Python Workshop
Plotting with `matplotlib`

Joseph D. Hughes

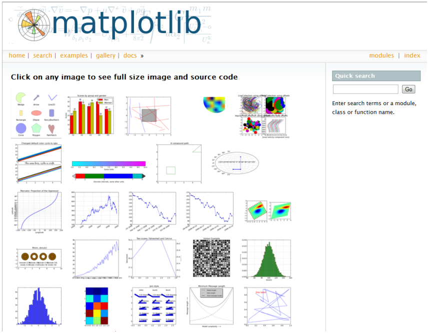
U.S. Geological Survey
Florida Water Science Center, Tampa, Florida USA

USGS National Groundwater Workshop, August 2012

Background information

`matplotlib` resources:

`http://www.matplotlib.sourceforge.net`



matplotlib

home | search | examples | gallery | docs »

modules | index

Click on any image to see full size image and source code

Quick search

Go

Enter search terms or a module, class or function name.

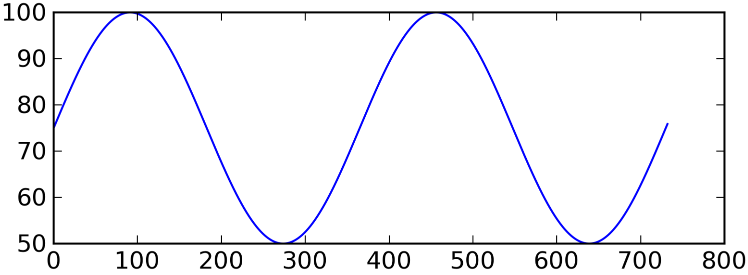
Creating a super simple plot (1)

SuperSimplePlot.py

```python
import numpy as np
import pylab as pl
import matplotlib as mpl
#--load flow data
q = np.genfromtxt( '..\\data\\USInflow.dat', skip_header=4 )
#--create figure of upstream inflow
fig = pl.figure( figsize=(6.0, 2.0), facecolor='w' )
#--define the subplot
ax = fig.add_subplot(1,1,1)
#--plot the data
ax.plot(q[:,0],q[:,1])
#--output figure
#--png
fig.savefig('..\\figures\\SuperSimplePlot.png',dpi=300)
```

Creating a super simple plot (2)

`SuperSimplePlot.py`



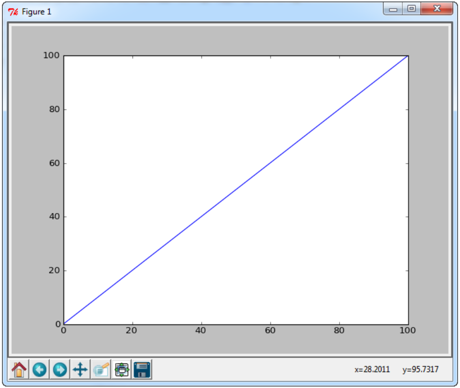
Creating a super simple plot (3)

- open the command line
- type `python`
- enter the text listed below

```python
import numpy as np
import matplotlib.pyplot as plt
y = np.arange(0,101,1)
fig = plt.figure()
ax = fig.add_subplot(111)
p = ax.plot(y)
plt.show()
```

Creating a super simple plot (4)

- everyone should see...



Creating a not so simple plot (1)

SimplePlot.py

```python
import sys
import string
import math
import numpy as np
import pylab as pl
import matplotlib as mpl
from matplotlib.font_manager import FontProperties
#--general specification data for matplotlib
mpl.rcParams['font.sans-serif']          = 'Univers 57 Condensed'
mpl.rcParams['font.serif']               = 'Times'
mpl.rcParams['font.cursive']             = 'Zapf Chancery'
mpl.rcParams['font.fantasy']             = 'Comic Sans MS'
mpl.rcParams['font.monospace']           = 'Courier New'
mpl.rcParams['mathtext.default']         = 'regular'
mpl.rcParams['pdf.compression']          = 0
mpl.rcParams['pdf.fonttype']             = 42
#--figure text sizes
mpl.rcParams['legend.fontsize']  = 7
mpl.rcParams['axes.labelsize']   = 8
mpl.rcParams['xtick.labelsize']  = 7
mpl.rcParams['ytick.labelsize']  = 7
```

Creating a not so simple plot (2)

SimplePlot.py

```
#--create upstream inflow data
#--temporal dimensions
nper    = 365 * 2 + 1
ntsp    = np.ones( (nper), np.int )
tsp_len = 1.0 #day
tmax    = float( nper ) * tsp_len
simtime = np.arange(0.0,tmax+2.*tsp_len,tsp_len)
#--generate a sinusoidal inflow function
q = np.zeros( (len(simtime)), np.float )
qbase, qptrb = 75.00, 25.00
tp    = 365.
ipos = 0
for ipos,t in enumerate( simtime ):
    qp = qptrb * math.sin( 2.0 * math.pi * t / tp )
    q[ipos] = qbase + qp
```

Creating a not so simple plot (3)

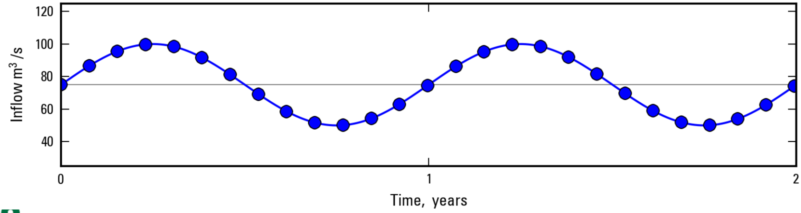
SimplePlot.py

```
#--create figure of upstream inflow
#--how big to make the figure and where to place it
fwid, fhgt = 6.00, 1.50
flft, frgt = 0.10, 0.95
fbot, ftop = 0.20, 0.95
fig = pl.figure( figsize=(fwid, fhgt), facecolor='w' )
fig.subplots_adjust(wspace=0.25,hspace=0.25,left=flft,right=frgt,bottom=fbot,top=ftop)
#--define the subplot
ax = fig.add_subplot(1,1,1)
#--plot the data
ax.plot([0,10], [qbase,qbase], color='0.5', linewidth=0.5, label='_Zero')
ax.plot(simtime/365,q, color='b', linewidth=1, label='Inflow', marker='o', markevery=28)
#--titles and axes
ax.set_ylabel( r'Inflow $m^3/s$' )
ax.set_ylim(25,125)
ax.set_xlabel('Time,  years')
ax.set_xlim(0,2)
ax.set_xticks( np.arange(0,3,1) )
```

Creating a not so simple plot (4)

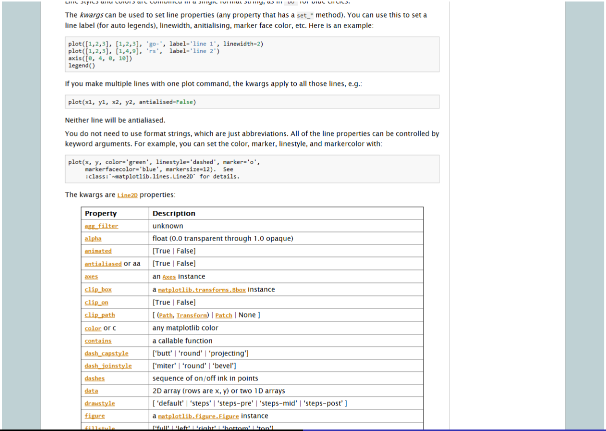
SimplePlot.py

```
#--output figure
#--png
outfigpng = '..\\figures\\Inflow.png'
fig.savefig(outfigpng,dpi=300)
print 'created...', outfigpng
#--pdf
outfigpdf = '..\\figures\\Inflow.pdf'
fig.savefig(outfigpdf,dpi=300)
print 'created...', outfigpdf
```



Creating a not so simple plot (5)

http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.plot



Line styles and colors are combined in a single format string, as in 'bo' for blue circles.

The *kwargs* can be used to set line properties (any property that has a set_* method). You can use this to set a line label (for auto legends), linewidth, anitialising, marker face color, etc. Here is an example:

```
plot([1,2,3], [1,2,3], 'go-', label='line 1', linewidth=2)
plot([1,2,3], [1,4,9], 'rs',  label='line 2')
axis([0, 4, 0, 10])
legend()
```

If you make multiple lines with one plot command, the kwargs apply to all those lines, e.g.:

```
plot(x1, y1, x2, y2, antialised=False)
```

Neither line will be antialiased.

