Interact with files

June 1, 2016

```
In [1]: # Just to know last time this was run:
        import time
        print time.ctime()
```

```
Wed Jun  1 16:56:44 2016
```

1 C: How to read and write files (ASCII and FITS)

This is part of the Python lecture given by Christophe Morisset at IA-UNAM. More informations at: http://python-astro.blogspot.mx/

Some informations are here: http://www.tutorialspoint.com/python/python_files_io.htm

1.1 Reading a simple ascii file

```
In [2]: # numpy is needed in some part of the lecture
        import numpy as np
```

First of all, we will have to have some files on the hard drive to read them The following notebook cell will write a file in the same directory where the notebook has been started.

```
In [3]: %%writefile data1.dat
        1   2.3  6   8 star
        2   3.5  7   9 galaxy
        3  -4.2  5   7 cluster
```

```
Overwriting data1.dat
```

Now the goal is to read this file. The first way is to open the file, read it completely in a variable and close the file. Then we can play with the content of the file.

```
In [4]: datafile = open('data1.dat', 'r') # Open the file to read it
```

```
In [5]: data = datafile.readlines() # The variable data will receive the content of
```

```
In [6]: datafile.close() # Not need anymore of the file.
```

```
In [7]: print type(data) # The data file is stored in the form of a list, each elem
```

```
<type 'list'>
```

```
In [8]: print data # Each row is a string and terminates with \n, symbol of END OF
```

```
['1   2.3  6   8 star\n', '2   3.5  7   9 galaxy\n', '3  -4.2  5   7 cluster']
```

```
In [9]: print len(data) # number of rows
```

```
3
```

```
In [10]: print data[0], 'tralala'
```

```
1   2.3  6   8 star
tralala
```

```
In [11]: for row in data:
             print row
```

```
1   2.3  6   8 star

2   3.5  7   9 galaxy

3  -4.2  5   7 cluster
```

```
In [12]: for row in data:
             print row,
```

```
1   2.3  6   8 star
2   3.5  7   9 galaxy
3  -4.2  5   7 cluster
```

```
In [13]: print type(data[0]) # Each element is a string
```

```
<type 'str'>
```

Now it is easy to separate each field with the split command:

```
In [14]: for row in data:
             print row.split()
```

```
['1', '2.3', '6', '8', 'star']
['2', '3.5', '7', '9', 'galaxy']
['3', '-4.2', '5', '7', 'cluster']
```

```
In [15]: # One can also transform the data if the type is known:
         for row in data:
             this_data = row.split()
             print('N = {0:2d} f = {1:5.2f} type = {2:>10s}'.format(int(this_data[0
                                                                    float(this_data
                                                                    this_data[4]))

N =  1 f =  2.30 type =       star
N =  2 f =  3.50 type =     galaxy
N =  3 f = -4.20 type =    cluster
```

```
In [16]: # One can even fill a list with the data, by column:
         N = []
         f = []
         type_ = [] # take care, type is a python command, you can erase it if you
         for row in data:
             this_data = row.split()
             N.append(int(this_data[0]))
             f.append(float(this_data[1]))
             type_.append(this_data[4])
         print N
         print f
         print type_
         N = np.array(N)
         print N

[1, 2, 3]
[2.3, 3.5, -4.2]
['star', 'galaxy', 'cluster']
[1 2 3]
```

```
In [17]: # If the file number of rows is not too big, you can use list comprehensio
         N = np.array([int(row.split()[0]) for row in data])
         f = np.array([float(row.split()[1]) for row in data])
         # Each one of this command scans all the rows, don't use for huge files
         print N
         print f

[1 2 3]
[ 2.3  3.5 -4.2]
```

1.2 How to treat special rows (headers, comments)

```
In [18]: %%writefile data2.dat
         # The following data are for test purpose
         N    f   x   y type
```

```
1   2.3  6   8 star
2   3.5  7   9 galaxy

3  -4.2  5   7 cluster
#4  -10.5  5  7 test

Overwriting data2.dat
```

```
In [19]: !cat data2.dat # Just to check that the # comments are also in the file

# The following data are for test purpose
N    f   x   y type
1   2.3  6   8 star
2   3.5  7   9 galaxy

3  -4.2  5   7 cluster
#4  -10.5  5  7 test
```

The file has to be read row by row, to be sure that special cases are treated.

```
In [20]: datafile = open('data2.dat', 'r') # Open the file to read it
         row = datafile.readline() # this reads only one line
         first_comment = row
         print first_comment,
         row = datafile.readline() # this reads only one line
         header = row
         print header,
         data = []
         while True: # loops until exit by break command
             row = datafile.readline()
             if row == '':
                 break
             if row[0] != '#' and row[0] != '\n': # comment lines are skipped
                 data.append(row)
         datafile.close()
         print data

# The following data are for test purpose
N    f   x   y type
['1   2.3  6   8 star\n', '2   3.5  7   9 galaxy\n', '3  -4.2  5   7 cluster\n']
```

```
In [21]: datafile = open('data2.dat', 'r') # Open the file to read it
         row = datafile.readline() # this reads only one line
         first_comment = row
         print first_comment,
         row = datafile.readline() # this reads only one line
         header = row
```

```
        print header,
        data = []
        row = datafile.readline()
        while row != '': # loops until exit by break command
            if row[0] != '#': # comment lines are skipped
                data.append(row)
            row = datafile.readline()
        datafile.close()
        print data
```

```
# The following data are for test purpose
N    f   x   y type
['1   2.3  6   8 star\n', '2   3.5  7   9 galaxy\n', '\n', '3  -4.2  5   7 cluster\
```

In [22]:
```
        # very shorter way to deal with the file. No need to look for the end of t
        datafile = open('data2.dat', 'r') # Open the file to read it
        data = []
        for row in datafile:
            if row[0] != '#': # comment lines are skipped
                data.append(row)
        datafile.close()
        print data
        # This way will include the header in the data... Not what we want
```

```
['N    f   x   y type\n', '1   2.3  6   8 star\n', '2   3.5  7   9 galaxy\n', '\n',
```

