Lecture02

March 2, 2023

0.1 Data Structures for Algebra:

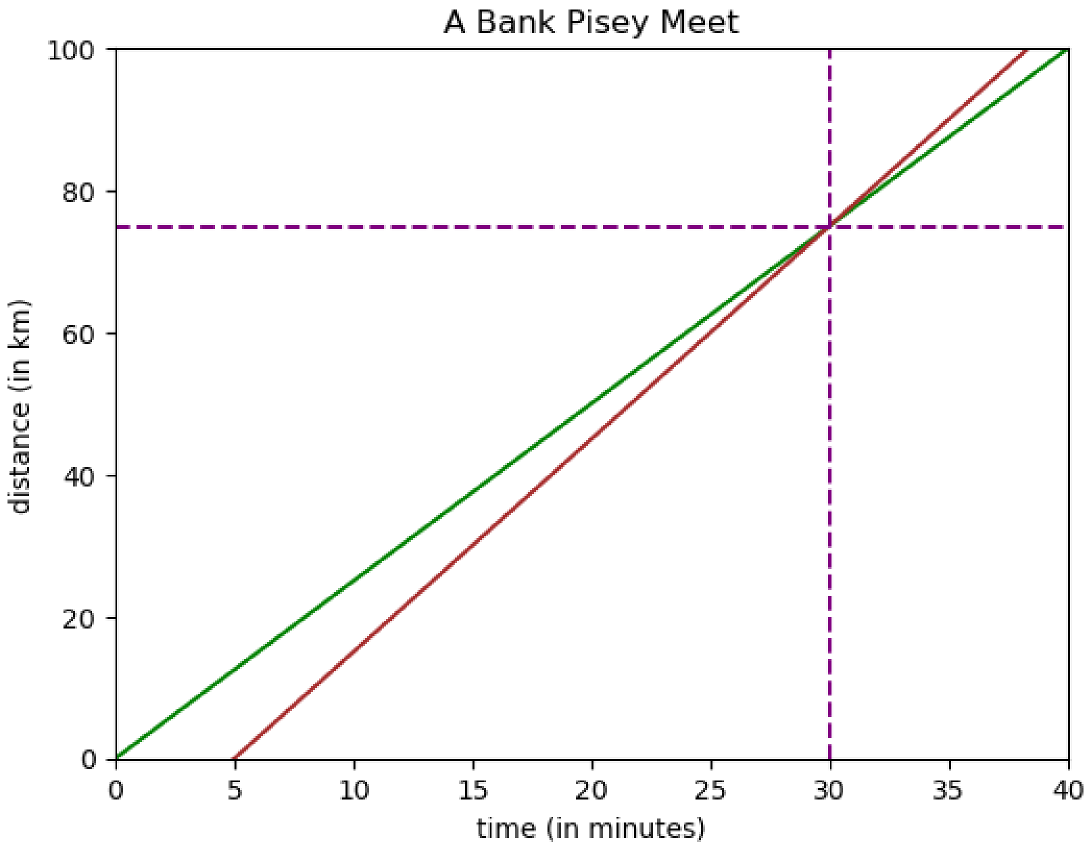
[4]:
```
import numpy as np
import matplotlib.pyplot as plt



t= np.linspace(0, 40, 1000) # start, finish, n points
# distance traveled by Pisey: d = 2.5t
d_p= 2.5*t
# distance traveled by Dara
d_r= 3*(t-5)

fig, ax= plt.subplots()
plt.title('A Bank Pisey Meet')
plt.xlabel('time (in minutes)')
plt.ylabel('distance (in km)')
ax.set_xlim([0,40])
ax.set_ylim([0,100])
ax.plot(t, d_p, c= 'green')
ax.plot(t,d_r, c= 'brown')
plt.axvline(x=30, color= 'purple', linestyle = '--')
plt.axhline(y= 75, color = 'purple', linestyle = '--')
```

