Merging data

- Process multiple files, merge data
- sys.argv : input from command line
- sys.stderr : messages, warnings,
  status reports
- multiline records – fasta

Practical Computing for Biologists

Chapter 11

Peter Brooks

Jan 13 2012

Managing input / output files and parameters

- Hardcoded: all file names and parameters in lines of code

  Poor practice – may need to modify code often.
  Exception: invariants; early in code, assign to variable names

- User input: OutFileN = raw_input(“Enter the output file name: ”)

  May be tedious for sequential runs with same parameters.

- Command line: specify order of elements or use tags & values

  e.g. macs, a ChIP-seq peak caller:

  ```
  $ python macs.py -t oct4.bed -c gfp.bed –name=oct4
            --gsize=2700000000 --tsize=36 --mfold=5 --diag –wig
  ```

  This may become cumbersome, difficult to read and error-prone.

- Configuration file: an auxiliary file has file names and parameter values.

  Good practice for logging and recording for “lab” notebook.

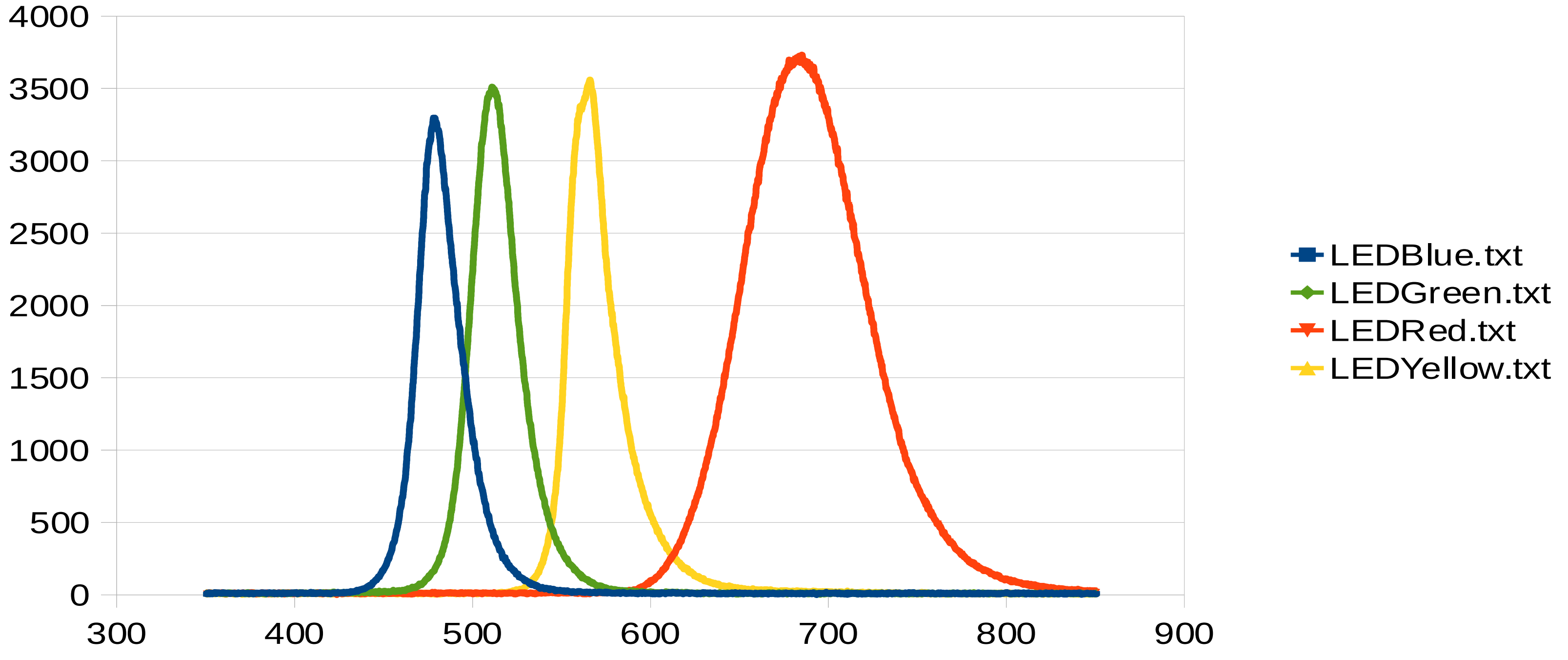
  ```
  $ python modelTest.py  20janConfig.txt
  ```

  Code opens the file and assigns the values to variables.

- As appropriate, use combinations of these styles.

Merge spectral data from several files into one file

(exercise in Chapter 11)



lambda	LEDBlue.txt	LEDGreen.txt	LEDRed.txt	LEDYellow.txt
530,06	97	1412	10	60
530,41	92	1397	11	64
530,75	93	1328	6	64
531,1	92	1297	9	72
531,45	86	1262	10	74

Quick file inspection with Unix tools before running programs

- Especially important if merging files – check for uniformity.
- Beginning of file – inspect any header lines

  ```
  $ head LEDblue.txt              shows first 10 lines
  $ head -4 *.txt        head -n  shows first n lines
  ```

- End of file : $ tail   same usage as head
- Count lines in file (and “words” and characters)

  ```
  $ wc *.txt
  ```

  Most useful is the line count.

  If < 64000 lines, spreadsheet may be tool of choice.