In [23]:
```
        # very shorter way to deal with the file:
        # we know that the header is the first no-comment line in the file.
        datafile = open('data2.dat', 'r') # Open the file to read it
        data = []
        comments = [] # we can keep the comments for some usage
        header_read = False # We will turn it to True once the header is read
        for row in datafile:
            if row[0] != '#': # comment lines are skipped
                if not header_read:
                    header = row
                    header_read = True # next time, data will be read
                else:
                    data.append(row)
            else:
                comments.append(row)
        datafile.close()
        print header,
        print data
        print comments
```

```
N    f   x   y type
['1   2.3  6   8 star\n', '2   3.5  7   9 galaxy\n', '\n', '3  -4.2  5   7 cluster\
```

```
['# The following data are for test purpose\n', '#4  -10.5  5  7 test']
```

```
In [24]: # Alternative way using "with". No need to close the file, done when the '
         data = []
         comments = []
         header_read = False
         def change_type(row_split):
             # This function change the type of the data read from the file from 5
             # It also return the result in form of a tuple
             return (int(row_split[0]),
                     float(row_split[1]),
                     float(row_split[2]),
                     float(row_split[3]),
                     row_split[4])
         with open('data2.dat', 'r') as datafile:
             for row in datafile:
                 if row[0] != '#' and row[0] != '\n': # comment lines are skipped
                     if not header_read:
                         header = row
                         header_read = True
                     else:
                         data.append(change_type(row.split()))
                 else:
                     comments.append(row)
         print header
         print data
         print comments
```

```
N    f    x    y type

[(1, 2.3, 6.0, 8.0, 'star'), (2, 3.5, 7.0, 9.0, 'galaxy'), (3, -4.2, 5.0, 7.0, 'clu
['# The following data are for test purpose\n', '\n', '#4  -10.5  5  7 test']
```

```
In [25]: # We can define the result as a structured array
         # We use the header to define the field names.
         # data must be a list of tuples.
         a = np.array(data, dtype={'names':header.split(),
                                   'formats':['i4','f16', 'f16', 'f16', 'S10']})
```

```
In [26]: a
```

```
Out[26]: array([(1, 2.2999999999999998224, 6.0, 8.0, 'star'),
                (2, 3.5, 7.0, 9.0, 'galaxy'),
                (3, -4.2000000000000001776, 5.0, 7.0, 'cluster')],
               dtype=[('N', '<i4'), ('f', '<f16'), ('x', '<f16'), ('y', '<f16'), ('
```

```
In [27]: print data[0]
```

```
(1, 2.3, 6.0, 8.0, 'star')
```

```
In [28]: print a[0]
```

```
(1, 2.2999999999999998224, 6.0, 8.0, 'star')
```

```
In [29]: # Easy access to the columns, by their name
         print a['N']
```

```
[1 2 3]
```

```
In [30]: print a['type']
```

```
['star' 'galaxy' 'cluster']
```

```
In [31]: # Easy combine the values of columns
         print np.sqrt(a['x']**2 + a['y']**2)
```

```
[ 10.0  11.401754  8.6023253]
```

1.2.1 Using numpy loadtxt

http://docs.scipy.org/doc/numpy/reference/generated/numpy.loadtxt.html

```
In [32]: # Fast way for reading the file
         # One hace to tell to skip the 2 first rows
         # skiprows
         b = np.loadtxt('data2.dat', skiprows=2, dtype='i4,f, f, f, S10')
```

```
In [33]: print b
```

```
[(1, 2.299999952316284, 6.0, 8.0, 'star') (2, 3.5, 7.0, 9.0, 'galaxy')
 (3, -4.199999809265137, 5.0, 7.0, 'cluster')]
```

```
In [34]: type(b)
```

```
Out[34]: numpy.ndarray
```

```
In [35]: # The names of the columns are f0, f1, f2, etc
         b.dtype
```

```
Out[35]: dtype([('f0', '<i4'), ('f1', '<f4'), ('f2', '<f4'), ('f3', '<f4'), ('f4',
```

1.2.2 Using numpy genfromtxt

http://docs.scipy.org/doc/numpy/reference/generated/numpy.genfromtxt.html

```
In [36]: # Fast and versatile way to read the file
         # the names are taken from the file
         # The types are defined automatically when reading the columns
         c = np.genfromtxt('data2.dat', names=True, dtype=None, skip_header=1)

In [37]: print c

[(1, 2.3, 6, 8, 'star') (2, 3.5, 7, 9, 'galaxy') (3, -4.2, 5, 7, 'cluster')]


In [38]: type(c)

Out[38]: numpy.ndarray

In [39]: c.dtype

Out[39]: dtype([('N', '<i8'), ('f', '<f8'), ('x', '<i8'), ('y', '<i8'), ('type', 'S

In [40]: c['f']

Out[40]: array([ 2.3,  3.5, -4.2])
```

Now a value of x is missing (not possible with space separator, so we use "," as separator):

```
In [41]: %%writefile data3.dat
         # The following data are for test purpose
         N,    f,   x,   y, type
         1,   2.3,  6,   8, star
         2,   3.5,   ,  9, galaxy
         3,  -4.2,  5,   7, cluster
         #4,  -10.5,  5,  7, test

Overwriting data3.dat


In [42]: d = np.genfromtxt('data3.dat', names=True, dtype=None, skip_header=1,
                           delimiter=',')

In [43]: # The missing value has been changed to -1
         d

Out[43]: array([(1, 2.3, 6, 8, ' star'), (2, 3.5, -1, 9, ' galaxy'),
                (3, -4.2, 5, 7, ' cluster')],
               dtype=[('N', '<i8'), ('f', '<f8'), ('x', '<i8'), ('y', '<i8'), ('typ

In [44]: # Th emissing value can be set to whatever you want (but non a NaN here, a
         d = np.genfromtxt('data3.dat', names=True, dtype=None, skip_header=1, deli
                           filling_values=-999)
```

```
In [45]: d['x'][1]

Out[45]: -999

In [46]: # ons can select the columns to be store
         e = np.genfromtxt('data3.dat', names=True, dtype=None, skip_header=1,
                           delimiter=',',usecols=(0,1,4))

In [47]: # The resulting array now contains only the given columns
         e

Out[47]: array([(1, 2.3, ' star'), (2, 3.5, ' galaxy'), (3, -4.2, ' cluster')],
               dtype=[('N', '<i8'), ('f', '<f8'), ('type', 'S8')])
```

1.2.3 Using recfrom to obtain a record array

```
In [48]: # Uses the same keywords than genfromtxt
         f = np.recfromtxt('data3.dat', names=True, dtype=None, skip_header=1,
                           delimiter=',',usecols=("N", "f", "type"))

In [49]: f

Out[49]: rec.array([(1, 2.3, ' star'), (2, 3.5, ' galaxy'), (3, -4.2, ' cluster')],
                   dtype=[('N', '<i8'), ('f', '<f8'), ('type', 'S8')])

In [50]: f.N

Out[50]: array([1, 2, 3])
```

