Vectors

March 8, 2023

1 Vectors in numpy

1.1 Pro Tip

When you have a collection of items a computer will sometimes give you two ways to deal with it: piece by piece or as a collection. Always prefer the "as a collection" method if you have the choice.

2 Warmup

Suppose we want to use Python lists to represent vectors. Suppose we want to have a function that takes two Python vectors and adds them together. Write this function.

[1]:
```
def addV(a,b):
    out = []
    for i in range(min(len(a),len(b))):
        out.append(a[i] + b[i])
    return out
addV([1,2,3],[9,3,6,3])
```

[1]:
```
[10, 5, 9]
```

[3]:
```
def addV(a,b):
    out = []
    for ax,bx in zip(a,b):
        out.append(ax + bx)
    return out
addV([1,2,3],[9,3,6,3])
```

[3]:
```
[10, 5, 9]
```

2.1 Ug, another data type? WHY???

Programming language people love data types. Scientitst who use programming languages do to. Why?

- A well chosen data type makes it easier to ask the questions we want to ask about our data.
- A well chosen data type makes it easier to write programs to work with our data.

2.2 Okay, fine. So tell me about vectors....

A vector is a `numpy` data type that holds a collection of values, kind of like a Python list. Here's what's different: - An individal =vector= can only hold one kind of data. - This greatly reduces our flexibility, but gives us a major advantage in return: speed. - Why do you think this is the case?

```
[4]: import numpy as np
```

```
[5]: a = np.array([1,2,4,8,12])
```

```
[6]: a
```

```
[6]: array([ 1,  2,  4,  8, 12])
```

```
[7]: type(a)
```

```
[7]: numpy.ndarray
```

```
[8]: a.dtype
```

```
[8]: dtype('int64')
```

```
[9]: a[1]
```

```
[9]: 2
```

```
[10]: a + a
```

```
[10]: array([ 2,  4,  8, 16, 24])
```

```
[11]: a * a
```

```
[11]: array([  1,   4,  16,  64, 144])
```

```
[12]: a ** a
```

```
[12]: array([            1,             4,           256,      16777216,
       8916100448256])
```

```
[13]: a.__dir__()
```

```
[13]: ['__new__',
        '__repr__',
        '__str__',
        '__lt__',
        '__le__',
        '__eq__',
'__ne__',
'__gt__',
'__ge__',
'__iter__',
'__add__',
'__radd__',
'__sub__',
'__rsub__',
'__mul__',
'__rmul__',
'__mod__',
'__rmod__',
'__divmod__',
'__rdivmod__',
'__pow__',
'__rpow__',
'__neg__',
'__pos__',
'__abs__',
'__bool__',
'__invert__',
'__lshift__',
'__rlshift__',
'__rshift__',
'__rrshift__',
'__and__',
'__rand__',
'__xor__',
'__rxor__',
'__or__',
'__ror__',
'__int__',
'__float__',
'__iadd__',
'__isub__',
'__imul__',
'__imod__',
'__ipow__',
'__ilshift__',
'__irshift__',
'__iand__',
'__ixor__',
'__ior__',
'__floordiv__',
'__rfloordiv__',
'__truediv__',
'__rtruediv__',
'__ifloordiv__',
'__itruediv__',
'__index__',
'__matmul__',
'__rmatmul__',
'__imatmul__',
'__len__',
'__getitem__',
'__setitem__',
'__delitem__',
'__contains__',
'__array__',
'__array_prepare__',
'__array_finalize__',
'__array_wrap__',
'__array_ufunc__',
'__array_function__',
'__sizeof__',
'__copy__',
'__deepcopy__',
'__reduce__',
'__reduce_ex__',
'__setstate__',
'dumps',
'dump',
'__complex__',
'__format__',
'__class_getitem__',
'all',
'any',
'argmax',
'argmin',
'argpartition',
'argsort',
'astype',
'byteswap',
'choose',
'clip',
'compress',
'conj',
'conjugate',
'copy',
'cumprod',
'cumsum',
'diagonal',
'dot',
'fill',
'flatten',
'getfield',
'item',
'itemset',
'max',
'mean',
'min',
'newbyteorder',
'nonzero',
'partition',
'prod',
'ptp',
'put',
'ravel',
'repeat',
'reshape',
'resize',
'round',
'searchsorted',
'setfield',
'setflags',
'sort',
'squeeze',
'std',
'sum',
'swapaxes',
'take',
'tobytes',
'tofile',
'tolist',
'tostring',
'trace',
'transpose',
'var',
'view',
'__dlpack__',
'__dlpack_device__',
'ndim',
'flags',
'shape',
'strides',
'data',
'itemsize',
'size',
'nbytes',
'base',
'dtype',
'real',
'imag',
'flat',
'ctypes',
'T',
'__array_interface__',
'__array_struct__',
'__array_priority__',
'__doc__',
'__hash__',
'__getattribute__',
'__setattr__',
'__delattr__',
'__init__',
'__subclasshook__',
'__init_subclass__',
'__dir__',
'__class__']
```

