Lecture 6

Numerical Computing

Prof. Steven Ludtke
N410.07, sludtke@bcm.edu

Lab #1

- Download the starter program from the class website. The starter program queries PubMed for articles written by a specific author, and prints the number of publications and number of coauthors. Try the program and understand what it's doing Use this program as a starting point for the following exercise. Each person in the group should select and complete one task (any of the 4), then one person should combine all of the work into a single program, which should be emailed to the TA before the end of class. You should coordinate your data representations so your code will all work together. Make sure each person in the group understands how all of the components work before leaving. As with lab 1, the tasks are ordered from most difficult to easiest:

1. Rather than simply counting the number coauthors in the retrieved records, tabulate the number of times each coauthor name appears, then print a) the number of coauthors and b) the top 10 coauthors with their publication count.

2. Similar to 1), tabulate all of the words present in the title of all of the retrieved records. Eliminate common words like "the" and "and", and print an ordered list of the top 20 occurring words.

3. Very often it is impossible to identify authors uniquely by their PubMed names. "Ludtke SJ" could be "Ludtke S". "Chen F" could represent dozens of different authors. While there is no perfect solution to this problem, to help assess the impact of the first issue, count the number of unique AU values, and also count the number of unique family names. Print the difference between these counts as the number of possible overlaps.

4. Modify the program to query based on a provided keyword rather than an author's name. Pubmed does not like overly large queries, so be sure to restrict the number of retrieved records to a reasonable number (no more than 1000). Be sure to print a warning if this limit was reached.

Lab 1 Review

```
from Bio import Entrez
from Bio import Medline

author=input("Enter author name: ")

# The following line MUST contain your valid email address
Entrez.email = "user@bcm.edu"

# Do a pubmed query to find all papers by a specific author
handle = Entrez.esearch(db="pubmed", term="{}[Author]".format(author), retmax=500)
record = Entrez.read(handle)
handle.close()

# record is a dictionary-like object. IdList is a list of pubmedid numbers matching the query
ids=record["IdList"]

# fetch all of the records in the list of matching ids
handle=Entrez.efetch(db="pubmed",id=tuple(ids),rettype="medline")
records = list(Medline.parse(handle))
handle.close()

# find all unique coauthors
coauth=set()
for r in records:
    coauth.update(r["AU"])

print("Coauthors:",coauth)
print(len(ids)," matching records")
print(len(coauth)," coauthors")
```

Medline Terms

Affiliation [AD]	Investigator [IR]	Pharmacological Action [PA]
Article Identifier [AID]	ISBN [ISBN]	Place of Publication [PL]
All Fields [ALL]	Issue [IP]	PMID [PMID]
Author [AU]	Journal [TA]	Publisher [PUBN]
Author Identifier [AUID]	Language [LA]	Publication Date [DP]
Book [book]	Last Author [LASTAU]	Publication Type [PT]
Comment Corrections	Location ID [LID]	Secondary Source ID [SI]
Corporate Author [CN]	MeSH Date [MHDA]	Subset [SB]
Create Date [CRDT]	MeSH Major Topic [MAJR]	Supplementary Concept[NM]
Completion Date [DCOM]	MeSH Subheadings [SH]	Text Words [TW]
EC/RN Number [RN]	MeSH Terms [MH]	Title [TI]
Editor [ED]	Modification Date [LR]	Title/Abstract [TIAB]
Entrez Date [EDAT]	NLM Unique ID [JID]	Transliterated Title [TT]
Filter [FILTER]	Other Term [OT]	UID [PMID]
First Author Name [1AU]	Owner	Version
Full Author Name [FAU]	Pagination [PG]	Volume [VI]
Full Investigator Name [FIR]	Personal Name as Subject [PS]	
Grant Number [GR]		

Lab 1 Review - 4

```
from Bio import Entrez
from Bio import Medline

words=input("Enter search words: ")

# The following line MUST contain your valid email address
Entrez.email = "user@bcm.edu"

# Do a pubmed query to find all papers by a specific author
handle = Entrez.esearch(db="pubmed", term="{}[TIAB]".format(words), retmax=500)
record = Entrez.read(handle)
handle.close()

# record is a dictionary-like object. IdList is a list of pubmedid numbers matching the query
ids=record["IdList"]

# fetch all of the records in the list of matching ids
handle=Entrez.efetch(db="pubmed",id=tuple(ids),rettype="medline")
records = list(Medline.parse(handle))
handle.close()

# find all unique coauthors
coauth=set()
for r in records:
    coauth.update(r["AU"])

print("Coauthors:",coauth)
print(len(ids)," matching records")
print(len(coauth)," coauthors")
```

