Ising Model
MVMISG
Métodos Numéricos Avanzados

Cristian Ontivero, Enzo Altamiranda, Mauricio Minestrelli, Valeria Serber

3 de octubre de 2014

Resumen

Se presentan los resultados que se obtuvieron al implementar diferentes algoritmos para calcular la matriz que tiene su origen en el Modelo de Ising, así como también, los autovalores de la misma, dado un tamaño de matriz. Se encontró que al momento de calcular la multiplicación matricial necesaria, es importante tener en cuenta tanto el algoritmo como la estructura de datos utilizada para almacenar los valores de modo que el método resulte eficiente a escala.

1. Palabras Clave

PALABRAS CLAVE

2. Introducción

El *Modelo de Ising* es un modelo físico utilizado en el estudio del comportamiento de materiales ferromagnéticos, propuesto por el físico *Wilhelm Lenz* en 1920 a su estudiante *Ernst Ising*, quién demostró que el modelo unidimensional no tenía transición de fase para su tesis en 1924. La matriz tratada en el presente trabajo tiene sus orígenes en este modelo, y surge de un producto de matrices que se explicará más adelante.

En este informe se describen las distintas instancias sucedidas en el desarrollo de un programa que calcula la matriz del *Modelo de Ising*, junto con sus autovalores. Se mencionan los algoritmos elegidos y cómo se fue optimizando los mismos, así como también las estructuras utilizadas, con el objetivo de lograr una menor complejidad en el código y una mayor rapidez para generar resultados. También se muestran las pruebas realizadas y gráficos que comparan las distintas iteraciones del desarrollo.

3. Metodología

3.1. Cálculo de la matriz del Modelo de Ising

La matriz del Modelo de Ising, llamada $\boldsymbol{A}$ a partir de ahora, puede calcularse como el producto de dos matrices $\boldsymbol{K}$ y $\boldsymbol{L}$, ambas de dimensión *2m* x *2m*, con m perteneciente a los naturales.

$$A = KL \tag{1}$$

donde

$$K = \begin{pmatrix} E & & & & \\ & E & & & \\ & & \ddots & & \\ & & & E & \\ & & & & E \end{pmatrix}, E = \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix} \tag{2}$$

$$L = \begin{pmatrix} \cos\beta & & & & -\sin\beta \\ & F & & & \\ & & \ddots & & \\ & & & F & \\ \sin\beta & & & & \cos\beta \end{pmatrix}, F = \begin{pmatrix} \cos\beta & \sin\beta \\ -\sin\beta & \cos\beta \end{pmatrix} \tag{3}$$

3.1.1. Primera Iteración

En la primera iteración se optó por almacenar cada matriz en un arreglo de arreglos. Se comenzó realizando una implementación del algoritmo estándar de multiplicación de matrices, debido a la simpleza del código correspondiente. Se probó esta versión creando las matrices K y L y multiplicándolas. El orden temporal de este algoritmo es cúbico, es decir, $O(n^3)$. Debido a esto, al calcular matrices pequeñas el algoritmo termina de forma rápida, sin embargo, al probar matrices de mayor tamaño, el tiempo crece en gran medida.

3.1.2. Segunda Iteración

Debido a los resultados anteriores, se hizo evidente que se debía mejorar el programa para disminuir su complejidad. Para ello, se utilizó el hecho de que las matrices K y L son **ralas**, es decir, matrices de gran tamaño, en la que la mayoría de los elementos son cero. Para optimizar el algoritmo, se optó por cambiar la estructura de datos que almacena las matrices, de modo que los ceros no se almacenen, y el algoritmo pueda saltear las operaciones que den como resultado cero. La estructura utilizada en esta iteración se conoce como *"Compressed column storage" (CCS)*, o alternativamente *"Compressed sparse column"*, y es la representación tradicional utilizada en *MATLAB* al usar la función *"sparse"*.