  To save time with large files, count only lines:

  ```
  $ wc -l *.txt
  ```

- Use same tools to check output files for expected content.
- How inspect 10 lines in the middle of a large file?

  (hint: recall Unix method to pass a result to another tool)

sys.argv

- Python module **sys** permits use of Unix tools.
- argv : "argument vector"

  “The argv library has been designed to handle the argument processing needs of most Unix software and to provide a consistent usage framework for user applications.”
- Command line elements become string elements of an argv list.

  ```
  $ python macs.py -t oct4.bed -c gfp.bed --name=oct4
            --gsize=2700000000 --tsize=36 --mfold=5 --diag --wig
  ```

  sys.argv[0] is 'macs.py', …..[1] is '-t'......
- Command line elements are arranged:
  - in a strictly defined order and using an obligatory set, or
  - by using tags-values, typically in any order;

    code provides default values for parameters not specified in command line.
- Redirected output with “ > “ is not included in argv.

Exercise with sys.argv

- Open a python program in text editor.
  Preferably in the scripts directory in specified in $PATH.
- Save with a new name, e.g. testargv.py
- Delete all except shebang.
- Usage = ' ' '

  testargv.py prints the argv list elements. It requires at least one argument.

  ' ' '
- Add 4 code lines: import sys print sys.argv

```
for MyArg in sys.argv:
    print MyArg        # Run to see program name as argv[0]
```

- Add a few words separated by spaces to command line.

```
if len(sys.argv) < 2:
    print Usage
```

  Run to see entire table of merged data.

Program for merging data

- Copy filestoXYYY.py to spectra folder in examples.
- Save with a new name, e.g. multiXYtoXYYY.py
- Edit Usage to include new name ...”modified from: ….”
  Replace combinedfile.dat with another output name.
- Avoid usage of entire program as part of an “Else”:
  Following the line: print Usage, add:
  raise SystemExit() (inside If...)
  (derived from “raise exception....” when errors encountered)
  Delete Else and dedent remaining lines.
- Before FileList = ….., add line: print sys.argv
- Run without redirection; scroll up through output to see argv and FileList print results. Try a run with no arguments.
- Alternative method: glob. Useful for data in remote directories and for defining file sets with regular expression terms.

Looping through the file list

1. Define a variable to handle input header lines.
2. Make a blank list that will contain the merged data.
3. Build a header line with each col labelled with input file name.
4. Initiate counters by setting to zero.

```
Header = 'lambda'      # column name for wavelengths
LinesToSkip=1          1

# change this for comma-delimited files
Delimiter='\t'         # Code does not use this variable.
MasterList=[]          2

FileNum=0
for InfileName in FileList:
     # use the name of the file (w/o extension) as the column Header
     Header += "\t" + InfileName   3

     Infile = open(InfileName, 'r') # it's ok for this to be in the file loop
  # the line number within each file, resets for each file
     LineNumber = 0 # reset for each file
4    RecordNum = 0

     for Line in Infile:

          .......Continued....
```

Looping through lines in each file in the file list

1. Assure more than 3 characters in the line. A blank line with only \n\r would have len=2.
2. Start a new data line only if file is first in the file list. A new line has the wavelength and the first color value (equiv. to elements 0 and 1). Problem ??
3. Build strings by concatenating the other 3 colors.
4. If data missing, then write an error message using the stderr tool of the sys module.

```
for Line in Infile:
        if LineNumber > (LinesToSkip-1) and len(Line)>3:  # skip first
Line and blanks     1

            Line=Line.strip('\n')
    2       if FileNum==0:
                MasterList.append(Line)
            else:
                ElementList=Line.split(Delimiter)
                if len(ElementList)>1:
    3               MasterList[RecordNum] += "\t" + ElementList[1]
                    RecordNum+=1
                else:
                    sys.stderr.write("Line %d not XY format in file
%s\n" % (LineNumber,InfileName))      4

        LineNumber+=1

    FileNum += 1 # the last statement in the file loop
    Infile.close()
```

Critique of program

- Program makes concatenated string outputs.
- Advantages?, disadvantages?
- Alternate strategies for “merging” data?
  Dependent on context and anticipated use.
- How manage rigorous inspection of input data?

Reading multiline records – e.g. fasta format

1. Empty list.
2. Empty dictionary.
3. Define fasta id as list entry or as dictionary key.
4. Concatenate sequential lines as list within list or as value for dictionary key.

```
1    RecordNum = -1  # don't have the zeroth record yet
     Sequences=[]
2    SeqDict={}
     for Line in Infile:
             Line = Line.strip()
                     if Line[0]=='>':
                     # we have a new record name
                     Name=Line[1:]  # chop off the > at the front
                     # Make a 2-item list with the name as the first element,
                     # and an empty string as the second
3                    Sequences.append([Name,' ' ])
                     RecordNum += 1 # Now we have a record
                     # Use the Name for the dictionary key
                     SeqKey = Name
                      # create a blank dictionary entry to append later
                     SeqDict[SeqKey] = " "

             else:  # this means we are not on a line with a name
                     if RecordNum > -1:  # are we past any header lines?
4                            # Add on to the end of the 2nd element of the list
                             Sequences[RecordNum][1] += Line
                             # Add to the dictionary value for the present Key
                             SeqDict[SeqKey] += Line
     Infile.close()
```