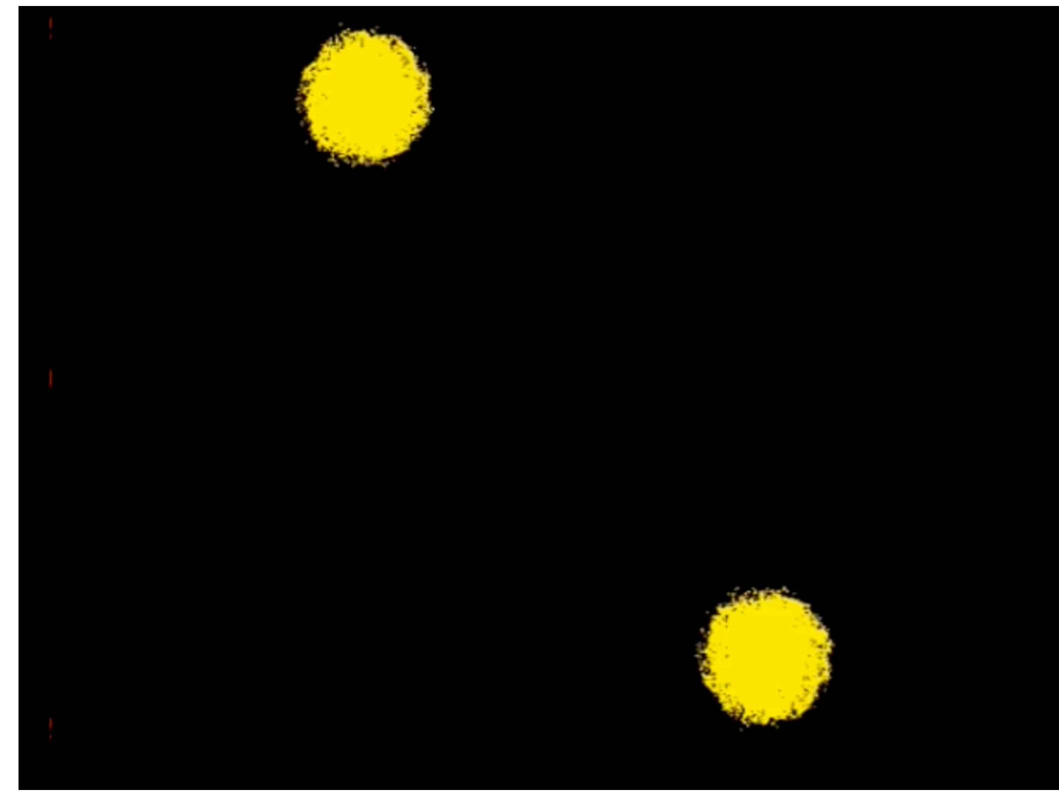
NagBody lectures: Recursion



Mario Alberto Rodríguez-Meza

Instituto Nacional de Investigaciones Nucleares

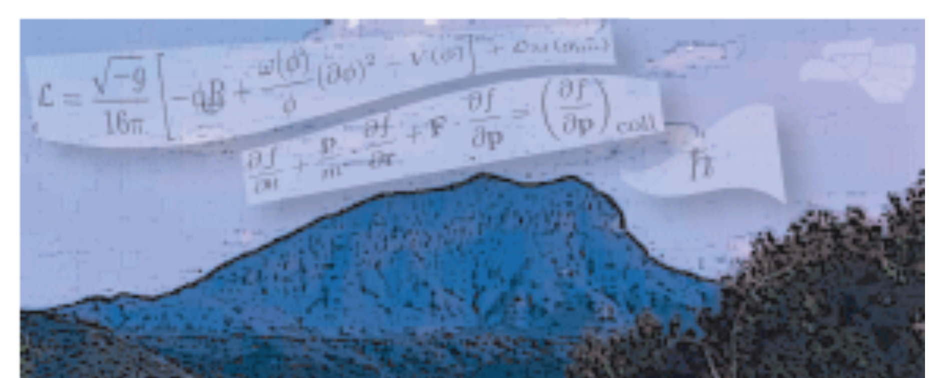
Correo Electrónico: marioalberto.rodriguez@inin.gob.mx

