DTW Tutorial

The goal of this week's tutorial is to build a flexible dynamic time warping algorithm that can take arbitrary distance functions as input and produce a distance and alignment between two sequences. The directions are a bit sparse as this is straightforward to do.

Preliminaries: [skip to Step 1 if you are Python expert]

First: `import numpy as np`

One thing to be aware of is that in Python you can pass a function as an argument. For example, let's assume we create function add that adds two numbers:

```
def add(x,y):
   return x+y
```

You can pass add to a function to an argument that needs to use it. For example:

```
def operate(operand1,operand2,function):
   return function(operand1,operand2)

>>> operate(2,3,add)
5
```

You can also define a default value for an argument (e.g. a default function):

```
def operate(operand1,operand2,function=add):
   return function(operand1,operand2)

>>> operate(3,4)
7
```

You can also have functions that return multiple values:

```
def addsub(x,y):
   return (x+y,x-y)

>>> operate(6,3,addsub)
(9, 3)
```

Step 1: Creating a distance function

Create a function absdiff that returns the absolute difference between two numbers.

Step 2: Creating a function that calls a function

The point of creating a distance function is that you can pass it as an argument to whatever other function you want. So, if you want to sum a distance function between two vectors, you can use the following (naïve) implementation:

```
def sumdist(x1,x2,distfunc):
    s=0
    for i in np.arange(0,x1.shape[-1]):
        s=s+distfunc(x1[i],x2[i])
    return s
```

You can call this with, for example

```
>>> sumdist(np.array([1,2,3]),np.array([4,5,6]),absdiff)
9
```

Try creating a squarediff function that returns the square of the difference of two numbers, and see what you get.

Note that a more pythonic version of the sumdist function would be to allow the function to apply to the vectors and then sum:

```
def sumdist(x1,x2,distfunc):
    return np.sum(distfunc(x1,x2))
```

Step 3: A basic DTW function

Let's start by coding up the basic DTW algorithm: write a basic function that will return the DTW score. Remember, for two vectors X (length m) and Y (length n) the algorithm is (in psuedocode)

---

for i ranging from 1 to m
  for j ranging from 1 to n
    D(i,j)=min(D(i-1,j),D(i-1,j-1),D(i,j-1))+dist(X(i),Y(j))
return D(m,n)

X and Y will at first be row vectors – each column will represent one point in time. Also, min([a,b,c]) will return the minimum of a, b, c.

```
>> dtw(np.array([1,2,4,2,3]),np.array([1,1,2,2,3]),absdiff)
2
```

Step 3a: Return the score matrix as well

You can adjust the algorithm to return a score matrix as well by just a tuple.

```
>>>
(s,D)=dtw(np.array([1,2,4,2,3]),np.array([1,1,2,2,3]),absdiff)
>>> s
2.0
>>> D
array([[ 0.,  0.,  1.,  2.,  4.],
       [ 1.,  1.,  0.,  0.,  1.],
       [ 4.,  4.,  2.,  2.,  1.],
       [ 5.,  5.,  2.,  2.,  2.],
       [ 7.,  7.,  3.,  3.,  2.]])
```

Step 4: Return a backtrace array

The easiest way that I found to do this is to return another 3-d array B which is m x n x 2. Set B[i,j,0] to -1 if the minimum comes from the previous column (0 otherwise), and B[i,j,1] to -1 if the minimum comes from the previous row (0 otherwise).

```
>>> B[:,:,0]
array([[ 0., -1., -1., -1., -1.],
    [ 0.,  0., -1., -1., -1.],
    [ 0.,  0.,  0.,  0., -1.],
    [ 0.,  0.,  0.,  0.,  0.],
    [ 0.,  0.,  0.,  0.,  0.]])
>>> B[:,:,1]
array([[ 0.,  0.,  0.,  0.,  0.],
    [-1., -1., -1.,  0.,  0.],
    [-1., -1., -1., -1., -1.],
    [-1., -1., -1., -1., -1.],
    [-1., -1., -1., -1., -1.]])
```

The neat thing is that you can use the quiver function to plot this backtrace array:

```
import matplotlib
# this works on my mac. May need to choose different backend
matplotlib.use('Qt5Agg')
import matplotlib.pyplot as plt
plt.quiver(B[:,:,0],B[:,:,1])
plt.show()
```

See if you can also compute the optimal backtrace.

Step 5: Speech alignment

Now, hook this up with the log spectrum calculation you did the last time (I can provide code if needed), and use the Euclidean distance between columns of your spectrogram as the distance function