2012 USGS National Groundwater Workshop
Using `python` to Improve Groundwater Model Effectiveness: A `matplotlib` Example Problems

August 9, 2012

1 Getting started

Open a command window and navigate to the MATPLOTLIB\python subdirectory (**fig. 1**).

2 `matplotlib` exercise 1 – create a simple plot

Download discharge data from NWIS for "USGS 06719505 CLEAR CREEK AT GOLDEN, CO" for the period from 8/1/2011 to 8/1/2012 and save as a space delimited file containing only the date and discharge data in the MATPLOTLIB\data subdirectory (a suggested name is `ClearCreek.txt`). If you do not have internet access, a prepared file is available (`06719505.txt`) in the MATPLOTLIB\data subdirectory. Create a new python script in the MATPLOTLIB\python subdirectory (a suggested name is `Ex1.py`). Open the python script you created in a text editor.

2.1 Read the Clear Creek data

Import the necessary datetime, pylab, numpy, and matplotlib modules and read the discharge data by adding the following to your blank python file (`Ex1.py`).

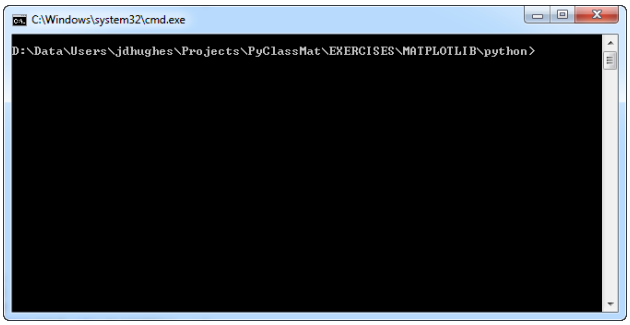

Figure 1: Getting started with the exercise.

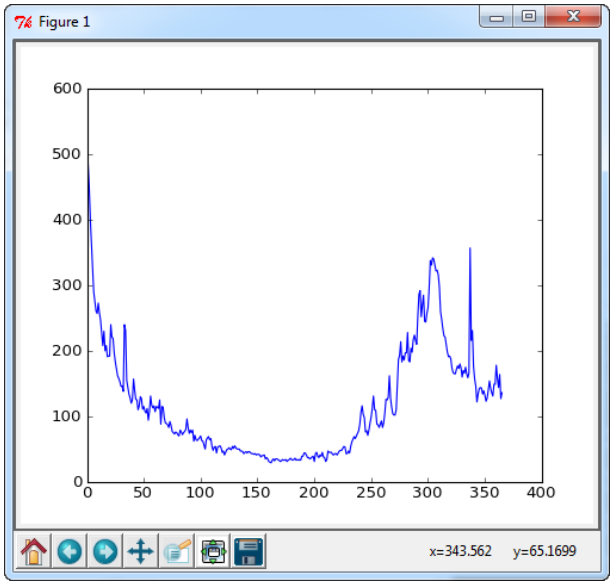

Figure 2: matplotlib exercise 1 screen output.

```
import datetime as dt
import numpy as np
import pylab as pl
import matplotlib as mpl
#--load flow data
q = np.genfromtxt( '..\\data\\ClearCreek.txt', \
                   names=['date','discharge'], dtype=[dt.datetime,'<f8'])
```

Depending on the date format you used when saving ClearCreek.txt python may experience some difficulty parsing the date. If this occurs, modify dt.datetime to '|S10' in the dtype= statement in np.genfromtext. To create a basic plot and print to the screen add the following lines to your python file.

```
#--create a figure of the discharge for the last year
pl.figure( figsize=(6.0, 5.0), facecolor='w' )
#--plot the data
pl.plot(q['discharge'])
#--show the plot
pl.show()
```

Run the script from the command line (**fig. 1**) by typing python Ex1.py and pressing enter. You should see the graphic shown in **figure 2** if the script executes correctly.

