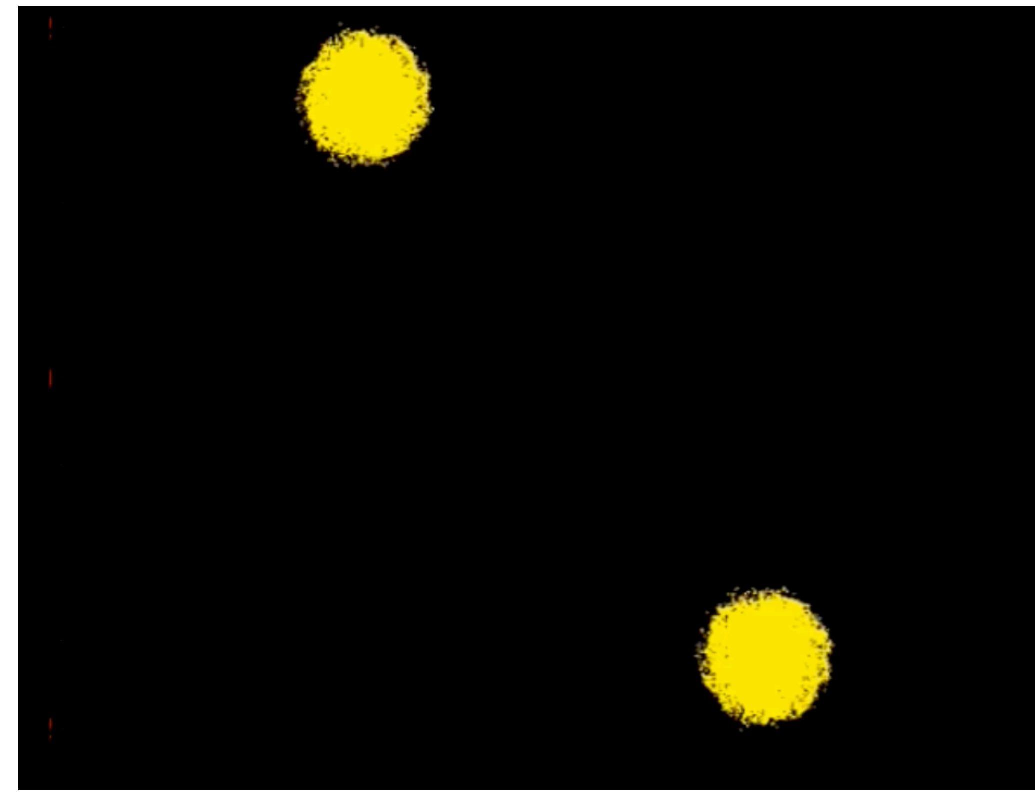
NagBody lectures: Complexity: analysis of algorithms



Mario Alberto Rodríguez-Meza

Instituto Nacional de Investigaciones Nucleares

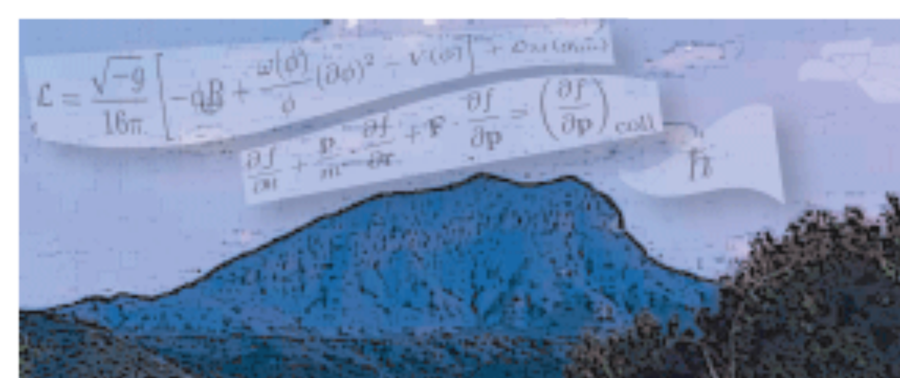
Correo Electrónico: marioalberto.rodriguez@inin.gob.mx

http://bitbucket.org/rodriguezmeza

Seminario de investigación,
Departamento de Física,
Universidad de Guanajuato
3 de febrero al XX de junio de 2022
Sesiones virtuales (Zoom, Meet, etcétera)

quintessence Group

INSTITUTO AVANZADO DE COSMOLOGIA



References and material

- Cosmología numérica y estadística: NagBody kit (http://bitbucket.org/rodriguezmeza). Mario A. Rodríguez-Meza. And: https://github.com/rodriguezmeza/NagBody_lectures.git

