

Métodos estadísticos en Economía y Sociología

Jaime Pérez Aparicio

30 de abril de 2015

Introducción

En este proyecto he aplicado métodos de la física estadística a otros temas. Concretamente he usado el algoritmo de Metrópolis, basado en el método de Montecarlo. Voy a introducir ambos métodos, pero para mejor entendimiento es preferible consultar bibliografía especializada en el tema.

Montecarlo

En el método de Montecarlo se tiene inicialmente un sistema en cierto estado. Aleatoriamente se cambia el estado de uno de los elementos que conforman al sistema. A este cambio se le asocia una puntuación. Si la puntuación es positiva o una determinada, se cambia definitivamente el sistema. Si no, se vuelve al estado anterior.

Si se quisiera hacer una inteligencia virtual que jugara al tres en raya, el ordenador iría comprobando cada posible jugada y asignándole una puntuación, en función de la probabilidad que tenga esa jugada de ganar la partida. Luego escogera la que tenga más puntuación. La desventaja de este método es que tarda mucho tiempo en comprobar todos los posibles estados, lo que se puede paliar limitando el número de estados que prevé.

En este caso, el programa comprueba solo el siguiente estado, por lo que no se da el error mencionado.

Metrópolis

La principal diferencia es que con este método se permite al sistema ir a un estado con menor puntuación siguiendo una ley de probabilidad exponencial. Esto hace que el sistema tarde mucho más en alcanzar su estado estable.

Para este trabajo se ha usado el algoritmo de Metrópolis.

Código

Como he invertido bastante tiempo en él, comentaré un poco los problemas que me han surgido y como los he arreglado. También explicaré como he organizado el programa. Intentaré comentar el programa para que, entre esto y los comentarios se pueda entender lo que hace y cómo lo hace. De todas formas lo explicaré muy abstractamente, sin meterme mucho en el código. Para entenderlo mejor, ver las anotaciones que haré en él.

He creado una matriz de objetos abstractos (en el ejemplo uso texto, pero se pueden usar números o otros objetos definidos a mano). Estos objetos pueden representar espines, opiniones, patrones de comportamiento o cualquier cosa. El programa va barriendo estos elementos aleatoriamente, los cambia y comprueba si ese cambio baja o sube la energía. En función de ello, calcula una probabilidad para que el sistema permanezca en el nuevo estado o vuelva al anterior. Haciendo varias veces esos barridos, el sistema tenderá hacia la posición de equilibrio más cercana (que no tiene por qué ser el estado de menor energía total, si no local).

La "librería" (no es exactamente una librería, si no código que se incrusta en el original) permite obtener la energía del sistema y la proporción de cada estado. Iba a intentar hacer que el programa calculara la entropía del sistema. Pero no se me ocurría como hacerlo y tampoco he tenido mucho tiempo para pensarlo. He conseguido que el algoritmo consuma más o menos pocos recursos (ha pasado de ocupar más de 1GB de RAM a cientos de MB), pero siempre es optimizable. Se pueden usar objetos más ligeros en ciertos casos, como booleans en vez de cadenas de caracteres, lo que disminuye la memoria usada.

En el programa principal (main) creo una matriz de estados, la relleno aleatoriamente (con un método implementado en la "librería") y hago evolu-

cionar el sistema mientras obtengo la energía y la distribución a cada paso. Luego paso los datos a unos archivos externos y lo dibujo con gnuplot (el proceso es automático). Hay una forma más eficaz de hacerlo usando el concepto de las pipes (pero no me da tiempo a enterarme de como usarlo correctamente). Es simplemente una idea para la mejora del programa.

Se pueden variar los principales parámetros del método: la temperatura, la constante de Boltzman, la J (que aparece en el modelo de Ising) a la que he llamado "influencia primeros vecinos" por simplificar, el campo externo (o influencia externa) y como afecta esa imposición.

He usado c++ por cuestiones prácticas. Se programar en otros lenguajes (Matlab/Octave, Python, Java, PHP y Bash) pero creo que c++ es el más rápido y fácil de usar para el usuario final (siempre que se le de el programa ya compilado, claro). Lo único que hecho de menos en c++ es la facilidad de algunos de los otros de crear GUIs para manejarlo todo gráficamente. De todas formas la idea tampoco era hacer un programa comercial.

Simulaciones

Sistema de dos niveles (espines up y down)

Este modelo sería igual a uno normal de Metrópolis de 2 niveles. La diferencia sólo es la generalización de la librería.

Puede verse cómo la energía disminuye con los pasos y el sistema se decanta por una de las 2 magnetizaciones (para el caso $T=0$). Aumentando la T se ve como se recupera la simetría, por lo que la magnetización se anula (el número de espines hacia arriba y hacia abajo es más o menos el mismo). No he visto a que temperatura está la transición de fase porque me llevaría un par de días y no los tengo. De todas formas el método es sencillo. Es dejar que el sistema evolucione y al cabo de un rato ver la magnetización. Si se repite el proceso para varias temperaturas, se verá que en cierto momento la magnetización se hace depreciable. Había conseguido hacerlo, más o menos, pero el programa empezó a dar fallos así que tuve que volver a una versión anterior en la que aún no lo había programado.

