Lecture 12 - Numpy Linear Algebra & Computational Cost

Outline

I. Numpy Linear Algebra

II. Computational Cost

III. Asymptotic Behavior

I. Numpy Linear Algebra

* Modules & Built in functions make our life easier! :)

Numpy (numerical python) → lin alg (linear algebra)

* Example functions

- Array math (element wise addition, subtraction, multiplication, scalar multiplication)

$\underline{v} + \underline{w}$, $\underline{v} * \underline{w}$ (vector, matrix)

- Sums & dot/inner products

  - np.sum($\underline{v}$)

```
sum_v
for i in range(len(v)):
    sum_v += v[i]
```

  - np.dot(v, w) ← vectors
  - np.dot(A, B) ← matrices
  - np.dot(A, v) ← A: matrix, v: vector

- Transpose

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

A → A.T, A^T

- $\|v\|$ (norm) - $\left(\sum_i v[i] ** 2\right)^{1/2}$

np.linalg.norm(v) or np.linalg.norm(A)

• Linear solutions ← Gauss Elimination
(FE + BS → 3 nested loops
" " 2 " ")

(1) $x = np.linalg.solve(A, b)$

(easiest, fastest for "dense" matrix)

(2) $A_{inv} = np.linalg.inv(A)$

$x = np.dot(A_{inv}, b)$

(2 steps, need to store A^{-1})

But, easy for multiple b's

$\underline{\underline{A}} \cdot \underline{x} = \underline{b}$

$\underline{\underline{A}}^{-1} \underline{\underline{A}} \cdot \underline{x} = \underline{\underline{A}}^{-1} \cdot \underline{b}$

$\underline{\underline{I}} \cdot \underline{x} = \underline{\underline{A}}^{-1} \cdot \underline{b}$

$\underline{x} = \underline{\underline{A}}^{-1} \cdot \underline{b}$

$Ax = b_1$
$Ax = b_2$
⋮

(Bonus)
LU decomposition

Example video ④

II. Computational Cost.

* When we run an algorithm, how many resources are we consuming?
  - Memory — can I store 100k data points?
  - CPU time — 5min? 5hours? 5years?
    → Cell phone vs. Supercomputer

* Memory: count # of bits you use.

array of 500 floats (double = 8 bytes = 64 bits per float)

$8 \frac{\text{bytes}}{\text{float}} \times 500 \text{ floats} = 4000 \text{ bytes} = 4 \text{ kB}$

1000 B = 1kB
1000 kB = 1MB ✓
1000 MB = 1GB ✓
1000 GB = TB ✓

* If I run a program, the memory consumed consists of all of my variables (+ a little extra)
  → O.S. call functions, etc.

  Big calculation → dominated by big arrays

* Running Time (CPU)

  - Determined by the # of clock cycles on your CPU.
    ↳ exact time is not easy to predict
    But # of executed lines of code is a good proxy.

Example:

		cost	# of times
1:	for i in range(n): (n: some number e.g. 10.)	c_1	n
2:	print (i)	c_2	n

>> 0
1
⋮
n-1

Total time: $T(n) = c_1 \cdot n + c_2 n$
$= (c_1 + c_2) n$

Big Calculations → dominated by big loops

↑ want to know this dependence.

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

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

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

III. Asymptotic Behavior

* How can I know how long my code is going to run, when I have a "big" system

$n = 5$ (small) — fast

$n = 10^3$ (big) — ?

$n \to \infty$ (asymptotic behavior)

* $T_1(n) = c_1 n^2$ $\qquad$ $T_2(n) = c_2 n^2 + c_3 n$

suppose $n = 10^5$ $\qquad$ $c_1 \approx 1.1$ sec. $\qquad$ $c_2 = c_3 \approx 1$ sec.

$T_1 = 1.1 \times 10^{10}$ sec.

(3600 sec/hr)

$= 1.10000 \cdot 10^{10}$ sec.

$T_2(n) = 1 \cdot 10^{10} + 1 \cdot 10^5$

$= 1.00001 \cdot 10^{10}$

* when n is big, bigger powers matter most!

* We really only care about the biggest power!

$T(n) = c_1 n^2 + c_2 n + \cdots = O(n^2)$ "order n^2" (for big n, n^2 only matters)

$T(n) = c_1 n^3 + c_2 n^2 + \cdots = O(n^3)$ "order n^3"

loops.

Gauss Elim: FE $\to O(n^3)$ ✓ $\leftarrow 10^9$ $\qquad$ $n = 10^3$

BS $\to O(n^2)$ ✓ $\leftarrow 10^6$

GE? $\longrightarrow O(n^3)$

How is this practical

* Just Back Sub: $T(n) = \Theta(n^2) \approx c n^2$

Suppose @ $n = 10^2$, $T(n) = 10$ sec. ← experiment measured.

How long will it take for $n = 10^4$?

depends on Mac? PC? Intel? i7, i5? compiled C++? Python?

$$10 \text{ sec} = c (10^2)^2 \Rightarrow c = \frac{10 \text{ sec}}{10^4} = 10^{-3} \text{ sec.}$$

$$T(10^4) = c (10^4)^2 = c \cdot 10^8 = 10^{-3} \text{ sec.} \cdot 10^8$$

$$= 10^5 \text{ sec.} \approx 30 \text{ hrs} = \underline{1 \text{ day.}}$$

3600 sec = 1 hr.
$\approx 10^3$

$10 \text{ sec.} (100 \times)^2 = 10^5 \text{ sec.}$ ← alternate.

$\left(\frac{10^4 \leftarrow \text{new}}{10^2 \leftarrow \text{old}}\right)^2$ ← order

multiplier $= 10^4 \cdot 10 \text{ sec} = 10^5 \text{ sec.}$ ← meas.