1.3 Fixed size ascii files

```
In [51]: %%writefile data4.dat
         #  Line       Iobs     lambda  relat_error Obs_code
         H   1   4861A 1.00000    4861. 0.08000  Anabel
         H   1   6563A 2.8667     6563. 0.19467  Anabel
         H   1   4340A 0.4933     4340. 0.03307  Anabel
         H   1   4102A 0.2907     4102. 0.02229  Anabel
         H   1   3970A 0.1800     3970. 0.01253  Anabel
         N   2   6584A 2.1681     6584. 0.08686  Anabel
         N   2 121.7m 0.00446 1217000. 0.20000  Liu
         O   1   6300A 0.0147     6300. 0.00325  Anabel
         TOTL   2326A 0.07900    2326. 0.20000  Adams
         C   2 157.6m 0.00856 1576000. 0.20000  Liu
         O   1 63.17m 0.13647  631700. 0.10000  Liu
         O   1 145.5m 0.00446 1455000. 0.200    Liu
         TOTL   3727A 0.77609    3727. 0.200    Torres-Peimbert
         S II   4070A 0.06174    4070. 0.200    Torres-Peimbert
         S II   4078A 0.06174    4078. 0.200    Torres-Peimbert

Overwriting data4.dat
```

```
In [52]: # Here we cannot use SPACE as a separator, as some strings contains spaces
         # "delimiter" is used to specify the size (in characters in the file) of e
         # The types must be clearly defined too.
         obs  = np.genfromtxt('data4.dat',
                              dtype=["a11","float","float","float","a2"],
                              delimiter=[11,8,9,10,2],
                              names = True
                              )

In [53]: obs # The same delimiter (fixed sizes) is applied to the names. May not be

Out[53]: array([('H  1  4861A', 1.0, 4861.0, 0.08, 'An'),
                ('H  1  6563A', 2.8667, 6563.0, 0.19467, 'An'),
                ('H  1  4340A', 0.4933, 4340.0, 0.03307, 'An'),
                ('H  1  4102A', 0.2907, 4102.0, 0.02229, 'An'),
                ('H  1  3970A', 0.18, 3970.0, 0.01253, 'An'),
                ('N  2  6584A', 2.1681, 6584.0, 0.08686, 'An'),
                ('N  2 121.7m', 0.00446, 1217000.0, 0.2, 'Li'),
                ('O  1  6300A', 0.0147, 6300.0, 0.00325, 'An'),
                ('TOTL  2326A', 0.079, 2326.0, 0.2, 'Ad'),
                ('C  2 157.6m', 0.00856, 1576000.0, 0.2, 'Li'),
                ('O  1 63.17m', 0.13647, 631700.0, 0.1, 'Li'),
                ('O  1 145.5m', 0.00446, 1455000.0, 0.2, 'Li'),
                ('TOTL  3727A', 0.77609, 3727.0, 0.2, 'To'),
                ('S II  4070A', 0.06174, 4070.0, 0.2, 'To'),
                ('S II  4078A', 0.06174, 4078.0, 0.2, 'To')],
               dtype=[('Line', 'S11'), ('Iobs', '<f8'), ('lambda', '<f8'), ('relat_

In [54]: # Defining the names:
         obs2  = np.genfromtxt('data4.dat', skip_header=1,
                              dtype=None,
                              delimiter=[11,8,9,10,2],
                              names = ['label', 'i_obs', 'lambda', 'e_obs', 'observ
                              )

In [55]: obs2

Out[55]: array([('H  1  4861A', 1.0, 4861.0, 0.08, 'An'),
                ('H  1  6563A', 2.8667, 6563.0, 0.19467, 'An'),
                ('H  1  4340A', 0.4933, 4340.0, 0.03307, 'An'),
                ('H  1  4102A', 0.2907, 4102.0, 0.02229, 'An'),
                ('H  1  3970A', 0.18, 3970.0, 0.01253, 'An'),
                ('N  2  6584A', 2.1681, 6584.0, 0.08686, 'An'),
                ('N  2 121.7m', 0.00446, 1217000.0, 0.2, 'Li'),
                ('O  1  6300A', 0.0147, 6300.0, 0.00325, 'An'),
                ('TOTL  2326A', 0.079, 2326.0, 0.2, 'Ad'),
                ('C  2 157.6m', 0.00856, 1576000.0, 0.2, 'Li'),
                ('O  1 63.17m', 0.13647, 631700.0, 0.1, 'Li'),
                ('O  1 145.5m', 0.00446, 1455000.0, 0.2, 'Li'),
```

```
                ('TOTL  3727A', 0.77609, 3727.0, 0.2, 'To'),
                ('S II  4070A', 0.06174, 4070.0, 0.2, 'To'),
                ('S II  4078A', 0.06174, 4078.0, 0.2, 'To')],
               dtype=[('label', 'S11'), ('i_obs', '<f8'), ('lambda', '<f8'), ('e_ob
```

In [56]:
```
%%writefile data5.dat
#  Line      Iobs    lambda  relat_error Obs_code
H  1  4861A 1.00000    4861. 0.08000 x Anabel
H  1  6563A 2.8667     6563. 0.19467 x Anabel
H  1  4340A 0.4933     4340. 0.03307 x Anabel
H  1  4102A 0.2907     4102. 0.02229 x Anabel
H  1  3970A 0.1800     3970. 0.01253 t Anabel
N  2  6584A 2.1681     6584. 0.08686 x Anabel
N  2 121.7m 0.00446 1217000. 0.20000 g Liu
O  1  6300A 0.0147     6300. 0.00325 t Anabel
TOTL  2326A 0.07900    2326. 0.20000 g Adams
C  2 157.6m 0.00856 1576000. 0.20000 t Liu
O  1 63.17m 0.13647  631700. 0.10000 g Liu
O  1 145.5m 0.00446 1455000. 0.200   g Liu
TOTL  3727A 0.77609    3727. 0.200   g Torres-Peimbert
S II  4070A 0.06174    4070. 0.200   g Torres-Peimbert
S II  4078A 0.06174    4078. 0.200   g Torres-Peimbert
```

```
Overwriting data5.dat
```

In [57]:
```
# Here we want to skip one column:
obs3  = np.genfromtxt('data4.dat', skip_header=1,
                      dtype=None,
                      delimiter=[11, 8, 9, 8, 2, 2],
                      names = ['label', 'i_obs', 'lambda', 'e_obs', 'na', '
                      usecols = (0, 1, 2, 3, 5)
                      )
```

