02-HOMEWORK-ChangeOfBasis

September 16, 2021

1 Homework 2

In this assignment, we will explore computational aspects of matrix inversion and changes of basis.

[2]:
```
# Import numpy to handle numerical linear algebra
import numpy as np
```

1.1 Matrix Inversion

We introduced the the notion of an inverse matrix in class. This can be computed by hand (see your textbook), but we can also use built-in `numpy` functions to do it for us.

Consider the matrix `A` defined below.

[3]:
```
A = np.array([[1,1,0,0],
              [2,0,2,0],
              [3,3,3,0],
              [0,0,4,4]])

print(A)
```

```
[[1 1 0 0]
 [2 0 2 0]
 [3 3 3 0]
 [0 0 4 4]]
```

Problem 1 Figure out how to compute the inverse of `A`. Call the result `A_inv`. Verify that your result has the correct inverse property; i.e., show that `A` times `A_inv` returns the identity matrix. Print out your answers.

General Remark: To reiterate a comment from the last coding HW: when doing coding exercies, feel free to use any resources you like when completing these assignments (e.g., Google `numpy` documentation).

Remark About Accuracy: Due to numerical errors, you will frequently get a result which you *expect* to be zero, but which the computer tells you is some extremely small number (e.g., `1e-17`). Don't worry about these numerical errors. If they bother you too much, feel free to figure out how to round your answers when printing them.

[4]:
```
## Your Code for Problem 1 Goes Here
A_inv = np.linalg.inv(A).round(9)
```

```
print(np.dot(A, A_inv))
```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

Next, consider the matrix `B` defined below.

[5]:
```
B = np.array([[1,1,0,0],
              [2,0,-2,0],
              [2,0,0,2],
              [-1,-1,-2,-2]])
```

Problem 2 Try computing the inverse of `B` using the same method from the last problem.

[6]:
```
## Your Code for Problem 2 Goes Here
B_inv = np.linalg.inv(B)
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
<ipython-input-6-04d8a93981dc> in <module>
      1 ## Your Code for Problem 2 Goes Here
----> 2 B_inv = np.linalg.inv(B)

<__array_function__ internals> in inv(*args, **kwargs)

F:\ProgramData\Anaconda3\lib\site-packages\numpy\linalg\linalg.py in inv(a)
    543     signature = 'D->D' if isComplexType(t) else 'd->d'
    544     extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
--> 545     ainv = _umath_linalg.inv(a, signature=signature, extobj=extobj)
    546     return wrap(ainv.astype(result_t, copy=False))
    547

F:\ProgramData\Anaconda3\lib\site-packages\numpy\linalg\linalg.py in␣
 ↪_raise_linalgerror_singular(err, flag)
     86
     87 def _raise_linalgerror_singular(err, flag):
---> 88     raise LinAlgError("Singular matrix")
     89
     90 def _raise_linalgerror_nonposdef(err, flag):

LinAlgError: Singular matrix
```

You will get an error!

Explain the mathematical reason that `B` is not invertible in the box below (convert it to a markdown cell and type your answer).

Your Explanation for Problem 2 Goes Here There are only 3 pivots in a 4x4 matrix, where the 4th row/column can be made from a linear combination of the other 3.

1.2 Transformation Matrices and Change of Basis

Let $V = \mathbb{R}^3$ and let $W = \mathbb{R}^2$. Choose ordered bases

$$B = \left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right)$$

for V, and

$$C = \left(\begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \end{bmatrix} \right)$$

for W.

Let $\Phi : V \to W$ be the linear transformation defined on the basis vectors of B by:

$$\Phi \left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\Phi \left(\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} -2 \\ -1 \end{bmatrix}$$

$$\Phi \left(\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right) = \begin{bmatrix} 1 \\ 5 \end{bmatrix}$$

Problem 3 Create the transformation matrix for Φ with respect to the bases B and C. Call the matrix `A_Phi` in your code; we will denote it A_Φ in the text below. Print your answer.

[49]:
```
## Your Code for Problem 3 Goes Here
B = np.array([[1,1,1],[0,1,1],[0,0,1]])
C = np.array([[1,-1],[1,1]])
A_Phi = np.array([np.linalg.inv(C)@[1,0],np.linalg.inv(C)@[-2,-1],np.linalg.
 ↪inv(C)@[1,5]]).transpose()
print(A_Phi)
```

```
[[ 0.5 -1.5  3. ]
 [-0.5  0.5  2. ]]
```

Now suppose that we have another basis $\overline{B}$ for V and another basis $\overline{C}$ for W. Theorem 2.20 from the text book gives a formula for computing the tranformation matrix for Φ with respect to these new bases—call this matrix $\overline{A}_\Phi$. The formula is

$$\overline{A}_\Phi = T^{-1} A_\Phi S,$$

where S is the transformation matrix for the identity map id_V with respect to the bases $\overline{B}$ and B and T is the transformation for the identity map id_W with respect to the bases $\overline{C}$ and C. T^{-1} is the matrix inverse of T.

