Python-based Performance Analysis Supporting System (PyPASS): A software tool set for the performance analysis of regulatory photochemical air quality modeling

Byeong-Uk Kim and Harvey E. Jeffries*
Department of Environmental Sciences and Engineering, University of North Carolina, Chapel Hill, NC, USA

1. INTRODUCTION

The Clean Air Act requires Regulatory photochemical modeling (RPAQM) as part of ozone control policy development processes. Standard guidance on RPAQM requires performance evaluation before applying the models to environmental decision making processes. The primary goal of model performance evaluation is to judge whether a model prediction of air quality is acceptable for its application.

Performance evaluation of RPAQM requires large amounts of information to be processed through statistical assessment and graphical analysis. During a model performance evaluation process, the information production task can be a bottle-neck for the whole process. Frequently, the performance evaluation for regulatory modeling is constrained by regulatory timelines so that saving time in generating information steps can be very important because evaluators can spend more time on the performance analysis itself. Also, there are cases where some information is not available because of the absence of proper data retrieval and/or information generating tools. Missing information may result in flaws in judgment of a model's adequacy for its policy application. To conduct a more comprehensive performance evaluation within the time allowed, methods for gathering as much information as fast as possible can be critical.

In this presentation, we introduce a new performance evaluation supporting tool, PyPASS. PyPASS is a software package composed of three major modules mostly written in Python. Each module is specialized in retrieving, processing, and generating information relevant for regulatory photochemical air quality modeling. In the following section, we present the overview of PyPASS and several examples of PyPASS applications. We discuss the performance of PyPASS compared with the existing tools, its known issues, and our plan for improvement.

*Corresponding author: Harvey E. Jeffries, Department of Environmental Sciences and Engineering, UNC-Chapel Hill, 120 Rosenau Hall, CB#7431, Chapel Hill, NC 27599-7431; e-mail: harvey@unc.edu

2. PyPASS OVERVIEW

2.1 Design goals and implementation

At the planning phase, we set several design goals to guide the development of this 'new tool'. First, the new tool should perform fast information generation. Second, it should provide information that may not be directly available through existing tools. Third, it has to be publicly-available with no-cost or very low-cost to acquire and use it. Last, it must be flexible and extensible to accommodate future changes of data formats and needs for new analysis measures.

To achieve these goals, we made several choices for the 'new tool' development and implementation. First, we selected the command-line as the user interface because batch-operations are more suitable than interactive operations to generate large sets of information efficiently. Second, we made the new tool generate 'pre-defined' graphics and we added some graphical measures that were not frequently used or unavailable in the past evaluation tasks. In other words, we have set the type of graphics we will make prior to information production, but we provide many options with regard to styles and formats of the graphics. Last, we have selected Python, Python Software Foundation (2005), as the main programming language and named our new tool as 'PyPASS' following a common Python user community's naming convention. Python is a scriptable and object-oriented language that takes advantages of scripting and object-orientation, e.g. fast code developing cycle and enhanced software reusability. Python is free and has large sets of supporting libraries for scientific computing and visualization. Details of external libraries and applications used for PyPASS are presented in the following section.

2.2 User interface and structure

In object-oriented design, a Use Case diagram, which is a part of Unified Modeling Language, is the method used to describe how users will typically interact with the application, Object Management Group (2005). This diagram gives an overview about what the system will do for users. Fig. 1. shows the Use Case diagram of PyPASS. Briefly, the symbol representing a human is called 'actor' describing a role of the system users. Ovals represent functionalities provided by PyPASS. Ovals linked to other ovals with arrow lines are representing specific functionalities derived from the functions exposed to users. The actual outcomes of PyPASS are made by functionalities described in the oval at the end of each branch. For details of each component in the diagram, we encourage users to refer to Folwer and Scott (1997).

