TBfile

```
import os.path
import utils
import numpy as np

class TBfile(object):
    def __init__(self,directory='Output'):
        self.header = {}
        self.ftype = 'spec'  # or 'img'
        self.directory = directory

    def __str__(self):
        print 'TBfile'

    def write(self,fn=None,directory=None):
        print "For now, this is done in planet.py starting at line 163"
        print "Eventually pull it over to TBfile.py"

    def flist(self,fd=None,directory=None):
        """This generates the list of filenames to be opened - doesn't check for existence"""
        if directory is not None:
            usedir = directory
        else:
            usedir = self.directory
        file_list = utils.ls(directory=usedir, show=False, returnList=True)

        get_fn = True
        files = []
        if type(fd) == int:
            ifile = [fd]
        elif fd == '?' or fd == None:
            for i,fn in enumerate(file_list):
                print '%d  -  %s' % (i,fn)
            sfile = raw_input('File numbers: ')
            if '-' in sfile:
                sfile = sfile.split('-')
                ifile = range(int(sfile[0]),int(sfile[1])+1)
            else:
                if ',' in sfile:
                    sfile = sfile.split(',')
                else:
                    sfile = sfile.split()
                ifile = []
                for i in sfile:
                    ifile.append(int(i))
        elif type(fd) == list and type(fd[0]) == int:
            ifile = fd
        elif type(fd) == list and type(fd[0]) == str:
            files = fd
        else:
            get_fn = False
            files = [fd]

        if get_fn:
            for i,fn in enumerate(file_list):
                if i in ifile:
                    files.append(fn)
```

```
    return files

def read(self,fd=None,directory=None):
    """reads brightness temperature file(s):
       fn = filename to read (but then ignores directory) | '?', '.' or None | integer [None]
       directory = subdirectory for data (not used if filename given) ['Output']"""

    if directory is not None:
        usedir = directory
    else:
        usedir = self.directory
    try_files = self.flist(fd,usedir)

    ### Read in two passes:  first header
    self.ftype = None
    headerText = []
    self.files = []
    for filename in try_files:
        try:
            fp = open(filename,'r')
        except IOError:
            print filename+" not found - removing from list"
            continue
        print "Reading "+filename
        self.files.append(filename)

        ## Get past any header and get first line
        line ='# file:  '+filename+'\n'
        for line in fp:
            if line[0]=='#':
                headerText.append(line)
                if 'img_size' in line:
                    self.ftype = 'img'
                #line = fp.readline()
            if line[0] == 'U':
                self.ftype = 'spec'
                spec_b = line
        fp.close()
    self.parseHeader(headerText)

    ### Second pass reads the data, split into 'img' and 'spec' types
    if self.ftype == 'img':
        imRow = imCol = 0
        data = []
        for filename in self.files:
            fp = open(filename,'r')
            for line in fp:
                if line[0]=='#':
                    continue
                imRow+=1
                vdat = []
                sdat = line.split()
                imCol = 0
                for v in sdat:
                    imCol+=1
                    vdat.append(float(v))
                data.append(vdat)
            fp.close()
        self.header['img_filename'] = filename
```

```
        self.header['img_size'] = [imRow,imCol]
        self.data = np.array(data)
        self.xyextents = [-self.resolution*len(self.data)/2.0,self.resolution*len(self.data)/2.0,-self!