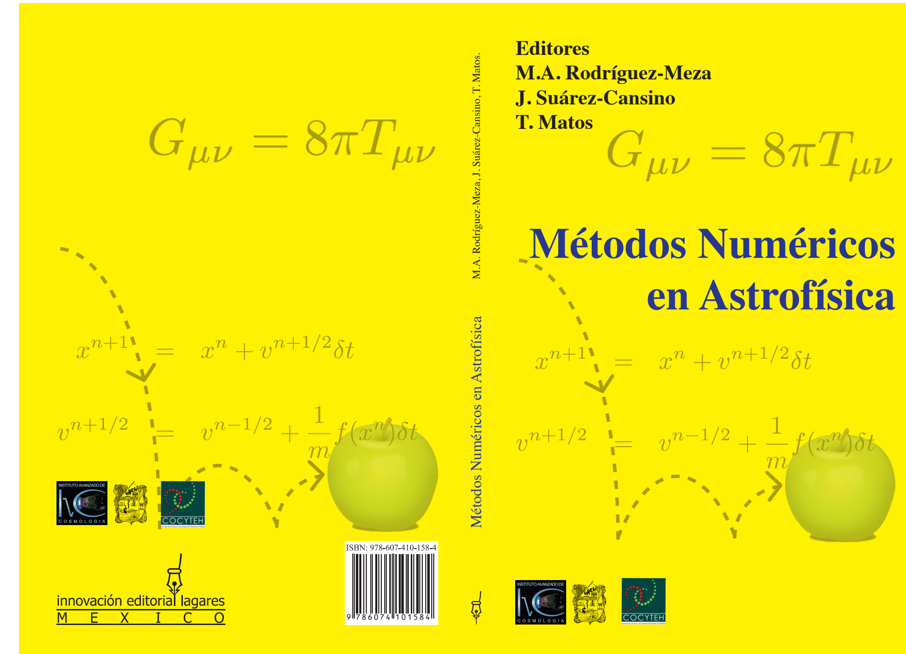
- Métodos numéricos en astrofísica, capítulo 1, Método de N-cuerpos en astrofísica. (https://www.researchgate.net/publication/316582859_Metodo_de_N-Cuerpos_en_Astrofisica)

- La estructura a gran escala del universo. Capítulo 22 en Travesuras cosmológicas de Einstein et al. https://www.researchgate.net/publication/316582400_La_estructura_a_gran_escala_del_universo_simulaciones_numericas

- https://www.researchgate.net/profile/Mario_Rodriguez-Meza

- https://www.researchgate.net/publication/314281416_Los_agujeros_negros_y_las_ondas_del_Dr_Einstein

- M.A. Rodriguez-Meza, Adv. Astron. 2012, 509682 (2012). arXiv:1112.5201. (https://www.researchgate.net/publication/51967093_A_Scalar_Field_Dark_Matter_Model_and_Its_Role_in_the_Large-Scale_Structure_Formation_in_the_Universe)



Content: Complexity

- Worst-case analysis
- *O*-notation
- Computational complexity

Concept of Analysis of algorithms

- In computing it is important to understand how an algorithm perform.
- When we are designing an algorithm usually the aspect of most interest is how fast it will run.
- Storage is also important.
- Therefore we need a formal and a deterministic way of analyze algorithms.

Concept of
We will cover:

- Worst-case analysis. The metric by which most algorithms are compared. Other cases are: the average case and the best case.

- *O*-notation. It is a formal method to express the performance of an algorithm. It will be the upper bond of a function within a constant factor.

- Computational complexity. It is the growth rate of the resources (CPU time, storage…) an algorithm requires with respect to the size of the data it processes. The *O*-notation is a formal expression of an algorithm's complexity.

Concept of Worst-case analysis:

- Algorithms do not perform the same in all situations.
- There are three cases: *best* case, *worst* case and *average* case.
- Example: linear search. A natural but inefficient way of searching a list: traverse data from one end to the other.
- Best case: element we are looking is in the first place.
- Worst case: element we are logic is in the last place.
- Average case: element is somewhere in the middle.

Concept of Worst-case analysis:

- *Reasons why we chose Worst-case analysis*:

  a. Many algorithms perform their worst case a large part of the time.

  b. Best case is not very informative. For example, all searching algorithms behave almost the same in this case. Comparison among algorithms is an important matter.

  c. Determine the average-case performance is not always easy. Or even what does it means “average”.

  d. The worst-case gives us an upper bound on performance.

