Python Biax Data Reader: biaxread

J.R. Leeman

May 9, 2014

1 Purpose

This script was designed to read data into an easy to use format from xlook output in ASCII or binary form. Data is read, the header parsed for column names, lengths, etc., and a rec array or Pandas dataframe object returned that is easy to access and call from within a script. Column names are used to call the data columns, so as long as consistent naming is used the column order in the file is irrelevant.

2 Description

2.1 ASCII Files

The function ReadAscii first opens the text file with a standard open command. We know that each column in the data is written as 12 characters wide. The first line of the header is the number of records, the second is the column number, the third the column headings, the fourth the column units, and the fifth the number of records in each column. This information if parsed and stored. Numerical data is read into an array. A rec array is created with the numpy package and the labels of the column names that were parsed or the data is converted into a dataframe object.

2.2 Binary Files

The function ReadBin opens the binary file in 'read binary' mode. The initial file information is stored as follows (in big-endian format):

Information	Format	Bytes
Name	20 characters	20
Number of Columns	int	4
Sweep	int	4
Date/Time	int	4

For each possible column there are 84 bytes of information. There are 32 possible columns. We read the information for each column and store information for columns that have data. Blank columns are identified by the first 6 characters of the name 'no_val'.

Information	Format	Bytes
Name	13 characters	13
Units	13 characters	13
Gain	int	4
Comment	50 characters	50
Number of Elements	int	4

Data is stored as the default machine format, which for modern Intel based computers in little-endian. This is the default for the module, but big-endian can be specified. Each column is written out as a sequence of doubles that are n_elem long. Doubles are read and stored in a numpy array. The array is then combined with datatype information collected from the headers and stored as a rec array.

To find the default machine binay format run: *python -c "import struct; print 'little' if ord(struct.pack('L', 1)[0]) else 'big'"*

3 Cautions

A few cautions should be observed when using the biaxread script:

1. The empty array is shaped by row number information from the header. If a datafile is cut, but the header is left unmodified there will be many extra zero data pairs at the end of the array.

4 Usage

4.1 Return Formats

By default data will be returned as a Numpy record array (recarray). Data can also be returned as a Pandas dataframe object that is indexed on the row number of the data. To do this, set *pandas=True* in the function call to either ReadAscii or ReadBin.

4.2 ASCII Files

First process the experimental data in xlook and output a text file with headers. To do this input *type 0 -1 1 12 pxxxx_data.txt* at the xlook command line or add it to the data reduction file. Be sure to correct any column or row number specifications here. Start IPython or open your own Python script. Import biaxread and pass the function ReadAscii the name of the text output file from xlook. The array is now returned and ready to use.

4.3 Binary Files

Process the experimental data in xlook and ouput a binary file. This is done with the 'write' command with the syntax *write filename*. In IPython or your Python script import biaxread and pass the function ReadBin the name of the binary file. The array is returned and ready to use.

Files that were written in a big-endian format by setting dataendianness to 'big'. The function call would look like *ReadBin(filename,dataendianness='big')*.

5 Acknowledgements

Thank you to Marco Scuderi for using the code and being patient as little issues were worked out. Please send any bug reports to kd5wxb@gmail.com or as an issue in the github repository.