2.2 Modifying the axes and adding labels

Add the following lines to modify the extent of the x-axis and add labels to the x- and y-axes.

```
pl.xlabel('Days')
pl.ylabel(r'Discharge ft$^3$/s')
pl.xlim(0,365)
```

These lines should be added before the pl.show() command. The r prior to the string in the pl.ylabel command allows LaTeX equations and equation formatting to be embedded in strings shown in the plot. The x-axis is limited to be between 0 and 365 using the xlim command because there are only 366 days in the dataset for Clear Creek (remember python is zero-based). After saving these modifications rerun the script. You should see the graphic shown in **figure 3** if the modified script executes correctly.

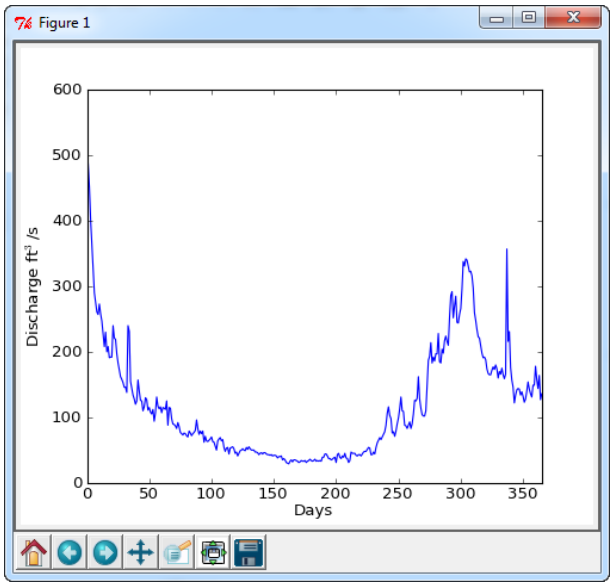

Figure 3: `matplotlib` exercise 1 screen output with axes modifications.

2.3 Saving the plot as an image

To save the plot to an image file modify the script by commenting out the `pl.show()` command and adding the `pl.savefig` command as shown below. The resulting figure is shown **figure 4**.

```
##--show the plot
#pl.show()
#--output figure
pl.savefig('..\\figures\\Ex1.png',dpi=300)
```

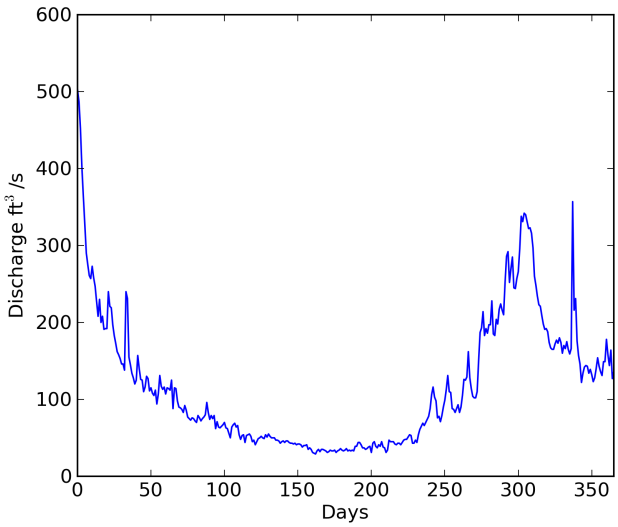

Figure 4: `matplotlib` exercise 1 exported image file.

2.4 Additional tasks

Some additional tasks that you can attempt if you have time are changing the color, line weight, line style, *etc.* of the Clear Creek discharge data. You should refer to http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.plot for guidance on modifying these (and other) options.

3 matplotlib exercise 2 – working with subplots

A base python script for exercise 2 (BarChartBase.py) has been provided. Make a copy of this file so you can backtrack in the event that you make and error and need to start over; a suggested sript name is Ex2.py.