In [58]:
```
obs3
```

Out[58]:
```
array([('H  1  4861A', 1.0, 4861.0, 0.08, 'An'),
       ('H  1  6563A', 2.8667, 6563.0, 0.19467, 'An'),
       ('H  1  4340A', 0.4933, 4340.0, 0.03307, 'An'),
       ('H  1  4102A', 0.2907, 4102.0, 0.02229, 'An'),
       ('H  1  3970A', 0.18, 3970.0, 0.01253, 'An'),
       ('N  2  6584A', 2.1681, 6584.0, 0.08686, 'An'),
       ('N  2 121.7m', 0.00446, 1217000.0, 0.2, 'Li'),
       ('O  1  6300A', 0.0147, 6300.0, 0.00325, 'An'),
       ('TOTL  2326A', 0.079, 2326.0, 0.2, 'Ad'),
       ('C  2 157.6m', 0.00856, 1576000.0, 0.2, 'Li'),
       ('O  1 63.17m', 0.13647, 631700.0, 0.1, 'Li'),
       ('O  1 145.5m', 0.00446, 1455000.0, 0.2, 'Li'),
       ('TOTL  3727A', 0.77609, 3727.0, 0.2, 'To'),
```

```
               ('S II  4070A', 0.06174, 4070.0, 0.2, 'To'),
               ('S II  4078A', 0.06174, 4078.0, 0.2, 'To')],
              dtype=[('label', 'S11'), ('i_obs', '<f8'), ('lambda', '<f8'), ('e_ob
```

```
In [59]: obs3['lambda']
```

```
Out[59]: array([    4861.,     6563.,     4340.,     4102.,     3970.,     6584.,
                 1217000.,     6300.,     2326.,  1576000.,   631700.,  1455000.,
                    3727.,     4070.,     4078.])
```

```
In [60]: new_obs3 = obs3.view(np.recarray)
```

```
In [61]: new_obs3.label
```

```
Out[61]: array(['H  1  4861A', 'H  1  6563A', 'H  1  4340A', 'H  1  4102A',
                'H  1  3970A', 'N  2  6584A', 'N  2 121.7m', 'O  1  6300A',
                'TOTL  2326A', 'C  2 157.6m', 'O  1 63.17m', 'O  1 145.5m',
                'TOTL  3727A', 'S II  4070A', 'S II  4078A'],
               dtype='|S11')
```

```
In [62]: new_obs3.lambda # lambda is reserved!!!
```

```
          File "<ipython-input-62-a5c3f73ef51d>", line 1
        new_obs3.lambda # lambda is reserved!!!
                        ^
    SyntaxError: invalid syntax
```

```
In [63]: new_obs3['lambda']
```

```
Out[63]: array([    4861.,     6563.,     4340.,     4102.,     3970.,     6584.,
                 1217000.,     6300.,     2326.,  1576000.,   631700.,  1455000.,
                    3727.,     4070.,     4078.])
```

Using masks on the structured array.

```
In [64]: mask_observer = obs3['observer'] == 'An'
         print obs3[mask_observer]
```

```
[('H  1  4861A', 1.0, 4861.0, 0.08, 'An')
 ('H  1  6563A', 2.8667, 6563.0, 0.19467, 'An')
 ('H  1  4340A', 0.4933, 4340.0, 0.03307, 'An')
 ('H  1  4102A', 0.2907, 4102.0, 0.02229, 'An')
 ('H  1  3970A', 0.18, 3970.0, 0.01253, 'An')
 ('N  2  6584A', 2.1681, 6584.0, 0.08686, 'An')
 ('O  1  6300A', 0.0147, 6300.0, 0.00325, 'An')]
```

```
In [65]: for o in obs3[mask_observer]:
             print('line {0[label]:4s}, wavelength={0[lambda]}A Intensity={0[i_obs]
```

```
line H  1  4861A, wavelength=4861.0A Intensity=1.000+/- 8.0%)
line H  1  6563A, wavelength=6563.0A Intensity=2.867+/-19.5%)
line H  1  4340A, wavelength=4340.0A Intensity=0.493+/- 3.3%)
line H  1  4102A, wavelength=4102.0A Intensity=0.291+/- 2.2%)
line H  1  3970A, wavelength=3970.0A Intensity=0.180+/- 1.3%)
line N  2  6584A, wavelength=6584.0A Intensity=2.168+/- 8.7%)
line O  1  6300A, wavelength=6300.0A Intensity=0.015+/- 0.3%)
```

1.4 Writing files

1.4.1 Simple "write" method from "open" class

```
In [66]: f = open('data10.dat', 'w')
```

```
In [67]: f.write('tralala')
         f.write('trololo')
```

```
In [68]: f.close()
```

```
In [69]: !cat 'data10.dat' # the writing method put everything together.
```

```
tralalatrololo
```

```
In [70]: f = open('data11.dat', 'w')
         f.write('tralala\n') # \n to indicate end of line
         f.write('trololo\n')
         f.close()
         !cat 'data11.dat'
```

```
tralala
trololo
```

```
In [71]: f = open('data11.dat', 'a') # Append to the edn of the file
         f.write('trilili\n') # \n to indicate end of line
         f.write('trululu\n')
         f.close()
         !cat 'data11.dat'
```

```
tralala
trololo
trilili
trululu
```

```
In [72]: a = 'Hola'
         b = 3
         with open('data12.dat', 'w') as datafile:
             datafile.write("""{0}
         This is a file
         with a lot of lines.
         It is easy to write it.
         Using the {1} \".
         Tralala.""".format(a, b))
         !cat "data12.dat"
```

```
Hola
This is a file
with a lot of lines.
It is easy to write it.
Using the 3 ".
Tralala.
```

1.4.2 Using pickle (and cpickle) python specific format

```
In [73]: # Let's define some stuffs we want to keep in a file (data and variable na
         a = 5
         b = 'Hola'
         c = np.array([1,2,3,4,5])
         def d(x):
             """ Function mia"""
             return x**2
```

```
In [74]: import pickle # The module we will use for this
```

```
In [75]: pickle.dump((a,b,c,d), file('Demo.pickle','w')) # Writing the variables
```

```
In [76]: res = pickle.load(file('Demo.pickle'))
```

```
In [77]: type(res)
```

```
Out[77]: tuple
```

```
In [78]: print res[0]
         print res[1]
         print res[2]
```

```
5
Hola
[1 2 3 4 5]
```

```
In [79]: res[3](5)
```

```
Out[79]: 25
```

```
In [80]: a,b,c,d = pickle.load(file('Demo.pickle'))

In [81]: a

Out[81]: 5

In [82]: help(d)

Help on function d in module __main__:

d(x)
    Function mia



In [83]: %timeit res = pickle.load(file('Demo.pickle'))

The slowest run took 6.35 times longer than the fastest. This could mean that an ir
10000 loops, best of 3: 110 µs per loop


In [84]: import cPickle # A newer version writen in C, faster.

In [85]: %timeit res = cPickle.load(file('Demo.pickle'))

The slowest run took 4.21 times longer than the fastest. This could mean that an ir
10000 loops, best of 3: 64.1 µs per loop


In [86]: huge_arr = np.random.rand(1000,100)
         pickle.dump(huge_arr, file('Demo2.pickle','w')) # Writing the variable

In [87]: %timeit res2 = pickle.load(file('Demo2.pickle'))

10 loops, best of 3: 23.3 ms per loop


In [88]: %timeit res2 = cPickle.load(file('Demo2.pickle')) # It is supposed to be f

1 loop, best of 3: 233 ms per loop


In [89]: import gzip
         pickle.dump((a,b,c,d), gzip.open('Demo.pklz','wb')) # Writing the variable

In [90]: f = gzip.open('Demo.pklz','rb')
         a, b, c, d = pickle.load(f)
         f.close()
```

1.4.3 FITS files

What is the FITS format? The FITS format is the most popular way to save and interchange astronomical data. The files are organized in units each of which contains a human readable header and a data. This structure is refereed as HDUs (Header/DATA Unit).