You do not need to use format strings, which are just abbreviations. All of the line properties can be controlled by keyword arguments. For example, you can set the color, marker, linestyle, and markercolor with:

```
plot(x, y, color='green', linestyle='dashed', marker='o',
     markerfacecolor='blue', markersize=12).  See
     :class:`~matplotlib.lines.Line2D` for details.
```

The kwargs are Line2D properties:

Property	Description
agg_filter	unknown
alpha	float (0.0 transparent through 1.0 opaque)
animated	[True | False]
antialiased or aa	[True | False]
axes	an Axes instance
clip_box	a matplotlib.transforms.Bbox instance
clip_on	[True | False]
clip_path	[(Path, Transform) | Patch | None]
color or c	any matplotlib color
contains	a callable function
dash_capstyle	['butt' | 'round' | 'projecting']
dash_joinstyle	['miter' | 'round' | 'bevel']
dashes	sequence of on/off ink in points
data	2D array (rows are x, y) or two 1D arrays
drawstyle	['default' | 'steps' | 'steps-pre' | 'steps-mid' | 'steps-post']
figure	a matplotlib.figure.Figure instance
fillstyle	['full' | 'left' | 'right' | 'bottom' | 'top']

Creating a bar chart (1)

MeterologicData.csv

```
Daily Date,M6888_Rain_inpd,OH515_EPT_mmpd,TA613_AIRT_MIN_C,TA613_AIRT_Max_C
1/1/2001,0,3.51,3.45,17.75
1/2/2001,0,3.15,7.51,20.15
```

```
12/29/2011,0,3.07,13.741,22.76
12/30/2011,0,3.31,13.551,24.887
12/31/2011,0,3.25,16.624,26.572
```

BarChart.py

```python
#--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)
```

```python
#--function for parsing string into a datetime
def mkdate(text):
    return dt.datetime.strptime(text, '%m/%d/%Y')
```

Creating a bar chart (2)

BarChart.py

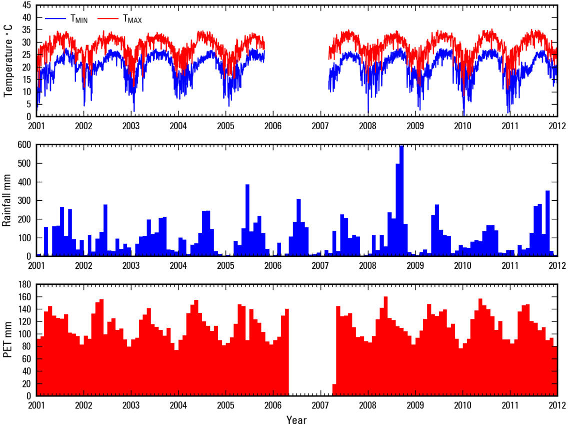
```
#--create monthly totals for rainfall and ETP
on_date = d['date'][0]
monthly_data, c = [], [0.0, 0.0]
for ipos,t in enumerate( d['date'] ):
    if t.month != on_date.month or ipos == len( d['date'] ) - 1:
        t_month = t_date.month
        t_day = 1 #int( t_date.day / 2 )
        t_year = t_date.year
        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 )
```

Creating a bar chart (3)

BarChart.py

```python
#--matplotlib date specification
years, months = mdates.YearLocator(), mdates.MonthLocator()  #every year, every month
yearsFmt = mdates.DateFormatter('%Y')
#--define the first subplot
ax = fig.add_subplot(3,1,1)
#--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')
```

Creating a bar chart (4)



Model coordinates

plotHeads.py

```
#--problem size
nlay, nrow, ncol = 1, 41, 40
#--coordinate information
dx, dy = 500., 500.
xOff, yOff = 0.0, 0.0
xcell = np.arange(xOff+dx/2.,xOff+(ncol*dx)+dx/2.0,dx)
ycell = np.arange(yOff+dy/2.,yOff+(nrow*dy)+dy/2.0,dy)
Xcell,Ycell = np.meshgrid(xcell,ycell)
xedge = np.arange(xOff,xOff+float(ncol)*dx+0.001,dx)
yedge = np.arange(yOff,yOff+float(nrow)*dy+0.001,dy)
Xedge,Yedge = np.meshgrid(xedge,yedge)
xmin,xmax = xOff, xOff+float(ncol)*dx
ymin,ymax = yOff, yOff+float(nrow)*dy
#--read MODFLOW data from external files and invert for plotting
ibound  = au.loadArrayFromFile(nrow,ncol,'..\\ref\\ibound.ref')
ibound   = np.flipud(ibound)
```

