Numerical Methods - Assignment 1

In [ ]:

1. Python and MATLAB

List at least 3 differences between Python and MATLAB.

Python is free and open source, meaning that you can manipulate other people's code to do what you want, and it's free so you do not need to spend money on licences like you need to with Matlab.

Python is not as fast as Matlab when working with computationally expensive numerical methods, which is something Matlab is used for, and why it was most likely a good option for the Numerical Methods ChBE course.

Matlab automatically comes with an IDE (integrated development environment) which is really helpful to those new to it to be able to debug and see what variables are valued at in certain points, for Python you would have to download the Spyder IDE that allows you to do this. Which is really cool because earlier last year I tried picking up Python and it would have been a lot better if I knew there was this IDE feature and I wouldn't just have to work in something that looks like a command prompt window! The fact that I couldn't find this I guess also goes to show the different versions of Python that are hard to manage, like referenced in the notes.

2. Plot Data

Read the data and create a plot.

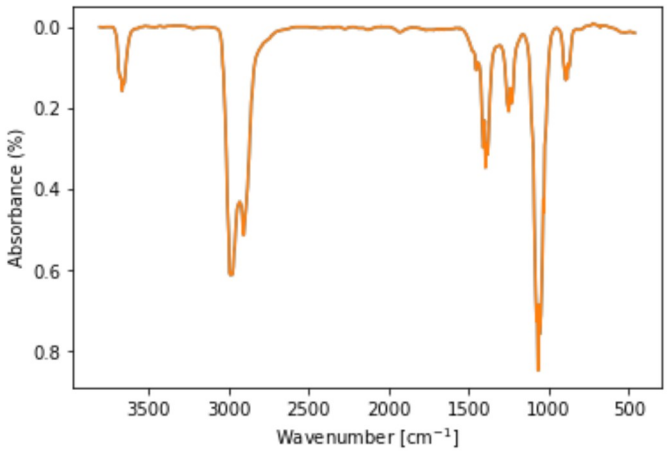
- Import `matplotlib` and `pandas` packages.
- Read in `data/ethanol_IR.csv` file and create a plot of IR spectra data.

In [1]:
```
import pylab as plt
import pandas as pd
%matplotlib inline
etoh_IR = pd.read_csv('data/ethanol_IR.csv')

wavenum = pd.DataFrame(etoh_IR, columns= ['wavenumber [cm^-1]'])
absorb = pd.DataFrame(etoh_IR, columns = ['absorbance'])

fig, ax = plt.subplots(); ax.plot(wavenum,absorb)
ax.invert_xaxis()
ax.invert_yaxis()
ax.set_xlabel('Wavenumber [cm$^{-1}$]')
ax.set_ylabel('Absorbance (%)')
plt.plot(wavenum,absorb)
```

Out[1]:
```
[<matplotlib.lines.Line2D at 0x1baeaae7760>]
```



Briefly describe the most prominent peaks in the dataset.

The peaks around 3000 cm^-1 show Carbon-Hydrogen stretching, the one slightly above 3000 cm^-1 is indicative of sp2 hybridization and the part of the peak below 3000^-1 is indicative of sp3 hybridization. The strong peak near 1000 cm^-1 is indicative of Carbon-Oxygen stretching, so this structure could have an ether or be classified just as an ether.

3. Matrix-vector Multiplication

Write a funcion that uses `for` loops.

This function should multiply an arbitrary matrix and vector.

In [2]:
```
import numpy as np
def mulMatVec(matrix, vector):
    result = []
    for row in matrix:
        dotprod = np.dot(row,vector)
        result.append(dotprod)
    return result
```

You can use the matrix and vector given below.

In [3]:
```
A = np.array([[1, 2], [-4, 5]])
B = np.array([-2, 3])
print(mulMatVec(A,B))
```

```
[4, 23]
```

Or create an arbitrary set of matrix and vector using `numpy.random.rand` .

In [4]:
```
#This was just for practice, please grade the one above!

numrows = np.random.randint(1,high = 5)
numcols = np.random.randint(1,high = 5)
matrix = []
for i in range(numrows):
    row = np.random.randint(1,high = 7,size = numcols)
    matrix.append(row)
vector = np.random.randint(1, high = 8, size = numcols)
print(matrix)
print(vector)
print(mulMatVec(matrix, vector))
```

```
[array([6, 6]), array([2, 5]), array([1, 3]), array([1, 4])]
[1 7]
[48, 37, 22, 29]
```

Show that your function is correct using `numpy.isclose` .

In [5]:
```
print(np.isclose(mulMatVec(A,B),[4,23]))
```

```
[ True  True]
```

4. Vandermonde Matrix

Use `numpy.hstack` to construct a 4th-order Vandermonde matrix.