Lab 1 Review - 3

```
from Bio import Entrez
from Bio import Medline

author=input("Enter author name: ")

# The following line MUST contain your valid email address
Entrez.email = "user@bcm.edu"

# Do a pubmed query to find all papers by a specific author
handle = Entrez.esearch(db="pubmed", term="{}[Author]".format(author), retmax=500)
record = Entrez.read(handle)
handle.close()

# record is a dictionary-like object. IdList is a list of pubmedid numbers matching the query
ids=record["IdList"]

# fetch all of the records in the list of matching ids
handle=Entrez.efetch(db="pubmed",id=tuple(ids),rettype="medline")
records = list(Medline.parse(handle))
handle.close()

# find all unique coauthors
coauth=set()
coauthlast=set()
for r in records:
     coauth.update(r["AU"])
     coauthlast.update(r["AU"].split()[0])  # <-- nope… won't work right

print("Coauthors:",coauth)
print(len(ids)," matching records")
print(len(coauth)," coauthors with ",len(coauth)-len(coauthlast),"possible overlaps")
```

Lab 1 Review - 3

```
from Bio import Entrez
from Bio import Medline

author=input("Enter author name: ")

# The following line MUST contain your valid email address
Entrez.email = "user@bcm.edu"

# Do a pubmed query to find all papers by a specific author
handle = Entrez.esearch(db="pubmed", term="{}[Author]".format(author), retmax=500)
record = Entrez.read(handle)
handle.close()

# record is a dictionary-like object. IdList is a list of pubmedid numbers matching the query
ids=record["IdList"]

# fetch all of the records in the list of matching ids
handle=Entrez.efetch(db="pubmed",id=tuple(ids),rettype="medline")
records = list(Medline.parse(handle))
handle.close()

# find all unique coauthors
coauth=set()
coauthlast=set()
for r in records:
    for a in r["AU"]:
        coauth.add(a)
        coauthlast.add(a.split()[0])

print("Coauthors:",coauth)
print(len(ids)," matching records")
print(len(coauth)," coauthors with ",len(coauth)-len(coauthlast),"possible overlaps")
```

Lab 1 Review - 1

```
from Bio import Entrez
from Bio import Medline

author=input("Enter author name: ")

# The following line MUST contain your valid email address
Entrez.email = "user@bcm.edu"

# Do a pubmed query to find all papers by a specific author
handle = Entrez.esearch(db="pubmed", term="{}[Author]".format(author), retmax=500)
record = Entrez.read(handle)
handle.close()

# record is a dictionary-like object. IdList is a list of pubmedid numbers matching the
query
ids=record["IdList"]

# fetch all of the records in the list of matching ids
handle=Entrez.efetch(db="pubmed",id=tuple(ids),rettype="medline")
records = list(Medline.parse(handle))
handle.close()

# find all unique coauthors
coauth=set()
for r in records:
    coauth.update(r["AU"])

print("Coauthors:",coauth)
```

Lab 1 Review - 1

```
handle=Entrez.efetch(db="pubmed",id=tuple(ids),rettype="medline")
records = list(Medline.parse(handle))
handle.close()

coauth=set()
for r in records:
    coauth.update(r["AU"])

print "Coauthors:",coauth
```

Lab 1 Review - 1

```
handle=Entrez.efetch(db="pubmed",id=tuple(ids),rettype="medline")
records = list(Medline.parse(handle))
handle.close()

auth={}
for r in records:
        for a in r["AU"]:
                try: auth[a]+=1
                except: auth[a]=1

def second(x): return x[1]

authl=sorted(auth.items(),key=second,reverse=True)

print("Coauthors:",authl[:10])
```

Lambda

- Unnamed single-use functions:

lambda x,y: x*y*23

- equivalent to:

def f(x,y) : return x*y*23

Lab 1 Review - 1

```
handle=Entrez.efetch(db="pubmed",id=tuple(ids),rettype="medline")
records = list(Medline.parse(handle))
handle.close()

auth={}
for r in records:
        for a in r["AU"]:
                try: auth[a]+=1
                except: auth[a]=1

authl=sorted(auth.items(),key=lambda a: a[1],reverse=True)

print("Coauthors:",authl[:10])
```