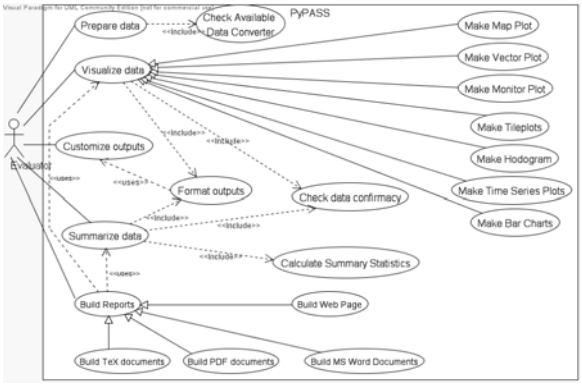

Fig. 1. Use Case Diagram of PyPASS. In this diagram, users, i.e. evaluators, mainly interact with five abstract-level functions provided by PyPASS; Prepare Data, Visualize Data, Customize Outputs, Summarize Data, and Build Reports.

To provide these functionalities, PyPASS utilizes many existing Python libraries contributed by other Python application developers. Table 1. shows all the libraries and applications necessary to run PyPASS. The table also shows how to acquire these software libraries/packages. All but one, are free. For the visualization library, we have chosen ChartDirector, Advanced Software Engineering Limited (2005). Even though ChartDirector is a commercial product, it costs relatively little while it has high quality graphics, object-orientation support, and cross-platform availability. Moreover, under a specific condition, we can provide PyPASS with ChartDirector libraries free to users.

Table 1. List of software libraries and applications necessary to run PyPASS. Note that each library listed in this table may require other libraries. For example, PyTables requires HDF5 library (refer to http://hdf.ncsa.uiuc.edu/HDF5/)

Library/Application Name (Version), Source
CAMxSubset/CMAQSubset (6.0), upon request to bukim@email.unc.edu
ChartDirector (4.0), http://www.advsofteng.com/download.html
mxDateTime (2.0.6), http://www.egenix.com/files/python/mxDateTime.html
numarray (1.3.3), http://www.stsci.edu/resources/software_hardware/numarray
PyTables (1.1), http://pytables.sourceforge.net/html/GetIt.html
Proj4 (4.4.6), http://www.remotesensing.org/proj/
pyRXP (1.0.7), http://www.reportlab.org/
ReportLab (1.19), http://www.reportlab.org/

Based on these external libraries, we wrote three major sets of PyPASS components. The first set is for processing air quality model files and configuration data. The second set is for visualizing and documenting all the information including sliced air quality model output data and stationary monitor data. The third set is to support the main functions of PyPASS, i.e. utility classes including projection calculation for monitors.

3. ILLUSTRATIVE EXAMPLES

3.1 Use of PyPASS

For now, PyPASS supports files used and produced in applications of CAMx by Environ (2005) and CMAQ by U.S. EPA (2005). However, if there are proper pre-procesors, other air quality model files can be used with PyPASS. The first step for using PyPASS is to prepare PyPASS input files. PyPASS input data files are outputs of CAMxSubset/CMAQSubset. These 'subset' processors, written in portable C, extract binary outputs and inputs of CAMx/CAMQ corresponding to a user-specified block of model domain. Two typical blocks are cells where stationary monitors are located and horizontal/vertical slices that users are interested in. The format of the CAMxSubset/CMAQSubset outputs is a fixed format chartacter header providing metadata information such as the grid numbers site ids and users' comments, followed by a binary section that is.an array or list of values in IEEE standard binary.

The second step for using PyPASS is to convert these binary subset files into PyTable files, which is based on HDF5 libraries. The second

step also includes merging some model meta-data into the PyTables. For example, we define monitor site information in an XML file and integrate it in each PyTables object. An example XML description of a monitor is shown in Fig. 2.

```
<monitor>
  <identity>
    <cams>C26</cams>
      <xcams>A110/X154</xcams>
      <airs>48-201-0029</airs>
      <label>HNWA</label>
      <name>NW Harris Co.(Tomball)</name>
  </identity>
  <location>
    <city>Tomball</city>
    <county>Harris</county>
    <state>Texas</state>
    <lon>-95.673889</lon>
    <lat>29.901111</lat>
    <ele>55</ele>
  </location>
  <measurements start="97" end="">
    <p>HH</p>
    <p>NO</p>
    <p>NO2</p>
    <p>O3</p>
    <p>WS</p>
    <p>WD</p>
    <p>AT</p>
    <p>DP</p>
    <p>TSR</p>
  </measurements>
</monitor>
```

Fig. 2. An example of monitor definition XML file. One of the advantages of using these meta-data is to integrate various model related information into one file.

