Python Biax Data Reader: biaxread

J.R. Leeman

July 19, 2013

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 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. Finally the array is populated with data and the final product returned to the calling script.

2.2 Binary Files

The function ReadBin opens the binary file in 'read binary' mode. The inital 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.

A new set of improvements would be to add a cutting tool or a way to interface the script with the Pandas data handling library.

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 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.

4.2 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. This code is free for all to use with proper acknowledgement. Please send any bug reports to kd5wxb@gmail.com.

6 Code

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

    f.close()

    return data_rec

def ReadBin(filename,dataendianness='little'):
    """
    Takes a filename containing the binary output from xlook and
    reads the columns into a rec array 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

    col_headings = []
    col_recs    = []
    col_units   = []

    # Unpack information at the top of the file about the experiment
    name = struct.unpack('20c',f.read(20))
    name = ''.join(str(i) for i in name)
    name = name.split("\0")[0]
    print "\nName: ",name

    # The rest of the header information is written in big endian format
```

```
109
110     # Number of records (int)
111     num_recs = struct.unpack('>i',f.read(4))
112     num_recs = int(num_recs[0])
113     print "Number of records: %d" %num_recs
114
115     # Number of columns (int)
116     num_cols = struct.unpack('>i',f.read(4))
117     num_cols = int(num_cols[0])
118     print "Number of columns: %d" %num_cols
119
120     # Sweep (int) - No longer used
121     swp = struct.unpack('>i',f.read(4))[0]
122     print "Swp: ",swp
123
124     # Date/time(int) - No longer used
125     dtime = struct.unpack('>i',f.read(4))[0]
126     print "dtime: ",dtime
127
128     # For each possible column (32 maximum columns) unpack its header
129     # information and store it. Only store column headers of columns
130     # that contain data. Use termination at first NUL.
131     for i in range(32):
132
133         # Channel name (13 characters)
134         chname = struct.unpack('13c',f.read(13))
135         chname = ''.join(str(i) for i in chname)
136         chname = chname.split("\0")[0]
137
138         # Channel units (13 characters)
139         chunits = struct.unpack('13c',f.read(13))
140         chunits = ''.join(str(i) for i in chunits)
141         chunits = chunits.split("\0")[0]
142
143         # This field is now unused, so we just read past it (int)
144         gain = struct.unpack('>i',f.read(4))
145
146         # This field is now unused, so we just read past it (50 characters)
147         comment = struct.unpack('50c',f.read(50))
148
149         # Number of elements (int)
150         nelem = struct.unpack('>i',f.read(4))
151         nelem = int(nelem[0])
152
153         if chname[0:6] == 'no_val':
154             continue # Skip Blank Channels
155         else:
156             col_headings.append(chname)
157             col_recs.append(nelem)
158             col_units.append(chunits)
159
160
161     # Show column units and headings
162     print "\n\n-----------------------------------------------"
163     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=[]
    for name in col_headings:
        dtype.append((name,'float'))
    dtype = np.dtype(dtype)

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

    for col in range(num_cols):
        for row in range(col_recs[col]):
            if dataendianness == 'little':
                data[row,col] = struct.unpack('<d',f.read(8))[0]
            elif dataendianness == 'big':
                data[row,col] = struct.unpack('>d',f.read(8))[0]
            else:
                print "Data endian setting invalid, please check and retry"
                return 0

    data_rec = np.rec.array(data,dtype=dtype)

    f.close()

    return data_rec
```