Python for Data Analysis 1

09/02/14

Blake Jacquot

Motivation

- Can Python accommodate routine data analysis task for 3D image arrays (row, column, time) in manner that compares favorably to Matlab?
  - The answer is ‘Yes’

- Python is free and open source with potential to replace costly Matlab

Key Points

- The Python array structure in the numerical python toolbox (numpy) allows very similar command structures to Matlab and comparable processing speeds

- Python processing speed compares favorably with Matlab
  - Python computes slightly faster than Matlab for median-based operations tested
  - Python computes slightly slower than Matlab for most other operations. But they are close.

- Python defaults reading with C style indexing. To enforce same 3D structure as Matlab use the ‘F’ option to force Fortran-style indexing when reshaping arrays (Matlab is based on Fortran syntax).
  - If Fortran order is not enforced, 3D arrays may have structure (z-dim, row, col)
  - C style has slowest varying index first, Fortran has it last
  - Numpy defaults to column-major order (C order), while Matlab defaults to column-major order (Fortran order)

Tabulated Process Times

Matlab Command	Result	Time Elapsed (sec)	Python Command	Result	Time Elapsed (sec)	Winning Platform for Operation (Time Difference)	Note
Median(matlab_data(:)	37.0	0.238	Np.median(data[:])	37.0	0.132	Python (0.106 sec)	Store all pixels in 1D array before processing
mean(matlab_data(:)	64.32	0.004	Np.mean(data[:])	64.32	0.013	Matlab (0.009 sec)	Store all pixels in 1D array before processing
std(matlab_data(:)	68.0	0.074	Np.std(data[:])	68.0	0.077	Matlab (0.003 sec)	Store all pixels in 1D array before processing
max(matlab_data(:)	255	0.006	Np.max(data[:])	255	0.011	Matlab (0.005 sec)	Store all pixels in 1D array before processing
median(matlab_data,3);		0.184	np.median(data, axis = 0)		0.131	Python (0.053sec)	Process pixels in 3D array (same as images on ‘Processing’ slide)
mean(matlab_data,3)		0.005	np.mean(data, axis = 0)		0.016	Matlab (0.011 sec)	Process pixels in 3D array (same as images on ‘Processing’ slide)
std(matlab_data,0,3);		0.084	np.std(data, axis = 0)		0.084	Tie	Process pixels in 3D array (same as images on ‘Processing’ slide)
max(matlab_data,[],3);		0.005	np.max(data, axis = 0)		0.015	Matlab (0.01 sec)	Process pixels in 3D array (same as images on ‘Processing’ slide)

- 1D processing is done for vector of 500 x 300 x 51 = 7,650,000 pixels in double precision
- 3D processing is done for 500 x 300 x 51 array in double precision

Viewing Images

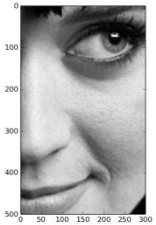
imtool(matlab_data(:,:,4))

```
import matplotlib.pyplot as plt
plt.imshow(data[:,:,2], cmap=plt.cm.gray)
plt.show()
```

Matlab = (row, col, z-dim) ⟵ | ⟶ Python = [row, col, z-dim]

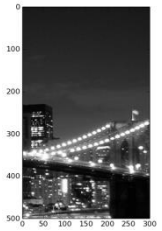
matlab_data(: , : , 1)

data[0 , : , :]



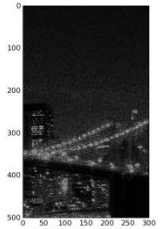
matlab_data(:,:,2)

data[1 , : , :]



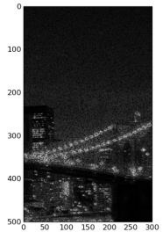
matlab_data(:,:,3)

data[2 , : , :]



matlab_data(:,:,4)

data[3 , : , :]



- Images are the same whether viewed in Matlab or Python
- Array used is 3D stack of 51 images with size row= 500, col = 300

Processing

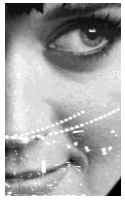
Matlab = (col, row, z-dir)

Python = [z-dir, row, col]

median(matlab_data,3);
Time for operation = 0.184 sec

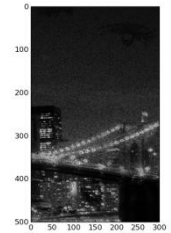
mean(matlab_data,3);
Time for operation = 0.005 sec

std(matlab_data,0,3);
Time for operation = 0.084 sec

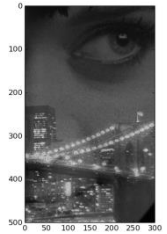


max(matlab_data,[],3);
Time for operation = 0.005 sec

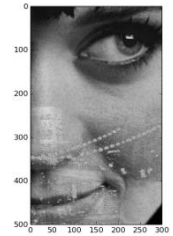
np.median(data, axis = 0)
Time for operation = 0.131 sec



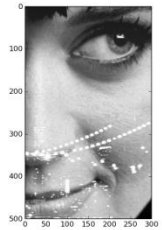
np.mean(data, axis = 0)
Time for operation = 0.016 sec



np.std(data, axis = 0)
Time for operation = 0.084 sec



np.max(data, axis = 0)
Time for operation = 0.015 sec



- Processing of 3D images take comparable time whether done in Matlab or Python
- Processing is done on 3D stack of 51 images with size row= 500, col = 300

Matlab vs. Python Command for IO and Reshaping

Matlab = (row,col,z-dim) Python = [row,col,z-dim]

3D array used is (500,300,51) with double precision

Write data to file from Matlab with double precision

```
%write image to binary file from Matlab with double precision.
fid = fopen('matlab_data.dat', 'w+');
fwrite(fid,matlab_data, 'double');
fclose(fid);
```

Write data to file in Python with double precision

Not explored here

Read data to file from Matlab with double precision

Not explored here

Read data from file in Python with double precision

```
import numpy as np #numerical computing library
import os #navigates operating system, used for changing folders
import struct #allows interpretation of strings into binary data

currpath = os.getcwd()
os.chdir(currpath)
currfile = 'matlab_data.dat'
f = open(currfile,'rb') #rb = read binary
data = f.read()
f.close()

#Since data is in string format, we need to convert to double
index = 0
pixels = []
while index < len(data):
    curr_string = data[index:index+8]
    curr_pix = struct.unpack('d',curr_string) #'d' is double
    pixels.append(curr_pix)
    index+=8 #eight string characters make up a double (8 bytes)

#reshape data
data = np.reshape(pixels,(500,300,51),'F') #'F' refers to interpreting as Fortran
ordering, instead of default C ordering
```

Notes

- Python environment: Enthought Canopy 1.3.0 (64 bit)
- 192 Gbytes or RAM
- 64 bit, multi-core processor
- Matlab and Python data stored with double precision