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

August 9-10, 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`).

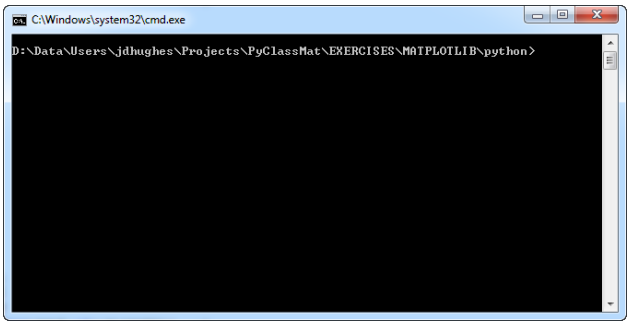

Figure 1: Getting started with the exercise.

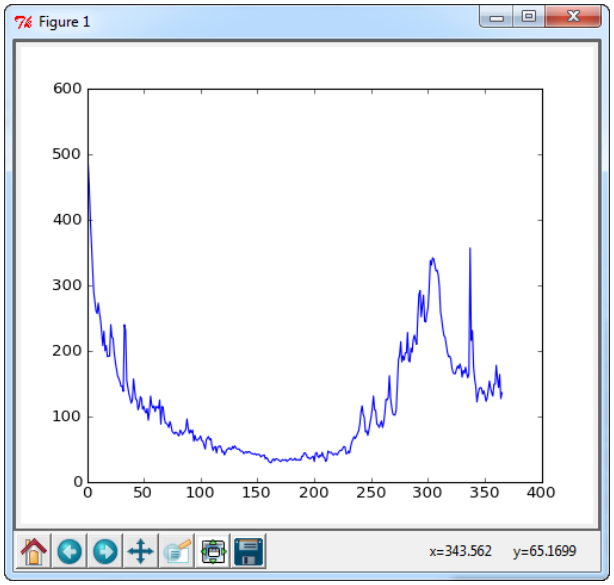

Figure 2: matplotlib exercise 1 screen output.

```
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.

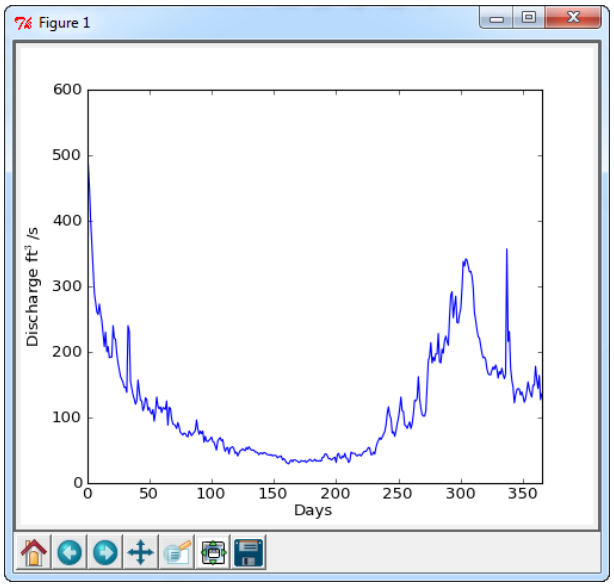

Figure 3: `matplotlib` exercise 1 screen output with axes modifications.

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)
```

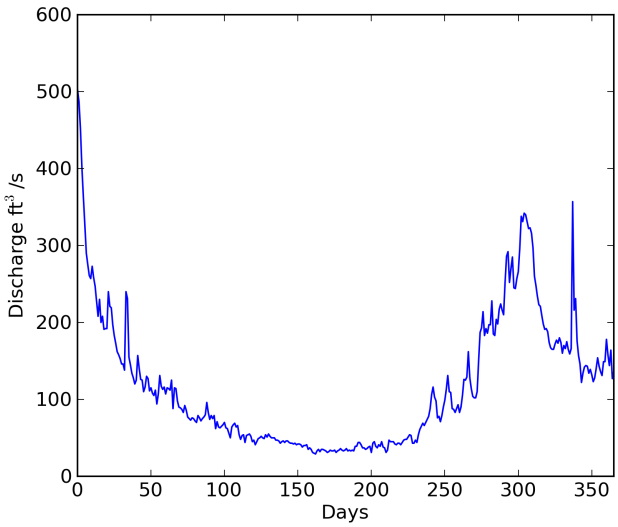

Figure 4: `matplotlib` exercise 1 exported image file.