A FITS file can contain one or more HDUs, the first of which is called "primary" and the rest are called "extensions". The primary HDU usually contains 1D spectrum, 2D image or 3D data cube, although any dimension from 0 to 999 are possible. The data are 1, 2 or 4 bytes integers or 4 or 8 bytes real numbers.

The extensions can contain or arrays as in the primary HDU or ascii tables or binary tables. If a FITS file contains only tables, it primary HDU does not contain data, but only header.

Both headers and data in a FITS file are organized in blocs of 2880 bytes. The header contain 80 bytes lines each of which consists of a keyword of 8 bytes followed in most of the cases by '= ' in the position 9 and 10 and then the value of the keyword. The rest of the line is a comment string beginning with '/'. Each header begins with the following lines

SIMPLE = T / file conforms to FITS standard BITPIX = 16 / number of bits per data pixel NAXIS = 2 / number of data axes NAXIS1 = 440 / length of data axis 1 NAXIS2 = 300 / length of data axis 2

which defines the format of the file as standard FITS, the data format and the dimensions of the stored data.

One block of 2880 bytes contains 36 lines of 80 characters per line. The header can have several blocks of 36 lines. The last block is identified by the presence of the keyword 'END' The next 2880 bytes block contains the first part of the data. The empty lines after 'END' keyword are filled with blanks and the unused bytes from the end of the data to the end of the 2880 bytes block are filled with NULLs.

```
In [91]: from astropy.io import fits

In [92]: # All of the functionality of PyFITS is now available in Astropy
         # from astropy.io import fits as pyfits
```

Manual here: https://pythonhosted.org/pyfits/

We will use one FITS files from San Pedro Martir echelle spectrograph. The file can be downloaded from: https://docs.google.com/open?id=0B4A0EADFiYFpVXo3QWdhbk5UM09Ld3BvSWl4d2tBUQ

```
In [93]: hdulist = fits.open('n10017o.fits')

In [94]: # The result hdulist is a list of HDU objects.
         # In the case of a simple file, there is only one primary HDU so the list
         len(hdulist)

Out[94]: 1

In [95]: # The information on what the file contains can be obtained by calling the
         hdulist.info()
         # The table said that there is only a primary HDU which contains 2154 X 2

Filename: n10017o.fits
No.    Name         Type      Cards   Dimensions   Format
0    PRIMARY     PrimaryHDU      62   (2154, 2048)   int16 (rescales to uint16)
```

In [96]:
```
# As described above, the HDU (header/data unit) contains header and data.
# To see what keywords were used in the header one can do:
hdulist[0].header.keys()
```

Out[96]:
```
['SIMPLE',
 'BITPIX',
 'NAXIS',
 'NAXIS1',
 'NAXIS2',
 'EXTEND',
 'COMMENT',
 'COMMENT',
 'BZERO',
 'BSCALE',
 'EXPTIME',
 'DETECTOR',
 'ORIGIN',
 'OBSERVAT',
 'TELESCOP',
 'LATITUDE',
 'LONGITUD',
 'ALTITUD',
 'SECONDAR',
 'TIMEZONE',
 'OBSERVER',
 'OBJECT',
 'INSTRUME',
 'GAINMODE',
 'FILTER',
 'IMGTYPE',
 'EQUINOX',
 'ST',
 'UT',
 'JD',
 'DATE-OBS',
 'CCDSUM',
 'RA',
 'DEC',
 'AH',
 'AIRMASS',
 'TMMIRROR',
 'TSMIRROR',
 'TAIR',
 'XTEMP',
 'HUMIDITY',
 'ATMOSBAR',
 'WIND',
 'WDATE',
```

```
 'DATE',
 'NAMPS',
 'CCDNAMPS',
 'AMPNAME',
 'CREATOR',
 'VERSION',
 'COMMENT',
 'COMMENT',
 'HISTORY',
 'HISTORY',
 'HISTORY',
 'HISTORY',
 'HISTORY',
 'HISTORY',
 'HISTORY',
 'HISTORY',
 'HISTORY',
 'HISTORY']
```

In [97]:
```
# and to get the value of a given keyword :
hdulist[0].header['OBJECT']
```

Out[97]:
```
'107 Psc'
```

In [98]:
```
hh = hdulist[0].header
hh?
```

In [99]:
```
hdulist[0].header
```

Out[99]:
```
SIMPLE  =                    T / conforms to FITS standard
BITPIX  =                   16 / array data type
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                 2154 / length of data axis 1
NAXIS2  =                 2048 / length of data axis 2
EXTEND  =                    T
COMMENT FITS (Flexible Image Transport System) format is defined in 'Astro
COMMENT and Astrophysics', volume 376, page 359; bibcode: 2001A&A...376..3
BZERO   =                32768 / BZERO
BSCALE  =                    1 / BSCALE
EXPTIME =                600.0 / Integration Time, sec.
DETECTOR= 'e2vm2 E2V-4240'     / CCD Type
ORIGIN  = 'UNAM    '           / OAN SPM, IA-UNAM
OBSERVAT= 'SPM     '           / Observatory
TELESCOP= '2.12m   '           / Telescope
LATITUDE= '+31:02:39'          / Latitude
LONGITUD= '115:27:49'          / Longitud
ALTITUD =                2800 / altitud
SECONDAR=                  -1 / F/ Secondary type
TIMEZONE=                   8 / Time Zone
```

```
OBSERVER= 'Leonid  '           / Observer's Name
OBJECT  = '107 Psc '           / Object
INSTRUME= 'Echelle '           / Instrument
GAINMODE=                    1 / Gain factor in the CCD
FILTER  = 'None    '           / Filter
IMGTYPE = 'object  '           / Image Type
EQUINOX =               2011.7 / Equinox
ST      = '01:25:41.3'         / Sideral Time
UT      = '10:34:27'           / Universal Time
JD      =            2455803.5 / Julian Date
DATE-OBS= '2011-08-30'         / Observation Date UTM
CCDSUM  = '1 1     '           / Binning [ Cols:Rows ]
RA      = ' 01:43:10.8'        / Right Ascension
DEC     = ' 20''19''43.0'      / Declination
AH      = ' -00:17:29.1'       / Hour Angle
AIRMASS =                 1.02 / Airmass
TMMIRROR=                    0 / Primary Mirror Temperature (celsius degre
TSMIRROR=                    0 / Secundary Mirror Temperature (celsius deg
TAIR    =                    0 / Internal Telescope Air Temperature (celsi
XTEMP   =                 14.7 / Exterior Temperature (celsius degree)
HUMIDITY=                 46.0 / % external Humidity
ATMOSBAR=                731.9 / Atmosferic Presure in mb
WIND    = 'S at 30.6 km/h'     / Wind Direction
WDATE   = '10:34:10, 08/30/11' / Weather Acquisition Date (Local time)
DATE    = '2011-08-30T10:34:29' / file creation date (YYYY-MM-DDThh:mm:ss
NAMPS   =                    1 / Number of Amplifiers
CCDNAMPS=                    1 / Number of amplifiers used
AMPNAME = '1 Channel'          / Amplifier name
CREATOR = 'Python Oan ccds'    / Name of the software task that created th
VERSION = '4.12D   '           / Application Software Version
COMMENT Visit our weather site http://www.astrossp.unam.mx/weather15
COMMENT for complete meteorological data of your observation night
HISTORY bin2fits V1.0
HISTORY Programmer: Enrique Colorado [ colorado@astrosen.unam.mx ]
HISTORY Observatorio Astronomico Nacional -UNAM
HISTORY V1.00 By Arturo Nunez and Colorado >Ported to Python using pyfits
HISTORY V0.50 By E. Colorado >Added interior mirrors temperatures
HISTORY V0.49 By E. Colorado >Added BIASSEC parameter
HISTORY V0.48 By E. Colorado >Aditional info for autofocus calculations
HISTORY V0.4 By E. Colorado >Now we include timezone, and remove lat. sign
HISTORY V0.3 By E. Colorado >Now we include weather data
HISTORY V0.2 By E. Colorado >General OAN Working Release
```