[4]:
```
<matplotlib.lines.Line2D at 0x7fed07778c70>
```



```
[15]: # Scalars (Rank 0 Tensors) in Base Python
      x = 25
      x
```

```
[15]: 25
```

```
[16]: type(x) # if we'd like more specificity (e.g., int16, uint8), we need Numpy or ␣
        ↪another numeric library
```

```
[16]: int
```

```
 [7]: y = 3
```

```
[19]: py_sum = x + y
      py_sum
```

```
[19]: 28
```

```
[20]: type(py_sum)
```

```
[20]: int
```

```
[21]: x_float = 25.0
      float_sum = x_float + y
      float_sum
```

```
[21]: 28.0
```

```
[22]: type (float_sum)
```

```
[22]: float
```

```
[24]: # Vectors (Rank 1 Tensors) in Numpy

      x = np.array([25, 2, 5]) # type argument is optional e.g. : dtype = np.float16
      x
```

```
[24]: array([25,  2,  5])
```

```
[25]: len(x)
```

```
[25]: 3
```

```
[26]: x.shape
```

```
[26]: (3,)
```

```
[27]: type(x[0])
```

```
[27]: numpy.int64
```

```
[28]: # Vector Transposition
      # Transposing a regular 1_D array has no effect...
      x_t = x.T
      x_t
```

```
[28]: array([25,  2,  5])
```

```
[29]: x_t.shape
```

```
[29]: (3,)
```

```
[30]: # ... but it dose we nested "matrix-style" brackets:
      y = np.array([[25, 2, 5]])
      y
```

```
[30]: array([[25,  2,  5]])
```

[31]:
```
y.shape
```

[31]:
```
(1, 3)
```

[32]:
```
# ...But can transpose a matrix with a dimension of length 1, which is mathematically equivalent:
y_t = y.T
y_t
```

[32]:
```
array([[25],
       [ 2],
       [ 5]])
```

[33]:
```
y_t.shape # this is a colum vector as it has 3 rows and 1 column
```

[33]:
```
(3, 1)
```

[34]:
```
# Column vector can be transposed back to original row vector
y_t.T
```

[34]:
```
array([[25,  2,  5]])
```

[37]:
```
# Zero Vectors
# Have no effect if added to another vector
z = np.zeros(3)
z
```

[37]:
```
array([0., 0., 0.])
```

[39]:
```
# Vector in PyTorch
import torch
x_pt = torch.tensor([25, 2, 5])
x_pt
```

[39]:
```
tensor([25,  2,  5])
```

[40]:
```
# L^2 Norm
x
```

[40]:
```
array([25,  2,  5])
```

[41]:
```
(25**2 + 2**2 + 5**2)**(1/2)
```

[41]:
```
25.573423705088842
```

[42]:
```
np.linalg.norm(x)
```

```
[42]: 25.573423705088842
```

```
[43]: # L^1 Norm
      x
```

```
[43]: array([25,  2,  5])
```

```
[44]: np.abs(25) + np.abs(2) + np.abs(5)
```

```
[44]: 32
```

```
[45]: # Squared L^2 Norm
      x
```

```
[45]: array([25,  2,  5])
```

```
[46]: (25**2 + 2**2 + 5**2)
```

```
[46]: 654
```

```
[47]: # Max Norm
      x
```

```
[47]: array([25,  2,  5])
```

```
[48]: np.max([np.abs(25), np.abs(2), np.abs(5)])
```

```
[48]: 25
```

```
[50]: # Orthogonal Vector
      i = np.array([1,0])
      i
```

```
[50]: array([1, 0])
```

```
[51]: j = np.array([0,1])
```

```
[52]: np.dot(i,j)
```

```
[52]: 0
```

```
[53]: # Matrices (Rank 2 Tensor) in Numpy
      # Use array() with nested brackets:
      X = np.array([[25,2], [5, 26], [3,7]])
      X
```

```
[53]: array([[25,  2],
             [ 5, 26],
             [ 3,  7]])
```

```
[54]: X.shape
```

```
[54]: (3, 2)
```

```
[55]: X.size
```

```
[55]: 6
```

```
[56]: # Select left column of matrix X (zero-indexed)
      X[:,0]
```

```
[56]: array([25,  5,  3])
```

```
[57]: # Select middle row of matrix X:
      X[1,:]
```

```
[57]: array([ 5, 26])
```

```
[58]: # Another slicing-by-index example:
      X[0:2, 0:2]
```

```
[58]: array([[25,  2],
             [ 5, 26]])
```

```
[60]: # Matrices in PyTorch

      X_pt = torch.tensor([[25,2], [5,26], [3,7]])
      X_pt
```

```
[60]: tensor([[25,  2],
              [ 5, 26],
              [ 3,  7]])
```

```
[61]: X_pt.shape
```

```
[61]: torch.Size([3, 2])
```

```
[62]: X_pt[1, :]
```

```
[62]: tensor([ 5, 26])
```

1 # Common Tensor Tensor Opperation

[63]:
```
# Tensor Transposition
X
```

[63]:
```
array([[25,  2],
       [ 5, 26],
       [ 3,  7]])
```

[64]:
```
X.T
```

[64]:
```
array([[25,  5,  3],
       [ 2, 26,  7]])
```

[65]:
```
X_pt.T
```

[65]:
```
tensor([[25,  5,  3],
        [ 2, 26,  7]])
```

[67]:
```
# Basic Arithmetical Properties
X*2
```

[67]:
```
array([[50,  4],
       [10, 52],
       [ 6, 14]])
```

[68]:
```
X+2
```

[68]:
```
array([[27,  4],
       [ 7, 28],
       [ 5,  9]])
```

[69]:
```
X*2 +2  # Python operations are overloaded; could alternatively use torch
```