Note: This formula holds in general, not just for the specific example we are looking at!

Problem 4 Let $\overline{B}$ be the standard basis for $\mathbb{R}^3$ and $\overline{C}$ the standard basis for $\mathbb{R}^2$. Construct the matrices S and T from the formula for our specific example; call them `S` and `T` in your code. Use these matrices to construct $\overline{A}_\Phi$ using the formula above; call your answer `A_Phi_bar`. Print your answers `S`, `T` and `A_Phi_bar`.

[52]:
```
## Your Code for Problem 4 Goes Here
B_bar = np.array([[1,0,0],[0,1,0],[0,0,1]])
C_bar = np.array([[1,0],[0,1]])
S = np.dot(B_bar,np.linalg.inv(B))
T = np.dot(C_bar, np.linalg.inv(C))
A_Phi_bar = np.linalg.inv(T)@A_Phi@S
print(S, '\n', T, '\n', A_Phi_bar)
```

```
[[ 1. -1.  0.]
 [ 0.  1. -1.]
 [ 0.  0.  1.]]
 [[ 0.5  0.5]
 [-0.5  0.5]]
 [[ 1. -3.  3.]
 [ 0. -1.  6.]]
```

1.3 Principal Component Analysis

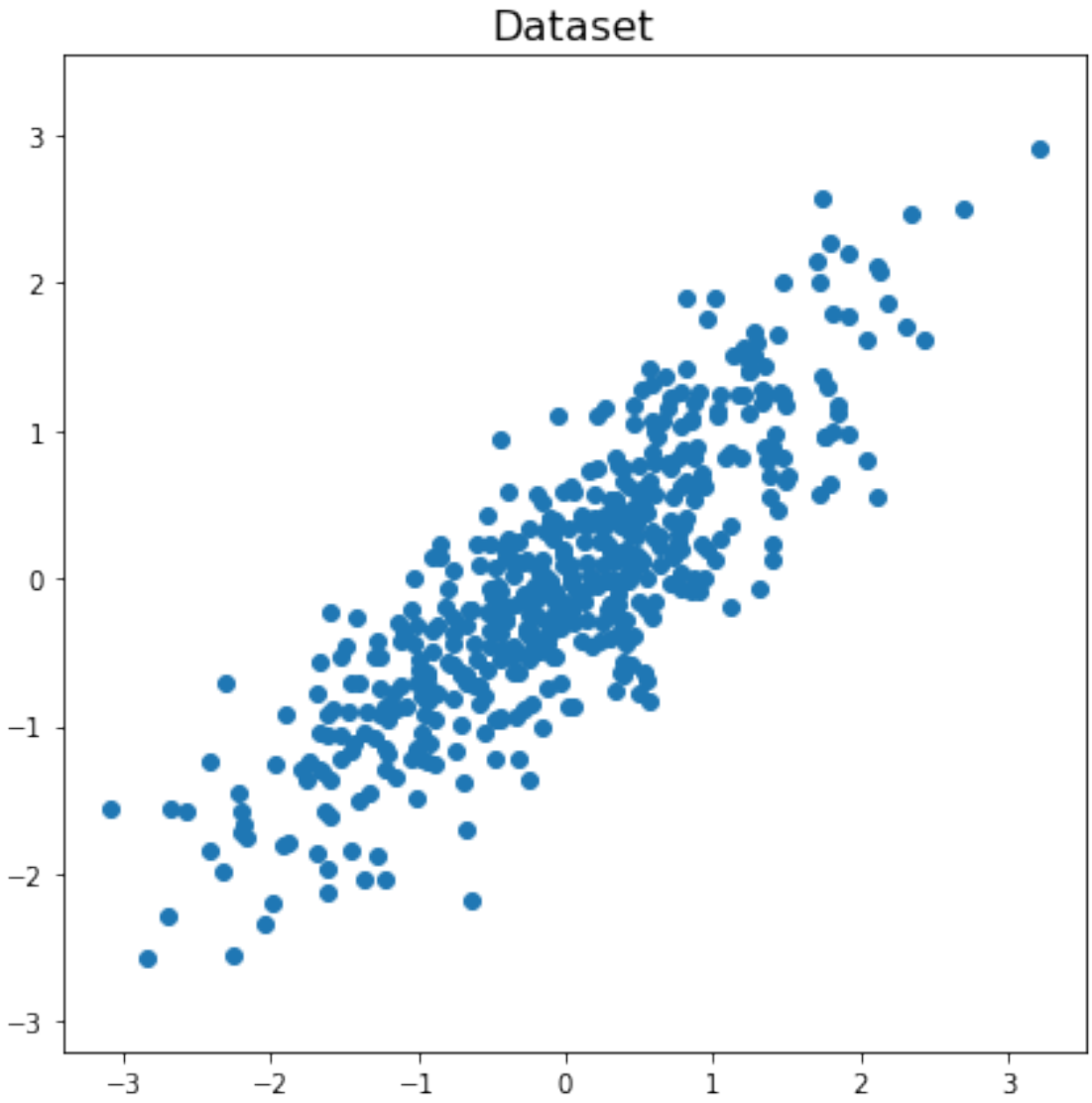
In this section, we will give a brief introduction to a tool which is ubiquitous in data science: principal component analysis. To start, we'll creat a toy data set.