[14]:
```
a.max()
```

[14]:
```
12
```

[17]:
```
a.dot(a)
```

[17]:
```
229
```

3 Linear Spaces

We can use the `linspace` function to create ranges. - It takes 3 arguments, `begin`, `end`, `count`. - Weirdly, the `end` is included; `range` and friends do not. - Why do you think they did that?

[24]:
```
list(range(0,10))
```

[24]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

[27]:
```
np.linspace(0,10.6,10)
```

[27]:
```
array([ 0.        ,  1.17777778,  2.35555556,  3.53333333,  4.71111111,
        5.88888889,  7.06666667,  8.24444444,  9.42222222, 10.6       ])
```

[ ]:
```

```

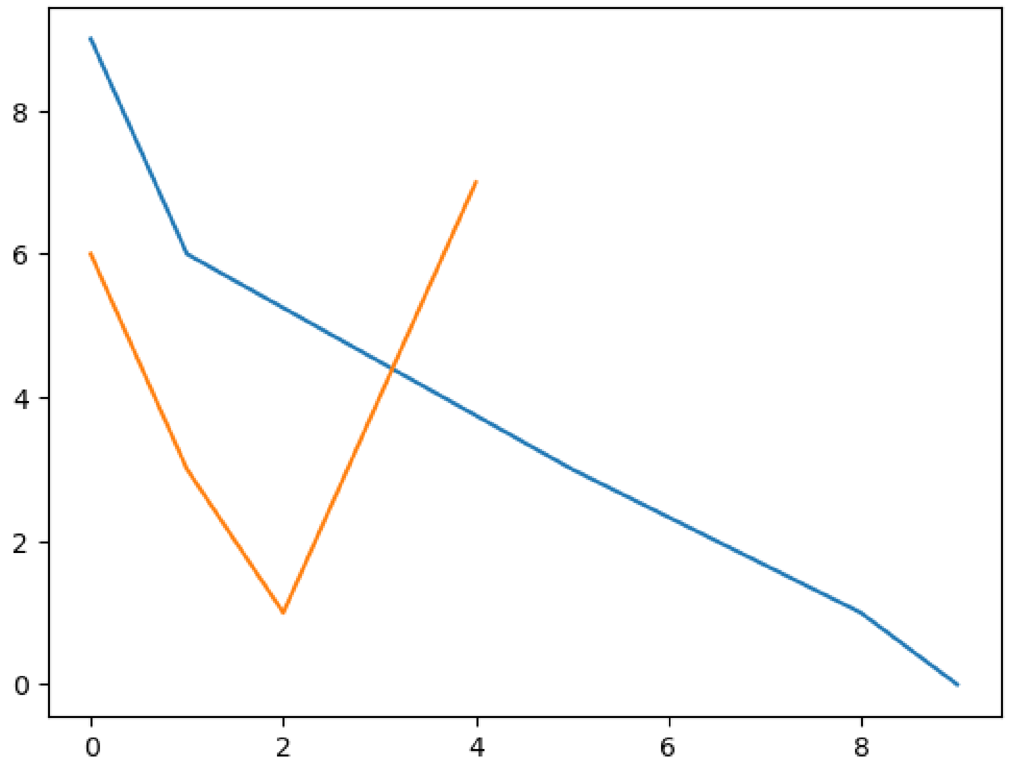
4 Plotting

Sometimes you want to plot things to the screen.