6 Code

```
1
2  import numpy as np
3  import struct
4  import pandas as pd
5
6  def ReadAscii(filename,pandas=False):
7      """
8      Takes a filename containing the text output (with headers) from xlook and
9      reads the columns into a rec array or dataframe object for easy data
10     processing and access.
11     """
12
13     try:
14         f = open(filename,'r')
15     except:
16         print "Error Opening %s" %filename
17         return 0
18
19     col_width = 12 # Columns are 12 char wide in header
20
21     # First line of the file is the number of records
22     num_recs = f.readline()
23     num_recs = int(num_recs.strip('number of records = '))
24     print "\nNumber of records: %d" %num_recs
25
26     # Second line is column numbers, we don't care so just count them
27     num_cols = f.readline()
28     num_cols = num_cols.split('col')
29     num_cols = len(num_cols)
30     print "Number of columns: %d" %num_cols
31
32     # Third line is the column headings
33     col_headings_str = f.readline()
34     col_headings_str = col_headings_str[5:-1]
35     col_headings = ['row_num'] # Row number the the first (unlabeled) column
36     for i in xrange(len(col_headings_str)/12):
37         heading = col_headings_str[12*i:12*i+12].strip()
38         col_headings.append(heading)
39
40     # Fourth line is column units
41     col_units_str = f.readline()
42     col_units_str = col_units_str[5:-1]
43     col_units=[]
44     for i in xrange(len(col_units_str)/12):
45         heading = col_units_str[12*i:12*i+12].strip()
46         col_units.append(heading)
47     col_units = [x for x in col_units if x != '\n'] #Remove newlines
48
49     # Fifth line is number of records per column
50     col_recs = f.readline()
51     col_recs = col_recs.split('recs')
52     col_recs = [int(x) for x in col_recs if x != '\n']
53
```

```
    # Show column units and headings
    print "\n\n-------------------------------------------------"
    print "|%15s|%15s|%15s|" %('Name','Unit','Records')
    print "-------------------------------------------------"
    for column in zip(col_headings,col_units,col_recs):
        print "|%15s|%15s|%15s|" %(column[0],column[1],column[2])
    print "-------------------------------------------------"

    # Read the data into a numpy recarray
    dtype=[]
    #dtype.append(('row_num','float'))
    for name in col_headings:
        dtype.append((name,'float'))
    dtype = np.dtype(dtype)

    data = np.zeros([num_recs,num_cols])

    i=0
    for row in f:
        row_data = row.split()
        for j in xrange(num_cols):
            data[i,j] = row_data[j]
        i+=1

    f.close()

    if pandas==True:
        # If a pandas object is requested, make a data frame
        # indexed on row number and return it
        dfo  = pd.DataFrame(data,columns=col_headings)
        dfo = dfo.set_index('row_num')
        return dfo

    else:
        # Otherwise return the default (Numpy Recarray)
        data_rec = np.rec.array(data,dtype=dtype)
        return data_rec

def ReadBin(filename,dataendianness='little',pandas=False):
    """
    Takes a filename containing the binary output from xlook and
    reads the columns into a rec array or dataframe object for easy
    data processing and access.

    The data section of the file is written in the native format of the machine
    used to produce the file. Endianness of data is little by default, but may
    be changed to 'big' to accomodate older files or files written on power pc
    chips.
    """

    try:
        f = open(filename,'rb')
    except:
        print "Error Opening %s" %filename
        return 0
```

```
109
110     col_headings = []
111     col_recs     = []
112     col_units    = []
113
114     # Unpack information at the top of the file about the experiment
115     name = struct.unpack('20c',f.read(20))
116     name = ''.join(str(i) for i in name)
117     name = name.split("\0")[0]
118     print "\nName: ",name
119
120     # The rest of the header information is written in big endian format
121
122     # Number of records (int)
123     num_recs = struct.unpack('>i',f.read(4))
124     num_recs = int(num_recs[0])
125     print "Number of records: %d" %num_recs
126
127     # Number of columns (int)
128     num_cols = struct.unpack('>i',f.read(4))
129     num_cols = int(num_cols[0])
130     print "Number of columns: %d" %num_cols
131
132     # Sweep (int) - No longer used
133     swp = struct.unpack('>i',f.read(4))[0]
134     print "Swp: ",swp
135
136     # Date/time(int) - No longer used
137     dtime = struct.unpack('>i',f.read(4))[0]
138     print "dtime: ",dtime
139
140     # For each possible column (32 maximum columns) unpack its header
141     # information and store it. Only store column headers of columns
142     # that contain data. Use termination at first NUL.
143     for i in range(32):
144
145         # Channel name (13 characters)
146         chname = struct.unpack('13c',f.read(13))
147         chname = ''.join(str(i) for i in chname)
148         chname = chname.split("\0")[0]
149
150         # Channel units (13 characters)
151         chunits = struct.unpack('13c',f.read(13))
152         chunits = ''.join(str(i) for i in chunits)
153         chunits = chunits.split("\0")[0]
154
155         # This field is now unused, so we just read past it (int)
156         gain = struct.unpack('>i',f.read(4))
157
158         # This field is now unused, so we just read past it (50 characters)
159         comment = struct.unpack('50c',f.read(50))
160
161         # Number of elements (int)
162         nelem = struct.unpack('>i',f.read(4))
163         nelem = int(nelem[0])
```

```
164
165         if chname[0:6] == 'no_val':
166             continue # Skip Blank Channels
167         else:
168             col_headings.append(chname)
169             col_recs.append(nelem)
170             col_units.append(chunits)
171
172
173     # Show column units and headings
174     print "\n\n------------------------------------------------"
175     print "|%15s|%15s|%15s|" %('Name','Unit','Records')
176     print "------------------------------------------------"
177     for column in zip(col_headings,col_units,col_recs):
178         print "|%15s|%15s|%15s|" %(column[0],column[1],column[2])
179     print "------------------------------------------------"
180
181     # Read the data into a numpy recarray
182     dtype=[]
183     for name in col_headings:
184         dtype.append((name,'double'))
185     dtype = np.dtype(dtype)
186
187     data = np.zeros([num_recs,num_cols])
188
189     for col in range(num_cols):
190         for row in range(col_recs[col]):
191             if dataendianness == 'little':
192                 data[row,col] = struct.unpack('<d',f.read(8))[0]
193             elif dataendianness == 'big':
194                 data[row,col] = struct.unpack('>d',f.read(8))[0]
195             else:
196                 print "Data endian setting invalid, please check and retry"
197                 return 0
198
199     data_rec = np.rec.array(data,dtype=dtype)
200
201     f.close()
202
203     if pandas==True:
204         # If a pandas object is requested, make a data frame
205         # indexed on row number and return it
206         dfo  = pd.DataFrame(data,columns=col_headings)
207         # Binary didn't give us a row number, so we just let
208         # pandas do that and name the index column
209         dfo.index.name = 'row_num'
210         return dfo
211
212     else:
213         # Otherwise return the default (Numpy Recarray)
214         data_rec = np.rec.array(data,dtype=dtype)
215         return data_rec
```