Lab 1 Review - 1

```
handle=Entrez.efetch(db="pubmed",id=tuple(ids),rettype="medline")
records = list(Medline.parse(handle))
handle.close()

from collections import Counter
c=Counter()
for r in records:
    c.update(r["AU"])

print("Coauthors:",c.most_common(10))
```

Homework Review

(print a log table)

```
from math import *

# The bases of our logs
cols=(2.0,e,10.0)

# The title row
print("    ",end="")
for col in cols: print("{:9.5f}".format(col),end="")
print("")

# rows are the outer loop
for row in range(1,33):
    # Start with row number (value) column
    print("{:<4.0f}".format(row),end="")

    # now show each column
    for col in cols:
        print("{:9.5f}".format(log(row,col)),end="")
    print("")
```

Homework Review

(min/max from 2 column text file)

```
from sys import argv

lines=file(argv[1],"rU")
xvalues=[]
yvalues=[]
for l in lines:
   v=l.split()
   xvalues.append(float(v[0]))
   yvalues.append(float(v[1]))

print("X: {} - {}".format(min(xvalues),max(xvalues)))
print("Y: {} - {}".format(min(yvalues),max(yvalues)))
```

Homework Review

```
import numpy
from sys import argv

xy=numpy.loadtxt(argv[1]).transpose()

print("X: {} - {}".format(min(xy[0]),max(xy[0])))
print("Y: {} - {}".format(min(xy[1]),max(xy[1])))
```

IEEE Floating Point

```
a=10000.0
b=0
for i in range(10000): b+=1
a==b

?
```

IEEE Floating Point

```
a=10000.0
b=0
for i in range(80000): b+=0.125
a==b

?
```

IEEE Floating Point

```
a=10000.0
b=0
for i in range(100000): b+=0.1
a==b

?
```

Decimal Numbers

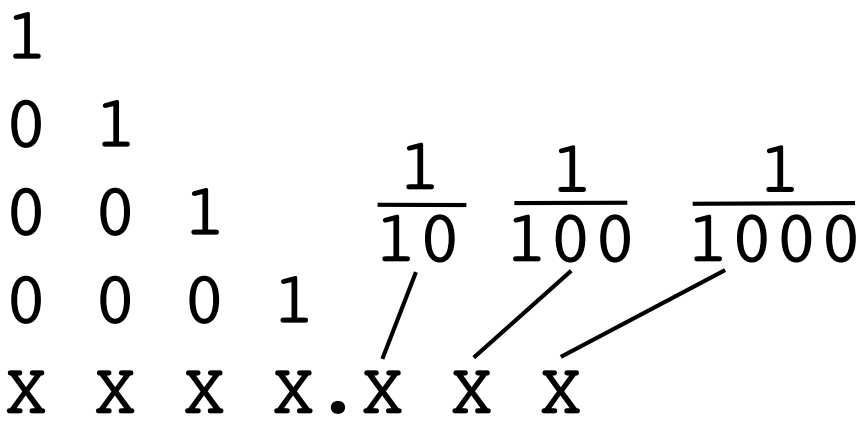
```
1
0  1
0  0  1
0  0  0  1
x  x  x  x
```

Binary Numbers

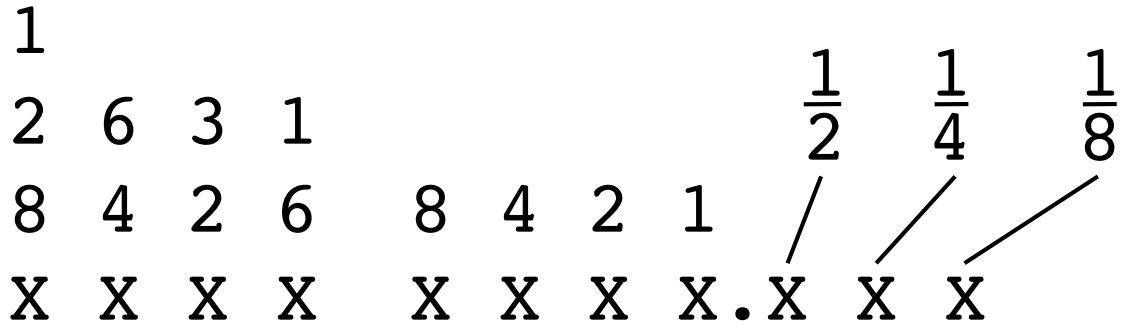
```
        1
        2  6  3  1
        8  4  2  6     8  4  2  1
        x  x  x  x     x  x  x  x
```