The third step for using PyPASS is to invoke proper PyPASS modules with command-line keywords and arguments. Depending on the plot types, each PyPASS module provides various options to control the details of the graphics such as line thickness. However, some of the graphical elements are not allowed to change. For example, all hourly averaged values are plotted with stair-step graphs and not the typical lines usually made by other applications. This is necessary to convey accurate information with regard to the average data we are dealing with and to reduce confusion of data properties. Example plots will be shown in the following section.

The last step is a documentation step where PyPASS provides automation functions to insert plots and text into target documents such as MS Word or LaTex. This is critical for users that have to produce a report containing a large set of consistent graphs because document automation saves a lot time if plot style modification is needed later on. However, the last step for using PyPASS is optional in the sense that users may skip this step to do more in-depth analysis or review more materials.

3.2 Outputs of PyPASS

In this section, we present various types of graphics and explain the important differences of PyPASS outputs compared with the typical graphics made by other applications. Note that our intention in this presentation is to introduce what PyPASS does and not to conduct an actual performance analysis. Therefore, we try not to go beyond discussing the graphics themselves, even though we will make some comments on the material used in generating graphics if necessary.

The dataset used in generating the following graphics is primarily from Houston-Galveston Mid-Course Review modeling conducted as part of developing TX 2000 State Implementation Plan. For details of the model configuration and other information, please refer to Texas Commission on Environmental Quality web site, http://www.tnrcc.state.tx.us/air/aqp/airquality_workshop.html.

The current version of PyPASS can make the following types of graphics:

- Bar charts for unpaired peak concentrations and hourly concentration change rates
- Scatter plots for chemical species and wind speed with guiding spline lines
- Time series plots for a single chemical or multiple chemicals spanning user-defined time periods
- Hodogram; time series plot for wind in polar coordinates
- Tile plots with optional components such as background maps, lines, monitors, and winds
- Standalone vector plots without chemical concentrations

Fig. 3. shows a bar chart of unpaired peak ozone concentrations at all monitor sites. Note that the monitors are sorted from west to east to give a sense of possible spatial discrepancies of model prediction. Users can choose to change the sorting direction easily if west-east is not the directional tendency that users want to examine. While four different model cases are plotted in Fig. 3., PyPASS can deal with an arbitrary number of cases.

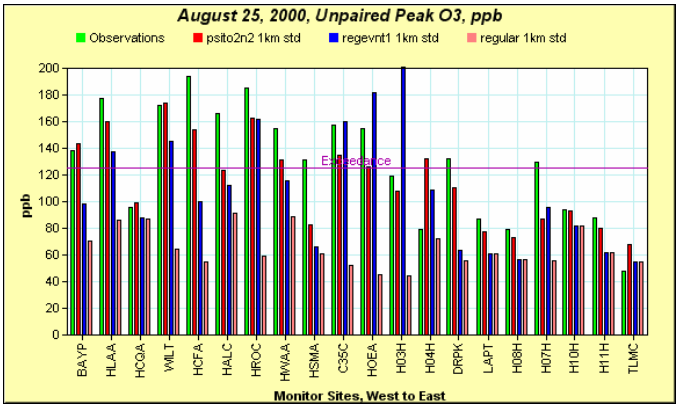

Fig. 3. An example of unpaired peak ozone concentrations bar chart. Different modeling cases are plotted with different colors and monitoring sites are sorted by its location from west to east of the study domain. X-axis label denotes the four-letter site codes and Y-axis shows ozone concentration along with NAAQS level depicted as the purple line with the label of 'Exceedance'.