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

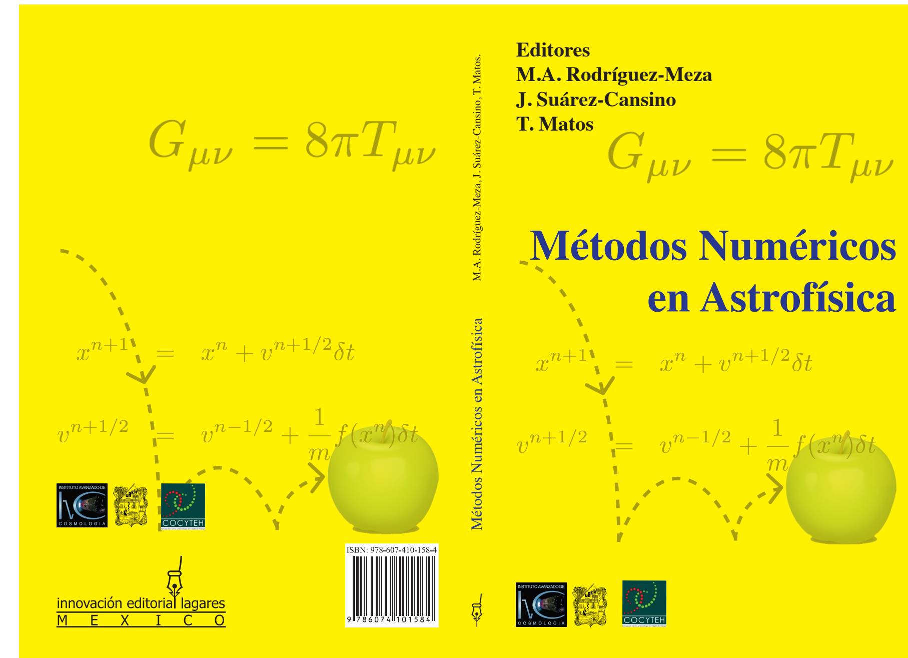
- Métodos numéricos en astrofísica, capítulo 1, Método de N-cuerpos en astrofísica. (https://www.researchgate.net/publication/316582859_Metodo_de_N-Cuerpos_en_Astrofisica)

- 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: Recursion

- Basic recursion
- Tail recursion

Concept of Recursion

- Recursion allows something to be defined in terms of smaller instances of itself.
- Nature: Fern leaves.
- Simplicity of using recursion… and powerful
- In C recursion is supported through recursive functions. A function that it calls itself.
- Some examples are in tree traversal and in sorting a list.

Basic recursion

Homework: make program
to compute the factorial function

- Basic recursion is the simplest way to use functions that call themselves.
- Tail recursion is an optimized way of using recursive functions. Most can be at compilation level.
- A simple example (not always consider as recursion process) is the factorial function: n! = n (n -1) (n - 2)… 1
- Consider the factorial as a product of smaller factorials: n! = n (n-1)! and so forth until n=1.

$$F(n) = \begin{cases} 1 & \text{if } n = 0,\ n = 1 \\ nF(n-1) & \text{if } n > 1 \end{cases}$$

Basic recursion

Homework: make program to compute the factorial function recursively

- Winding. Perpetuates itself making additional recursive calls. Terminates when one of the calls reach the termination condition, it returns instead of making another recursive call.
- Always must be at least one termination condition.
- Unwinding. When winding phase is complete the process enters the unwinding phase until the original call returns. And the recursive process is complete.

$F(4) = 4 \times F(3)$	winding phase
$F(3) = 3 \times F(2)$	.
$F(2) = 2 \times F(1)$	.
$F(1) = 1$	terminating condition
$F(2) = (2)(1)$	unwinding phase
$F(3) = (3)(2)$	.
$F(4) = (4)(6)$	.
24	recursion complete

$$F(n) = \begin{cases} 1 & \text{if } n = 0,\ n = 1 \\ nF(n-1) & \text{if } n > 1 \end{cases}$$

