Practice with NumPy

Aidan Hunt, University of Washington

Learning Objectives

After this lesson, students will be able to

- Generate NumPy arrays from the contents of files
- Traverse NumPy arrays with loops
- Visualize the contents of NumPy arrays with matplotlib

Check-in

- Homework 3 due Friday (start early!)
- Come to office hours or post on discussion board if you have issues
- Homework 4 will be posted on Friday (smaller problem plus creative programming)

Framing the Problem

Today, work an example that will allow us to practice with NumPy and plotting.

Objective: Post-process and visualize data from an Acoustic Doppler Velocimeter (ADV):

- Measures fluid velocity at a point using reflections of acoustic "pings"
- Measures X, Y, and 2 estimates of Z velocity (Z1 and Z2)

Provided text files:

- `vectrinoData.txt` contains raw data output from the sensor
  - First column: Timestamps (s)
  - Columns 2-5: Beam velocities (m/s)
  - Columns 6-9: Beam amplitudes
  - Columns 10-13: Beam correlation as integer between 0 and 255 (255 = perfect correlation)
- `vectrinoTransform.txt` contains a 4x4 coordinate transformation matrix for transforming from beam coordinates to XYZ coordinates

Post processing pipeline:

- Import data
- Filter readings with correlations below a certain % threshold
- Convert beam velocities to XYZ
- Plot filtered and unfiltered velocities

---

1) Pseudocode

Our pseudocode is essentially the post processing pipeline!

In [ ]:
```
import numpy as np

dataFile = 'vectrinoData.txt'
coordFile = 'vectrinoTransform.txt'

# Import data
    # Read from txt file using numpy?

# Clean data
    # Identify "bad readings" based on correlation threshold
    # Interpolate other these readings using linear interpolation

# Transform data to XYZ coordinates
    # Use transformation matrix for this

# Plot
    # Need both filtered and raw velocities
    # Select beam to plot
    # will need a plotting package for this
```

---

2) Write function for importing the data

NumPy provides a handy function for importing simple text data: `numpy.loadtxt`

In [ ]:
```
np.info(np.loadtxt)
```

Let's use this to import the data. Since this is a distinct task, define a function:

- (Note, we need to specify the delimiter or else we get an error)

In [ ]:
```
def importData(dataFile):
    '''
    Given a Vectrino data file, extracts and returns timestamps, beam
    velocities, and beam correlations.

    Parameters
    ----------
    dataFile : str
        Name of the Vectrino .txt file

    Returns
    -------
    timeStamps : numpy array of floats
        Timestamp for each sample, in seconds.
    beamVels : numpy array of floats
        nx4 array of velocity readings, in m/s. Rows represent samples,
        columns represent each beam.
    beamCorr : numpy array of floats
        nx4 array of beam correlations, in %. Rows represent samples, columns
        represent each beam.

    '''

    # Import full data matrix
```

```
    allData = np.loadtxt(dataFile, delimiter=',')

    # Extract quantities of interest
    timeStamps = allData[:,0]
    beamVels = allData[:,1:5]
    beamCorr = allData[:,9:] / 255 * 100

    return timeStamps, beamVels, beamCorr
```

In [ ]:
```
timeStamps, beamVels, beamCorr = importData('vectrinoData.txt')

print(beamVels)
```

---

3) Write function for cleaning velocity

Let's think about this a bit more (draw on board):

- We want to throw out all readings for a given sample if the correlation on any beam is less than a threshold
- For each beam, we want to replace those readings with linear interpolation based on the surrounding "good" points

Take 1 min to pseudocode this

In [ ]:
```
def cleanVelocity(timeStamps, beamVels, beamCorr):
    # Identify correlations below a particular threshold
        # Convert from elementwise true/false to row-wise true/false


    # For each beam
        # Identify "good" points to use as our interpolation basis
        # Generate "clean" time series using interpolation

    # Return the cleaned velocity vector
```

Code it up:

- Use `np.any` to consider entire rows as "bad" if one element is "bad".
- Use `np.interp` to perform interpolation.
- Add threshold as optional parameter because maybe we want to change it!

In [ ]:
```
def cleanVelocity(timeStamps, beamVel, beamCorr, corrThresh=75):
    '''
    Given Vectrino time stamps, beam velocities, beam correlations, removes
    readings where the correlation on any beam is below a specified threshold,
    and replaces them via linear interpolation of the surrounding points.
    The cleaned velocities are returned.

    Parameters
    ----------
    timeStamps : numpy array of floats
        Timestamp for each sample, in seconds.
    beamVels : numpy array of floats
        nx4 array of velocity readings, in m/s. Rows represent samples,
        columns represent each beam.
    beamCorr : numpy array of floats
        nx4 array of beam correlations, in %. Rows represent samples, columns
```

```
        represent each beam.
    corrThresh : float, optional
        Threshold (as a percentage) at and below which readings are replaced via linear
        interpolation. The default is 75.

    Returns
    -------
    cleanVel : numpy array of floats
        nx4 array of velocity readings with "bad" readings replaced, in m/s.
        Rows represent samples, columns represent each beam.

    '''

    # Identify "bad" readings
    badReading = beamCorr <= corrThresh
    badRows = np.any(badReading, axis=1) # If an element of a row is false, the whole ro

    # Preallocate matrix for filtered readings
    cleanVel = np.zeros(beamVel.shape)
    nBeams = beamVel.shape[-1]

    for i in range(nBeams): # For each beam of the Vectrino
        # Pull out "good" points to base interpolation on
        goodTime = timeStamps[~badRows]
        goodVel = beamVel[~badRows, i]

        # Interpolate all time stamps with good time stamps as basis
        cleanVel[:,i] = np.interp(timeStamps, goodTime, goodVel)

    return cleanVel
```

In [ ]:
```
beamVel = cleanVelocity(timeStamps, beamVels, beamCorr)
```

---

4) Write a function for performing the coordinate transformation

First, we need to import the transformation matrix (use `np.loadtxt` again).