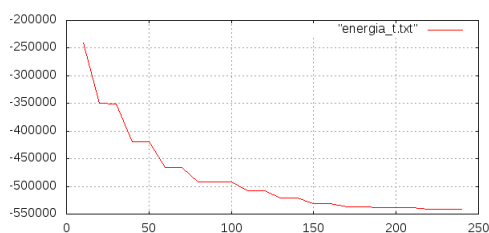


Figura 1: $E(t)$ con T nula

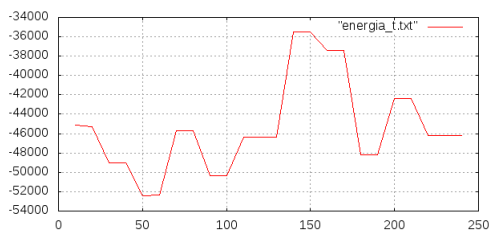


Figura 2: $E(t)$ con T elevada

Sistema de tres niveles (egoista, altruista, precavido)

Ahora ya hay que echarle un poco de imaginación al asunto. Suponiendo que hay tres tendencias en el comportamiento, cómo interactúan entre sí? Hay alguna que será más estable que las otras a lo largo del tiempo? En ese caso, se ha conseguido una estrategia evolutivamente estable (así se llaman en el libro del gen egoista). Hay concursos en los que se tiene que diseñar estrategias que ganen a las demás a lo largo del tiempo. Con este modelo se podría hacer algo parecido, aunque es muy sencillo hacer que una estrategia gane (sólo hay que aumentar un par de parámetros). En esos concursos las estrategias compiten entre sí.

En la figura con T nula, el sistema se ordena fácilmente. En la de T alta, al sistema le cuesta más ordenarse.

Puede verse que en la de T nula el sistema evoluciona hacia cierta distribución de los estados, mientras que a T elevada el sistema tiende a la equiprobabilidad de los estados (33 %).

En estos casos hay que predecir cómo interactuarán los estados entre sí. Para este caso he imaginado que un egoista junto a otro egoista no será productivo. Un egoista al lado de un altruista saldrá ganando, y el segundo perdiendo, lógicamente.

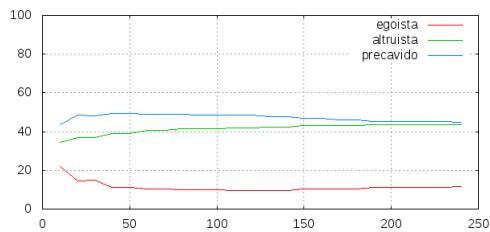


Figura 3: Proporciones para T nula

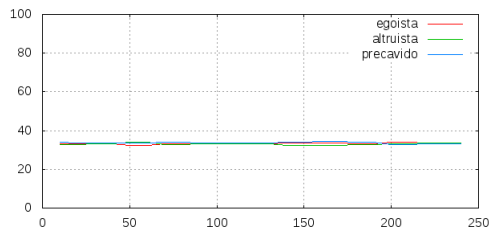


Figura 4: Proporciones para T alta

te. Altruista-altruista funcionará bien, pues saldrán ganando los dos. Luego he introducido al precavido, que lo que hace es no fiarse y dar solo si recibe algo, por lo que siempre gana. Junto al altruista ganan los dos, mientras que junto al egoista no pierde ninguno. Sería como una copia del vecino. Se puede ver que es la estrategia más estable. Su presencia impulsa a los altruistas. Sin ellos el comportamiento sería distinto, como se muestra en la otra figura. Se ve que la proporción se va a 50 %, más o menos. He mantenido el estado precavido porque era más fácil que volver a cambiar el programa a 2 estados. He hecho que desaparezca rápidamente con los factores.

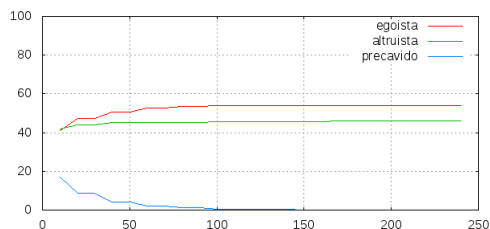


Figura 5: Sólo egoista y altruista. Temperatura nula.

Bibliografía

- **"El gen egoista"** de **Richard Dawkins**. Me he inspirado en él al realizar el ejemplo de egoistas, altruistas y precavidos. En su libro hay muchos más ejemplos interesantes de cómo evolucionan los patrones de comportamiento en la naturaleza.

- **"Programación con C++"** de **Al Stevens y Clayton Walnum, Ed Anaya**. Lo he usado como consulta para mirar las cosas que no recordaba de C++ y buscar información sobre como hacer otras que no conocía.