Basic recursion

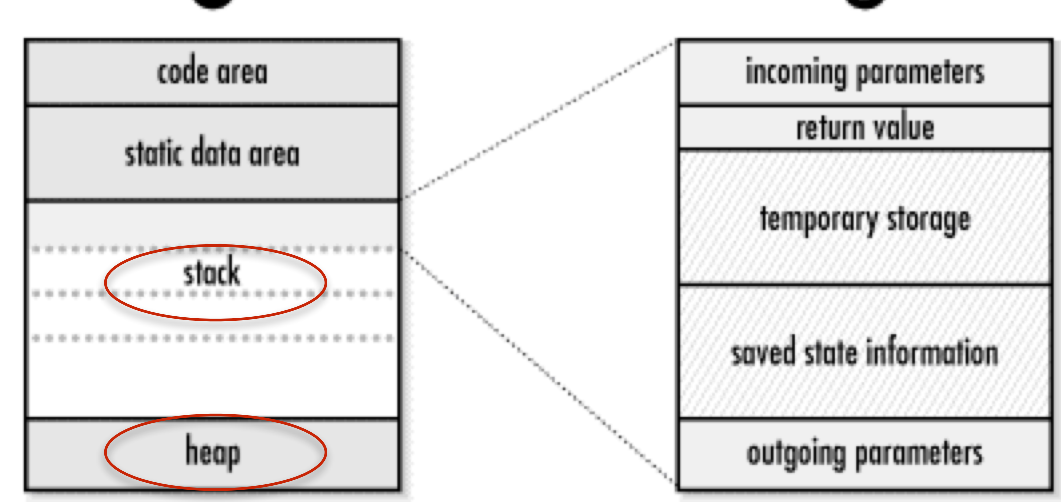
Homework: make program to compute the factorial function recursively using tail method

- Winding. Perpetuates itself making additional recursive calls. Terminates when one of the calls reach the termination condition, it returns instead of making another recursive call.
- Always must be at least one termination condition.
- Unwinding. When winding phase is complete the process enters the unwinding phase until the original call returns. And the recursive process is complete.

$F(4) = 4 \times F(3)$	winding phase
$F(3) = 3 \times F(2)$	.
$F(2) = 2 \times F(1)$	.
$F(1) = 1$	terminating condition
$F(2) = (2)(1)$	unwinding phase
$F(3) = (3)(2)$	.
$F(4) = (4)(6)$	.
24	recursion complete

$$F(n) = \begin{cases} 1 & \text{if } n = 0,\ n = 1 \\ nF(n-1) & \text{if } n > 1 \end{cases}$$