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**).

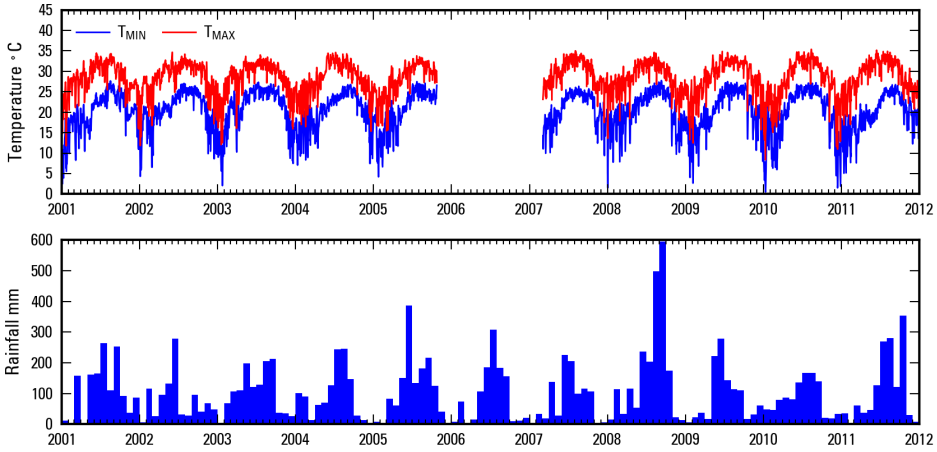

Figure 5: `matplotlib` exercise 2 initial figure output.

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**.

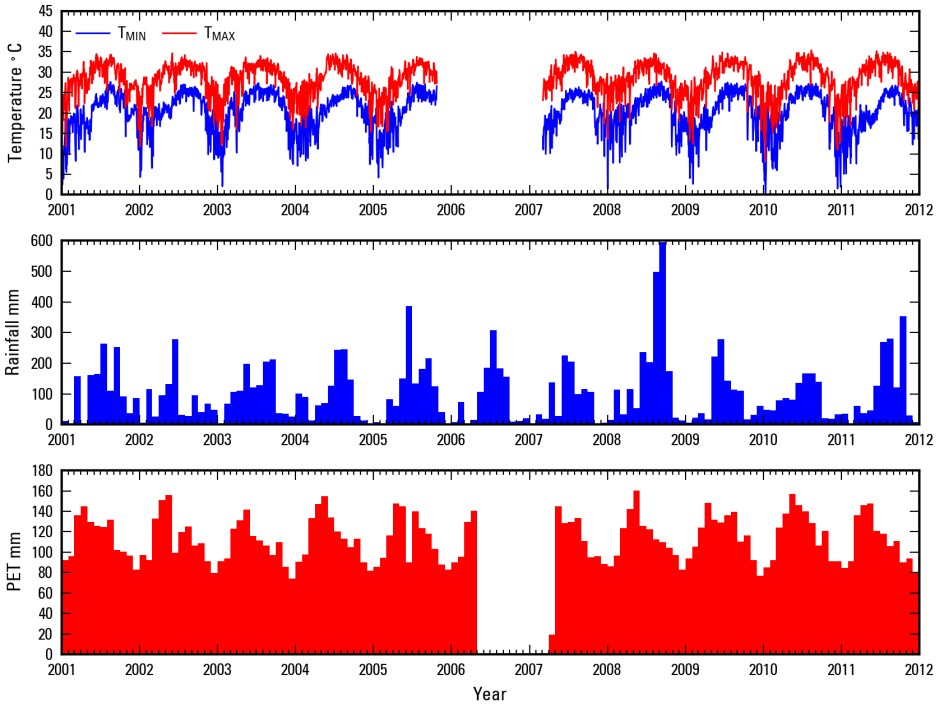

Figure 6: matplotlib exercise 2 figure output with the third subplot.

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.

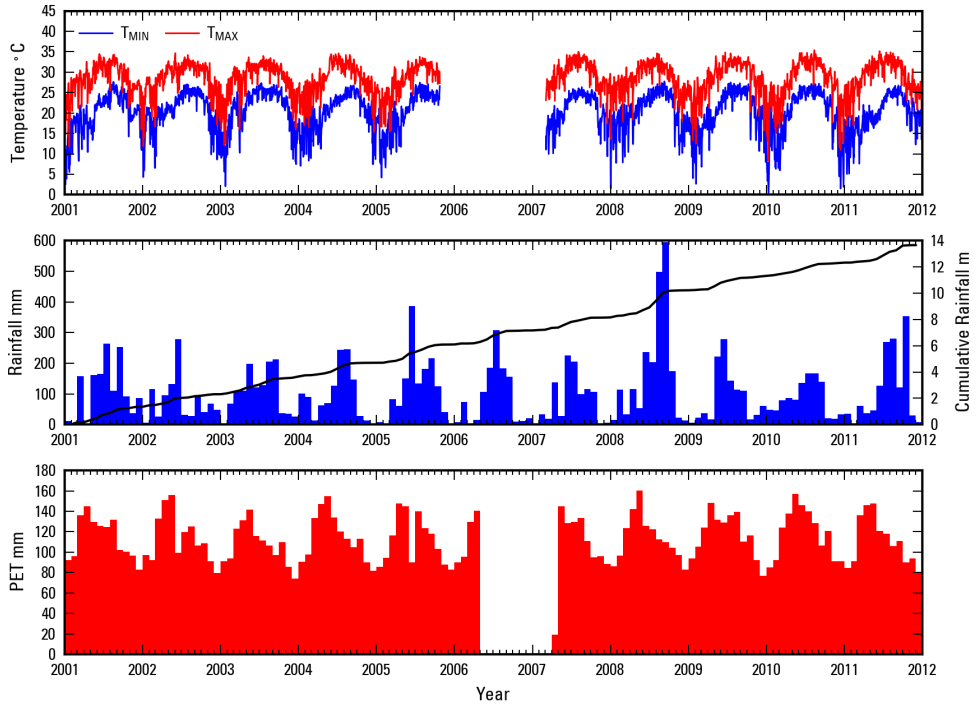

Figure 7: matplotlib exercise 2 figure output with the third subplot and a second axis on the second subplot.

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

In this exercise we are going to read data from a binary MODFLOW head file (data\CoastalAquifer.hds) and plot hydrographs and maps. The python\MFBinaryClass.py python script includes functions for reading binary data from MODFLOW head and cell-by-cell files.

The classes in python\MFBinaryClass.py include functionality to extract time series from specific cells and three-dimensional data from MODFLOW files. An example of how to read MODFLOW output data using python\MFBinaryClass.py is listed below. This example is from MODFLOWBase.py.

```
#--problem size
nlay,nrow,ncol = 1,41,40
#--default data if command line argument not defined for variable
head_file = '..\\data\\CoastalAquifer.hds'
#--get available times in the head file
get_cell = 1
headObj = mfb.MODFLOW_Head(nlay,nrow,ncol,head_file)
t = headObj.get_gage(get_cell)
ntimes = t.shape[0]
mf_times = np.zeros( (ntimes), np.float )
for i in range(0,ntimes):
    mf_times[i] = t[i,0]
```

