Computational complexity
dna47a 2d

Hello,

What is difference between asymptotic computational complexity and order of complexity?

As far as I understand in order of complexity we ignore any constants that is in front of the variable and so we say that the order of complexity of Backward Euler is M times more than Forward Euler but in asymptotic computational complexity we consider the constants like 2, 2/3 and so on?

Thanks.

Related to 12 Stiffness and Implicit Methods for IVPs / 12.3 Implementation of Implicit Methods / 12.3.1 Implementation of Implicit Methods for Linear Systems

Showing 2 responses Newest first

Add comment

My posts All posts Topics Learners

Search all posts

Add a post

All posts sorted by recent activity

[staff] Submission not graded? Hello, I have got ...
HusamMalkawi
18 1d

Computational complexity Hello,   What ...
dna47a
3 2d

Stability of Implicit Methods No preview available
sandipan_dey
6 1 New 1w

PSET2 plot_Y return values: grader accepts axes ...
sandipan_dey
2 1w

FE code submissions The grader passed my ini ...
m_powers
9 2w

Parachute is getting deployed much earlier ...
sandipan_dey
6 1w

These are good points to be clear about. @sandipan_dey gave a nice recap of what asymptotic computational complexity means and the associated notation. The details are covered in 6.00.1x (or on OpenCourseWare's 6.0001).

@sandipan_dey also makes a good point about the meaning of the term "order". However, the description given is about the *order of* ***accuracy*** for a given numerical method. You can review the definition in Section 9.2.1. Specifically, the *order* there refers to the degree of the polynomial in terms of Δt.

I believe your question is about ***runtime complexity***, which deals with how long it takes to arrive at an answer, rather than the accuracy of the result. In this context, the "asymptotic complexity" and "order of complexity" are indeed more or less interchangeable. (The former tends to be more formal; "order" may be used more colloquially, e.g., "The function $f(x) = 3x^2 + \sqrt{x}/8$ has order x^2, or is second-order.")

In Section 12.3.1 that you posted this in, we are using the term "asymptotic complexity" somewhat loosely. It is the case that strictly according to the big-O/Theta/Omega definitions, we do drop the constant, because the asymptotic behavior is the same after scaling. When comparing FE vs BE, though, tracking the constants allows us to more accurately identify the threshold of required timesteps where it's better to switch from one method to the other. If we had only said an FE step is $O(M^2)$ and a BE step is $O(M^3)$, we would only be able to conclude that BE is more efficient than FE when we need to run it for fewer timesteps by a factor of at least

$O(M)$ to achieve the same error. But is that factor closer to M, or $M/2$, or $M/10$, or $\frac{2}{3}M$? Knowing the constant can help you better pinpoint when to cross over.

Finally, note that the runtimes we claim are "asymptotic" in the sense that we still drop lower-order terms. For example, the matrix-vector multiplication in FE is not exactly $2M^2$ operations. Calculating each element in the new vector requires M multiplications and $M-1$ additions, so across all M elements, the total cost is $2M^2 - M$ operations. (Then, there are the remaining operations to calculate v^{n+1}.) The point is, the exact constant in front of the M^2 isn't necessarily 2, especially when M is small. But for large M where the highest-order polynomial term dominates, 2 is a pretty good approximation, so the final $1/3$ constant in our switchover threshold is also a pretty good ratio in those situations.

In summary, "constants don't matter until they do," and you have to use your understanding of the problem statement to decide when to pull hidden constants out of the definition of asymptotic complexity.

sandipan_dey 1d

In case of asymptotic computational complexity, we are often interested to provide an upper / lower / tight bound $O(f(|x|)),\ \Omega(f(|x|)),\ \Theta(f(|x|))$ for the runtime of the algorithm, as a function of the length of the input x (e.g., binary search has logarithmic runtime complexity) and in case of the deterministic algorithms, the output is expected to be computed by the algorithm by that time (when a Turing machine starts with the input x on its tape, it's guaranteed to stop after writing the correct output on its tape in time $O(f(|x|))$).

Here, for the iterative numerical methods, the order is defined by how fast the error of approximation decreases with number of iterations. We are not only interested how fast the algorithm terminates, but also how much accuracy it provides after it stops and produces the output and how fast the accuracy improves with iterations.

Add a response

Connect

Idea Hub
Contact Us
Help Center
Security
Media Kit

GET IT ON Google play
Download on the App Store