In [100]:
```
# The header can be printed as it appears in the file by
print hdulist[0].header.cards
```

```
('SIMPLE', True, 'conforms to FITS standard')
('BITPIX', 16, 'array data type')
```

```
('NAXIS', 2, 'number of array dimensions')
('NAXIS1', 2154, 'length of data axis 1')
('NAXIS2', 2048, 'length of data axis 2')
('EXTEND', True, '')
('COMMENT', "FITS (Flexible Image Transport System) format is defined in 'Astronomy
('COMMENT', "and Astrophysics', volume 376, page 359; bibcode: 2001A&A...376..359H'
('BZERO', 32768, 'BZERO')
('BSCALE', 1, 'BSCALE')
('EXPTIME', 600.0, 'Integration Time, sec.')
('DETECTOR', 'e2vm2 E2V-4240', 'CCD Type')
('ORIGIN', 'UNAM', 'OAN SPM, IA-UNAM')
('OBSERVAT', 'SPM', 'Observatory')
('TELESCOP', '2.12m', 'Telescope')
('LATITUDE', '+31:02:39', 'Latitude')
('LONGITUD', '115:27:49', 'Longitud')
('ALTITUD', 2800, 'altitud')
('SECONDAR', -1, 'F/ Secondary type')
('TIMEZONE', 8, 'Time Zone')
('OBSERVER', 'Leonid', "Observer's Name")
('OBJECT', '107 Psc', 'Object')
('INSTRUME', 'Echelle', 'Instrument')
('GAINMODE', 1, 'Gain factor in the CCD')
('FILTER', 'None', 'Filter')
('IMGTYPE', 'object', 'Image Type')
('EQUINOX', 2011.7, 'Equinox')
('ST', '01:25:41.3', 'Sideral Time')
('UT', '10:34:27', 'Universal Time')
('JD', 2455803.5, 'Julian Date')
('DATE-OBS', '2011-08-30', 'Observation Date UTM')
('CCDSUM', '1 1', 'Binning [ Cols:Rows ]')
('RA', ' 01:43:10.8', 'Right Ascension')
('DEC', " 20'19'43.0", 'Declination')
('AH', ' -00:17:29.1', 'Hour Angle')
('AIRMASS', 1.02, 'Airmass')
('TMMIRROR', 0, 'Primary Mirror Temperature (celsius degree)')
('TSMIRROR', 0, 'Secundary Mirror Temperature (celsius degree)')
('TAIR', 0, 'Internal Telescope Air Temperature (celsius deg')
('XTEMP', 14.7, 'Exterior Temperature (celsius degree)')
('HUMIDITY', 46.0, '% external Humidity')
('ATMOSBAR', 731.9, 'Atmosferic Presure in mb')
('WIND', 'S at 30.6 km/h', 'Wind Direction')
('WDATE', '10:34:10, 08/30/11', 'Weather Acquisition Date (Local time)')
('DATE', '2011-08-30T10:34:29', 'file creation date (YYYY-MM-DDThh:mm:ss UT)')
('NAMPS', 1, 'Number of Amplifiers')
('CCDNAMPS', 1, 'Number of amplifiers used')
('AMPNAME', '1 Channel', 'Amplifier name')
('CREATOR', 'Python Oan ccds', 'Name of the software task that created the file')
('VERSION', '4.12D', 'Application Software Version')
```

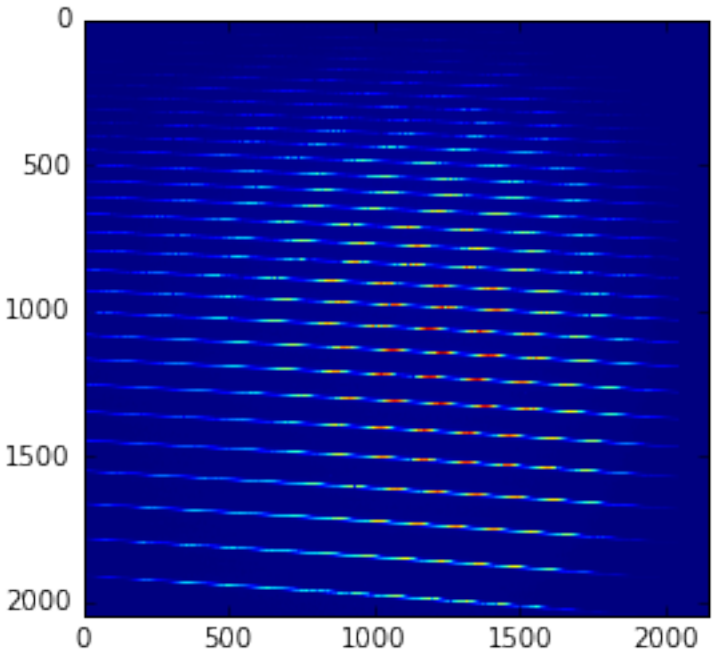
```
('COMMENT', 'Visit our weather site http://www.astrossp.unam.mx/weather15', '')
('COMMENT', 'for complete meteorological data of your observation night', '')
('HISTORY', 'bin2fits V1.0', '')
('HISTORY', 'Programmer: Enrique Colorado [ colorado@astrosen.unam.mx ]', '')
('HISTORY', 'Observatorio Astronomico Nacional -UNAM', '')
('HISTORY', 'V1.00 By Arturo Nunez and Colorado >Ported to Python using pyfits', ''
('HISTORY', 'V0.50 By E. Colorado >Added interior mirrors temperatures', '')
('HISTORY', 'V0.49 By E. Colorado >Added BIASSEC parameter', '')
('HISTORY', 'V0.48 By E. Colorado >Aditional info for autofocus calculations', '')
('HISTORY', 'V0.4 By E. Colorado >Now we include timezone, and remove lat. sign', '
('HISTORY', 'V0.3 By E. Colorado >Now we include weather data', '')
('HISTORY', 'V0.2 By E. Colorado >General OAN Working Release', '')
```

