ELAPSED_TIME}/86400)"
- Wave Absolute Error: "np.abs(np.subtract({Wave}, {Wave Model}))"

Time Filter: "({CLOCK_TIME} > 3*3600) & ({CLOCK_TIME} < 21*3600)"

Corrupt Values: [-999]

Range Bounds:
  Random: [0, 1]
  Wave: [-1, 1]
  Wave Absolute Error: [None, 0.2]

Increment Bounds:
  Linear: [0.0001, None]
  Random: [0.0001, None]
  Wave: [0.0001, 0.5]
```

In the configuration file, composite signals and time filters can be defined using strings of python code. Numpy (and other python modules if needed) can be used for computation. **Strings of python code should be thoroughly tested by the user.** A list of key:value pairs can be used to specify the order of evaluation.

When using a string of python code, keywords in {} are expanded using the following rules in this order:

1. Keywords ELAPSED_TIME and CLOCK_TIME return time in seconds
2. Keywords that are a key in the translation dictionary return 'pm.df[pm.trans[Keyword]]'
3. Keywords that are a key in a user specified dictionary of constants, specs, return 'specs[Keyword]'.

The values in specs can be used to generate a time filter, define upper and lower bounds in quality control tests, define system latitude and longitude, or other constants needed in composite equations, for example:

```
specs = {
  'Frequency': 900,
  'Multiplier': 10,
  'Latitude': 35.054,
  'Longitude': -106.539}
```

Strings are evaluated and added to the dataframe using the following code:

```
signal = pm.evaluate_string(raw_column_name, string, specs)
pm.add_signal(trans_column_name, signal)
```

If the string evaluation fails, the error message is printed in the final report.

The script, **simple_example_using_config.py**, in the examples/simple directory uses a configuration file and string evaluation.

11 Task scheduler

To run Pecos on an automated schedule, create a task using your operating systems task scheduler. On Windows, open the Control Panel and search for *Schedule Tasks*. The task can be set to run at a specified time and the action can be set to run a batch file (.bat or .cmd file name extension), which calls a python driver script. For example, the following batch file runs driver.py:

```
cd your_working_directory
C:\Python27\python.exe driver.py
```

12 Results

When a test fails, information is stored in:

```
pm.test_results
```

Test results includes the following information:

- System Name: System name associated with the data file
- Variable Name: Column name in the data file
- Start Date: Start time of the failure
- End Date: : End time of the failure
- Timesteps: The number of consecutive timesteps involved in the failure
- Error Flag: Error messages include:
  - Duplicate timestamp
  - Nonmonotonic timestamp
  - Missing data (used for missing data and missing timestamp)
  - Corrupt data
  - Data > upper bound, value
  - Data < lower bound, value
  - Increment > upper bound, value
  - Increment < lower bound, value

Pecos can be used to generate an HTML report, test results CSV file, and a performance metrics CSV File.

12.1 Monitoring report

The monitoring report include the start and end time for analysis, custom graphics and performance metrics, a table that includes test results, graphics associated with the test results (highlighting data points that failed a quality control tests), notes on runtime errors and warnings, and (optionally) the configuration options used in the analysis.

- **Custom Graphics:** Custom graphics can be created for a specific applications. These graphics are included at the top of the report. Any graphic with the name *custom* in the subdirectory are included in the custom graphics section. By default, no custom graphics are generated.
- **Performance Metrics:** Performance metrics are displayed in a table.

- **Test Results** Test results contain information stored in pm.test_results Graphics follow that display the data point(s) that caused the failure.
- **Notes:** Notes include Pecos runtime errors and warnings. Notes include:
  - Empty/missing database
  - Formatting error in the translation dictionary
  - Insufficient data for a specific quality control test
  - Insufficient data or error when evaluating string
- **Configuration Options:** Optional. Configuration options used in the analysis.

The following method can be used to write a monitoring report:

```
pecos.io.write_monitoring_report()
```

12.2 Test results

The test results CSV file contains information stored in pm.test_results.

The following method can be used to write test results:

```
pecos.io.write_test_results()
```

12.3 Performance metrics

The performance metrics CSV file contains metrics. Values are appended each time an analysis is run.

The following method can be used to write metrics to a file:

```
pecos.io.write_metrics()
```

12.4 Dashboard

To compare performance of several systems, key graphics and metrics can be gathered in a dashboard view. For example, the dashboard can contain multiple rows (one for each system) and multiple columns (one for each location). The dashboard can be linked to specific monitoring reports for more detailed information.

For each row and column in the dashboard, the following information can be specified

- Text (i.e. general information about the system/location)
- Graphics (i.e. a list of custom graphics)
- Table (i.e. a Pandas DataFrame with performance metrics)
- Link (i.e. the path to monitoring report for detailed information)

The following method can be used to write a monitoring report:

```
pecos.io.write_dashboard()
```

Pecos includes a dashboard example, **dashboard_example.py**, in the examples/dashboard directory.

13 Custom applications

Pecos can be customized for specific applications. Python scripts can be added to initialize data and add application specific models. Additional quality control tests can be added by inheriting from the PerformanceMonitoring class.

13.1 PV system monitoring

For PV systems, the translation dictionary can be used to group data according to the system architecture, which can include multiple strings and modules. The time filter can be defined based on sun position and system location. The data objects used in Pecos are compatible with pvlib, which can be used to model PV systems *[Stein2016]* (https://github.com/pvlib/pvlib-python). Pecos also includes a function to read Campbell Scientific ascii file format and functions to compute pv performance metrics (i.e. performance ratio, clearness index).

Pecos includes a PV system example, **pv_example.py**, in the examples/pv directory. The example uses graphics functions in **pv_graphics.py**.

13.2 Performance metrics

The performance metrics file, created by Pecos, can be used in additional analysis to track system health over time.

Pecos includes a performance metrics example (based on PV metrics), **metrics_example.py**, in the examples/metrics directory.

14 Copyright and license

15 References

References

[Hunter2007] John D. Hunter. Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, 9, 90-95 (2007), DOI:10.1109/MCSE.2007.55

[McKinney2013] McKinney W. (2013) Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython

[Stein2016] J.S. Stein, W.F. Holmgren, J. Forbess, and C.W. Hansen, PVLIB: Open Source Photovoltaic Performance Modeling Functions for Matlab and Python, in 43rd Photovoltaic Specialists Conference, 2016

[vanderWalt2011] Stefan van der Walt, S. Chris Colbert and Gael Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering, 13, 22-30 (2011), DOI:10.1109/MCSE.2011.37