In Fig. 3. one of the interesting sites is HALC. This is because most of monitors on the western side of the study domain show ozone exceedances and one modeled case, 'psito2n2 1km std', shows a similar pattern in terms of exceedances, except at HALC. Users may want to look at the meteorological predictions and observations at the HALC site.

PyPASS provides three major graphics for winds: wind speed scatter plots, wind hodogram, and wind error hodogram. A wind speed scatter plot is shown in Fig. 4. This specific graph shows the fact that the model wind becomes twice as fast as the observed winds starting late afternoon on August 25, 2000. Also, another important phenomena revealed by this plot is that there are cases where the modeled winds are very slow while the observed winds are strong enough to move about 4 km per hour. Given that the finest model grid resolution was 1 km, the real world winds could move airmass up to four cells away while the model holds the airmass near the HALC monitor. A hodogram is shown in Fig. 5. This plot uses the same data of Fig. 4. It is now clear that the winds from 1300-1600 is not in good agreement in direction even though wind speeds are similar. Also, this plot gives an idea that the main wind direction of observation and prediction are not quite the same.

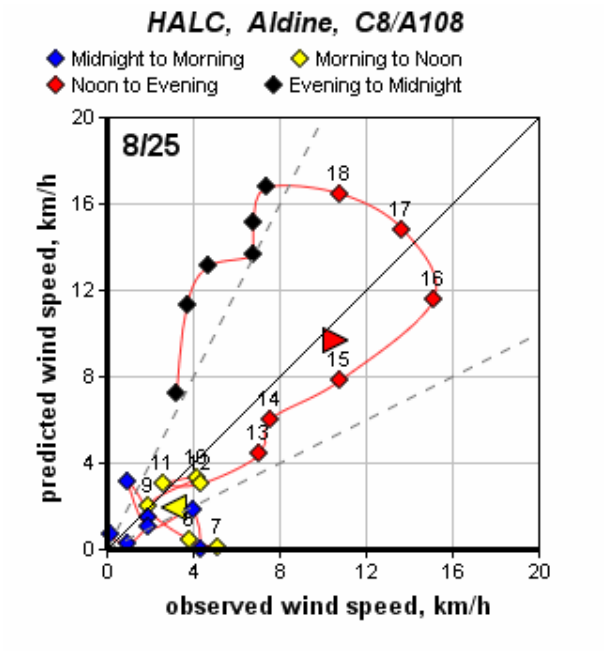

Fig. 4. An example of wind speed scatter plots. X-axis is for the observed winds and Y-axis is for the predicted winds. The red spline curve is for helping users to follow each pair in time sequence. Four different colors are used to distinguish four time periods of a day: 0000-0600, 0700-1200, 1300-1800, and 1900-2400.

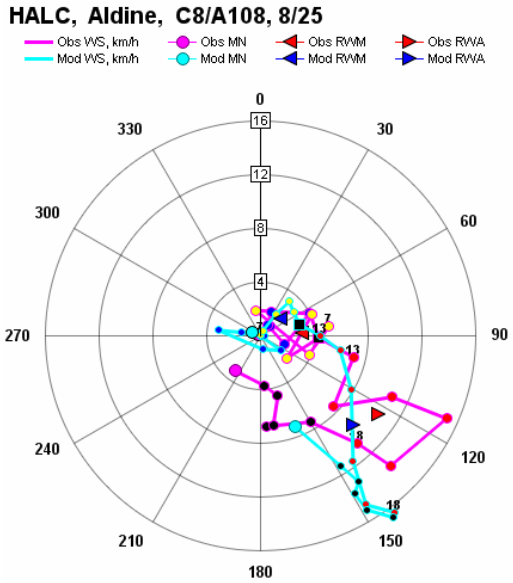

Fig. 5. An example of hodogram. The radial axis shows wind speed in km/h and the angular axis shows wind direction in degree with 0 degree as N winds. Each filled circle is the start point of a wind vector. Purple is for observations and cyan is for predictions. Triangles are for resultant winds; left-facing one is for morning and right-facing one is for afternoon. Some hours of day data points are labeled with its hour of day such as '13'.