[69]:
```
array([[52,  6],
       [12, 54],
       [ 8, 16]])
```

[70]:
```
torch.add(torch.mul(X_pt,2), 2)
```

[70]:
```
tensor([[52,  6],
        [12, 54],
        [ 8, 16]])
```

[71]:
```
# Hadamard product

X
```

```
[71]: array([[25,  2],
             [ 5, 26],
             [ 3,  7]])
```

```
[72]: A = X +2
      A
```

```
[72]: array([[27,  4],
             [ 7, 28],
             [ 5,  9]])
```

```
[73]: A + X
```

```
[73]: array([[52,  6],
             [12, 54],
             [ 8, 16]])
```

```
[74]: A*X
```

```
[74]: array([[675,   8],
             [ 35, 728],
             [ 15,  63]])
```

```
[75]: A_pt = X_pt + 2
```

```
[76]: A_pt + X_pt
```

```
[76]: tensor([[52,  6],
              [12, 54],
              [ 8, 16]])
```

```
[77]: A_pt*X_pt
```

```
[77]: tensor([[675,   8],
              [ 35, 728],
              [ 15,  63]])
```

```
[78]: x
```

```
[78]: array([25,  2,  5])
```

```
[79]: y = np.array([0, 1, 2])
```

```
[80]: 25*0 + 2*1 + 5*2
```

```
[80]: 12
```

[81]:
```
np.dot(x, y)
```

[81]:
```
12
```

[82]:
```
x_pt
```

[82]:
```
tensor([25,  2,  5])
```

[83]:
```
y_pt = torch.tensor([0,1, 2])
y_pt
```

[83]:
```
tensor([0, 1, 2])
```

[84]:
```
np.dot(x_pt, y_pt)
```

[84]:
```
12
```

[87]:
```
torch.dot(torch.tensor([25, 2, 5.]), torch.tensor([0, 1, 2.]))
```

[87]:
```
tensor(12.)
```

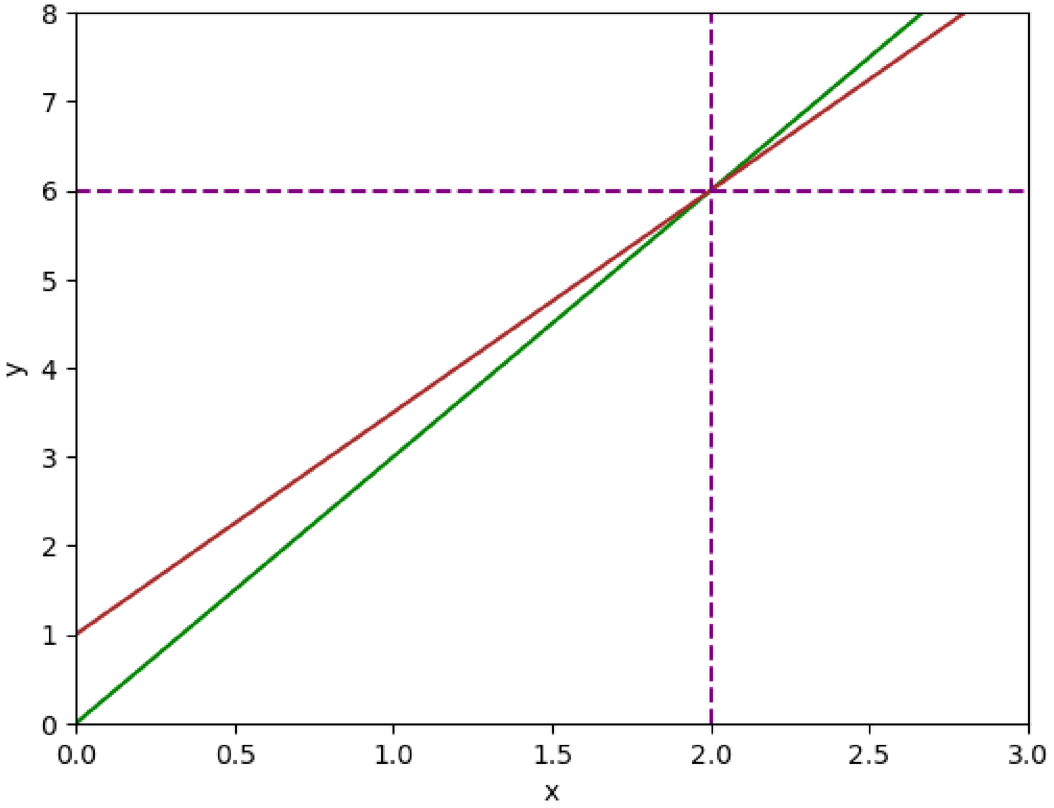
[99]:
```
# Solving Linear Systems
x = np.linspace(-10, 10, 1000) # start, finish, n points
```