dec	binary
1	1
2	10
4	100
8	1000
15	1111
212	1101 0100

Decimal Numbers



Binary Numbers




```
0.25      0000 0000.010
0.625     0000 0000.101
0.3       0000 0000.0100 1100 1100 ...
```

IEEE Floating Point

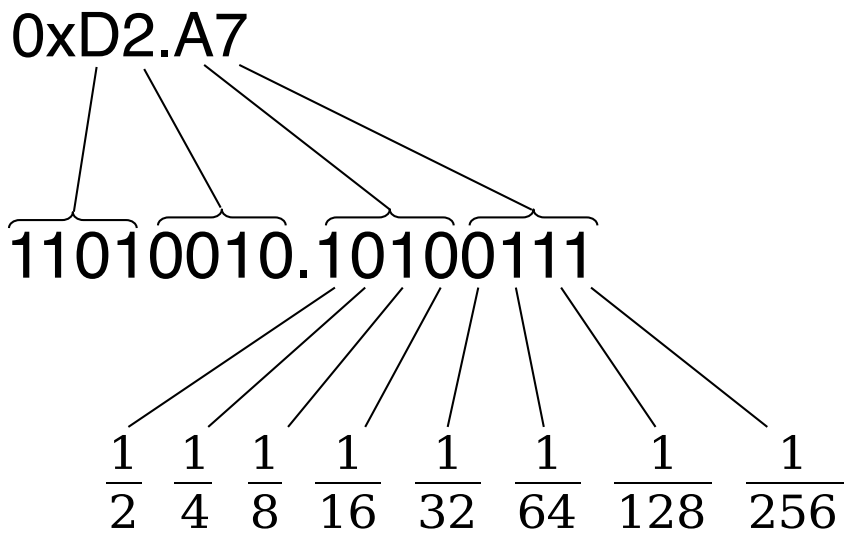
- "Binary Scientific Notation"
- 127.25 1111111.01
- $1.2725 \times 10^{2} \rightarrow 1.11111101 \times 10^{110}$

- Single:
- SEEEEEEE EMMMMMMM MMMMMMMM MMMMMMMM
- S – Sign bit 0=+
- E – Exponent, bias 127
- M – Significand (Mantissa), implicit 1 when normalized

the 'decimal point' in binary

Hexadecimal

Binary



=210.65234375
=$2.1065234375 \times 10^{2}$
or
$1.10100101010011 1 \times 10^{111}$

Digital Representation of Numbers

- `Bit` `0-1`
- `Nibble` `(4 bits)` `0-15`
- `Byte (char)` `(8 bits)` `0-255`
- `Word (short)` `(16 bits)` `0-65,535`
- `Longword (long)` `(32 bits)` `0-4,294,967,296`
- `Long Longword` `(64 bits)` `0-1.844x10`^{19}
- `Float` `(32 bits)` `10`^{38} `(7 digits)`
- `Double` `(64 bits)` `10`^{308} `(15 digits)`

normal Python

Python “long integers” are a special object, not a standard number.

Numerical Processing

- decimal - When you need accurate decimal values
- fractions - Rational fractions

- NumPy - Fast arrays, linear algebra, etc.
- Matplotlib - Matlab-style plotting interface
- SciPy - Large library of numerical computing algorithms

Why is this slow?

```
from math import *

x=[i/1000000.0 for i in xrange(10000000)]
y=[sin(i) for i in x]
```

How about this ?

```
from numpy import *

x=arange(0,10.0,0.000001)
y=sin(x)
```

NumPy

- **http://csc.ucdavis.edu/~chaos/courses/nlp/Software/NumPyBook.pdf** # numpy book

- from numpy import *
- a=arange(60)
- b=a.reshape(10,6) # make 2-D matrix
- c=a.reshape(3,4,5) # make 3-D (tensor)
- b.shape # current dimensions
- b.size # total number of elements
- b.ndim # dimensionality
- b.dtype # type of value stored
- b.astype("")