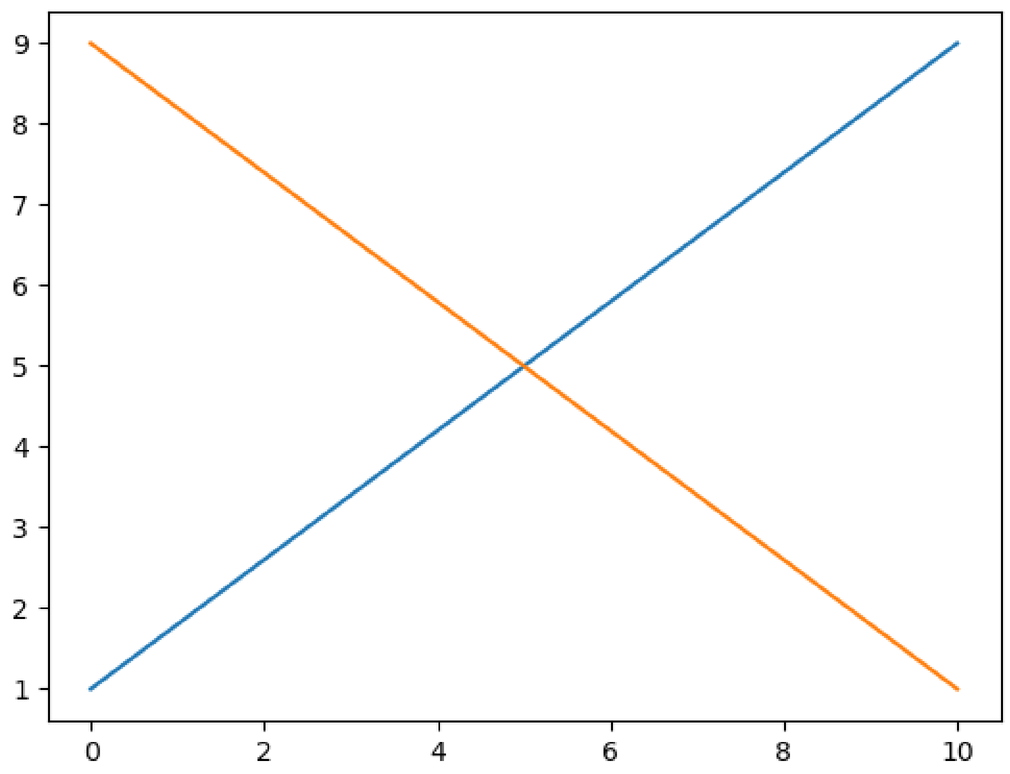
[28]:
```
import matplotlib.pyplot as plt
```

[29]:
```
a = np.array([0,1,5,8,9])
b = np.array([9,6,3,1,0])
c = np.array([6,3,1,4,7])
plt.plot(a,b,c)
```

[29]:
```
[<matplotlib.lines.Line2D at 0x7fb978417100>,
 <matplotlib.lines.Line2D at 0x7fb9784480a0>]
```



[33]:
```
plt.plot(np.array([0,10]),np.array([1,9]))
plt.plot(np.array([0,10]),np.array([9,1]))
plt.show()
```



```
[ ]:
```

5 Problem for you!

Write a program called `rosette` that takes an integer argument n. This will create n equispaced points on a unit circle and connect each point to every other point (a "complete graph"). Use `linspace` and vectors to do this.

```
[34]: np.pi
```

```
[34]: 3.141592653589793
```

```
[47]: pts = np.linspace(0,np.pi * 2, 8)
```

```
[48]: pts
```

```
[48]: array([0.        , 0.8975979 , 1.7951958 , 2.6927937 , 3.5903916 ,
              4.48798951, 5.38558741, 6.28318531])
```

```
[49]: ys = np.sin(pts)
      ys
```