          .resolution*len(self.data)/2.0,self.resolution*len(self.data)/2.0]
        self.x = []
        self.y = []
        for i in range(len(self.data)):
            self.x.append(self.xyextents[0] + i*self.resolution)
            self.y.append(self.xyextents[2] + i*self.resolution)
        self.x = np.array(self.x)
        self.y = np.array(self.y)
    elif self.ftype == 'spec':
        bvals = []
        indata = []
        f = []
        wavel = []
        n = 0
        for filename in self.files:
            fp = open(filename,'r')
            for line in fp:
                if line[0] == '#':
                    continue
                if line[0] == 'U':
                    labels = line.split()
                    xlabel = labels[0]
                    del(labels[0])
                    print 'b = ',
                    for b in labels:
                        print ' '+b,
                        bb = b.split('(')[1].strip(')').split(',')
                        bb = [float(bb[0]),float(bb[1])]
                        bvals.append(bb)
                    b = np.array(bvals)
                    print ''
                    ylabels = labels
                    continue
                indata = line.split()
                f.append(float(indata[0]))
                wavel.append((speedOfLight/1E7)/float(indata[0]))
                del(indata[0])
                t = []
                for d in indata:
                    t.append(float(d))
                data.append(t)
                n+=1
            fp.close()
        self.data = np.array(indata).transpose()
        self.f = np.array(f)
        self.wavel = np.array(wavel)
        print 'Freq:  %.3f - %.3f %s  (%d points)' % (f[0],f[-1],xlabel,n)


def parseHeader(self, headerText):
    """Parses the pyPlanet image header"""
    for hdr in headerText:
        hdr = hdr.strip('#').strip()
        print hdr
        updateKey = False
        if ':' in hdr:
            updateKey = True
```

```
                hdrkey = hdr.split(':')[0].split()[0]
                hdr = hdr.split(':')[1]
                h = []
                hdr = hdr.split()
                for dat in hdr:
                    dat = dat.strip('[').strip(']').strip(',')
                    try:
                        h.append(float(dat))
                    except ValueError:
                        h.append(dat)
            else:
                hdrkey = hdr.split()[0]
                if not self.header.has_key(hdrkey):
                    updateKey = True
                    h = [None]
            if updateKey:
                self.header[hdrkey] = h
                self.__dict__[hdrkey] = h
        ### set any header-derived values
        if self.header.has_key('res'):
            self.resolution = self.header['res'][0]   # keep this for backward compatibility
        if self.header.has_key('freqs'):
            self.freq = self.header['freqs'][0]
            self.header['band'] = utils.getRFband(self.freq,'GHz')



def plotTB(fn=None,xaxis='Frequency',xlog=False, justFreq=False, directory='Output',distance!
  =4377233696.68):
    """plots brightness temperature against frequency and disc location:
           fn = filename to read (but then ignores directory) | '?', '.' or None | integer [None]
           xaxis = 'f[requency]' | 'w[avelength' ['freq']
           xlog = True | False [False]
           justFreq = True | False [False]
           directory = subdirectory for data (not used if filename given) ['Output']
           distance = distance for angular size plot in km [4377233696 km for Neptune]"""

    filename,Tb,f,wavel,b,xlabel,ylabels = readTB(fn=fn,directory=directory)
    title = filename.split('/')

    ## Frequency plot
    plt.figure('TB')
    for i in range(len(b)):
        if xaxis[0].lower() == 'f':
            plotx = f
        else:
            plotx = wavel
            xlabel='Wavelength [cm]'
        if xlog:
            plt.semilogx(plotx,Tb[i],label=ylabels[i])
        else:
            plt.plot(plotx,Tb[i],label=ylabels[i])
        #plt.plot(f,Tb[i],'x')
    #plt.legend()
    plt.xlabel(xlabel)
    plt.ylabel('Brightness Temperature [K]')
    plt.title(title[-1])

    if justFreq:
        return len(f)
```

```
## b plot
plt.figure('b')
for i in range(len(f)):
    plt.plot(b[:,0],Tb[:,i],label=str(f[i]))
    plt.plot(b[:,0],Tb[:,i],'o')
plt.legend()
plt.title(title[-1])
plt.xlabel('km')
plt.ylabel('Brightness Temperature [K]')

## b plot vs angle
angle = []
for r in b[:,0]:
    angle.append( (r/distance)*(180.0/np.pi)*3600.0 )
plt.figure('b_vs_angle')
for i in range(len(f)):
    plt.plot(angle,Tb[:,i],label=str(f[i]))
    plt.plot(angle,Tb[:,i],'o')
plt.legend()
plt.title(title[-1])
plt.xlabel('arcsec')
plt.ylabel('Brightness Temperature [K]')

return len(b)
```