3.1 Getting started with subplots

In this exercise we are going to read some meteorologic data (data*backslash*MeterologicData.csv) and plot multiple plots on a single matplotlib figure. The portion of the python script that reads and processes the data is shown below. The np.genfromtxt is quite complicated the primary task that this command is performing is parsing datetime data and filling missing data values. The data file data\MeterologicData.csv contains rainfall, potential evapotranspiration (PET), minimum air temperature, and maximum air temperature. The script also includes logic for calculating monthly rainfall and PET.

```
#--read data
metnames = ['date','Rain_inpd','ETP_mmpd','AirTMin_C','AirTMax_C']
d = np.genfromtxt( '..\\data\\MeterologicData.csv', skip_header=1, delimiter=',', \
                   missing_values=('MISSING','MISSING','MISSING','MISSING','MISSING'), \
                   filling_values=(dt.date(1900, 1, 1),0.0,0.0,np.NAN,np.NAN), \
                   names=metnames, dtype=None, converters={'date':mkdate} )
datemin = dt.date(d['date'].min().year  , 1, 1)
datemax = dt.date(d['date'].max().year+1, 1, 1)
#--create monthly totals for rainfall and ETP
on_date = d['date'][0]
monthly_data, c = [], [0.0, 0.0]
for ipos,t in enumerate( d['date'] ):
    if t.month != on_date.month or ipos == len( d['date'] ) - 1:
        t_month = t_date.month
        t_day = 1 #int( t_date.day / 2 )
        t_year = t_date.year
        monthly_data.append( [ dt.date(t_year, t_month, t_day), c[0], c[1], t_date.day ] )
        c[0] = 0.0
        c[1] = 0.0
        on_date = t
    c[0] += d['Rain_inpd'][ipos]
    c[1] += d['ETP_mmpd'][ipos]
    t_date = t
monthly_data = np.array( monthly_data )
```

The add_subplot(row,col,subplot) command is used to create subplots in a matplotlib figure. The same matplotlib commands are used to create and format the subplot as is used for single plot matplotlib figures.

```
#--plot the temperature data
ax.plot(pl.date2num(d['date']),d['AirTMin_C'], color='b', linewidth=0.7, label=r'T$_{MIN}$')
ax.plot(pl.date2num(d['date']),d['AirTMax_C'], color='r', linewidth=0.7, label=r'T$_{MAX}$')
#--legends and axes
leg = ax.legend(loc='upper left',ncol=2,labelspacing=0.25,columnspacing=1,\
                handletextpad=0.5,handlelength=2.0,numpoints=1)
leg._drawFrame=False
ax.xaxis.set_major_locator(years), ax.xaxis.set_minor_locator(months)
ax.xaxis.set_major_formatter(yearsFmt)
ax.set_xlim(datemin, datemax)
ax.set_ylabel( r'Temperature $^{\circ}$C' )
ax.set_ylim(0,45)
#--define the second subplot
ax = fig.add_subplot(3,1,2)
```

```
#--plot the rainfall data
ax.bar(pl.date2num(monthly_data[:,0]),monthly_data[:,1]*25.4, \
       color='b', width=monthly_data[:,3], linewidth=0, label='Rainfall')
#--axes
ax.xaxis.set_major_locator(years), ax.xaxis.set_minor_locator(months)
ax.xaxis.set_major_formatter(yearsFmt)
ax.set_xlim(datemin, datemax)
ax.set_ylabel( 'Rainfall mm' )
```

This script plots air temperature on the first subplot using a line graph (using the `plot` command) and monthly rainfall on the second subplot using a bar chart (using the `bar` command). The command `pl.date2num(d['date'])` converts the `datetime` data to a number format that can be plotted by `matplotlib`. Because we are working with `datetime` there are some commands for using dates on the x-axis. Please take some time to review the file before you begin modifying it. Prior to making any modifications to your file run the script which will create an output image named `figures\Ex2.png` (**fig. 5**).