[98]:
```
y1 = 3*x
```

[95]:
```
y2 = 1 + (5*x)/2
```

[97]:
```
fig, ax = plt.subplots()
plt.xlabel('x')
plt.ylabel('y')
ax.set_xlim([0,3])
ax.set_ylim([0,8])
ax.plot(x, y1, c = 'green')
ax.plot(x, y2, c = 'brown')
plt.axvline( x = 2, color = 'purple', linestyle = '--')
plt.axhline( y = 6, color = 'purple', linestyle = '--')
```

[97]:
```
<matplotlib.lines.Line2D at 0x7fed0ae55760>
```



[102]:
```
 # 2x - 3y = 15, 4x +10y = 14
y1 = -5 + (2*x)/3
```

[103]:
```
y2 = (7-2*x)/5
```

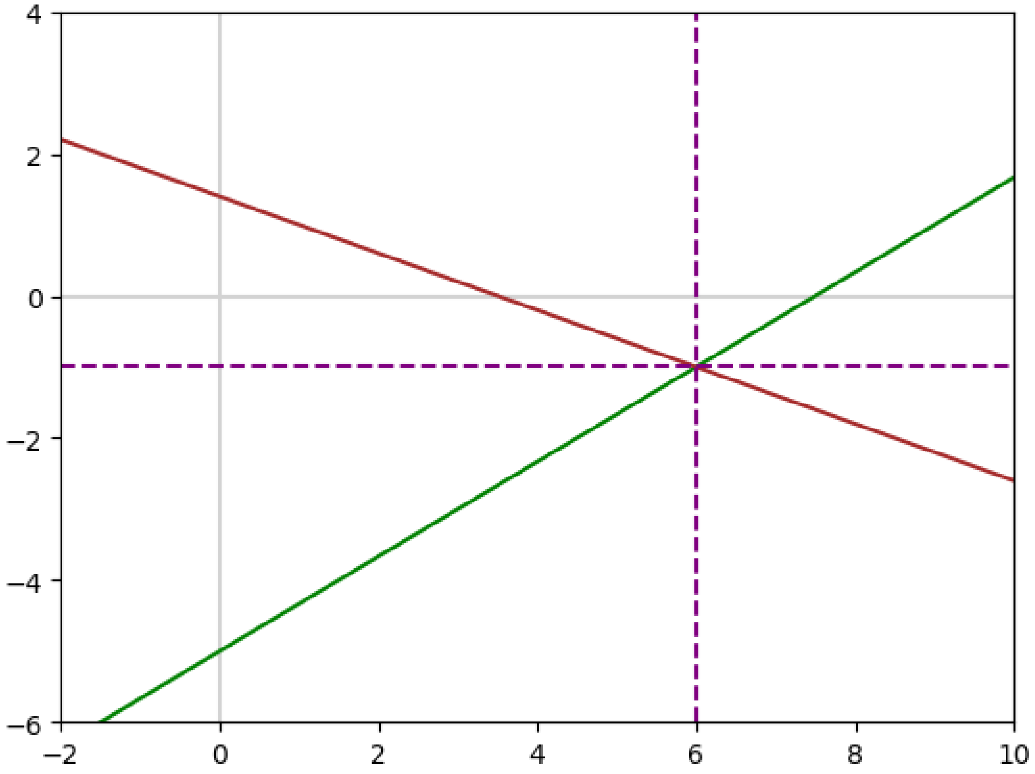
[104]:
```
fig, ax = plt.subplots()
plt.xlabel = ('x')
plt.ylabel = ('y')

# Add x and y axex:
plt.axvline(x= 0, color = 'lightgray')
plt.axhline(y = 0, color = 'lightgray')



ax.set_xlim([-2, 10])
ax.set_ylim([-6, 4])
ax.plot(x, y1, c = 'green')
ax.plot(x, y2, c = 'brown')
plt.axvline(x = 6, color = 'purple', linestyle= '--')
plt.axhline( y= -1, color = 'purple', linestyle = '--')
```

[104]:
```
<matplotlib.lines.Line2D at 0x7fed0afcad90>
```



[ ]: