Costo computacional de un algoritmo

Significado de $O(t)$

La notación $O(t)$ es de uso habitual en matemática y tiene un significado específico dado por la siguiente definición [1]:

Dadas dos funciones φ, ψ con valores reales, la igualdad

$$\varphi(t) = O(\psi(t)), \tag{1}$$

indica que existe una constante C tal que, para todo t suficientemente cercano a un valor límite t_0 (por ejemplo, $t \to 0$, $t \to \infty$, etc.), se tiene

$$|\varphi(t)| \leq C\psi(t). \tag{2}$$

Como ejemplo, recordemos la relación $\frac{\mathrm{sen}^2(\mathrm{t})}{\mathrm{t}^2} \leq 1$ que puede observarse de la Figura 1-(a). Esto nos permite afirmar que $\mathrm{sen(t)}^2 \leq t^2$ cuando $t \to 0$. De esta manera,

$$|\mathrm{sen}^2(\mathrm{t})| = \mathrm{sen}^2(\mathrm{t}) \leq C\ t^2 \quad \text{con} \quad C = 1, \tag{3}$$

y diremos que $\varphi(t) = O(\psi(t))$ cuando $t \to 0$. Particularmente si $\varphi(t)/\psi(t) = 1$ para $t \to t_0$ (como en el caso analizado), decimos que $\varphi(t)$ tiende asintóticamente a $\psi(t)$, lo que significa que en ese límite ambas funciones son practicamente indistinguiles (ver Figura 1-(b)).

Como otro ejemplo, vemos que la función $\varphi(x) = x^2 + x$ es $O(x^2)$ cuando $t \to \infty$, ya que

$$\begin{aligned} |x^2 + x| \quad &\leq \quad |\ x^2\ | + |\ x\ | \quad \leq \quad |\ x^2\ | + |\ x^2\ | \quad \text{para} \ \ |x| \geq 1, \\ &\to \quad |x^2 + x| \leq 2\ |\ x^2\ | = 2x^2. \end{aligned} \tag{4}$$

Esta relación es váida siempre que $|x| \geq 1$ y por lo tanto será válida en el límite $t \to \infty$. Por otra parte

$$\frac{x^2 + x}{x^2} = 1 + \frac{1}{x} \to 1 \ \text{ cuando } \ x \to \infty, \tag{5}$$

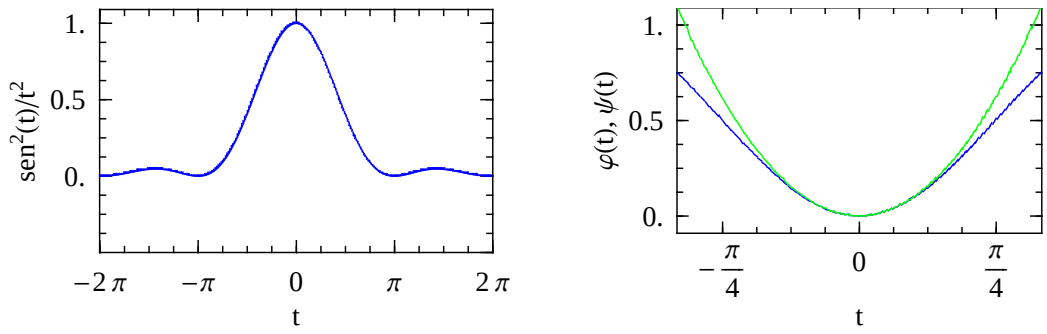

Figura 1: (a) Gráfico de la función $\text{sen}^2(\text{t})/t^2$. (b) Gráficos de la funciones $\varphi(t) = \text{sen}^2(\text{t})$ (línea azul) y $\psi(t) = t^2$ (línea verde).

y $\varphi(x)$ tiende asintóticamente a x^2.

Número de operaciones

Cuando contamos con un algoritmo que permite resolver algún problema de cálculo, es importante tener conocimiento de cual es su costo de cálculo, es decir, cual es el número de operaciones que deberán realizarse para completarlo y obtener el resultado que estamos buscando. Esto permite calificar el algoritmo y comparar su eficiencia con respecto a otro que resuelva el mismo problema.

En una computadora, todos los cálculos matemáticos se reducen a un conjunto de operaciones aritméticas elementales denominadas operaciones de punto flotante[1], *flops*. Estas operaciones son las de suma, resta, multiplicación y división.

Tomemos como ejemplo un polinomio genérico de grado 2

$$p(x) = a_0 + a_1 x + a_2 x^2. \tag{6}$$

Supongamos que queremos conocer el número de operaciones necesarias para evaluar dicho polinomio en un valor $x = x_0$, esto es, cuántas operaciones elementales deben realizarce para conocer el valor que toma $p(x)$ cuando $x = x_0$:

$$p(x_0) = a_0 + a_1 x_0 + a_2 x_0^2. \tag{7}$$

En el primer término no es necesaria ninguna operación ya que a_0 es conocido; en el segundo término debemos resolver el producto $a_1 . x_0$ lo que implica una operación elemental; el valor del

[1]La *aritmética de punto flotante* está basada en una *representación de punto flotante* del conjunto de números reales. En un sistema numérico de punto flotante, la posición del punto decimal (o binario) se almacena separadamente de los dígitos, y la precisión con que puede representarse un número es proporcional al valor del propio número. Esto lo distingue de una *representación de punto fijo*, donde la precisión es constante.

último término se encuentra luego de realizar 2 productos, uno para hallar el valor de $x_0^2 = x_0.\ x_0$, y luego uno más para hallar $a_2 x_0^2 = a_2.\ x_0^2$. Finalmente debemos sumar los tres términos, lo que agrega dos operaciones de suma al cómputo. En resumen, tenemos 3 productos y 2 sumas, lo que da un total de 5 flops para la evaluación del polinomio.

Pero la expresión (6) no es la única para el polinomio $p(x)$. Por ejemplo, sacando factor común x entre el segundo y tercer término, $p(x)$ también puede ser escrito como

$$p(x) = a_0 + x(a_1 + a_2 x). \tag{8}$$

En esta forma, el número de flops necesarias para su evaluación se reduce a 4: 2 productos y 2 sumas. Esto no parece un ahorro de cálculo muy importante, pero veamos que pasa en el caso más general...

Sea el polinomio de grado n

$$q(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + ... + a_{n-1} x^{n-1} + a_n x^n. \tag{9}$$

Para evaluarlo en $x = x_0$ necesitaremos resolver:

ninguna operación en el primer término,

un producto en el segundo término $a_1.\ x_0$,

dos productos en el tercer término, $a_2.\ x_0.\ x_0$,

tres productos en el cuarto término, $a_3.\ x_0.\ x_0.\ x_0$,...

y así siguiendo...

en el ante último término : $a_{n-1}.\ x_0^{n-1} = a_{n-1}.\ \underbrace{x_0.\ x_0.\ x_0 ... \ x_0}_{n-2 \text{ productos}} \rightarrow n-1$ productos,

en el último término : $a_n.\ x_0^n = a_n.\ \underbrace{x_0.\ x_0.\ x_0 ... \ x_0.\ x_0}_{n-1 \text{ productos},} \rightarrow n$ productos,

lo que nos da un total de

$$0 + 1 + 2 + 3 + ... + n - 1 + n = \sum_{k=0}^{n} k = \frac{n}{2}(n+1) \ \text{ productos.} \tag{10}$$

Finalmente, debemos sumar los $n+1$ términos de la expresión (9), lo que implica realizar n sumas. Entonces, el número de flops requeridas para evaluar un polinomio de grado n escrito en su *forma canónica* es $\varphi(n) = \frac{n}{2}(n+3) = \frac{n^2}{2} + \frac{3}{2}n \sim \frac{n^2}{2}$.

Es habitual, al indicar el número de operaciones de un algoritmo, despreciar los términos de menor orden, como hicimos en este caso, ya que los mismos suelen no ser significativos a menos que n sea pequeño. Por otra parte, podemos utilizar la notación "O" para calificar el costo de cálculo. Vemos que si $n > 2$ esta expresión es menor a $2n^2$, y por lo tanto diremos que el algoritmo es

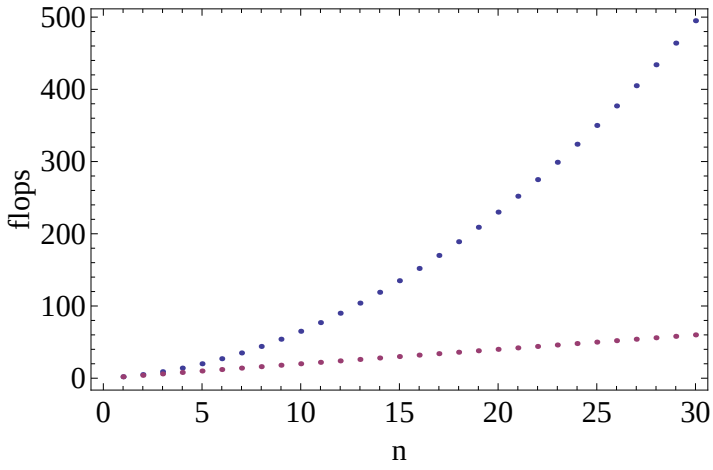

Figura 2: Número de operaciones empleadas en la evaluación de un polinomio de grado n escrito en la forma canónica (puntos azules) y en la forma de Horner (puntos rosas)

$O(n^2)$. Más todavía, $\lim_{n\to\infty} \frac{\varphi(n)}{n^2/2} = \lim_{n\to\infty} 1 + \frac{3}{n} = 1$ y ambas funciones tienden asintóticamente al mismo valor, lo que justifica el no tener en cuenta el término $\frac{3}{2}n$ para n grande.

Con esto en mente, diremos que todos los algoritmos que sean $O(n)$ son comparables respecto de su costo de cómputo; todos los que sean $O(n^2)$ son comparables entre sí, y así en general, todos los algoritmos $O(n^k)$ tendrán una eficiencia computacional similar.

Pasemos ahora al análisis para el mismo polinomio pero escrito como en (8)

$$q(x) = (...((a_n x + a_{n-1})x + a_{n-2})x + ... + a_1)x + a_0. \tag{11}$$

A esta forma se la conoce como *forma de Horner*. Esta expresión requiere de n productos y n sumas para su evaluación [2]. Entonces el número de operaciones requeridas es $2n$. Este algoritmo es $O(n)$, un orden menor al caso anterior, y por lo tanto su costo computacional también lo es.

En la figura (2) se graficó, para ambos casos, la cantidad de flops necesarias en función del grado del polinomio. Claramente podemos ver cuánto más eficiente es la forma de Horner al momento de evaluar un polinomio. El programa de cálculo Matlab contiene los comandos **tic** y **toc** que nos permiten comparar el costo de cómputo entre distintos algoritmos. Escribiendo la sentencia tic al inicio del proceso y toc al final, obtendremos el tiempo empleado por la computadora para completar el mismo. Como ejemplo de su utilización, en la figura (3) se muestra una secuencia de sentencias para evaluar un polinomio de grado $n = 10^8$ en el valor $x_0 = 1$, tanto para la forma estándar como para la forma de Horner[3]. Al finalizar el proceso, se obtiene en pantalla el tiempo

[2] Le dejamos la demostración como ejercicio. No es difícil.

[3] Vea por ejemplo, cómo van creciendo los tiempos de cómputo con el grado del polinomio

```
clc,clear
N=1e8;a=1:N;x=1;

tic,p=a(1);for k=2:N,p=p+a(k)*x^k;
end,
toc % metodo estandar

tic,pn=a(1);
for k=2:N,pn=pn*x+a(k);
end,
toc % metodo de Horner
```

Figura 3:

empleado en cada caso:

Elapsed time is 7.134219 seconds.

Elapsed time is 2.269280 seconds.

Por último, aunque no nos enfoquemos en este punto, debemos tener presente que en el costo computacional de un algoritmo hay mucho más que el número de operaciones que el mismo requiere. Por ejemplo, en una computadora que cuente con un único procesador, el tiempo de ejecución está afectado por el movimiento de los datos entre distintos componentes de la memoria y por otros trabajos que estén ejecutándose al mismo tiempo. En una máquina con un procesador múltiple, también hay que tener en cuenta el tiempo empleado en la comunicación entre procesadores. Por el momento sólo nos enfocaremos en lo que respecta a la aritmética de un algoritmo.

Bibliografía

[1] Lloyd N. Trefethen and David Bau III, *Numerical Linear Algebra* (Society for Industrial and Applied Mathematics, Philadelphia, UK, 1997).