In [ ]:
```
tMat = np.loadtxt('vectrinoTransform.txt', delimiter=',')
```

Now perform the transformation.

- Coordinate transformation -> matrix product
- Transformation matrix usually operates on column vector (where rows are unique dimenisons).
- But here, the columns are unique dimensions

So how do we apply it?

- Use `@` operator to perform matrix multiplication
- Transpose beam velocities to match convention we are used to for rotation matrices
- Transpose back so that matrix is shaped how we expect

In [ ]:
```
def coordinateTransform(beamVels, tMat):
    '''
    Converts velocities in beam coordinates to velocities in xyz coordinates

    Parameters
```

```
    ----------
    beamVels : numpy array of floats
        nx4 array of velocity readings in beam coordinates. Rows represent
        samples, columns represent each beam.
    tMat : numpy array of floats
       Coordinate transformation matrix for beam to xyz, as a 4x4 matrix.

    Returns
    -------
    numpy array of floats
        nx4 array of velocity readings in xyz coordinates. Rows represent
        samples, columns represent each beam (X, Y, Z1, Z2).

    '''
    xyzVels = tMat @ beamVels.T
    return xyzVels.T
```

---

5) Put it all together!

In [ ]:
```
import numpy as np

# Define data files
dataFile = 'vectrinoData.txt'
transformFile = 'vectrinoTransform.txt'

# Import data from these files
time, beamVel, beamCorr = importData(dataFile)

# Clean velocity time series based on correlation values
beamFilt = cleanVelocity(time, beamVel, beamCorr, corrThresh=95)

# Import transformation matrix and apply coordinate transformation
tMat = np.loadtxt(transformFile, delimiter=',')
vel = coordinateTransform(beamVel, tMat)
velFilt = coordinateTransform(beamFilt, tMat)
```

6) Plot using `matplotlib`

Plotting package provides simple plotting interface (we'll do a deep dive later in the quarter)

In [ ]:
```
import matplotlib.pyplot as plt

def plotVelocity(time, vel, velFilt, coordInd):
    '''
    Plots a time series of the measured velocity (both filtered and unfiltered)
    in the given coorindate direction

    Parameters
    ----------
    time : numpy vector of floats
        Time stamps of each velocity reading, in seconds
    vel : numpy array of floats
        nx4 array of raw velocity readings in beam coordinates. Rows represent
        samples, columns represent each beam.
    velFilt : numpy array of floats
        nx4 array of filtered velocity readings in beam coordinates. Rows represent
        samples, columns represent each beam.
    coordInd : int
        Integer that represents which coordinate direction to plot:
```

```
            0 = X
            1 = Y
            2 = Z1
            3 = Z2

    Returns
    -------
    None.

    '''
    # Generate plot of velocity in given coordinate direction
    plt.figure()
    plt.plot(time, vel[:,coordInd])
    plt.plot(time, velFilt[:,coordInd])
    plt.grid()
    plt.xlabel('Time [s]')
    plt.ylabel('Velocity [m/s]')
    plt.legend(['Unfiltered', 'Filtered'])

    # Generate text for title based on coordinate index
    coords = ['X', 'Y', 'Z1', 'Z2']
    plt.title('Velocity in ' + coords[coordInd] + ' direction')
```

8) Compare the results with data cleaning to those without

Adjust correlation threshold parameter to change how much filtering is imposed on the velocity readings.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define data files
dataFile = 'vectrinoData.txt'
transformFile = 'vectrinoTransform.txt'

# Import data from these files
time, beamVel, beamCorr = importData(dataFile)

# Clean velocity time series based on correlation values
beamFilt = cleanVelocity(time, beamVel, beamCorr, corrThresh=95)

# Import transformation matrix and apply coordinate transformation
tMat = np.loadtxt(transformFile, delimiter=',')
vel = coordinateTransform(beamVel, tMat)
velFilt = coordinateTransform(beamFilt, tMat)

# Plot the velocity in a given direction
beamInd = 0
plotVelocity(time, vel, velFilt, beamInd)
```

What if we wanted to use this for several data files? Define a function that performs all of the processing (and, if you want, produces a plot), given a data file as an input. Then pass in each file using a loop.

Other packages use NumPy!

Many other packages build upon the scientific computing foundation provided by NumPy:

Matplotlib

- Popular plotting package
- MATLAB-like plotting functionality

To import and use Matplotlib, typically import `matplotlib.pyplot`: the interface that gives us useful scripting commands

SciPy

- "Scientific Python"
- Many useful numerical routines
  - Numerical integration (trapezoidal method, ODE solvers)
  - Root-finding
  - Optimization

Scikit-learn

- Popular machine learning package
- Built on NumPy and SciPy

Pandas

- Useful package for working with tabular data (i.e., spreadsheets)
- More on this next week

In [ ]: