The logic of zoom/Pan:

My Requirement is I have to zoom/pan with a buttons. The frame work I am using is PyQt. The following logic I have used. When I test with terminal it works well. The same when I called with button in an GUI, it shows strange behavior. I am using Ubuntu 10.4. Since my graph/plot often switch from log to linear scale, I have choosen algorithm like this.

Please help me to clear the problem.

The axes coordinates remains same over all the axes and extends from (0,0), to (1,1).

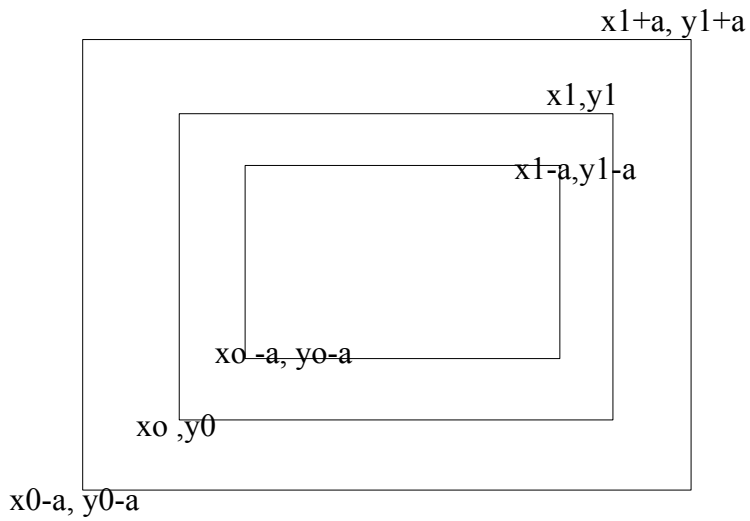


Fig (tool.a)

The data coordinates is the coordinate against which data are plotted. The pixel coordinate is the coordinate where all the details picture is present.

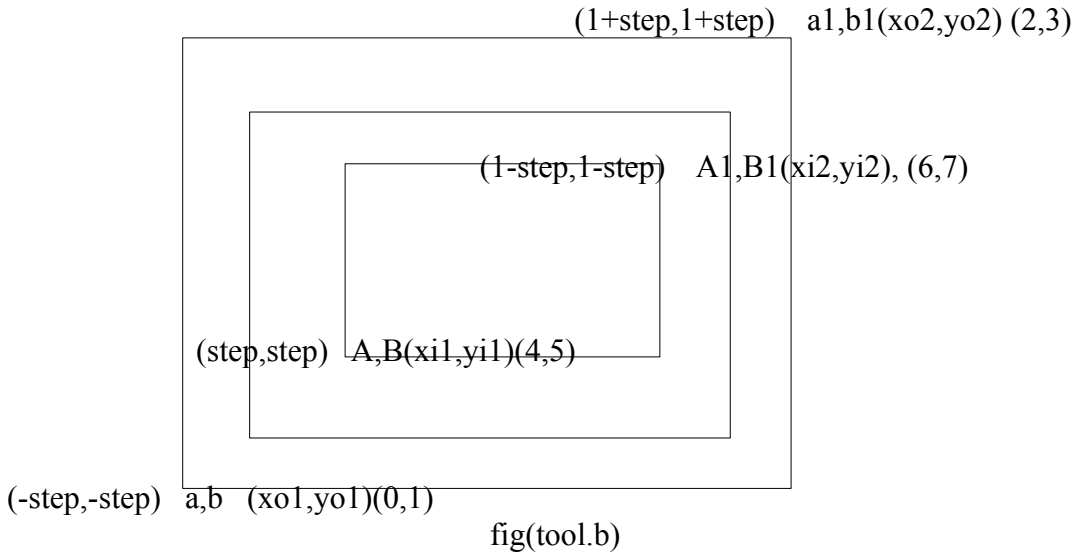
1. In the fig, a rectangle from (xo,yo) to (x1, y1) represents the orginal plotted area.
2. The inner rectangle area shows area we are going to zoom out
3. The outer rectangle area shows area we are going to zoom in
4. All the points in the above fig show in Axes cordinate which never gets changed.

Algorithm:

1. Firstly we get the details of steps to reach from one of the graph to another end.
2. a = 1/step, # single step
3. Compute various points in the fig.
4. Covert all the points to pixels coordinate system. It too remains same unless the subplot area is changed. We never going to change this in our project. So, in the initialization part we

convert these points into pixels preferably with a method. Which we can connect with area changed signal of the subplot.

