rootsumsquare_speedtest

October 5, 2020

[1]:

```
import numpy as np # Using Numpy functions

import functools # For timing decorator
import time

from typing import Callable, Tuple # Py3.8 type hints
```

[2]:

```
arr = np.random.rand(10000,3)*100
arr
# Has shape (10e3, 3)
# Contains values [0,100)
```

[2]:

```
array([[61.13728195, 69.18745048, 54.88712317],
       [65.87910921, 28.81345076, 98.17994997],
       [73.62183883, 15.98124766, 50.68841608],
       …,
       [58.1146232 , 37.86288826, 90.78003775],
       [54.26021097, 25.82608218,  4.63050161],
       [13.25944118, 29.12721574,  8.56601449]])
```

[3]:

```
def timer(func: Callable) -> Tuple[Callable, float]:
    """Time the execution of a function.
    """
    @functools.wraps(func)
    def wrapper_timer(*args, **kwargs):
        tic = time.perf_counter()
        value = func(*args, **kwargs)
        toc = time.perf_counter()
        elapsed = toc-tic
        return value, elapsed
    return wrapper_timer
```

[4]:

```
@timer
def longWay(a: np.ndarray) -> np.ndarray:
    """Calculates the root sum of the squares.
    Uses a longer method involving sqrt, sum and power.
    """
    return np.sqrt(np.sum(np.power(a, 2), axis=1))

@timer
def shortWay(a: np.ndarray) -> np.ndarray:
    """Calculates the root sum of the squares.
    Calls numpy.linalg.norm() to calculate.
    """
    return np.linalg.norm(a, axis=1)
```

[5]:

```
longWay(arr) # shape [[10000],1]
```

[5]:

```
(array([107.41166993, 121.6945954 ,  90.8013819 , …, 114.24501298,
         60.27104247,  33.12980654]),
 0.0014043209957890213)
```

[6]:

```
shortWay(arr) # shape [[10000], 1]
```

[6]:

```
(array([107.41166993, 121.6945954 ,  90.8013819 , …, 114.24501298,
         60.27104247,  33.12980654]),
 0.0004620629988494329)
```

[7]:

```
mean_long = []
mean_short = []
samples = 10000
for i in range(samples):
    arr = np.random.rand(10000,3)*100 # Generate dummy arrays
    mean_long.append(longWay(arr)[1]) # Calculate 'long' times
    mean_short.append(shortWay(arr)[1]) # Calculate 'short' times
long_time = np.mean(mean_long)
short_time = np.mean(mean_short)

# Return Stats
print(f"""Over n={samples} samples, with shape (10000, 3)\n
Average time to compute:
np.sqrt(np.sum(np.power(arr, 2), axis=1))   :    {long_time*1e6:0.0f}␣
 ↪microseconds
                                    sigma   :    {np.std(np.
 ↪array(mean_long)*1e6):0.3f}\n
np.linalg.norm(arr, axis=1)                 :    {short_time*1e6:0.0f}␣
 ↪microseconds
                                    sigma   :    {np.std(np.
 ↪array(mean_short)*1e6):0.3f}\n
A difference of                             :    {short_time*100./long_time:0.
 ↪2f}%""")
```

```
Over n=10000 samples, with shape (10000, 3)
```

```
Average time to compute:
np.sqrt(np.sum(np.power(arr, 2), axis=1))    :   657 microseconds
                                    sigma    :   100.355

np.linalg.norm(arr, axis=1)                  :   178 microseconds
                                    sigma    :   18.306

A difference of                              :   27.13%
```