File "plottinginputs.py" contains function "readplotsettings" This function accepts 1 input pfilename for reading the user defined settings of the figure to be generated. It returns the objects required as input in the function "create_subplots"

In [1]:

```
def readplotsettings(pfilename):
    '''Function to read input for the plots'''
    import numpy as np

    print('Reading plot settings.')
    plot_inputs=np.loadtxt("plotdetails.txt", dtype='str', delimiter=';', skiprows=2)
    filenames = plot_inputs[0,1:5]
    figtitles = plot_inputs[1,1:5]
    xtitles = plot_inputs[2,1:5]
    ytitles = plot_inputs[3,1:5]
    xcoord = plot_inputs[4,1:5]
    ycoord = plot_inputs[5,1:5]
    xcoord = xcoord.astype('float64')
    ycoord = ycoord.astype('float64')
    print('Done')

    return (filenames, figtitles, xtitles, ytitles, xcoord, ycoord) # pack the data read from the file into a tup
```

This is a function to map and scale colorbar in the plots. "cbar_axismap" function is saved in file "colormap.py" and imported in function "create_subplot" in file "plotting.py".

"cbar_axismap" function accepts 2 arguments, mappable and ctitle. Use ctitle ="sample text" to set colorbar title.

In [2]:

```
def cbar_axismap(mappable, ctitle='set title here'):
    '''Function to control the size of colorbar (aspect ratio) in the
      figure with 2 or 4 subplots.'''
    from mpl_toolkits.axes_grid1 import make_axes_locatable
    import matplotlib.pyplot as plt

    last_axes = plt.gca()
    ax = mappable.axes
    fig = ax.figure
    divider = make_axes_locatable(ax)
    cax = divider.append_axes('right', size='5%', pad=0.05)
    cbar = fig.colorbar(mappable, cax=cax, format='%.0e')
    cbar.set_label(ctitle, rotation=270, size=12, labelpad=15)
    plt.sca(last_axes)

    return cbar
```

File "plotting.py" contains functions "create_subplots", "read2ofiles" and "read4ofiles". "read2ofiles" and "read4ofiles" are the functions to read the simulation results for visualization and these functions are called by the function "create_subplots".

"create_subplots" accepts 9 arguments. Use nplots = 3 or 5, to generatre a figure with 2 subplots or 4 subplots, respectively. fname = filename, a list of file names where simulations results are saved. ftitle = figure titles, a user defined string list to set subplot titles xtitle = x-axis label, a user defined string list to set x-axis label ytitle = y-axis label, a user defined string list to set y-axis label\ a = coordinates of x and y rectangular patch 1, list of floating point numbers (leave this unchanged if patch1 = 'false') b = coordinates of x and y rectangular patch 2, list of floating point numbers (leave this unchanged if patch2 = 'false') patch1 = True or False, default = False patch2 = True or False, default = False

In [3]:

```
def create_subplots(nplots, fname, ftitle, xtitle, ytitle, a, b, patch1='False', patch2='False' ):
    '''Function to create a figure with 2 or 4 subplots.'''
    import matplotlib.pyplot as plt
    import matplotlib.patches as patches
    #from colorbar import colorbar

    print("Formatting figure fonts.")
    plt.rc('font', family='serif', size=12) # assign fonts and size in the figure

    print("Creating figures axes.")
    fig, ax = plt.subplots(figsize=(10,8), dpi=150) # assign size of plot and resolution


    da = a[1] - a[0]
    db = b[1] - b[0]
    dm = a[3] - a[2]
    dn = b[3] - b[2]

    if nplots == 5:
        print("Reading files.")
        x1, y1, p1, x2, y2, p2, x3, y3, p3, x4, y4, p4, im, jm = read4ofiles(fname)
    elif nplots == 3:
        print("Reading files.")
        x1, y1, p1, x2, y2, p2, im, jm = read2ofiles(fname)
    else:
        print('Incorrect values of nplots.')
        print('STOP')
        sys.exit()

    imax = im[0]
    jmax = jm[0]
    xmin = x1[0][0]
    xmax = x1[imax-1][jmax-1]
    ymin = y1[0][0]
    ymax = y1[imax-1][jmax-1]

    print('\nGenerating data plot.')
    for i in range (1,nplots):
        if nplots == 5:
            ax = plt.subplot(2,2,i)
        elif nplots == 3:
            ax = plt.subplot(1,2,i)

        if i == 1:
            img1 = plt.pcolormesh(x1, y1, p1, cmap='jet', shading='gouraud')
            cbar_axismap(img1)
            if patch1 == 'True':
                rect1 = patches.Rectangle((a[0], b[0]), da, db, edgecolor='r', facecolor='none')
                ax.add_patch(rect1)
            if patch2 == 'True':
                rect2 = patches.Rectangle((a[2], b[2]), dm, dn, edgecolor='c', facecolor='none')
                ax.add_patch(rect2)

        elif i == 2:
            img2 = plt.pcolormesh(x2, y2, p2, cmap='jet', shading='gouraud')
            cbar_axismap(img2)
            if patch1 == 'True':
                rect1 = patches.Rectangle((a[0], b[0]), da, db, edgecolor='r', facecolor='none')
                ax.add_patch(rect1)
            if patch2 == 'True':
                rect2 = patches.Rectangle((a[2], a[2]), dm, dn, edgecolor='c', facecolor='none')
                ax.add_patch(rect2)

        elif nplots == 5 and i == 3:
            img3 = plt.pcolormesh(x3, y3, p3, cmap='jet', shading='gouraud')
            cbar_axismap(img3)
            if patch1 == 'True':
                rect1 = patches.Rectangle((a[0], b[0]), da, db, edgecolor='r', facecolor='none')
                ax.add_patch(rect1)
            if patch2 == 'True':
                rect2 = patches.Rectangle((a[2], b[2]), dm, dn, edgecolor='c', facecolor='none')
                ax.add_patch(rect2)

        elif nplots == 5 and i == 4:
            img4 = plt.pcolormesh(x4, y4, p4, cmap='jet', shading='gouraud')
            cbar_axismap(img4)
            if patch1 == 'True':
                rect1 = patches.Rectangle((a[0], b[0]), da, db, edgecolor='r', facecolor='none')
                ax.add_patch(rect1)
            if patch2 == 'True':
                rect2 = patches.Rectangle((a[2], b[2]), dm, dn, edgecolor='c', facecolor='none')
                ax.add_patch(rect2)

        plt.xlabel(str(xtitle[i-1]), size=12)
        plt.ylabel(str(ytitle[i-1]), size=12)
        plt.title(str(ftitle[i-1]), size=12)
        plt.xlim(xmin, xmax)
        plt.ylim(ymin, ymax)
        #plt.clim(-0.03,0.03)
        plt.tight_layout()

        plt.gca().set_aspect('equal', adjustable='box') # set aspect ratio of the plot

    plt.show()
    return
```