Range should be from -1 to 1 with a resolution of 25 (i.e. the number of rows should be 25).

In [6]:
```
import numpy as np
resolution = 25
xi = np.linspace(-1,1,resolution)
xi = xi.reshape(-1,1)
xvdm = np.hstack((xi**0,xi**1,xi**2,xi**3,xi**4))
print(xvdm)
```

```
[[ 1.00000000e+00 -1.00000000e+00  1.00000000e+00 -1.00000000e+00
   1.00000000e+00]
 [ 1.00000000e+00 -9.16666667e-01  8.40277778e-01 -7.70254630e-01
   7.06066744e-01]
 [ 1.00000000e+00 -8.33333333e-01  6.94444444e-01 -5.78703704e-01
   4.82253086e-01]
 [ 1.00000000e+00 -7.50000000e-01  5.62500000e-01 -4.21875000e-01
   3.16406250e-01]
 [ 1.00000000e+00 -6.66666667e-01  4.44444444e-01 -2.96296296e-01
   1.97530864e-01]
 [ 1.00000000e+00 -5.83333333e-01  3.40277778e-01 -1.98495370e-01
   1.15788966e-01]
 [ 1.00000000e+00 -5.00000000e-01  2.50000000e-01 -1.25000000e-01
   6.25000000e-02]
 [ 1.00000000e+00 -4.16666667e-01  1.73611111e-01 -7.23379630e-02
   3.01408179e-02]
 [ 1.00000000e+00 -3.33333333e-01  1.11111111e-01 -3.70370370e-02
   1.23456790e-02]
 [ 1.00000000e+00 -2.50000000e-01  6.25000000e-02 -1.56250000e-02
   3.90625000e-03]
 [ 1.00000000e+00 -1.66666667e-01  2.77777778e-02 -4.62962963e-03
   7.71604938e-04]
 [ 1.00000000e+00 -8.33333333e-02  6.94444444e-03 -5.78703704e-04
   4.82253086e-05]
 [ 1.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00]
 [ 1.00000000e+00  8.33333333e-02  6.94444444e-03  5.78703704e-04
   4.82253086e-05]
 [ 1.00000000e+00  1.66666667e-01  2.77777778e-02  4.62962963e-03
   7.71604938e-04]
 [ 1.00000000e+00  2.50000000e-01  6.25000000e-02  1.56250000e-02
   3.90625000e-03]
 [ 1.00000000e+00  3.33333333e-01  1.11111111e-01  3.70370370e-02
   1.23456790e-02]
 [ 1.00000000e+00  4.16666667e-01  1.73611111e-01  7.23379630e-02
   3.01408179e-02]
 [ 1.00000000e+00  5.00000000e-01  2.50000000e-01  1.25000000e-01
   6.25000000e-02]
 [ 1.00000000e+00  5.83333333e-01  3.40277778e-01  1.98495370e-01
   1.15788966e-01]
 [ 1.00000000e+00  6.66666667e-01  4.44444444e-01  2.96296296e-01
   1.97530864e-01]
 [ 1.00000000e+00  7.50000000e-01  5.62500000e-01  4.21875000e-01
   3.16406250e-01]
 [ 1.00000000e+00  8.33333333e-01  6.94444444e-01  5.78703704e-01
   4.82253086e-01]
 [ 1.00000000e+00  9.16666667e-01  8.40277778e-01  7.70254630e-01
   7.06066744e-01]
 [ 1.00000000e+00  1.00000000e+00  1.00000000e+00  1.00000000e+00
   1.00000000e+00]]
```

Create an orthonormal version of the Vandermonde matrix.

Orthonormal means:

- the L_2 norm of each column is 1.
- the inner product between any 2 columns is 0.

Print the orthonormalized Vandermonde matrix.