Type	Bit-Width	Character
bool_	boolXX	'?'
byte	intXX	'b'
short		'h'
intc		'i'
int_		'l'
longlong		'q'
intp		'p'
ubyte	uintXX	'B'
ushort		'H'
uintc		'I'
uint		'L'
ulonglong		'Q'
uintp		'P'
single	floatXX	'f'
float_		'd'
longfloat		'g'
csingle	complexXX	'F'
complex_		'D'
clongfloat		'G'
object_		'O'
str_		'S#'
unicode_		'U#'
void		'V#'

NumPy

- a=zeros((nx,ny,...))
- a=fromfunction(lambda i,j:i+j,(4,5))
- a=loadtxt(filename) # read multicolumn data from a text file
- a=arange(0,20,.1) # fractional version of range()
- a*10 # multiply each element !
- b=sin(a) # sin() of each element
- c=a[c>0] # condition, returns elements >0
- c.sort() # sort values in-place
- c.mean(),var(),std(),prod() # average, variance, standard dev, product
- inner(a,b), outer(a,b) # inner and outer matrix products
- dot(a,b), cross(a,b) # dot and cross products (similar to above)
- histogram(a,bins,range) # compute a histogram of 'a'

NumPy

- a=mat(arange(8))
- b=mat(arange(8)).transpose()
- a*b
- b*a
- numpy.linalg.eig(b*a)

SciPy

- Clustering package (scipy.cluster)
- Constants (scipy.constants)
- Fourier transforms (scipy.fftpack)
- Integration and ODEs (scipy.integrate)
- Interpolation (scipy.interpolate)
- Input and output (scipy.io)
- Linear algebra (scipy.linalg)
- Maximum entropy models (scipy.maxentropy)
- Miscellaneous routines (scipy.misc)
- Multi-dimensional image processing (scipy.ndimage)
- Orthogonal distance regression (scipy.odr)
- Optimization and root finding (scipy.optimize)
- Signal processing (scipy.signal)
- Sparse matrices (scipy.sparse)
- Sparse linear algebra (scipy.sparse.linalg)
- Spatial algorithms and data structures (scipy.spatial)
- Special functions (scipy.special)
- Statistical functions (scipy.stats)
- Image Array Manipulation and Convolution (scipy.stsci)

matplotlib (pylab)

- Matlab-like plotting library
- **http://matplotlib.sourceforge.net/users/pyplot_tutorial.html**

```
ipython –pylab=tk      # special mode for interaction with pylab
x=arange(0,4*pi,0.05)  # from numpy
y=sin(x)               # easy to apply a function to a list of values
plot(x,y)              # plot x,y and open a display window
```