In [101]:
```
# The data in the file are accessible with
data = hdulist[0].data
```

In [102]:
```
# and can be seen with [we need to import matplotlib.pyplot as plt before
%matplotlib inline
import matplotlib.pyplot as plt
plt.imshow(data)
```

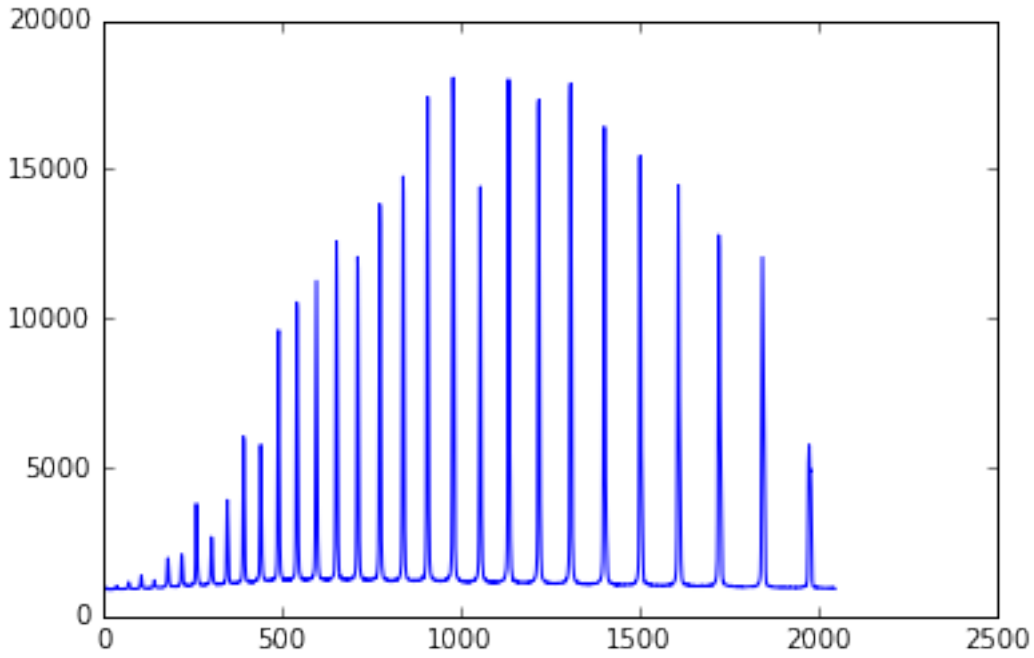
```
/Users/christophemorisset/anaconda/lib/python2.7/site-packages/matplotlib/font_mana
  warnings.warn('Matplotlib is building the font cache using fc-list. This may take
```

Out[102]:
```
<matplotlib.image.AxesImage at 0x10de8a1d0>
```



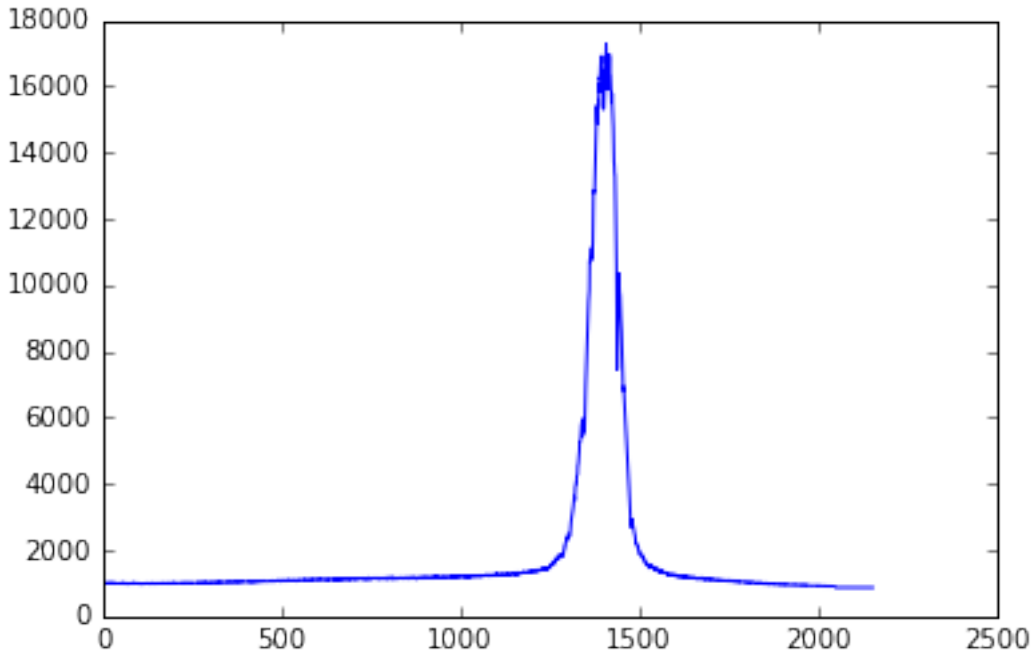
```
In [103]: # A column from the data can be plotted with
          plt.plot(data[:,1000])
          # where I am plotting the column number 1000.

Out[103]: [<matplotlib.lines.Line2D at 0x1175315d0>]
```




```
In [104]: # In the same way a line from the data is plotted with:
          plt.plot(data[1000,:])

Out[104]: [<matplotlib.lines.Line2D at 0x1175ce890>]
```




```
In [105]: # For this example I'll use a spectrum obtain with the high dispersion ca
          # The file is opened as usual:
          hdulist = fits.open('swp04345.mxhi')
```