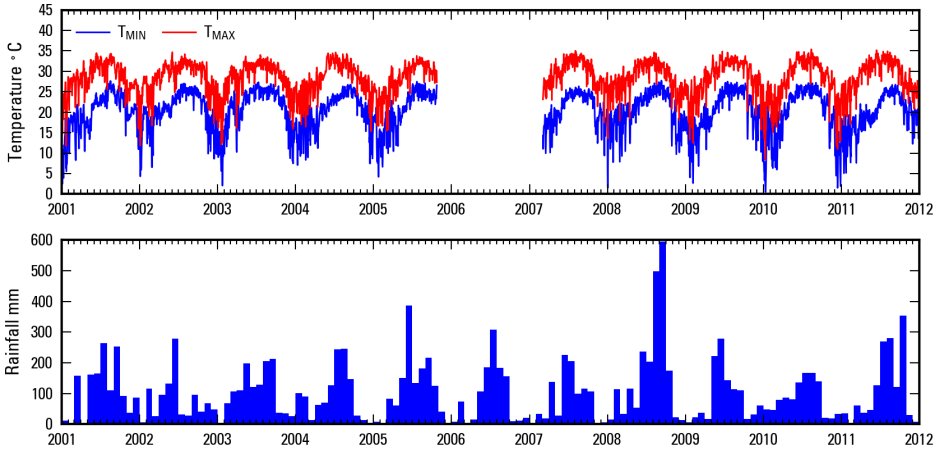

Figure 5: `matplotlib` exercise 2 initial figure output.

3.2 Add a third subplot

What you are going to be doing is adding a third subplot and plotting monthly PET on a bar chart. The easiest thing to do is copy and paste the code for the second subplot and modifying it to plot the PET data. Your modifications should look something like the code listed below.

```
#--define the third subplot
ax = fig.add_subplot(3,1,3)
#--plot the potential evapotranspiration data
ax.bar(pl.date2num(monthly_data[:,0]),monthly_data[:,2], \
       color='r', width=monthly_data[:,3], linewidth=0, label='Rainfall')
#--axes
ax.xaxis.set_major_locator(years), ax.xaxis.set_minor_locator(months)
ax.xaxis.set_major_formatter(yearsFmt)
ax.set_xlim(datemin, datemax)
ax.set_xlabel( 'Year' )
ax.set_ylabel( 'PET mm' )
```

After saving and running the revised script the output image should look similar to **figure 6**.

3.3 Add a second axis to the second subplot

It is relatively easy to add a second axis to individual subplots using the `twinx()` command. You will be adding a second axis to the second subplot showing the cumulative rainfall. Add the following to your script.

```
ax2 = ax.twinx()
ax2.plot(pl.date2num(monthly_data[:,0]),np.cumsum( monthly_data[:,1]*25.4/1000. ), color='k' )
```

These two lines should be added immediately after the `ax.bar` command. The second axis label can be added by adding the following line after the `ax.set_ylabel` command.

```
ax2.set_ylabel( 'Cumulative Rainfall m' )
```

The resulting image should look similar to **figure 7**

3.4 Additional tasks

Some additional tasks that you can attempt if you have time are changing the colors, line weights, line styles, *etc.* on any of the subplots. You should refer to `http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.plot` and `http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.bar` for guidance on modifying these (and other) options.

4 matplotlib exercise 3 – working with MODFLOW output

A base `python` script for exercise 3 (`MODFLOWBase.py`) has been provided. Make a copy of this file so you can backtrack in the event that you make and error and need to start over; a suggested sript name is `Ex3.py`.