Binary head data

plotHeads.py

```python
#--get available times
headObj = mfb.MODFLOW_Head(nlay,nrow,ncol,head_file)
t = headObj.get_gage(1)
ntimes = t.shape[0]
mf_times = np.zeros( (ntimes), np.float )
for i in range(0,ntimes):
    mf_times[i] = t[i,0]


#--create figures for each output time
for ipos,on_time in enumerate( mf_times ):
    #--build output file name
    output_name = '{0}{1}_{2:05d}.{3}'.format(base_dir,base_name,int(ipos),extension)
    fnames.append( output_name )
    #--read head data
    headObj = mfb.MODFLOW_Head(nlay,nrow,ncol,head_file)
    totim,kstp,kper,h,success = headObj.get_record(on_time)
    hd = np.copy( h[0,:,:] )
```

Create the map (1)

plotHeads.py

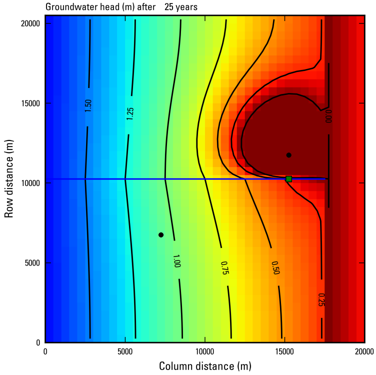
```
    #--invert rows for plotting and mask data in inactive areas
    hd       = np.flipud(hd)
    hd       = np.ma.masked_where(ibound<1,hd)


    #--figure
    ztf = figure(figsize=(4.0,4.0), facecolor='w')
    ztf.subplots_adjust(wspace=0.2,hspace=0.2,left=0.1,right=0.9,bottom=0.1,top=0.9)
    ax = ztf.add_subplot(1,1,1,aspect='equal')
    iyears = int( on_time / 365. )
    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)
```

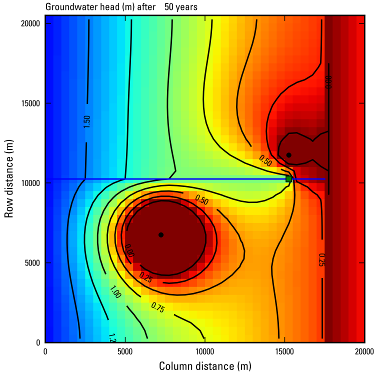
Create the map (2)

plotHeads.py

```
    hp = ax.pcolor(Xedge,Yedge,hd,\
                   vmin=0,vmax=2,cmap='jet_r',alpha=1.0,edgecolors='None')
    ch = ax.contour(xcell,ycell,hd,\
                    levels=hdcontour,colors='k',linewidths=1)
    ax.clabel(ch,inline=1,fmt='%5.2f',fontsize=6)
    ax.plot([xedge[0],xedge[35]],[ycell[20],ycell[20]],linewidth=1,color='b',label='River')
    ax.plot(xcell[struct_loc[1]],ycell[struct_loc[0]],'gs',markersize=4,label='Structure')
    ax.plot(xcell[well_loc[0,1]],ycell[well_loc[0,0]],'ko',markersize=3,label='PW-1')
    ax.plot(xcell[well_loc[1,1]],ycell[well_loc[1,0]],'ko',markersize=3,label='PW-2')
    #--plot limits
    ax.set_xlim(xmin,xmax)
    ax.set_ylim(ymin,ymax)
    xlabel('Column distance (m)')
    ylabel('Row distance (m)')
    #--save figure
    ztf.savefig(output_name,dpi=600)
    close(ztf)
```

Final maps


Groundwater head (m) after 25 years
Row distance (m)
Column distance (m)
0
5000
10000
15000
20000
0
5000
10000
15000
20000
1.50
1.25
1.00
0.75
0.50
0.25
0.00