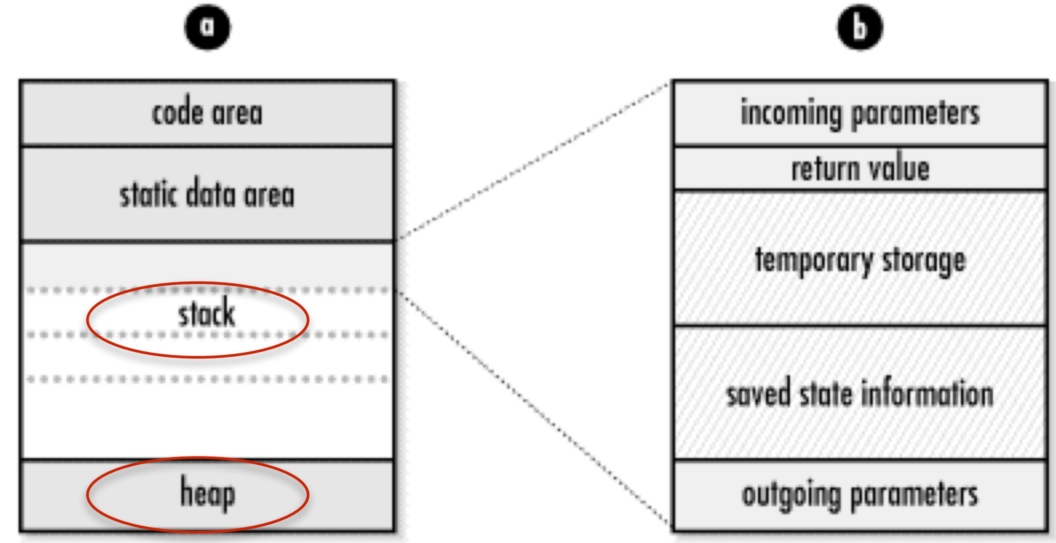
How basic recursion works



Five regions

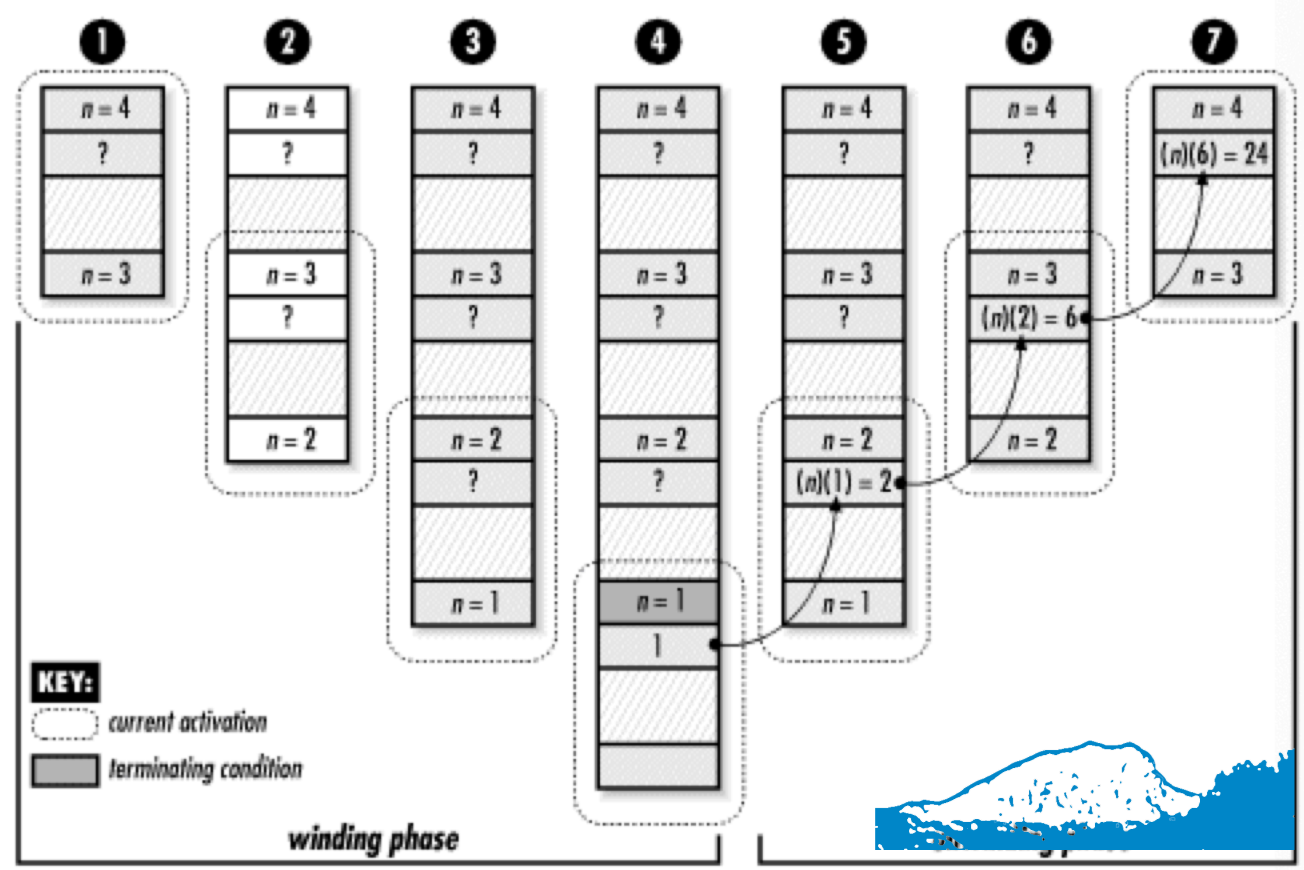
- How recursion works. Look at how C functions are executed.
- How a C program is organized in memory.
- Four areas as it executes: code area; static area; heap and stack.
- Code area: contains machine instructions. Executed as program runs.
- Static area: contains data the persist throughout the life of the program, such global variables and static local variables.
- Heap contains dynamical allocated storage, such memory allocated using malloc.
- The stack contains information about functions calls: *the activation record*. Remains in the stack until the call terminates.
- When a function is called in a C program, a block of storage is allocated on the stack to keep track of information associated with the call.

How recursion works



Five regions

- What happens on the stack when one computes 4!
- The initial call to fact results in one activation record being placed on the stack with incoming parameter of n=4.
- This activation does not meet any of the terminating conditions and fact is recursively called with n set to 3. This places another activation record of fact on the stack but with an incoming parameter of n=3 and son…
- Until the termination condition is met and the unwinding process begins.