4.1 Getting started with MODFLOW output

Text

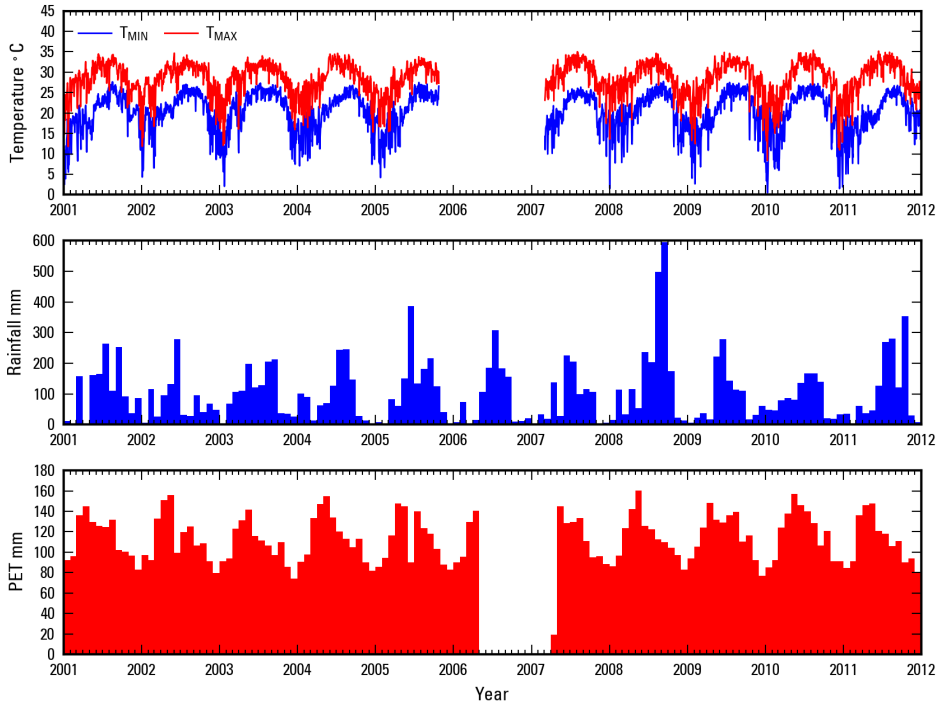

Figure 6: `matplotlib` exercise 2 figure output with the third subplot.

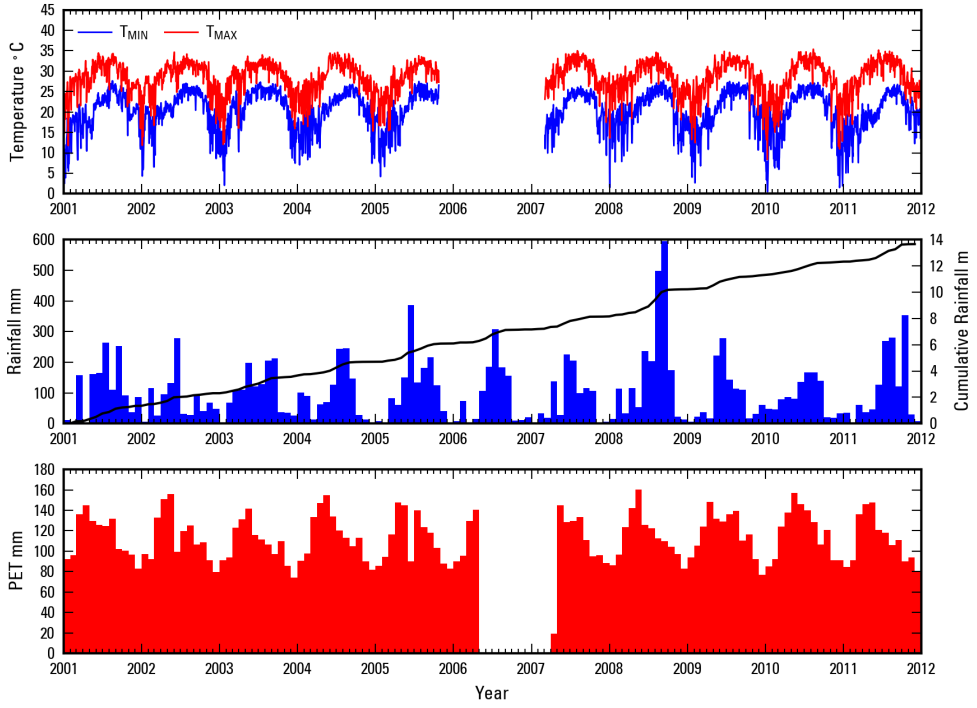

Figure 7: `matplotlib` exercise 2 figure output with the third subplot and a second axis on the second subplot.