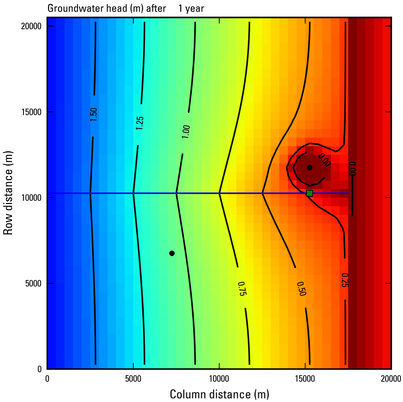

Groundwater head (m) after 50 years
Row distance (m)
Column distance (m)
0
5000
10000
15000
20000
0
5000
10000
15000
20000
1.50
0.00
0.50
0.50
0.00
0.25
1.00
0.75
1.2
0.25

Using ffmpeg.exe

plotHeads.py

```python
import sys
import os
import subprocess

#--animate head data
coutf = '{0}{1}.swf'.format(base_dir,base_name)
cline = 'ffmpeg.exe -i {0}{1}_%05d.png {2} -y'.format( base_dir,base_name,coutf )
try:
    os.remove(coutf)
except:
    print 'could not remove...{0}'.format( coutf )
subprocess.call(cline, stdin=None, stdout=None, stderr=None, shell=False)
#--delete temporary png files
for f in fnames:
    os.remove(f)
```

Using `ffmpeg.exe`

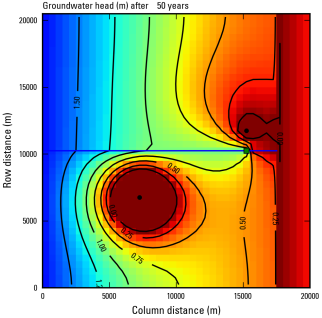


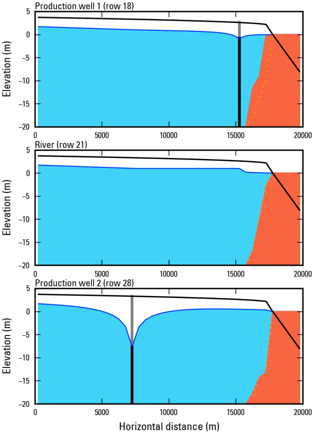
Adding cross-sections (1)

Cross-sectionSample.py

```
    ix = 23
    ax = ztf.add_subplot(3,2,2)
    text(0.0,1.01,'Production well 1 (row 18)',\
         horizontalalignment='left',verticalalignment='bottom',size=7,transform=ax.transAxes)
    t = np.copy( top[ix,:] )
    h = np.copy(  hd[ix,:] )
    h[35:] = 0.0
    z = np.copy(  zs[ix,:] )
    z[35:] = 0.0
    zt = np.copy(  z_steady[ix,:] )
    f  = ax.fill_between(xcell,y1=h,y2=z,color='#40d3f7')
    s  = ax.fill_between(xcell,y1=z,y2=-25.,color='#F76541')
    ax.plot(xcell,zt,linestyle=':',color='#FFA500')
    ax.plot(xcell,t,'k-',zorder=100)
    ax.plot([xcell[30],xcell[30]],[t[30],h[30]],linestyle='solid',color='0.5',linewidth=2)
    ax.plot([xcell[30],xcell[30]],[h[30],-25.],'k-',linewidth=2)
    ax.plot(xcell[0:36],h[0:36],'b-',linewidth=0.5)
    #--plot limits
    ax.set_xlim(xmin,xmax)
    ax.set_ylim(-20,5)
    ylabel('Elevation (m)')
```

Adding cross-sections (2)





Automating model runs and figure preparation

AutomationSample.bat

```
rem ***Sample 02
cd ..\SWRSample02\
mf2005-swr_x64.exe SWRSample02.nam
mf2005-swr_x64.exe SWRSample02.02.nam
cd ..\Python\
python SWRSample02.py
python SWRSample02.02.py
python SWRSample02v.py
python SWRSample02v.02.py
rem ***Sample 16
cd ..\SWRTestSimulation16\
mf2005-swr_x64.exe SWRTestSimulation16.level.nam
mf2005-swr_x64.exe SWRTestSimulation16.tilted.nam
mf2005-swr_x64.exe SWRTestSimulation16.tilted.IC.nam
cd ..\python\
python SWRTestSimulation16.level.py
python SWRTestSimulation16.tilted.py
python SWRTestSimulation16.tilted.IC.py
rem END OF BATCH FILE
cd ..\
pause
```