The file is there: https://docs.google.com/open?id=0B4A0EADFiYFpTlN5Wkd1cU9SVXFNdzg0WDlWV196

```
In [106]: #but now hdulist has 2 elements (2 header/data units):
          len(hdulist)

Out[106]: 2

In [107]: # We can see that the primary header has dimension (), son does not conta
          # The data are in the extension.
          hdulist.info()

Filename: swp04345.mxhi
No.    Name         Type      Cards   Dimensions   Format
0    PRIMARY     PrimaryHDU     421   ()
1    MEHI        BinTableHDU     61   60R x 17C    [1B, 1I, 1D, 1I, 1D, 1E, 1E, 768


In [108]: # The first header contains the minimal infirmation:
          print hdulist[0].header.cards[:5]

('SIMPLE', True, 'Standard FITS Format')
('BITPIX', 8, 'Binary data')
```

```
('NAXIS', 0, 'Two-dimensional image')
('EXTEND', True, 'Extensions are present')
('TELESCOP', 'IUE', 'International Ultraviolet Explorer')
```

In [109]:
```
# The number of axis is 0 which means there is no data block in the prima
# The header of the second HDU begins with the keyword XTENSION and with
print hdulist[1].header.cards[:5]
```

```
('XTENSION', 'BINTABLE', 'Binary table extension')
('BITPIX', 8, 'Binary data')
('NAXIS', 2, 'Two-dimensional table array')
('NAXIS1', 16961, 'Width of row in bytes')
('NAXIS2', 60, 'Number of orders')
```

In [110]:
```
# To progress further we need to know what is in the table.
# As usual, the columns have names and type of the stored data.
# These information can be obtained using the column attribute of hdulist
cols = hdulist[1].columns
```

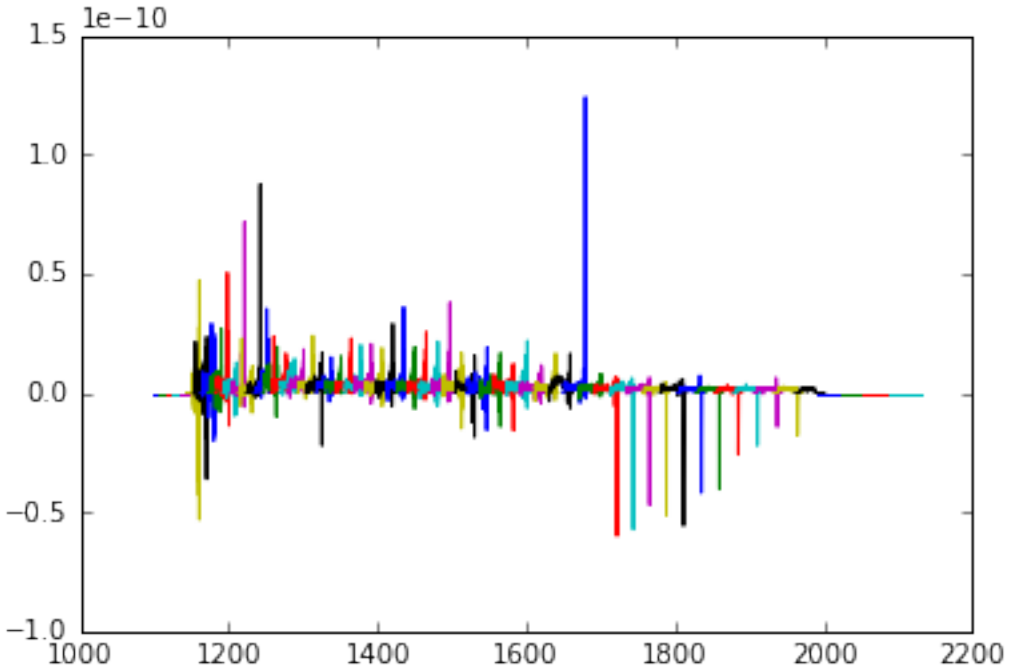
In [111]:
```
# the cols.info returns the names of the columns and the information of t
cols.info
```

Out[111]:
```
<bound method ColDefs.info of ColDefs(
    name = 'ORDER'; format = '1B'
    name = 'NPOINTS'; format = '1I'
    name = 'WAVELENGTH'; format = '1D'; unit = 'ANGSTROM'
    name = 'STARTPIX'; format = '1I'; unit = 'PIXEL'
    name = 'DELTAW'; format = '1D'; unit = 'ANGSTROM'
    name = 'SLIT HEIGHT'; format = '1E'; unit = 'PIXEL'
    name = 'LINE_FOUND'; format = '1E'; unit = 'PIXEL'
    name = 'NET'; format = '768E'; unit = 'FN'
    name = 'BACKGROUND'; format = '768E'; unit = 'FN'
    name = 'NOISE'; format = '768E'; unit = 'FN'
    name = 'QUALITY'; format = '768I'
    name = 'RIPPLE'; format = '768E'; unit = 'FN'
    name = 'ABS_CAL'; format = '768E'; unit = 'ERG/CM2/S/A'
    name = 'START-BKG'; format = '1I'; unit = 'PIXEL'
    name = 'END-BKG'; format = '1I'; unit = 'PIXEL'
    name = 'SCALE_BKG'; format = '1E'
    name = 'COEFF'; format = '7E'
)>
```

In [112]:
```
# The data are available using (this example is NOT the right way of plot
# and don't forget to import numpy as np to have np.arange working]:

data1 = hdulist[1].data
DTs =  data1.ABS_CAL
```

```
WLs = data1.WAVELENGTH
DWs = data1.DELTAW
for WL, DW, DT in zip(WLs, DWs, DTs):
    plt.plot(WL + np.arange(len(DT)) * DW, DT)
```



1.4.4 Writing FITS files

```
In [113]: # Creation of numpy array with the data.
          x = np.arange(100)

In [114]: # Creation of the HDU from the data.
          hdu = fits.PrimaryHDU(x)
          print hdu.header.cards
```

```
('SIMPLE', True, 'conforms to FITS standard')
('BITPIX', 64, 'array data type')
('NAXIS', 1, 'number of array dimensions')
('NAXIS1', 100, '')
('EXTEND', True, '')
```

```
In [115]: #Adding additional keywords to the header.
          # The automatically created header contains only the required minimum of
          # If additional keywords are needed they are added with:
          hdu.header['testkey'] = (0.001,'some test value')
```

```
In [116]: print hdu.header.cards

('SIMPLE', True, 'conforms to FITS standard')
('BITPIX', 64, 'array data type')
('NAXIS', 1, 'number of array dimensions')
('NAXIS1', 100, '')
('EXTEND', True, '')
('TESTKEY', 0.001, 'some test value')


In [117]: hdulist = fits.HDUList([hdu])
          hdulist.writeto('new.fits', clobber=True) # clobber is to overwrite is al
          hdulist.close()
```

Another way to deal with FITS tables is to use the ATpy library, we'll see this later