[19]:
```
# import matplotlib for plotting
import matplotlib.pyplot as plt
```

[28]:
```
N = 500 # Number of points to sample
m = 0.8 # Slope

xs = np.random.normal(0,1,N)
ys = m*xs + np.random.normal(0,0.5,N)

plt.figure(figsize = (7,7))
plt.scatter(xs,ys)
plt.title('Dataset',fontsize = 16)
plt.axis('equal')
plt.show()
```



Now imagine that each dot in this 'point cloud' is a data point. In practice, data would be high dimensional and hard to visualize, so it is frequently useful to find a 'good basis' which captures the variability of the data. Intuitively, a good basis should line up with the line along which this point cloud appears to be clustered. Such a good basis is frequently used for 'dimension reduction': one can project the data onto a small number of directions which are able to capture the general shape of the data.

One process for finding a good basis is called Principal Component Analysis. We will cover the theory behind this in detail in the next few weeks of the course. For now, we will use a 'black box' algorithm to find the basis.

[29]:
```
# Import the PCA module from the sklearn package
from sklearn.decomposition import PCA
```

```
X = np.column_stack((xs,ys))
pca = PCA(n_components=2)
pca.fit(X)
eVec = pca.components_ # New basis vectors
eVals = pca.singular_values_
```

We plot the point cloud below, with our new PCA basis drawn over the top. The basis vectors have been scaled according to the variability that they capture (using the `eVals` data from the PCA computation).

Observe that the new basis aligns very well with the point cloud.

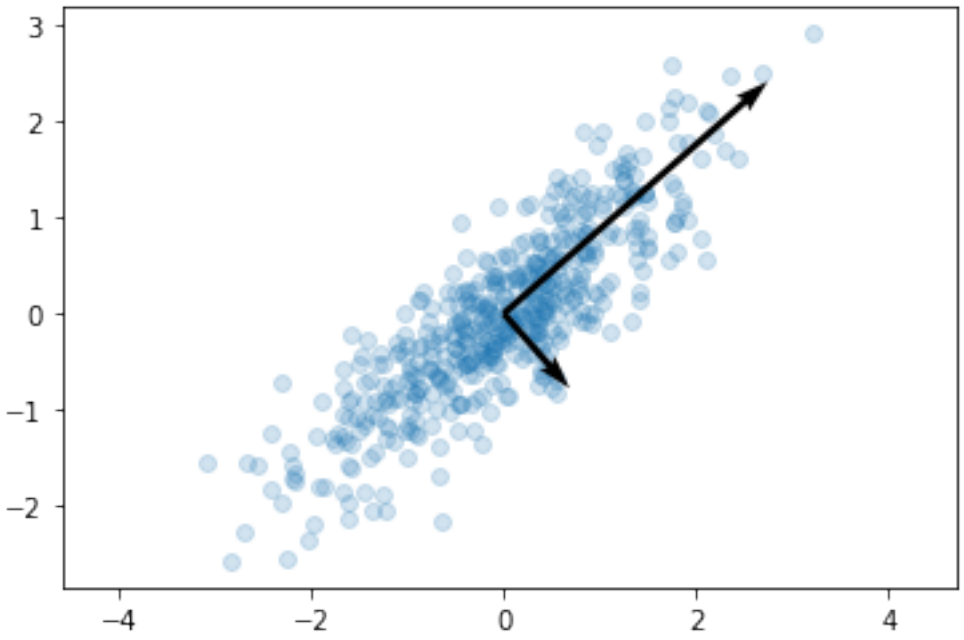
[30]:
```
eVals
```

[30]:
```
array([29.48688603,  8.33969818])
```

[31]:
```
plt.scatter(X[:, 0], X[:, 1], alpha=0.2)

# Syntax for quiver:
# plt.quiver(xVal for basepoint, yVal for basepoint, xVal for vector, yVal for␣
 ↪vector, scale = )
plt.quiver(0, 0, eVals[0]*eVec[0,0], eVals[0]*eVec[0,1], scale=75)
plt.quiver(0, 0, eVals[1]*eVec[1,0], eVals[1]*eVec[1,1], scale=75)

plt.axis('equal');
```



Problem 5 Play around with the various parameters in the code for this section and try to figure out what they do. **You don't have to turn in an answer for this problem.**