Esta representación cuenta con tres arreglos. El primero, al cual llamaremos ***values***, contiene los valores no nulos de la matriz, de tamaño ***nnz*** (del inglés *"number of nonzeros"*). El segundo, $\boldsymbol{r_i}$, indica el índice de la fila del elemento que se encuentra en el primer arreglo y en la misma posición. Dicho arreglo también tiene tamaño *nnz*. El tercero, ***cp***, tiene como tamaño la cantidad de columnas más uno, donde en la posición j del arreglo se guarda el índice en el arreglo de valores, en el que se encuentra el primer elemento no nulo de la columna j. El último elemento de *cp* es el valor *nnz*. Si alguna columna tuviera todos ceros, en el índice de esa columna se coloca el mismo valor que en la próxima columna. Utilizando el arreglo *cp* se puede conocer la cantidad de elementos no nulos presentes en cada columna j,

para ello basta con calcular la resta: *cp[j+1] - cp[j]*.

Ejemplo de almacenamiento de una matriz utilizando Compressed column storage

$$M = \begin{pmatrix} 0 & 3 & 0 & 5 & 7 \\ 0 & 0 & 0 & 3 & 8 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 5 \end{pmatrix}$$

Notar que la tercer columna no tiene valores. Esto se representa usando el mismo valor en *cp* que en la próxima columna. Esto es consistente con el hecho de que la cantidad de elementos no nulos presentes en la columna j se puede conocer con la resta

cp[j+1] - cp[j].

$$values = \left[\begin{array}{cccccccc} 1, & 3, & 2, & 5, & 3, & 7, & 8, & 5 \end{array} \right]$$

$$ri = \left[\begin{array}{cccccccc} 2, & 0, & 3, & 0, & 1, & 0, & 1, & 3 \end{array} \right]$$

$$cp = \left[\begin{array}{cccccc} 0, & 1, & 3, & 3, & 5, & 8 \end{array} \right]$$

Complejidad espacial de la representación CCS

Para una **matriz densa**, es decir, que no tiene elementos nulos, de m x n, se necesitan m punteros en memoria, cada uno apuntando a los m arreglos de n elementos. En comparación con una matriz en representación *CCS*, se necesita espacio para *nnz·sizeof(elems) + (nnz + cols)·sizeof(índices)*. Para una aproximación más fácil de entender, si suponemos que el tamaño de índices, elementos, y punteros en memoria usan todos a misma cantidad de bytes, se tiene que la complejidad espacial de la matriz densa es $m+m·n$, es decir, $O(m·n)$. Como en el caso del Modelo de Ising las matrices son cuadradas, queda de orden espacial $O(n^2)$. En cambio, la complejidad espacial de la **matriz rala** es $2·nnz+n$, es decir orden $O(nnz+n)$, que es *lineal*.

Algoritmo de multiplicación CCS

Al utilizar la representación *CCS*, se logró un algoritmo de multiplicación significativamente más eficiente, ideal para matrices ralas, ya que se pudo evitar realizar todos los productos con elementos nulos. A continuación se observa en pseudocódigo, el algoritmo implementado para multiplicar utilizando la estructura *CCS*, que difiere del algoritmo estándar utilizado anteriormente. Cabe destacar que si este algoritmo fuera utilizado con matrices densas en vez de ralas, no sería más eficiente que el estándar.

```
CCSMatrix matriz = nueva matriz de m x n;

Por cada columna de la 2da matriz {
    Si la columna no tiene elementos entonces saltearla;

        Por cada fila de la 1ra matriz {

                currentVal = suma de los productos de los valores no nulos de la
                    columna actual, con los correspondientes de la fila actual;

                Si currentVal es cero entonces saltear la fila;

                Si aun no se registro el valor que comienza la columna actual {

                        Registrar;

                        Si habia columnas salteadas por no tener elementos entonces
                            propagar el valor actual hacia atras;
                }

                Almacenar el valor no nulo actual;

                Almacenar el indice de la fila del valor no nulo actual;
        }
}

Almacenar el numero de valores no nulos en la matriz;

Almacenar el numero de valores no nulos en la posicion final del arreglo de
    punteros de columnas;

Retornar matriz;
```

Algoritmo de multiplicación CCS: Se muestra en pseudocódigo el algoritmo que calcula la multiplicación de las matrices K y L aprovechando la estructura de datos CCS.

