Lecture 12 - Numpy Linear Algebra Tools

* AMA / Quiz / Prayer

I. Computational Cost.

* Last time, we talked about how direct methods were slow & iterative were faster. Lets be a little more quantitative about this.

* As we discussed earlier in the class, there are two general resources a computer has: memory and the CPU.

* we measure the memory a program uses in bits.

e.g. array of 500 floats.

1 float = 64 bits

500 floats = 64 bits $\times$ 500 = 32,000 bits

$= 32{,}000 \times \frac{\text{byte}}{8\text{bits}} = 4 \times 10^3$ bytes

$= 4$ kB.

* The total program memory is mostly made of the number of bits you have used in variables.

* We measure the amount of CPU used in the total running time of the calculation. The total run time is determined by the number of "steps" or operations the CPU needs to execute.

* We can count this by counting the number of lines of code that will execute.

e.g. How long will this code take to execute?

```
1: for i in range(n):
2:     print(i)
```

	cost	# of times
1:	c_1	n
2:	c_2	n

↑ How long it takes CPU to execute line #

```
>>> 0
    1
    2
    3
    ⋮
    n-1
```

total time: $T(n) = c_1 n + c_2 n = (c_1 + c_2) n$

Activity: Calculate how long it takes the CPU to execute this code:

```
for i in range(n):
    for j in range(n):
        A[i,j] = i * j
```

	cost	# of times
for i in range(n):	c_1	n
for j in range(n):	c_2	n^2
A[i,j] = i * j	c_3	n^2

$$T(n) = c_1 n + c_2 n^2 + c_3 n^2$$

$$= c_1 n + (c_2 + c_3) n^2$$

* when calculating cost, we often only care about the asympotic behavior when $n \to \infty$. Because of this we usually ignore the numerical prefactors

* So, in our first example, we would say:

$$T = O(n)$$

"scales like" " is of the order of "

"asymptotically approaches"

Our second example would be:

$T(n) = O(n^2)$ ← This is the largest power, the rest is not very important when n is big.

* Example:

$$T_1(n) \approx c_1 \cdot n^2$$

$$T_2(n) \approx c_2 \cdot n^2 + c_3 \cdot n$$

$c_1 = 1.1$ $c_2 = 1$, $c_3 = 1$, $n = 10^5$

$T_1(n) = 1.1 \times 10^{10}$, $T_2(n) = 10^{10} + 10^5$

$= 10^{10} + 10^9$

This term is way less important than the constant, c_1.

II. Numpy Linear Algebra

* We have spent a couple of days learning some important numerical analysis concepts around linear algebra. But the numpy module has everything we've done already built in!

* Useful numpy functions:

- Array addition, subtraction, scalar multiplication

$\underline{v} + \underline{w}$ → +

$\underline{v} - \underline{w}$ → −

$c\underline{v}$ → *

- matrix / array multiplication : $\underline{\underline{A}} \cdot \underline{v} \rightarrow$ np.dot(A, v)

  $\underline{v} \cdot \underline{w} \rightarrow$ np.dot(v, w)

- transpose : $\underline{\underline{A}}^T \rightarrow$ A.T

- matrix or vector norm : $\|\underline{\underline{A}}\| \rightarrow$ np.linalg.norm(A)

- solve a linear system (2 ways)

  (*) $\underline{\underline{A}}\,\underline{x} = b \rightarrow$ x = np.linalg.solve(A, b)

  $\underline{x} = \underline{\underline{A}}^{-1}\,\underline{b} \rightarrow$ Ainv = np.linalg.inv(A)

  x = np.dot(Ainv, b)

The better (faster!) way.

Activity — Practice w/ numpy linear algebra systems.

(See python activity).