5. For any operation we convert the pixel point to data coordinate point and set the limit and draw.



fig(tool.b)

The code:

```
from PyQt4 import QtGui, QtCore
from matplotlib.backends.backend_qt4agg import FigureCanvasQTAgg as FigureCanvas

from matplotlib.figure import Figure


class MplCanvas(FigureCanvas):
   """Class to represent the FigureCanvas widget"""


   def __init__(self):
      # setup Matplotlib Figure and Axis
      self.fig = Figure()
      self.ax = self.fig.add_subplot(111)
      self.ax.set_autoscale_on(True)
      self.ax.hold(False)

      # initialization of the canvas
      FigureCanvas.__init__(self, self.fig)
      # we define the widget as expandable
      FigureCanvas.setSizePolicy(self,
                           QtGui.QSizePolicy.Expanding,
                           QtGui.QSizePolicy.Expanding)
      # notify the system of updated policy
      FigureCanvas.updateGeometry(self)



class MplWidget(QtGui.QWidget):
   """Widget defined in Qt Designer"""
   def __init__(self, parent = None):
```

```
        # initialization of Qt MainWindow widget
        QtGui.QWidget.__init__(self, parent)
        # set the canvas to the Matplotlib widget
        self.canvas = MplCanvas()
        # create a vertical box layout
        self.vbl = QtGui.QVBoxLayout()
        # add mpl widget to the vertical box
        self.vbl.addWidget(self.canvas)
        # set the layout to the vertical box
        self.setLayout(self.vbl)




class Toolbar:
    def __init__(self, fig, step = 20, cursor = None):
        self.step = (2.0/step)
        self.fig  = fig
        self.cursor = cursor


        self.pix = self.toPix()

    def toPix(self):
        ax = self.fig.gca()
        step= self.step
        x1,y1 = ax.transAxes.transform((-step,-step))
        x2,y2 = ax.transAxes.transform((1+step,1+step))
        x3,y3 = ax.transAxes.transform((step,step))
        x4,y4 = ax.transAxes.transform((1-step,1-step))
        return (x1,y1,x2,y2,x3,y3,x4,y4)

    def toData(self):
        ax = self.fig.gca()
        pix = self.pix
        a1,b1 = ax.transData.inverted().transform((pix[0],pix[1]))
        a2,b2 = ax.transData.inverted().transform((pix[2],pix[3]))
        a3,b3 = ax.transData.inverted().transform((pix[4],pix[5]))
        a4,b4 = ax.transData.inverted().transform((pix[6],pix[7]))
        return (a1,b1,a2,b2,a3,b3,a4,b4)

    def zoomHplus(self):
        dat = self.toData()
        ax = self.fig.gca()
        ax.set_xlim(dat[4], dat[6])
        self._update()
    def zoomHminus(self):
        dat = self.toData()
        ax = self.fig.gca()
        ax.set_xlim(dat[0], dat[2])
        self._update()
    def zoomVplus(self):
        dat = self.toData()
        ax = self.fig.gca()
        ax.set_ylim(dat[5], dat[7])
        self._update()
    def zoomVminus(self):
        dat = self.toData()
        ax = self.fig.gca()
        ax.set_ylim(dat[1], dat[3])
        self._update()
```

```
    def panHplus(self):
        dat = self.toData()
        ax = self.fig.gca()
        ax.set_xlim(dat[4], dat[2])
        self._update()
    def panHminus(self):
        dat = self.toData()
        ax = self.fig.gca()
        ax.set_xlim(dat[0], dat[6])
        self._update()
    def panVplus(self):
        dat = self.toData()
        ax = self.fig.gca()
        ax.set_ylim(dat[5], dat[3])
        self._update()
    def panVminus(self):
        dat = self.toData()
        ax = self.fig.gca()
        ax.set_ylim(dat[1], dat[7])
        self._update()

    def _update(self):
        a = self.fig.gca()
        xaxis = getattr(a, 'xaxis', None)
        yaxis = getattr(a, 'yaxis', None)
        locators = []
        if xaxis is not None:
            locators.append(xaxis.get_major_locator())
            locators.append(xaxis.get_minor_locator())
        if yaxis is not None:
            locators.append(yaxis.get_major_locator())
            locators.append(yaxis.get_minor_locator())

        for loc in locators:
            loc.refresh()

        self.fig.canvas.draw_idle()


if __name__ == "__main__":
    import sys
    import numpy as np
    from PyQt4.QtCore import QTimer

    app = QtGui.QApplication(sys.argv)
    mp = MplWidget()

    t = np.arange(0.0, 3.0, 0.01)
    s = np.cos(2*np.pi*t)
    mp.canvas.ax.plot(t, s)


    mp.show()
    sys.exit(app.exec_())
```