4. Resultados

4.1. Comparación entre ambas iteraciones

Se realizaron pruebas en ambas iteraciones para descubrir las diferencias entre ellas. Se utilizaron matrices de diferentes tamaños, y se utilizó el comando *"time"* de *Linux* para medir el tiempo de cada caso. A continuación se muestra una tabla con los valores que se utilizaron en las pruebas, y luego los gráficos que se realizaron a partir de ellas.

Multiplicación estándar		Multiplicación CCS	
m	**tiempo (seg)**	**m**	**tiempo (seg)**
100	0.02	100	0
150	0.05	150	0
200	0.13	200	0
250	0.25	250	0.01
300	1.32	300	0.01
350	2.88	350	0.01
400	4.24	400	0.01
450	6.51	450	0.02
500	8.90	500	0.02
550	12.05	550	0.02
600	15.81	600	0.03
650	20.22	650	0.03
700	25.25	700	0.04
750	32.36	750	0.04
800	36.33	800	0.05
850	45.21	850	0.06
900	46.49	900	0.06
950	54.95	950	0.06
1000	62.82	1000	0.07
		10000	3.19
		20000	7.14
		25000	19.59
		30000	28.22
		35000	39.30
		40000	52.50
		45000	63.79

Tabla 1: Muestra el tiempo en segundos, que tardaron los dos algoritmos implementados respectivamente, a partir de distintos valores de m.

4.1.1. Primera iteración

En el siguiente gráfico se representa la complejidad del algoritmo de multiplicación estándar de matrices. El eje x corresponde al valor de m, y el eje y al *tiempo* en segundos, que tarda el algoritmo en resolver.

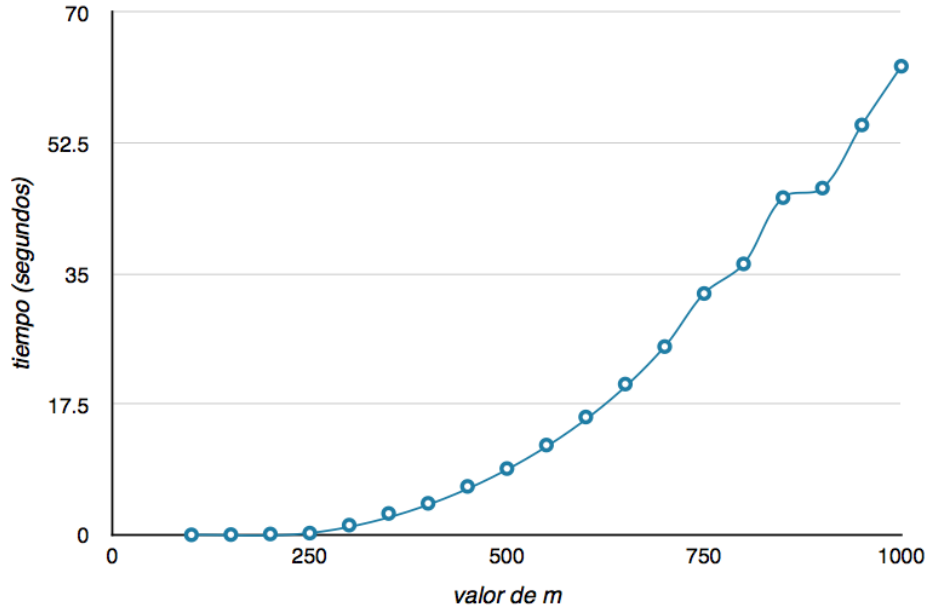

Figura 1: Multiplicación estándar de matrices

4.1.2. Segunda iteración

En el siguiente gráfico se representa la complejidad del algoritmo de multiplicación de matrices utilizando la estructura de datos *CCS*. Los ejes representan lo mismo que el gráfico anterior.