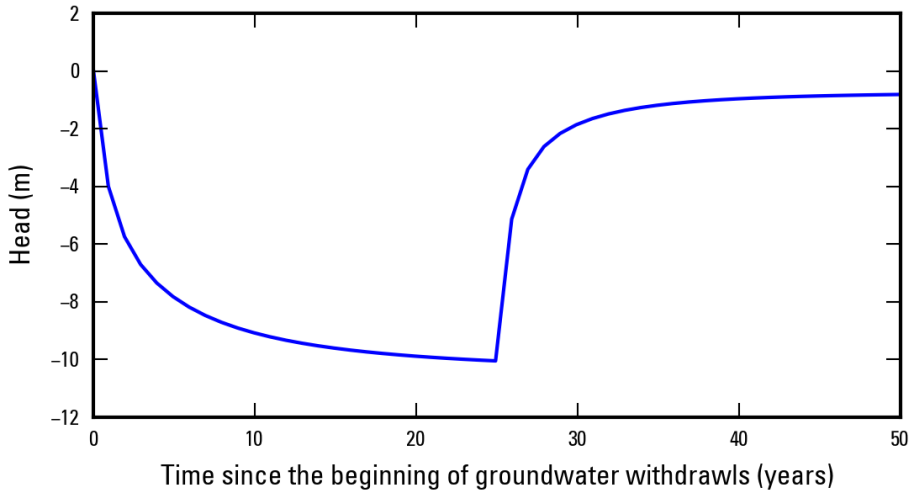

Figure 8: matplotlib exercise 3 groundwater head hydrograph for layer 1, row 18, and cell 31.

The python command mfb.MODFLOW_Head(nlay,nrow,ncol,head_file) creates and instance of the MODFLOW_Head class, initializes the instance, and opens data\CoastalAquifer.hds. The method get_gage extracts the head values from a specified cell number – layer 1, row 1, column 1 or cell 1 in this case. The remainder of the example code stores the TOTIM values from data\CoastalAquifer.hds in a numpy array.

4.2 Creating a hydrograph from MODFLOW head output

What you are going to be doing is extracting head data at the location of a groundwater withdrawal at layer 1, row 18, column 31, creating a hydrograph, and saving the hydrograph to an image file. The easiest thing to do is copy and paste the code for extracting the TOTIM values from data\CoastalAquifer.hds and adding code to plot this data to the bottom of your working script. Your new code should look something like the code listed below.

```
#--create a time-series plot of head
headObj = mfb.MODFLOW_Head(nlay,nrow,ncol,head_file)
get_cell = mfb.icrl_from_kij(1,18,31,nlay,nrow,ncol)
t = headObj.get_gage(get_cell)
ztf = figure(figsize=(4, 2), facecolor='w')
ztf.subplots_adjust(wspace=0.2,hspace=0.2,left=0.15,right=0.95,bottom=0.15,top=0.95)
ax = ztf.add_subplot(1,1,1)
hp = ax.plot(mf_times/365.25-100.,t[:,1])
ax.set_xlim(0,50)
xlabel('Time since the beginning of groundwater withdrawls (years)')
ylabel('Head (m)')
output_name = '..\\figures.MF\\Cell_{0}.png'.format( get_cell )
ztf.savefig(output_name,dpi=300)
```

The function mfb.icrl_from_kij(1,18,31,nlay,nrow,ncol) determines the cell number at layer 1, row 18, column 31. The command mf_times/365.25-100. within the ax.plot() command converts days to years and shifts the time axis so 0.0 corresponds to the period when groundwater withdrawals start in the model. The x-axis is restrict to only show the period with groundwater withdrawals between shifted time 0 and 50 years. The resulting image should look similar to **figure 8**

4.3 Modifying hydrograph output to drawdown

Copy and paste the code you developed to create the hydrograph figure at the bottom of your working script. In this case, the initial head can be calculated using h0 = t[:,1].max(). Modify

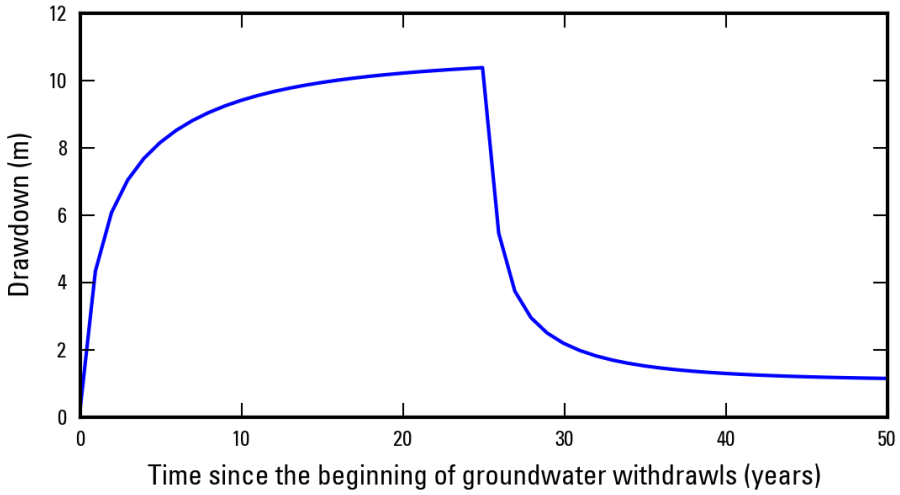

Figure 9: matplotlib exercise 3 drawdown hydrograph for layer 1, row 18, and cell 31.

the hydrograph to show drawdown. Your new code should look something like the code listed below.

```
#--create a time-series plot of drawdown
h0 = t[:,1].max()
ztf = figure(figsize=(4, 2), facecolor='w')
ztf.subplots_adjust(wspace=0.2,hspace=0.2,left=0.15,right=0.95,bottom=0.15,top=0.95)
ax = ztf.add_subplot(1,1,1)
hp = ax.plot(mf_times/365.25-100.,h0-t[:,1])
ax.set_xlim(0,50)
xlabel('Time since the beginning of groundwater withdrawls (years)')
ylabel('Drawdown (m)')
output_name = '..\\figures.MF\\Cell_{0}_drawdown.png'.format( get_cell )
ztf.savefig(output_name,dpi=300)
```

The resulting image should look similar to **figure 9**

4.4 Creating a map from MODFLOW head output within a loop

Now we will extract the head data from `data\CoastalAquifer.hds` for each `TOTIM` values and create a figure within a loop. The approach for extracting three-dimensional head data is similar to the approach used to extract data for an individual cell. The method `get_record(findtime)` is used to extract three-dimensional data.

We will be using the `matplotlib` commands `pcolor`, `contour`, and `clabel` to create the map of the MODFLOW head data. The coordinates at the center and edges of the cells will be needed to correctly plot the data using `pcolor` and `contour`. Suggested code for calculating the model coordinates to add to the bottom of your working script is listed below.

```
#--coordinate information
dx,dy   = 500., 500.
xcell = np.arange(dx/2.,(ncol*dx)+dx/2.0,dx)
ycell = np.arange(dy/2.,(nrow*dy)+dy/2.0,dy)
Xcell,Ycell = np.meshgrid(xcell,ycell)
xedge = np.arange(0.0,float(ncol)*dx+0.001,dx)
yedge = np.arange(0.0,float(nrow)*dy+0.001,dy)
Xedge,Yedge = np.meshgrid(xedge,yedge)
xmin,xmax = 0.0,float(ncol)*dx
ymin,ymax = 0.0,float(nrow)*dy
```

The `numpy` command `meshgrid` creates a two-dimensional array of x- and y-coordinates from one-dimensional arrays and are used by `pcolor` and `contour`. The `hd numpy` array will be used to Try to add a loop that extracts the head data and plots the data for each saved MODFLOW time step. A possible approach for coding this is listed below.

```
#--common data for each figure
hdcontour  = np.arange(0.0,10.0,0.25)
#--create figures for each output time
for on_time in mf_times:
    #--build output file name
    output_name = '..\\figures.MF\\Head_{0:05d}.png'.format( int( on_time ) )
    #--read head data - final zeta surface
    headObj = mfb.MODFLOW_Head(nlay,nrow,ncol,head_file)
    totim,kstp,kper,h,success = headObj.get_record(on_time)
    hd = np.copy( h[0,:,:] )
    #--invert rows for plotting with pcolor
    hd       = np.flipud(hd)
    #--calculate ranges
    minh, maxh = np.min(hd), np.max(hd)
    #--print summary of min and max head and zeta
    print 'Head                [{0:10.3f},{1:10.3f}]'.format(minh,maxh)
    #-Make figures
    print 'creating figure...{0}'.format( output_name )
    #--figure
    ztf = figure(figsize=(3, 3), facecolor='w')
    ztf.subplots_adjust(wspace=0.2,hspace=0.2,left=0.15,right=0.95,bottom=0.15,top=0.95)
    ax = ztf.add_subplot(1,1,1,aspect='equal')
    iyears = int( on_time / 365. ) - 100
    ctime = 'years'
    if iyears == 1:
        ctime = 'year'
    ctitle = 'Groundwater head (m) after {0:5d} {1}'.format( iyears, ctime )
    text(0.0,1.01,ctitle,horizontalalignment='left',verticalalignment='bottom',size=7,transform=ax.transAxes)
    #hp = ax.pcolor(Xedge,Yedge,hd,vmin=0,vmax=2,cmap='jet_r',alpha=1.0,edgecolors='None')
    hp = ax.imshow(np.flipud(hd),vmin=0,vmax=2,cmap='jet_r',alpha=1.0,extent=[xmin,xmax,ymin,ymax])
    ch = ax.contour(xcell,ycell,hd,levels=hdcontour,colors='k',linewidths=1)
    ax.clabel(ch,inline=1,fmt='%5.2f',fontsize=6)
    #--plot limits
    ax.set_xlim(xmin,xmax), ax.set_ylim(ymin,ymax)
    xlabel('Column distance (m)')
    ylabel('Row distance (m)')
    #--save the figure
```