Concept of *O*-notation

- It is the most common notation used to express algorithms' performance.

- It is an upper bound of a function within a constant factor:

- If $g(n)$ is an upper bound of $f(n)$, then for some constant c it is possible to find a value of n, called it n_0, for which any value of n >= n_0 will result in $f(n)$ <= c g(n).

- We *express performance of an algorithm* as a function of the size of the data it processes. Then $f(n)$ will such an function and n the size of the data.

- We are only interested in the *algorithm's growth rate*, or the *order of the grow*. It will describe how efficient an algorithm is for arbitrary input.

- And *O*-notation is such a function.

Simple rules for *O*-notation

- We can ignore constant terms because as the value of n becomes larger and larger, eventually constant terms become insignificant: $T(n) = n + 50$.

- We can ignore constant multipliers of terms because they too will become insignificant as values of n increases. $T_1(n) = n^2$ and $T_2(n) = 10\ n$. Value of n has only be greater than 10 for T_1 to becomes greater than T_2.

- We only consider the highest-order therm because, again as n increases, higher-order terms quickly outweigh the lower-order term ones. For example, $T(n) = n^2 + n$, for n=1024, the lesser order term of this expression constitutes less than 0.1% of the running time.

Simple rules for *O*-notation

- These ideas are formalized in the following rules:

  a. Constant terms are expressed as $O(1)$. Regardless of the size of the data. formally: $O(c) = O(1)$.

  b. Multiplicative constants are omitted. Formally stated, for a constant c: $O(c\,T) = c\,O(T) = O(1)$. If three tasks each run in time $T(n)$, the result is $O(3n)$.

  c. Addition is performed by taking the maximum. Formally stated: $O(T_1) + O(T_1 + O(T_2) = \mathrm{Max}(O(T_1), O(T_2))$. For example: $T_1(n)=n$ and $T_2(n)=n^2$ describe two task executed sequentially, the result is $O(n) + O(n^2)$. Which simplifies to $O(n^2)$.

  d. Multiplication is not changed but often is rewritten more compactly. Formally: $O(T_1)\ O(T_2) = O(T_1\ T_2)$. For example, in a nested loop where the outer loop are described by T_1 and whose inner iterations by T_2 and if they both are equal to n, then the result is n^2.

Concept of Computational complexity:

Homework: analyse nbody_n2 and gbsph codes..
Compare computing time.

- When speaking of the performance of an algorithm usually the aspect of interest is its *complexity*. The *grow rate* of the resources (time and storage: memory and disk) with respect to the size of the data.
- *O*-notation describes algorithm's complexity.
- We frequently can describe *worst-case complexity* by simply inspecting its overall structure. Other times it is helpful to employ techniques involving recurrences and summation formulas.
- To understand complexity it is important to look at the way to surmise the resources an algorithm will require.
- CPU time can be measured at each one of the major pieces that form the computational flow.

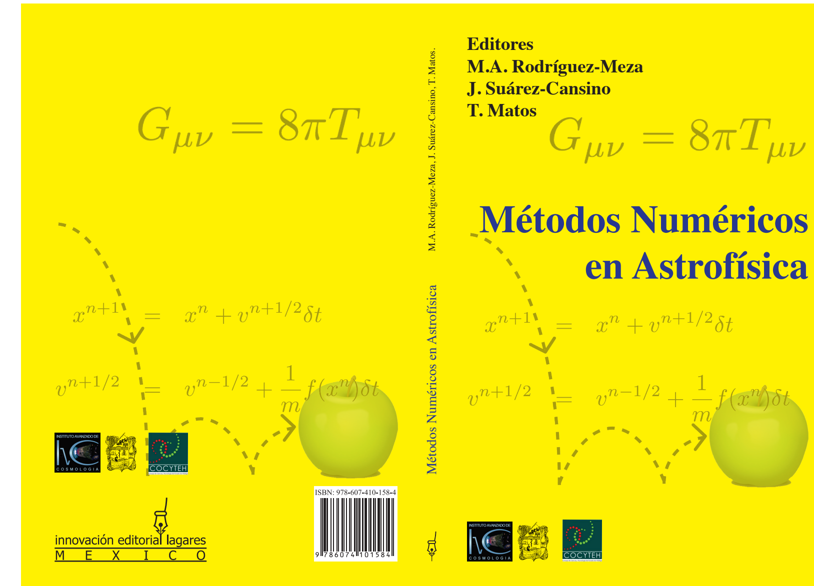
Conclusions: Recursion. Part II

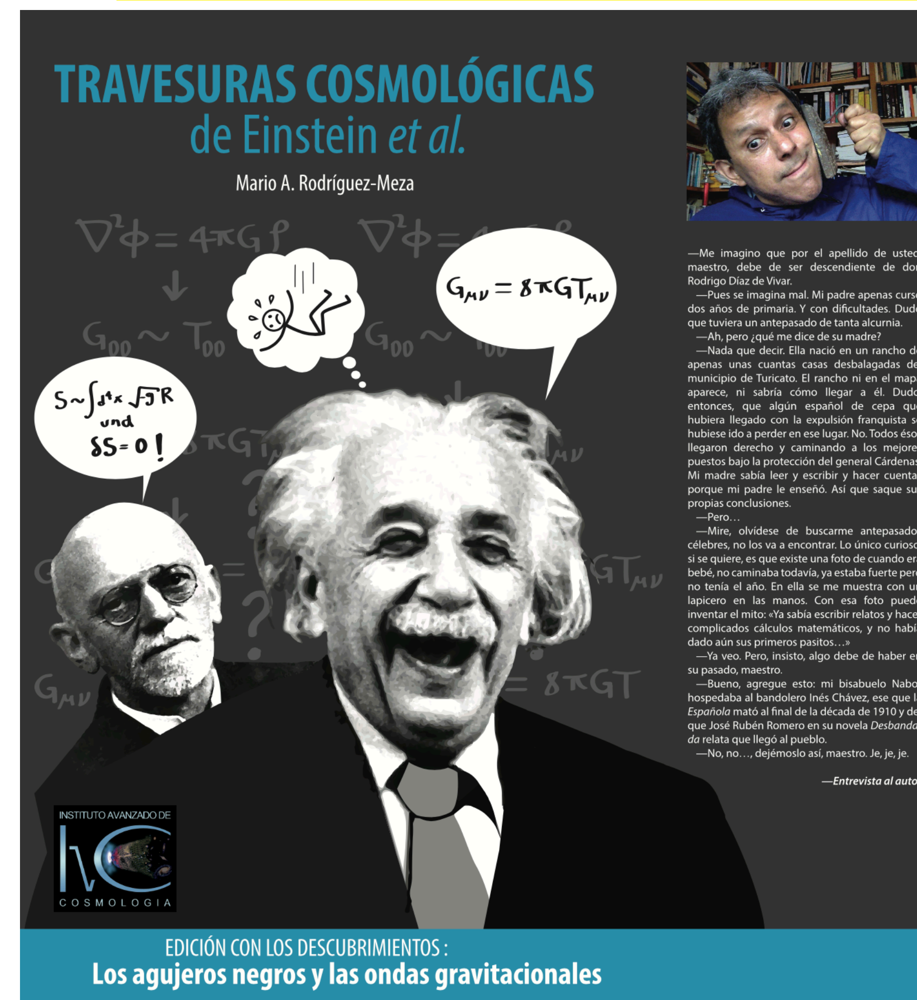
We have seen:

- Worst-case analysis. Against best and average cases.
- *O*-notation. Simple rules, examples and why it works.
- Computational complexity. CPU time, memory consuming and hard-disk storage.

References and material

- Cosmología numérica y estadística: NagBody kit (http://bitbucket.org/rodriguezmeza). Mario A. Rodríguez-Meza. And: https://github.com/rodriguezmeza/NagBody_lectures.git
- Métodos numéricos en astrofísica, capítulo 1, Método de N-cuerpos en astrofísica. (https://www.researchgate.net/publication/316582859_Metodo_de_N-Cuerpos_en_Astrofisica)
- La estructura a gran escala del universo. Capítulo 22 en Travesuras cosmológicas de Einstein et al. https://www.researchgate.net/publication/316582400_La_estructura_a_gran_escala_del_universo_simulaciones_numericas
- https://www.researchgate.net/profile/Mario_Rodriguez-Meza
- https://www.researchgate.net/publication/314281416_Los_agujeros_negros_y_las_ondas_del_Dr_Einstein
- M.A. Rodriguez-Meza, Adv. Astron. 2012, 509682 (2012). arXiv:1112.5201. (https://www.researchgate.net/publication/51967093_A_Scalar_Field_Dark_Matter_Model_and_Its_Role_in_the_Large-Scale_Structure_Formation_in_the_Universe)





See you!