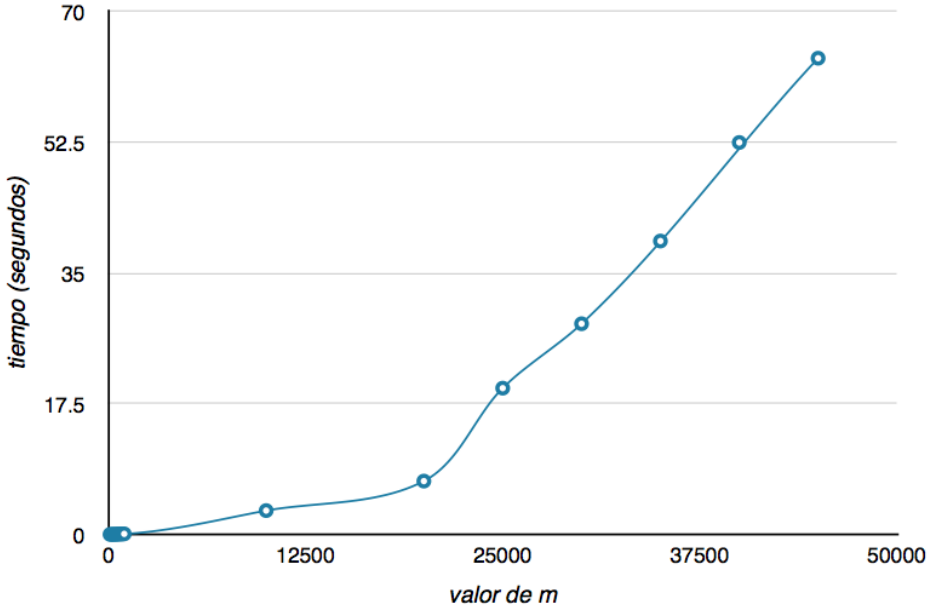

Figura 2: Multiplicación CCS

4.2. Descripción de los resultados

El resultado más importante que se puede obtener de los gráficos anteriores es que en el intervalo de 0 a 70 segundos, el primer algoritmo puede resolver hasta matrices de $m = 1000$, mientras que el segundo puede resolver hasta matrices de $m = 500000$ aproximadamente. Además, se puede observar que la curva del primer algoritmo crece mucho más rápido que la segunda. Esto puede corresponderse con el hecho de que el primer algoritmo es de $O(n^3)$, mientras que el otro es de orden menor.

5. Conclusiones

5.1. Cálculo de A

Si bien se logró optimizar bastante el algoritmo que calcula la matriz A, mientras se realizaban pruebas se pudo observar un patrón en la construcción de A. Se notó que a partir de $m = 3$, la matriz A puede escribirse genéricamente, ya que pequeños bloques de elementos se repiten a lo largo de la estructura, aumentando la cantidad de bloques lógicamente mientras m aumenta. Por lo tanto, se podría evitar tener que calcular A a partir del producto entre K y L, y en vez de eso, seguir la siguiente regla para representarla:

Cuando $m = 1$,

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{4}$$

Cuando $m = 2$,

$$A = \begin{pmatrix} T & Q & R & S \\ R & S & T & Q \end{pmatrix} \tag{5}$$

Cuando $m = 3$, la matriz comienza a expandirse dejando ceros en varias posiciones. Además puede verse que el bloque del medio compuesto por Q, R, S y T comienza a repetirse.

$$A = \begin{pmatrix} T & Q & R & & & S \\ & S & T & Q & R & \\ R & & & S & T & Q \end{pmatrix} \tag{6}$$

Entonces, para $m >= 3$, la matriz puede escribirse de forma genérica de la siguiente manera:

$$A = \begin{pmatrix} T & Q & R & & & & & & & S \\ & S & T & Q & R & & & & & \\ & & & S & T & \ddots & & & & \\ & & & & & \ddots & & & & \\ & & & & & & Q & R & & \\ & & & & & & S & T & Q & R & \\ R & & & & & & & S & T & Q \end{pmatrix} \tag{7}$$

Siendo

$$Q = \begin{pmatrix} \sin\alpha & \cos\beta \\ \cos\alpha & \cos\beta \end{pmatrix} \tag{8}$$

$$R = \begin{pmatrix} \sin\alpha & \sin\beta \\ \cos\alpha & \sin\beta \end{pmatrix} \tag{9}$$

$$S = \begin{pmatrix} -\cos\alpha & \sin\beta \\ \sin\alpha & \sin\beta \end{pmatrix} \tag{10}$$

$$T = \begin{pmatrix} \cos\alpha & \cos\beta \\ -\sin\alpha & \cos\beta \end{pmatrix} \tag{11}$$

6. Bibliografía