The `numpy` command `flipud(hd)` flips the data in the y-direction so it is oriented in a cartesian coordinate system for `pcolor`. The resulting image after 1 year should look similar to **figure 10**

Modify your code to use `imshow` instead of `pcolor`. You should refer to `http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.imshow` for guidance on using `imshow`. One thing to note is `imshow` expects the y-axis for data to be oriented in the same direction as MODFLOW so `hd` will need to be flipped again. Since `hd` is also being used to plot the contours, `hd` should be flipped in the call to `imshow` or a new variable defined. A possible approach for using `imshow` is listed below.

```
hp = ax.imshow(np.flipud(hd),vmin=0,vmax=2,cmap='jet_r',alpha=1.0,extent=[xmin,xmax,ymin,ymax])
```

The resulting image after 1 year using `imshow` should look similar to **figure 11**

Evaluate the result of not flipping `hd`.

4.5 Additional tasks

Some additional tasks that you can attempt if you have time are changing the color map used to create the map and contour levels *etc.* on the maps of hydraulic head. You should refer to `http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.pcolor` and `http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.contour` for guidance on modifying these (and other) options.

It may be useful to review the additional functionality available in `python\MFBinaryClass.py` for reading binary MODFLOW output.

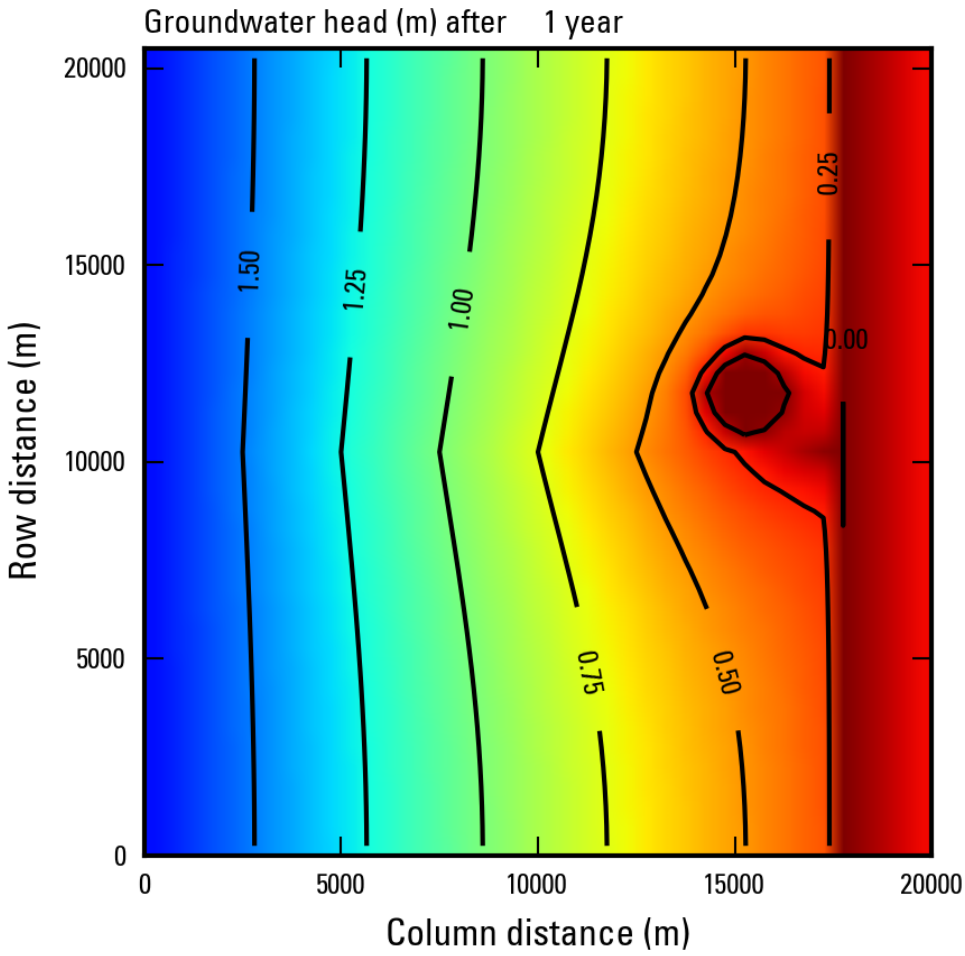

Figure 10: `matplotlib` exercise 3 simulated groundwater head after 1 year of groundwater withdrawals using `pcolor`.

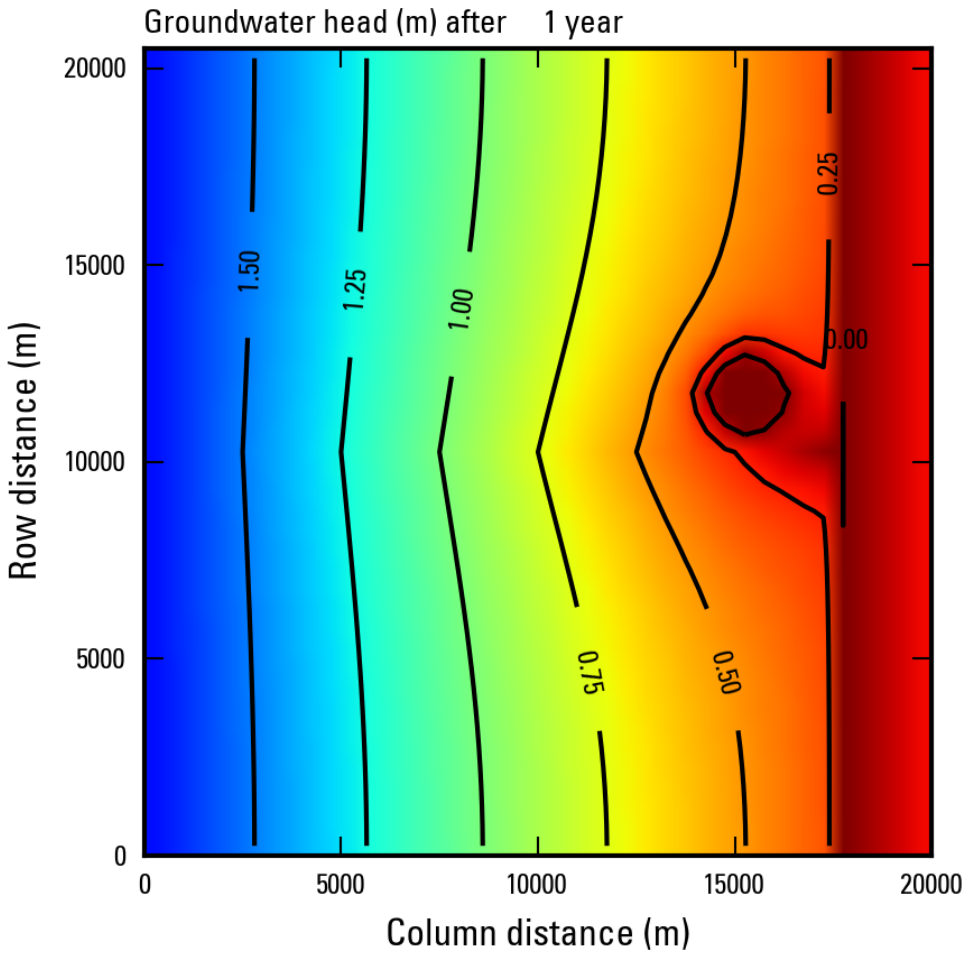

Figure 11: `matplotlib` exercise 3 simulated groundwater head after 1 year of groundwater withdrawals using `imshow`.