Lecture 5

BioPython
NumPy

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

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 and Plotting

- decimal - When you need accurate decimal values ($)
- fractions - Rational fractions

- NumPy - Fast arrays, linear algebra, etc. (http://www.numpy.org)
- Pandas - Spreadsheet-style data manipulation (https://pandas.pydata.org/)
- SciPy - Large library of numerical computing algorithms (https://scipy.org)
- Matplotlib - Matlab-style plotting interface - publication quality output
- Bokeh - Web-based interactive plotting (https://bokeh.pydata.org)
- Jupyter ipywidgets - Interactive web-based parameters (https://ipywidgets.readthedocs.io)

Why is this slow?

```
from math import *

x=[i/5000000.0 for i in range(50000000)]
y=[sin(i) for i in x]

print(y[-1])
```

How about this ?

```
from numpy import *

x=arange(0,10.0,0.0000002)
y=sin(x)

print(y[-1])
```

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#'

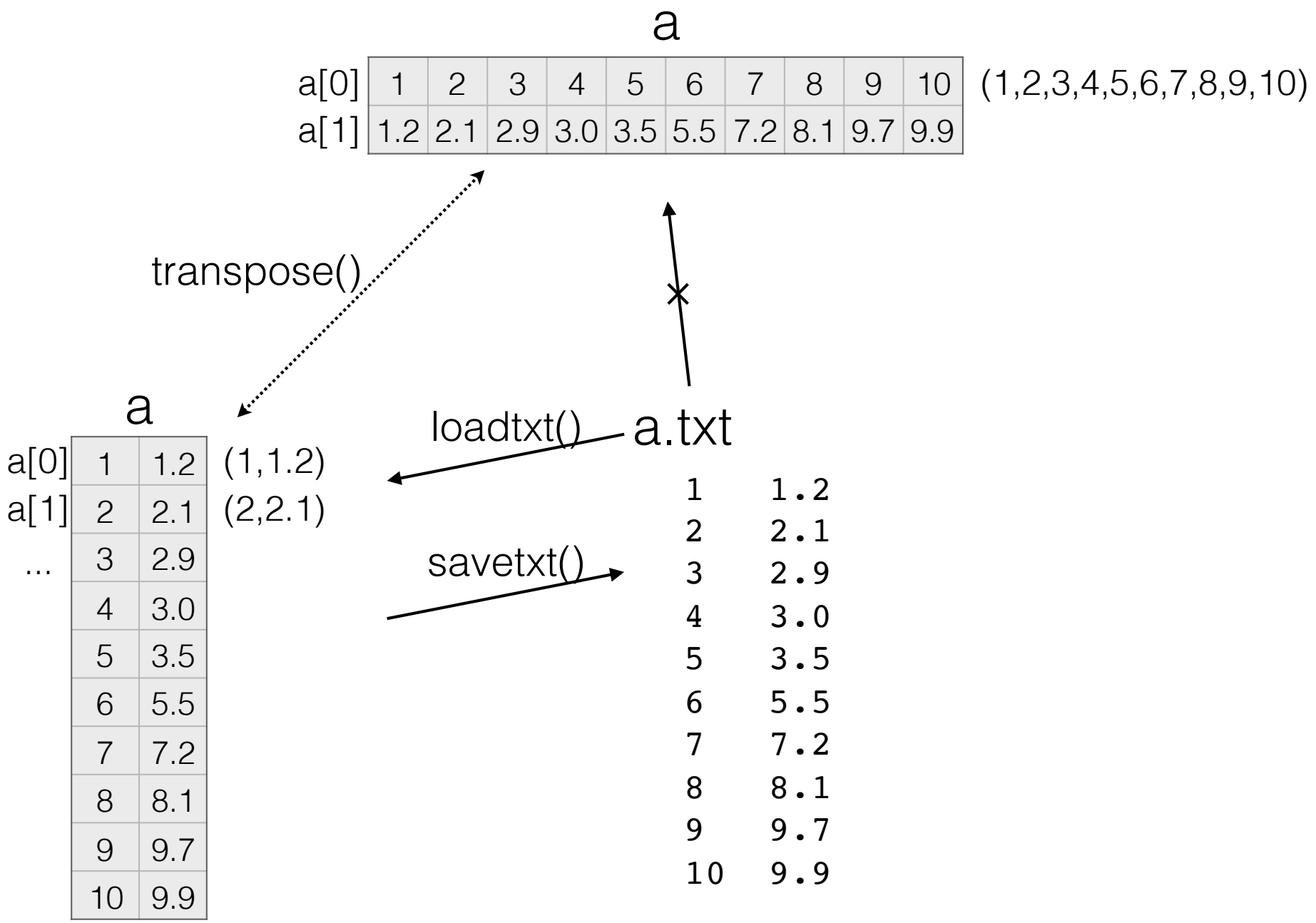
NumPy

- 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)

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

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**

```
Using Jupyter:

from pylab import *

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
```

OR (from the command line):

```
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
```

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")
```

Jupyter ipywidgets

```
from pylab import *
from ipywidgets import interact,interact_manual

def f(a=1.0):
    x=arange(0,4*pi,.01)
    y=sin(x*a)
    plot(x,y)
    show()
interact(f,a=(0.1,10.0))
```

Bokeh

```
from bokeh.plotting import figure, output_notebook, show

# prepare some data
x = [0.1, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0]
y0 = [i**2 for i in x]
y1 = [10**i for i in x]

output_notebook()      # output to Jupiter

# create a new plot
p = figure(
   tools="pan,box_zoom,reset,save",
   y_axis_type="log", y_range=[0.001, 10**11], title="log axis example",
   x_axis_label='sections', y_axis_label='particles'
)

# add some renderers
p.line(x, x, legend="y=x")
p.circle(x, x, legend="y=x", fill_color="white", size=8)
p.line(x, y0, legend="y=x^2", line_width=3)
p.line(x, y1, legend="y=10^x", line_color="red")
p.circle(x, y1, legend="y=10^x", fill_color="red", line_color="red", size=6)

show(p)
```