In [7]:
```
numcols = np.size(xvdm[0,:])
vec = list(range(0,5))

for i in vec:
    if i==0:
        xvdm[:,i] = (xvdm[:,i])/(np.linalg.norm(xvdm[:,i]))
    j = i-i
    while j != i:
        ortho = xvdm[:,i] - np.dot(xvdm[:,j],xvdm[:,i])*xvdm[:,j]
        xvdm[:,i] = ortho
        j = j+1
    xvdm[:,i] = xvdm[:,i]/(np.linalg.norm(xvdm[:,i],2))
print(xvdm)
```

```
[[ 2.00000000e-01 -3.32820118e-01  3.96566460e-01 -4.15922412e-01
   4.01324069e-01]
 [ 2.00000000e-01 -3.05085108e-01  2.97424845e-01 -2.07961206e-01
   6.68873448e-02]
 [ 2.00000000e-01 -2.77350098e-01  2.06904240e-01 -4.52089579e-02
  -1.36682835e-01]
 [ 2.00000000e-01 -2.49615088e-01  1.25004645e-01  7.64442378e-02
  -2.37146040e-01]
 [ 2.00000000e-01 -2.21880078e-01  5.17260600e-02  1.61108286e-01
  -2.59618073e-01]
 [ 2.00000000e-01 -1.94145069e-01 -1.29315150e-02  2.12893092e-01
  -2.26570966e-01]
 [ 2.00000000e-01 -1.66410059e-01 -6.89680800e-02  2.35908562e-01
  -1.57832983e-01]
 [ 2.00000000e-01 -1.38675049e-01 -1.16383635e-01  2.34264600e-01
  -7.05886207e-02]
 [ 2.00000000e-01 -1.10940039e-01 -1.55178180e-01  2.12071111e-01
   2.06213948e-02]
 [ 2.00000000e-01 -8.32050294e-02 -1.85351715e-01  1.73438002e-01
   1.03900105e-01]
 [ 2.00000000e-01 -5.54700196e-02 -2.06904240e-01  1.22475177e-01
   1.69994319e-01]
 [ 2.00000000e-01 -2.77350098e-02 -2.19835755e-01  6.32925410e-02
   2.12294616e-01]
 [ 2.00000000e-01  1.66277051e-17 -2.24146260e-01 -3.18152729e-17
   2.26835343e-01]
 [ 2.00000000e-01  2.77350098e-02 -2.19835755e-01 -6.32925410e-02
   2.12294616e-01]
 [ 2.00000000e-01  5.54700196e-02 -2.06904240e-01 -1.22475177e-01
   1.69994319e-01]
 [ 2.00000000e-01  8.32050294e-02 -1.85351715e-01 -1.73438002e-01
   1.03900105e-01]
 [ 2.00000000e-01  1.10940039e-01 -1.55178180e-01 -2.12071111e-01
   2.06213948e-02]
 [ 2.00000000e-01  1.38675049e-01 -1.16383635e-01 -2.34264600e-01
  -7.05886207e-02]
 [ 2.00000000e-01  1.66410059e-01 -6.89680800e-02 -2.35908562e-01
  -1.57832983e-01]
 [ 2.00000000e-01  1.94145069e-01 -1.29315150e-02 -2.12893092e-01
  -2.26570966e-01]
 [ 2.00000000e-01  2.21880078e-01  5.17260600e-02 -1.61108286e-01
  -2.59618073e-01]
 [ 2.00000000e-01  2.49615088e-01  1.25004645e-01 -7.64442378e-02
  -2.37146040e-01]
 [ 2.00000000e-01  2.77350098e-01  2.06904240e-01  4.52089579e-02
  -1.36682835e-01]
 [ 2.00000000e-01  3.05085108e-01  2.97424845e-01  2.07961206e-01
   6.68873448e-02]
 [ 2.00000000e-01  3.32820118e-01  3.96566460e-01  4.15922412e-01
   4.01324069e-01]]
```

Show that the L_2 of 5th column is 1.

In [8]:
```
print(np.isclose(np.linalg.norm(xvdm[:,4],2),1))
```

```
True
```

Show that the inner product between 1st column & 4th column is 0.

In [9]:
```
print(np.isclose(np.dot(xvdm[:,0],xvdm[:,3]),0))
```

```
True
```

Compute the rank of the orthonormalized Vandermonde matrix.

In [10]:
```
xvdmrank = np.linalg.matrix_rank(xvdm)
print(xvdmrank)
```

```
5
```

Show that the rank is equal to the number of columns.

In [11]:
```
numcols = np.size(xvdm[0,:])
print(numcols == xvdmrank)
```

```
True
```

Change the resolution to 30 and show that the rank is independent of the number of rows.

In [12]:
```
resolution = 30
xi = np.linspace(-1,1,resolution)
xi = xi.reshape(-1,1)
xvdm = np.hstack((xi**0,xi**1,xi**2,xi**3,xi**4))

numcols = np.size(xvdm[0,:])
vec = list(range(0,5))

for i in vec:
    if i==0:
        xvdm[:,i] = (xvdm[:,i])/(np.linalg.norm(xvdm[:,i]))
    j = i-i
    while j != i:
        ortho = xvdm[:,i] - np.dot(xvdm[:,j],xvdm[:,i])*xvdm[:,j]
        xvdm[:,i] = ortho
        j = j+1
    xvdm[:,i] = xvdm[:,i]/(np.linalg.norm(xvdm[:,i],2))
xvdmrank = np.linalg.matrix_rank(xvdm)
numcols = np.size(xvdm[0,:])
print(numcols == xvdmrank)
```

```
True
```