Fig. 6. shows the wind error hodogram with the same data used in Fig. 5. Interesting fact is that this site does not show good agreement over more than half of the hours for the day if +/- 50% of wind speed and +/- 30 degree of wind direction are used for the guidance. Especially, during two nighttime periods, 0000-0600 and 1900-2400, predicted winds are very different from the observed winds in speed.

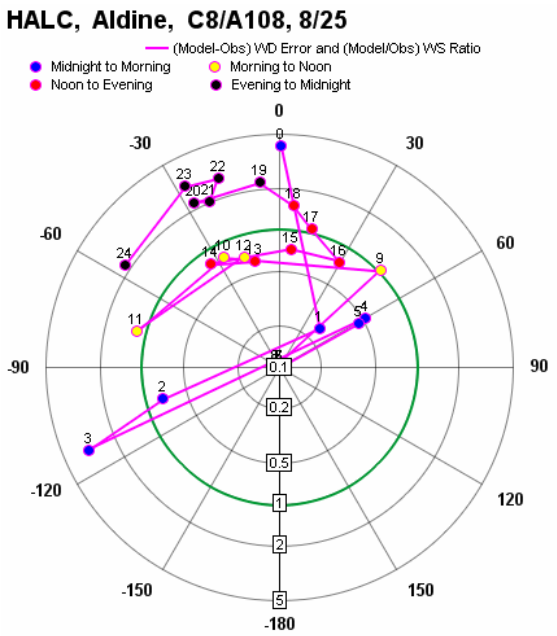

Fig. 6. An example of wind error hodogram. The radial axis is the ratio of modeled wind speeds to observed wind speeds. The angular axis is the degree of wind directional differences between observations and predictions. The green circle is the case when observed wind speeds are same as predicted wind speed. In an ideal case, all points are located at the cross point of 0 degree line and the green circle.

It is hard to judge whether these differences are important without examining chemical signals. An analysis to explore whether the given wind differences are important will be quite complex. To assist possible complex analysis, PyPASS provides various graphics for chemical signals: scatter plots and time series. Fig. 7. is the scatter plot for NO. One of the clear phenomena observed in this scatter plot is that NO is very biased in the model.

Fig. 8. is an example of a time series plot. Users can specify the maximum value on the Y-axis and the time span used for the X-axis. Different chemicals are shown in different colors. Observations are always plotted with solid lines while model cases are shown in different line types. Associated with Fig. 7, we can find that NO underestimation occurs early in the morning and there is NO agreement for 0700 while NO2 overestimation accompanies. Also, ozone is well underestimated for the 1200-1600 period.

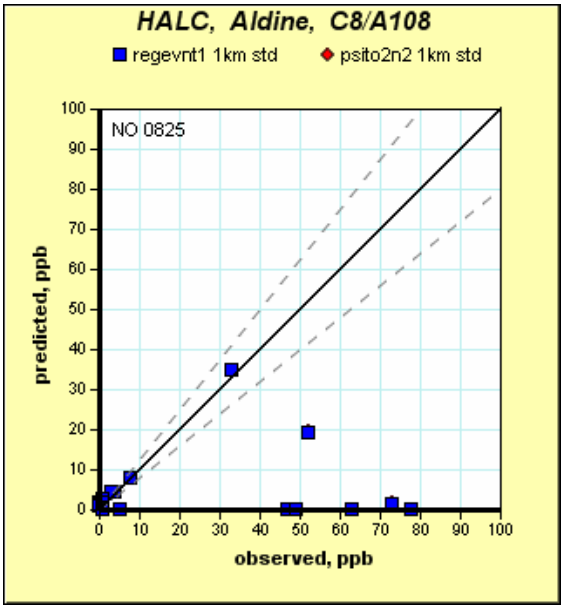

Fig. 7. An example of scatter plot. This example shows that two different modeling scenarios are almost identical in terms of NO prediction. Dotted lines are for +/- 20% biases of observed NO.

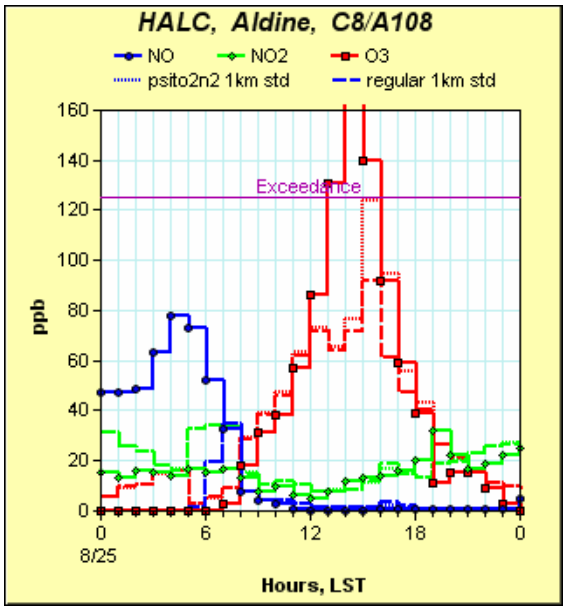

Fig. 8. An example of time series plot for chemicals. Solid lines are for observations. Three chemicals are plotted with different colors. This plot shows that the large underestimation of NO found in Fig. 7. occurs during the early morning and there is a large ozone underestimation for 1200-1600.

PyPASS can generate all plots with a transparent background, which is very helpful when users want to overlay and stack multiple plots. Fig. 9. is an overlaid plot of Fig. 5. on a GIS raster map showing important emission sources. The GIS map is scaled to match the number of pixels used in the radial axis of Fig. 5., i.e. four grid spacing in the GIS map has same pixels of 4km/h interval of Fig. 5. This pixel size matching gives users the opportunity to perform quick quantitative wind analysis.

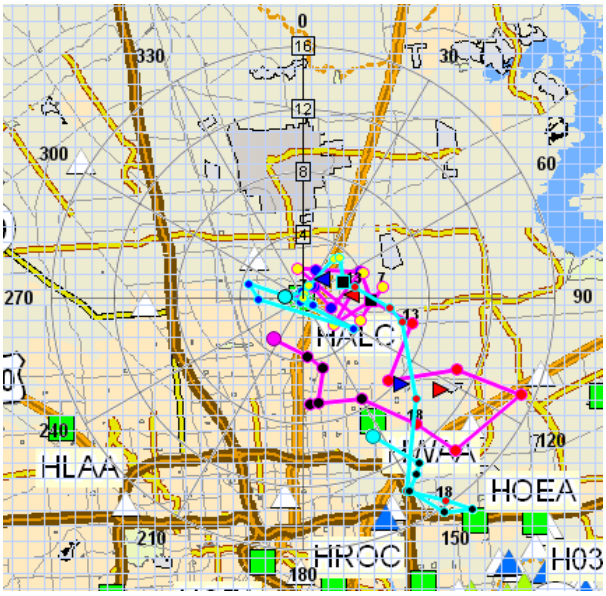

Fig. 9. An example of using transparent plots. This example is the results of overlaying Fig. 5. on a GIS map containing important emission sources (triangles) and monitors (squares with four-letter site codes).

4. DISCUSSION

PyPASS produces other plots such as wind vector plots. In the presentation, more PyPASS graphics will be introduced and discussion about PyPASS' performance in term of data storage and CPU time usage will be presented. In summary, PyPASS can produce lots of information necessary for the model performance evaluation while it improves the quality of graphical measures used in the performance analysis of regulatory photochemical air quality modeling.

5. REFERENCES

Advanced Software Engineering Limited, 2005: ChartDirector Version 4.0, http://www.advsofteng.com/product.html

Environ, 2005: CAMx Version 4.20, http://www.camx.com/

Fowler, M. and Scott, K., 1997: UML distilled, Addison-Wesley, 43-51

Object Management Group, 2005: Unified Modeling Language, http://www.uml.org

Python Software Foundation, 2005, Python Version 2.3, http://www.python.org/

U.S. EPA, 2005: CMAQ Version 4.4., http://www.epa.gov/asmdnerl/CMAQ/