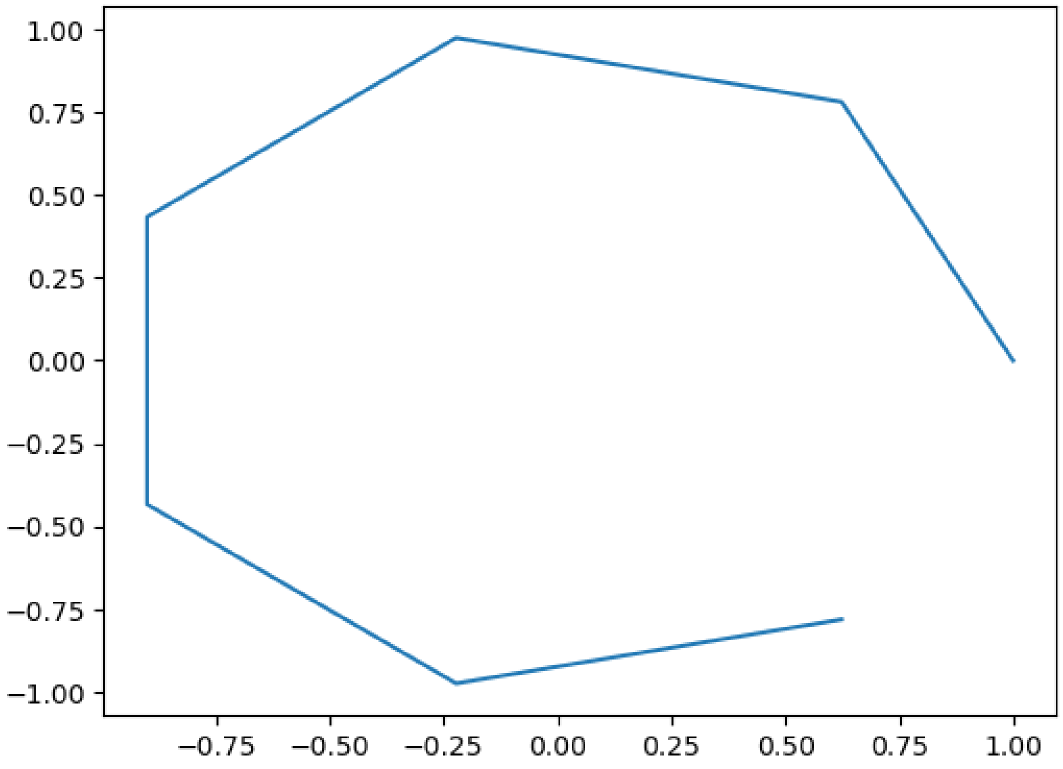
```
[49]: array([ 0.00000000e+00,  7.81831482e-01,  9.74927912e-01,  4.33883739e-01,
              -4.33883739e-01, -9.74927912e-01, -7.81831482e-01, -2.44929360e-16])
```

```
[50]: xs = np.cos(pts)
      xs
```

```
[50]: array([ 1.        ,  0.6234898 , -0.22252093, -0.90096887, -0.90096887,
              -0.22252093,  0.6234898 ,  1.        ])
```

```
[51]: plt.plot(xs[:-1],ys[:-1])
```

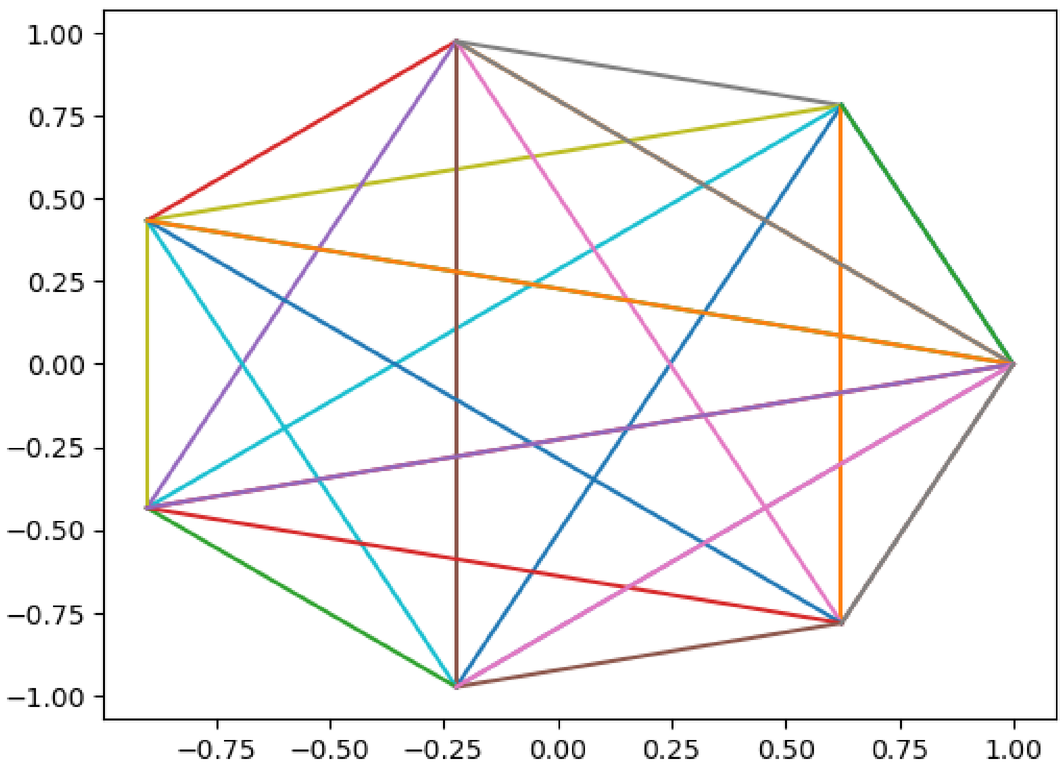
```
[51]: [<matplotlib.lines.Line2D at 0x7fb8e1017c70>]
```



```
[52]: len(xs)
```