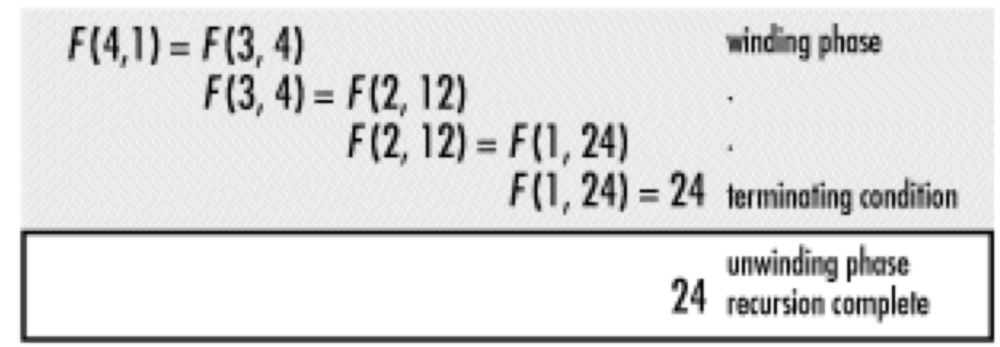
Basic recursion

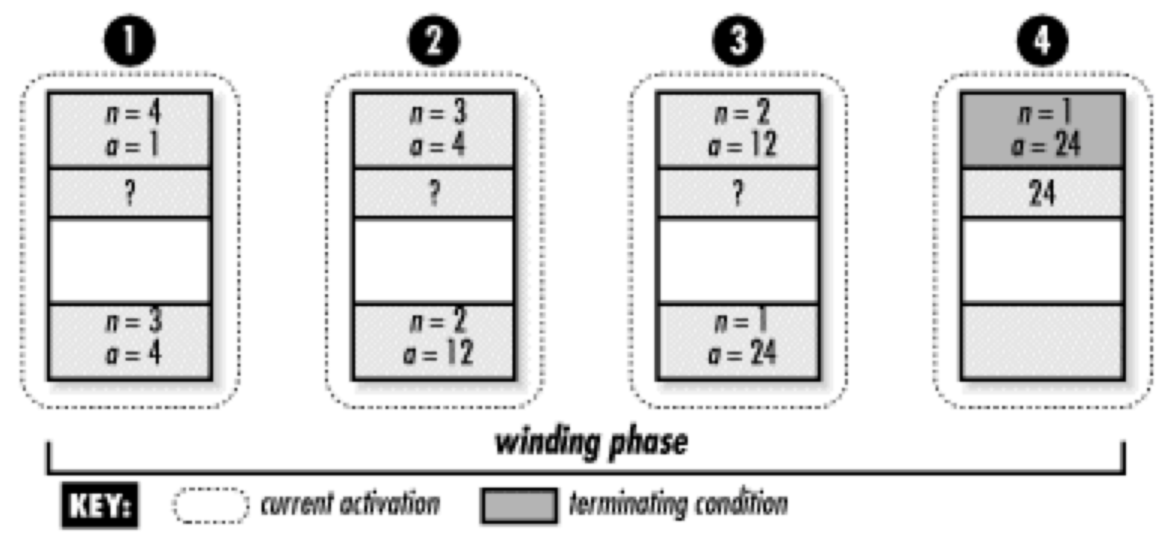
- The stack used in the basic recursion does have a few drawbacks. Maintaining information until it returns takes considerable amount of space.
- Generating and destroying activation records takes time because there is a significant amount of information must be saved and restored.
- Under concerning process time we may consider iterative method instead.
- Fortunately with have an alternative recursive approach. The tail recursion method.

Trail recursion

Homework: make program to compute the factorial function recursively using trail method

- A recursive function is said to be tail recursive if all recursive calls within it are tail recursive.
- A recursive call is tail recursive when it is the last statement that will be executed within the body of a function and its return value is not a part of an expression.
- Tail recursive function does not have the unwinding phase. Compilers take advantage of it. They overwrite the current activation record instead of pushing a new one onto the stack.





Factorial function redefined

$$F(n, a) = \begin{cases} a & \text{if } n = 0,\ n = 1 \\ F(n-1, na) & \text{if } n > 1 \end{cases}$$

Conclusions: Recursion

We have seen:

- Basic recursion functions.
- Tail recursion method.
- Computation of the factorial function as an example.