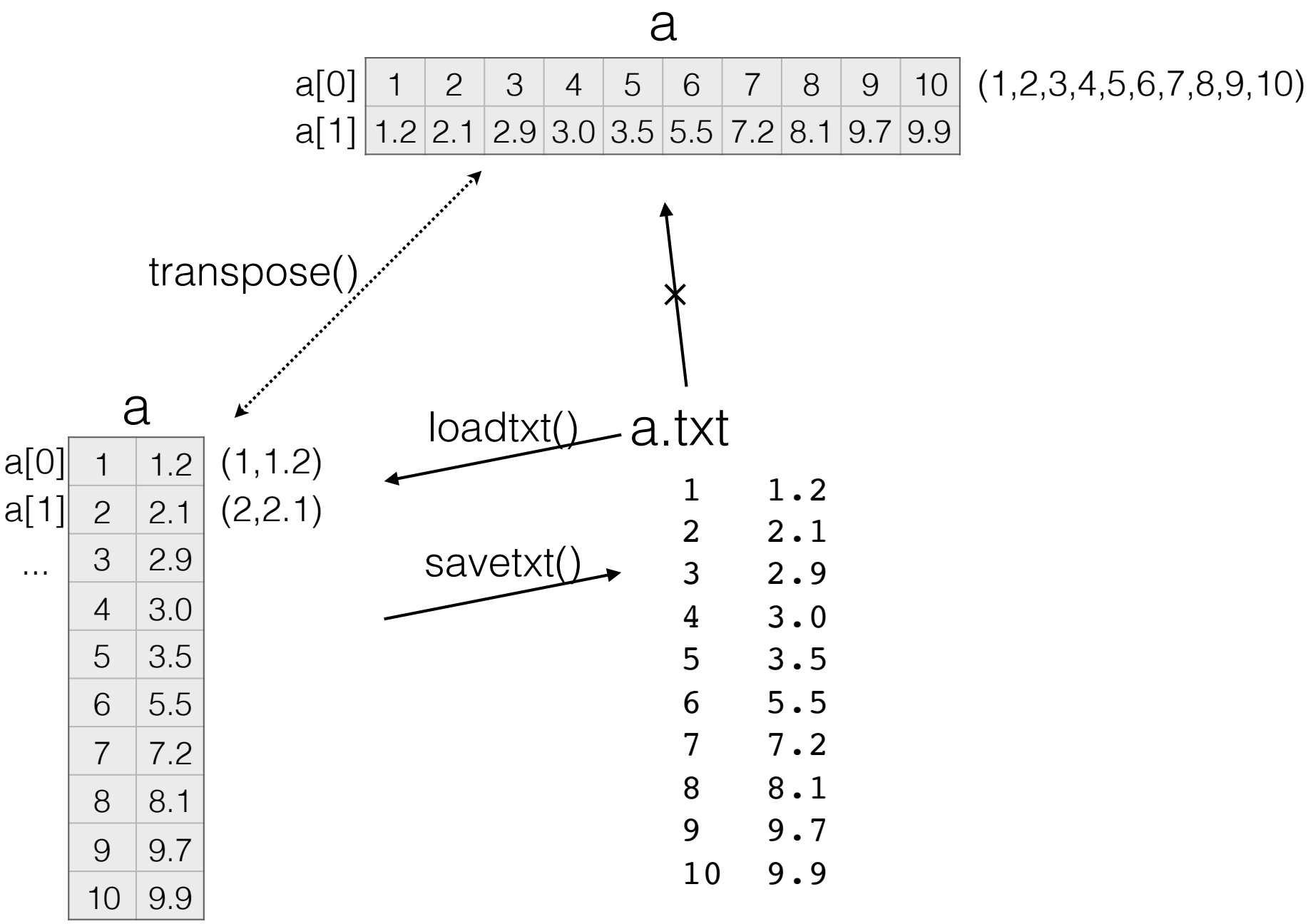
OR

```
python
from pylab import *    # <-- only if you don't use ipython
x=arange(0,4*pi,0.05)
y=sin(x)               # easy to apply a function to a list of values
plot(x,y)              # plot x,y and open a display window
show()                 # opens the plot window (blocks on some machines)
```

matplotlib (pylab)

```
ipython --pylab=tk     # special mode for interaction with pylab

x=arange(0,4*pi,0.05)  # from numpy

y=sin(x)               # easy to apply a function to a list of values

y2=cos(x)

figure(1)              # start a new figure

subplot(211)           # make a 1x2 set of plots and move to 1st

plot(x,y)              # plot x,y and open a display window

ylabel("sin(x)")

subplot(212)           # start on the 2nd subplot

plot(x,y2)             # second plot

ylabel("cos(x)")

xlabel("x")
```

NumPy.array


a
a[0] 1 2 3 4 5 6 7 8 9 10 (1,2,3,4,5,6,7,8,9,10)
a[1] 1.2 2.1 2.9 3.0 3.5 5.5 7.2 8.1 9.7 9.9
transpose()
a
a[0] 1 1.2 (1,1.2)
a[1] 2 2.1 (2,2.1)
... 3 2.9
4 3.0
5 3.5
6 5.5
7 7.2
8 8.1
9 9.7
10 9.9
loadtxt()
savetxt()
a.txt
1 1.2
2 2.1
3 2.9
4 3.0
5 3.5
6 5.5
7 7.2
8 8.1
9 9.7
10 9.9

Homework 6

1. When you email your homework (make sure to cc sludtke@bcm.edu), it should include a description of your planned class project. You should state the general purpose of the program, define what inputs your program will take, and what outputs it will produce.

2. Write a simple X-Y plotting program. It should ask the user for the name of a text file containing columns of numbers (sample file on website), and the number of the column containing the 'x' values and the number of the column containing the 'y' values. Plot the data (scatter plot or line plot) and save the result as a PDF file.