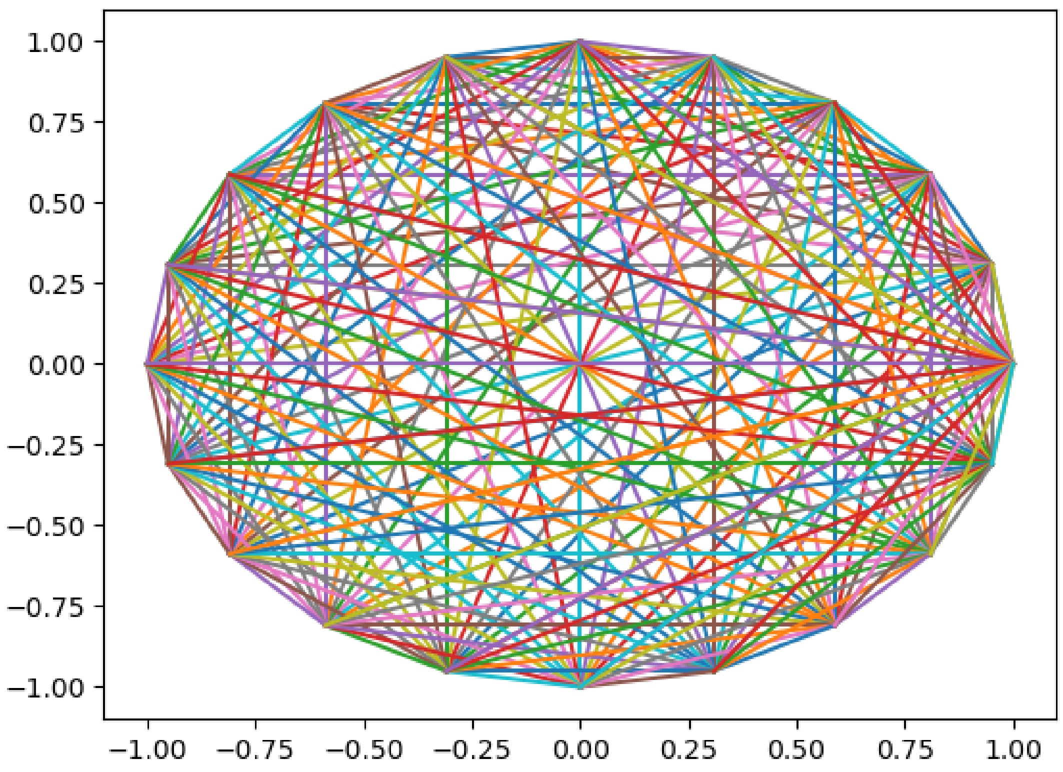
```
[52]: 8
```

```
[53]: for i in range(len(xs)):
          for j in range(i+1,len(ys)):
              plt.plot([xs[i],xs[j]], [ys[i],ys[j]])
```



[54]:
```
def rosette(n):
    pts = np.linspace(0,np.pi * 2, n+1)
    xs = np.cos(pts)
    ys = np.sin(pts)
    for i in range(len(xs)):
        for j in range(i+1,len(ys)):
            plt.plot([xs[i],xs[j]], [ys[i],ys[j]])
```

[55]:
```
rosette(20)
```


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

```
[ ]:
```