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

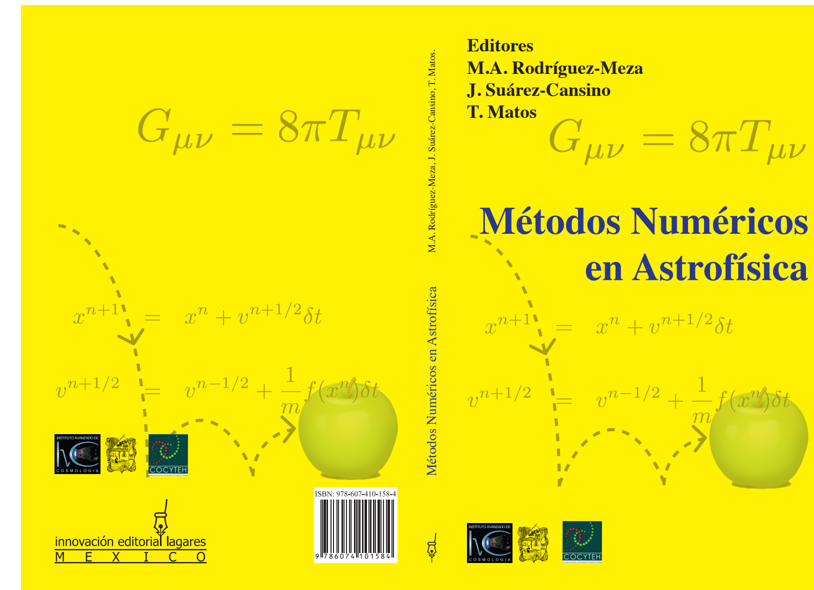
- Métodos numéricos en astrofísica, capítulo 1, Método de N-cuerpos en astrofísica. (https://www.researchgate.net/publication/316582859_Metodo_de_N-Cuerpos_en_Astrofisica)

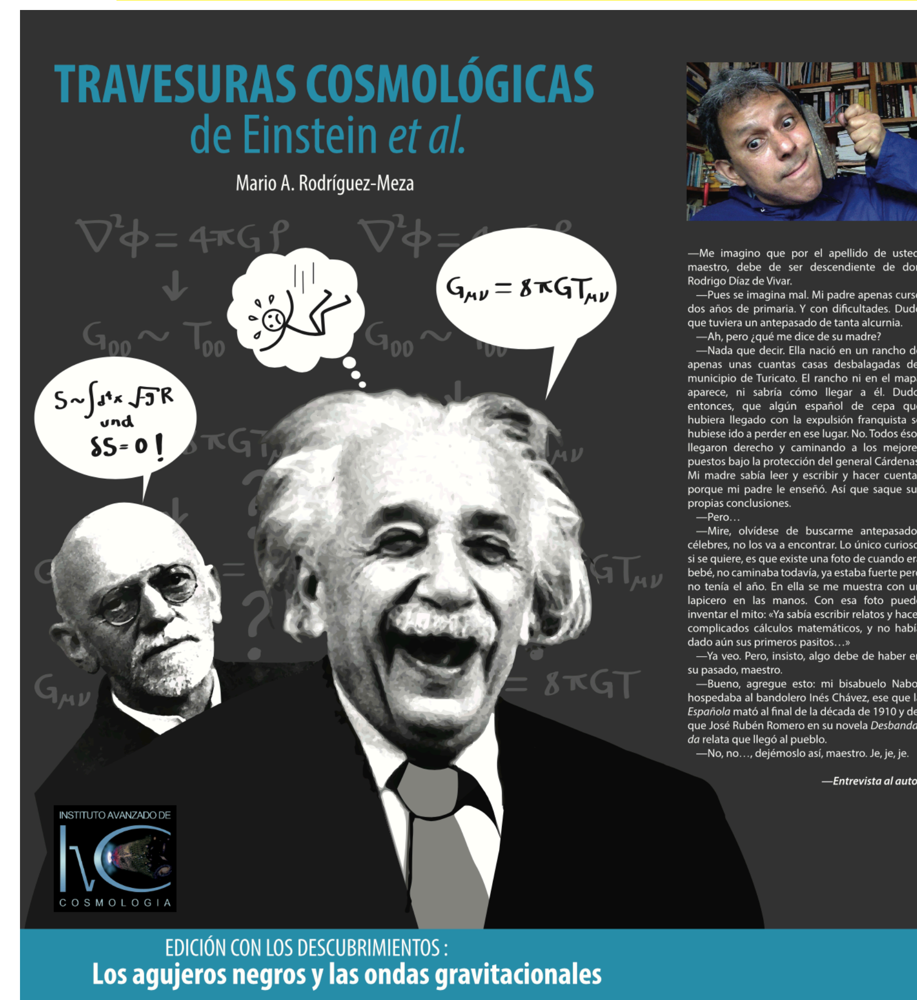
- La estructura a gran escala del universo. Capítulo 22 en Travesuras cosmológicas de Einstein et al. https://www.researchgate.net/publication/316582400_La_estructura_a_gran_escala_del_universo_simulaciones_numericas

- 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!