In [4]:

```
def read2ofiles(filename):
    '''Function to read 2 output files for visualization'''
    import numpy as np
    import math

    print("Reading plotting data.")
    x1, y1, p1 = np.loadtxt(filename[0], delimiter='\t', unpack=True)
    x2, y2, p2 = np.loadtxt(filename[1], delimiter='\t', unpack=True)

    print("Calculating variables.")
    im1 = int(math.sqrt(len(x1)))
    jm1 = int(math.sqrt(len(y1)))
    im2 = int(math.sqrt(len(x2)))
    jm2 = int(math.sqrt(len(y2)))

    IM = [im1, im2]
    JM = [jm1, jm2]

    print("Organizing data.")
    X1 = np.reshape(x1, (im1,jm1))
    Y1 = np.reshape(y1, (im1,jm1))
    P1 = np.reshape(p1, (im1,jm1))
    X2 = np.reshape(x2, (im2,jm2))
    Y2 = np.reshape(y2, (im2,jm2))
    P2 = np.reshape(p2, (im2,jm2))

    return (X1, Y1, P1, X2, Y2, P2, IM, JM)
```

In [5]:

```
def read4ofiles(filename):
    '''Function to read 4 output files for visualization'''
    import numpy as np
    import math

    print("Reading plotting data.")
    x1, y1, p1 = np.loadtxt(filename[0], unpack=True)
    x2, y2, p2 = np.loadtxt(filename[1], delimiter='\t', unpack=True)
    x3, y3, p3 = np.loadtxt(filename[2], delimiter='\t', unpack=True)
    x4, y4, p4 = np.loadtxt(filename[3], delimiter='\t', unpack=True)

    print("Calculating variables.")
    im1 = int(math.sqrt(len(x1)))
    jm1 = int(math.sqrt(len(y1)))
    im2 = int(math.sqrt(len(x2)))
    jm2 = int(math.sqrt(len(y2)))
    im3 = int(math.sqrt(len(x3)))
    jm3 = int(math.sqrt(len(y3)))
    im4 = int(math.sqrt(len(x4)))
    jm4 = int(math.sqrt(len(y4)))

    IM = [im1, im2, im3, im4]
    JM = [jm1, jm2, jm3, jm4]

    print("Organizing data.")
    X1 = np.reshape(x1, (im1,jm1))
    Y1 = np.reshape(y1, (im1,jm1))
    P1 = np.reshape(p1, (im1,jm1))
    X2 = np.reshape(x2, (im2,jm2))
    Y2 = np.reshape(y2, (im2,jm2))
    P2 = np.reshape(p2, (im2,jm2))
    X3 = np.reshape(x3, (im3,jm3))
    Y3 = np.reshape(y3, (im3,jm3))
    P3 = np.reshape(p3, (im3,jm3))
    X4 = np.reshape(x4, (im4,jm4))
    Y4 = np.reshape(y4, (im4,jm4))
    P4 = np.reshape(p4, (im4,jm4))

    return (X1, Y1, P1, X2, Y2, P2, X3, Y3, P3, X4, Y4, P4, IM, JM)
```

Now let's run the scripts: see example below

See saved file "plotdetails.txt" which contains the user defined inputs and settings.

Call function "readplotsettings" which returns the packed tuple. To unpack the tuple, create 6 objects as shown an run this script (Shift+Enter in Jupyter notebook)

```
In [6]: fn, ft, xt, yt, x1, y1 = readplotsettings("plotdetails.txt") # unpack tuple into 6 variables
```

```
Reading plot settings.

Done
```

Then, generate plot by the calling the function "create_subplots" as shown in the script below

To generate 4 subplots, use nplots = 5 and use the lists stored in

```
In [7]: create_subplots(nplots=5, fname=fn, ftitle=ft, xtitle=xt, ytitle=yt, a=x1, b=y1)
```

```
Formatting figure fonts.

Creating figures axes.

Reading files.

Reading plotting data.

Calculating variables.

Organizing data.


Generating data plot.
```

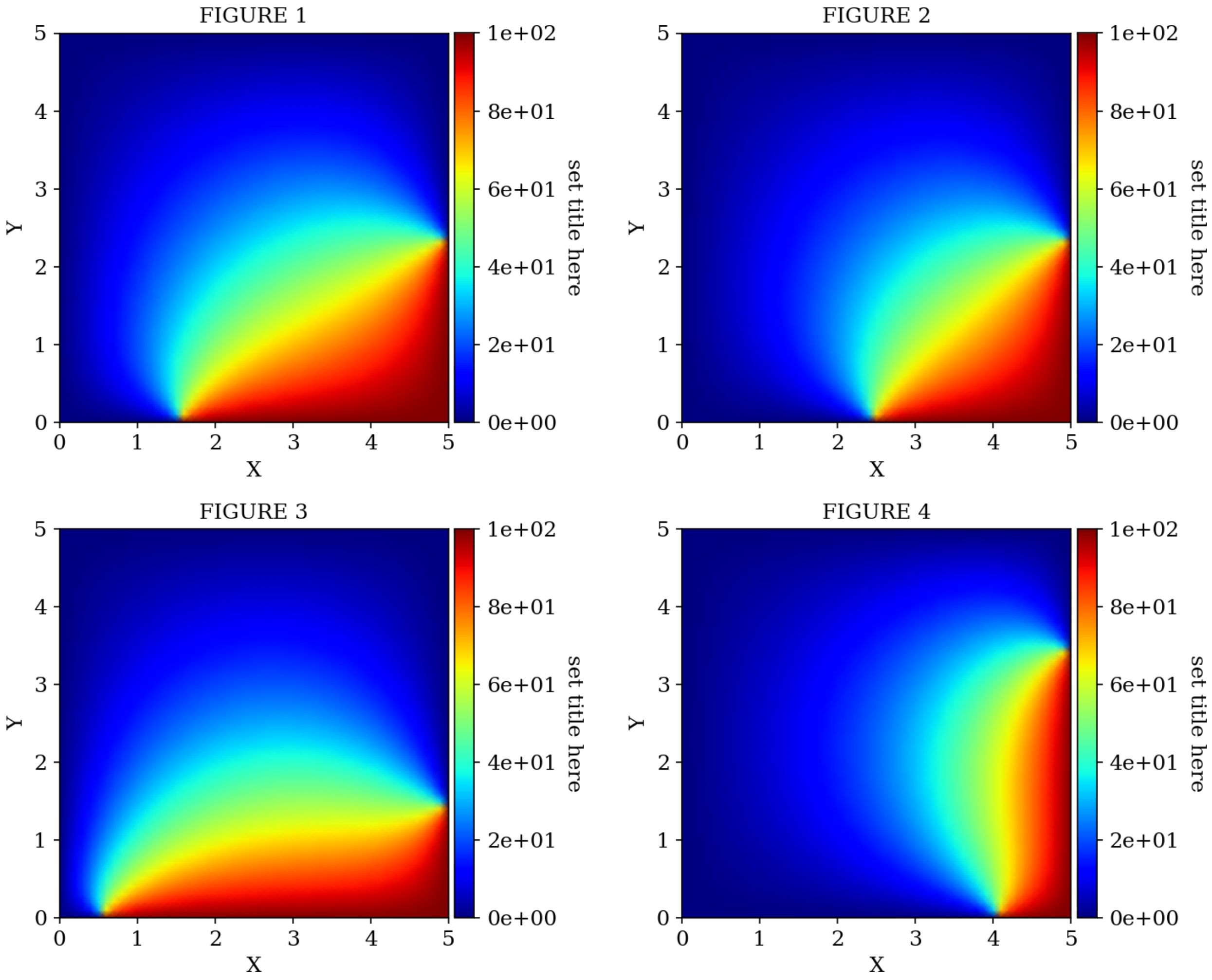
FIGURE 1
FIGURE 2
FIGURE 3
FIGURE 4
X
Y
0
1
2
3
4
5
0e+00
2e+01
4e+01
6e+01
8e+01
1e+02
set title here

In [ ]: