UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE CIENCIAS
DEPARTAMENTO DE MATEMÁTICA Y CIENCIA DE LA COMPUTACIÓN
LICENCIATURA EN CIENCIA DE LA COMPUTACIÓN
Profesor(es): Nicolas Thériault
Primer Semestre de 2021

Programación Avanzada – Laboratorios 1, 2 y 3

1. Objetivos

Objetivo principal: Implementar algoritmos para resolver sistemas lineales de gran tamaño.

Objetivos secundarios:

- Implementar algoritmos clásicos de eliminación gausiana (triangular superior o completa).
- Resolver sistemas lineales a través del cálculo de la matriz inversa.
- Obtener una implementación eficaz aplicables a sistemas lineales grandes.
- Implementar del algoritmo clásico de multiplicación matricial.
- Implementar el algoritmo de multiplicación matricial de Strassen.
- Obtener una implementación eficaz aplicables a sistemas lineales grandes.
- Implementar el algoritmo Dividir-y-Conquistar de inverso matricial.
- Obtener una implementación eficaz aplicables a sistemas lineales grandes.
- Resolver sistemas lineales grandes de forma más eficiente.

Se trabajará en grupos de 2 o 3 estudiantes, entregando las respuestas (con desarrollo) de la parte teórica en formato pdf o jpg, y el programa (parte práctica) en un archivo .c.

2. Programación

La programación debe ser en C (no en C++), utilizando a lo más las librerías estándares siguientes:

- stdio.h
- stdlib.h
- math.h
- time.h

Además, para evitar cualquier problemas de compatibilidad, se recomienda utilizar solamente la función *clock()* para la medición de tiempo.

El álgebra lineal se trabajará módulo $p = 37819961384332681 99$ (aritmética exacta) para evitar temas de redondeo. Las funciones de aritmética básica (suma, resta, productos y inverso modular) están disponibles en el archivo "formato_lab1.c" (disponible en la página web del curso).

En general se recomienda utilizar el generador aleatorio *arc4random* (con mejores propiedades aleatorias), pero se puede utilizar *rand* en versiones más antiguas del compilador C.

Observación: trabajando módulo $p = 37819961384332681 99$, la probabilidad que una matriz cuadrada $n \times n$ con entradas aleatorias no sea invertible es $\approx 2^{-62}$.

3. Eliminación Gaussiana (laboratorio 1)

Para el laboratorio 1, se consideran solamente técnicas de simplificación (y resolución) de sistemas lineales basados en *operaciones de filas* (o lineas, renglones, etc.)

Por razones historicas y temas de traducción, las diferentes técnicas de simplificación de sistemas lineales pueden llamarse de diferentes forma según la fuente de información utilizada. Para evitar confusiones, utilizaremos la siguiente nomenclatura

- Se llaman *pivote* los valores distintos de 0 lo más a la izquierda en una fila.
- En un sistema reducido, hay a lo más un pivote por columna.
- Consideramos la *eliminación Gaussian* como el proceso de simplificación de sistemas lineales hasta que los pivotes (que van bajando de linea desde la izquierda a la derecha) tienen valor 1 y todos los otros valores en sus columnas son 0.
- En la *eliminación triangular superior*, los pivotes (que van bajando de linea desde la izquierda a la derecha) tienen valor 1 pero solamente los valores en filas debajo de los pivote deben tener valor 0.
- La *sustitución reversa* resuelve el sistema desde la forma triangular superior, empezando con las últimas variables.
- Para calcular el inverso de una matriz A, se empieza con el sistema aumentado $[A|I_n]$ y se hace la eliminación Gaussiana hasta obtener un sistema $[I_n|B]$, donde $B = A^{-1}$.

4. Multipicación de matrices

El algoritmo de Strassen es una forma alternativa de multiplicación de matrices, más eficiente que la formula clásica a grande escala. Permite obtener un algoritmo de tipo Dividir-y-Conquistar: Separa los multiplicantes A y B en 4 submatrices cada una ($A_{11}, A_{12}, A_{21}, A_{22}$ y $B_{11}, B_{12}, B_{21}, B_{22}$) para luego hacer 7 multiplicaciones de submatrices (en vez de 8 con una formula "clásica") y finalmente recombinarlas en la matriz producto.

No es necesario que las matrices sean cuadradas, pero las dimensiones de A y B deben ser compatibles (es decir, el número de columnas de A es igual al número de filas de B) y se suponen pares dado que todas las submatrices de A deben tener las mismas dimensiones, y todas las submatrices de B deben tener las mismas dimensiones (en caso contrario, el algoritmo no se podría aplicar por las sumas de submatrices).

Indicación: Para la implementación, pueden suponer que el algoritmo de Strassen se aplica solamente a matrices cuadradas.

Notación: Supondremos que M_{11} es el cuarto superior izquierdo de la matriz M, M_{12} es el cuarto superior derecho, M_{21} es el cuarto inferior izquierdo, y M_{22} es el cuarto inferior derecho.

Podemos separar el algoritmo de multiplicación de Strassen en tres fases:

Preparación:

$$\begin{aligned} S_1 &= A_{11} + A_{22} & T_1 &= B_{11} + B_{22} \\ S_2 &= A_{21} + A_{22} & T_2 &= B_{21} + B_{22} \\ S_3 &= A_{11} + A_{12} & T_3 &= B_{11} + B_{12} \\ S_4 &= A_{21} - A_{11} & T_4 &= B_{21} - B_{11} \\ S_5 &= A_{12} - A_{22} & T_5 &= B_{12} - B_{22} \end{aligned}$$

Multiplicaciones:

$$\begin{aligned} M_1 &= S_1 \cdot T_1 \\ M_2 &= S_2 \cdot B_{11} \\ M_3 &= A_{11} \cdot T_5 \\ M_4 &= A_{22} \cdot T_4 \\ M_5 &= S_3 \cdot B_{22} \\ M_6 &= S_4 \cdot T_3 \\ M_7 &= S_5 \cdot T_2 \end{aligned}$$

Recombinación:

$$\begin{aligned} C_{11} &= M_1 + M_4 - M_5 + M_7 \\ C_{12} &= M_3 + M_5 \\ C_{21} &= M_2 + M_4 \\ C_{22} &= M_1 - M_2 + M_3 + M_6 \end{aligned}$$

Observación: Es posible cambiar el orden de varias operaciones en la formula de Strassen sin realmente cambiar la formula (algebraica). La separación en fases "preparación", "multiplicaciones" y "recombinación" es solamente para facilitar la visualización del algoritmo.

5. Inverso de matrices

Se estudia una forma alternativa de calculo de inverso de matrices, más eficiente que la formula clásica a grande escala. Permite obtener un algoritmo de tipo Dividir-y-Conquistar: Separa la matriz A en 4 submatrices $(A_{11}, A_{12}, A_{21}, A_{22})$ para luego hacer 2 inversos y 6 productos de submatrices (en vez de utilizar la eleimnación Gaussiana) y finalmente recombinarlas en la matriz inversa.

Para ser más eficiente, este algoritmo de inverso matricial requiere la implementación de multiplicación matricial optimizada. Como se trata de matrices inversas, es necesario que las matrices sean cuadradas pero la dimensión (cantidad de filas/columnas) puede ser impar.

Notación: Supondremos que M_{11} es el cuarto superior izquierdo de la matriz M, M_{12} es el cuarto superior derecho, M_{21} es el cuarto inferior izquierdo, y M_{22} es el cuarto inferior derecho.

Suponemos que A_{11} es invertible, y también lo es $A_{22} - A_{21}A_{11}^{-1}A_{12}$.
El inverso de la matriz A se calcula como:

$$A^{-1} = \begin{pmatrix} A_{11}^{-1} + A_{11}^{-1}A_{12}(A_{22} - A_{21}A_{11}^{-1}A_{12})^{-1}A_{21}A_{11}^{-1} & -A_{11}^{-1}A_{12}(A_{22} - A_{21}A_{11}^{-1}A_{12})^{-1} \\ -(A_{22} - A_{21}A_{11}^{-1}A_{12})^{-1}A_{21}A_{11}^{-1} & (A_{22} - A_{21}A_{11}^{-1}A_{12})^{-1} \end{pmatrix}$$

Observación: Es posible cambiar el orden de varias operaciones en la formula de inverso sin realmente cambiar la formula (algebraica). Incluso, se pueden agrupar las multiplicaciones de más de una manera para llegar al mínimo de 6 productos de submatrices.

6. Requerimientos, Laboratorio 1

El programa deberá ser capaz de trabajar con sistemas lineales de gran tamaño:

- Debe poder trabajar sistemas con un mínimo de 10^8 entradas (por ejemplo, con 10000 filas y 10000 columnas).
- Puede utilizar las funciones *malloc* y/o *calloc* para asignar arreglos grandes, pero estos no deberán ser de más de 10^5 espacios de memoria consecutivos.

Como mínimo, crear funciones para:

1. Generar un sistema lineal al azar del tamaño entregado por el usuario / la usuaria;
2. Leer archivos conteniendo un sistema lineal, donde la primera entrada es el número n de filas (int), la segunda entrada es el número m de columnas (int), luego de cuales las nm entradas de la matriz (aumentada) se entregan por filas, separados por un espacio o un cambio de linea;
3. Resolver el sistema lineal por eliminación Gaussiana hasta un sistema en forma escalonada reducida (eliminación Gaussiana completa);

4. Resolver el sistema por eliminación triangular superior (que elimna valores debajo del pivote pero no arriba) y substitución reversa;

5. Calcular el inverso de una matriz cuadrada (por eliminación Gaussiana) y utilizar este para resolver el sistema asociado.

Se podrán definir otras funciones (funciones auxiliares, etc) si deseado.
Además, para facilitar el testeo, se recomienda tener funciones para escribir el sistema y/o su solución cuando el tamaño del sistema es pequeño (hasta 6×6).

El programa debe utilizar un sistema de menú para el usuario / la usuaria, y debe liberar la memoria antes de cerrar el proceso. El menú debe permitir medir los tiempos de trabajos para sistemas grandes (sin mostrar el resultado del sistema). El trabajo para generar un sistema aleatorio o leer el sistema desde un archivo no se debe incluir en este tiempo.

7. Requerimientos, Laboratorio 2

El programa deberá ser capaz de trabajar con sistemas lineales de gran tamaño:

- Debe poder trabajar sistemas con un mínimo de 10^8 entradas (por ejemplo, con 10000 filas y 10000 columnas).
- Puede utilizar las funciones *malloc* y/o *calloc* para asignar arreglos grandes, pero estos no deberán ser de más de 10^5 espacios de memoria consecutivos.

Como mínimo, crear funciones para:

1. Generar una matriz al azar del tamaño entregado por el usuario / la usuaria;

2. Leer archivos conteniendo una matriz, donde la primera entrada es el número n de filas (int), la segunda entrada es el número m de columnas (int), luego de cuales las nm entradas de la matriz se entregan por filas, separados por un espacio o un cambio de linea;

3. Multliplicar matrices por el algoritmo clásico;

4. Multliplicar matrices por el algoritmo de Strassen recursivo (hasta un tamaño dado);

Se podrán definir otras funciones (funciones auxiliares, etc) si deseado. Además se recomienda tener funciones para escribir la matriz cuando el tamaño del sistema es pequeño (hasta 10×10).

El programa debe utilizar un sistema de menú para el usurio / la usuaria, y debe liberar la memoria antes de cerrar el proceso. El menú debe permitir medir los tiempos de trabajos para sistemas grandes (sin mostrar el resultado del sistema). El trabajo para generar un sistema aleatorio o leer el sistema desde un archivo no se debe incluir en el tiempo de trabajo.

8. Requerimientos, Laboratorio 3

El programa deberá ser capaz de trabajar con sistemas lineales de gran tamaño:

- Debe poder trabajar sistemas con un mínimo de 10^6 entradas (por ejemplo, con 1000 filas y 1000 columnas).
- Puede utilizar las funciones *malloc*, *calloc* y/o *realloc* para asignar arreglos grandes, pero estos no deberán ser de más de 10^5 espacios de memoria consecutivos.

Como mínimo, crear funciones para:

1. Generar una matriz al azar del tamaño entregado por el usuario / la usuaria;
2. Calcular el inverso de matrices lo más rápidamente posible;

Se podrán definir otras funciones (funciones auxiliares, etc) si deseado. Además se recomienda tener funciones para escribir la matriz cuando el tamaño del sistema es pequeño (hasta 10×10).

El programa debe utilizar un sistema de menú para el usurio / la usuaria, y debe liberar la memoria antes de cerrar el proceso. El menú debe permitir medir los tiempos de trabajos para sistemas grandes (sin mostrar el resultado del sistema). El trabajo para generar un sistema aleatorio o leer el sistema desde un archivo no se debe incluir en el tiempo de trabajo.

9. Evaluación de la programación

La nota del laboratorio se calculará según la ponderación siguiente:

- Redacción [**10 %**]
  El programa está escrito de forma que puede ser leído y/o re-utilizado fácilmente por otros programadores: la redacción es limpia (con espacios y divisiones claras) y bien documentada, las sub-funciones y las variables tienen nombres naturales (que indican a que sirven) o van acompañadas de comentarios aclarando a que sirven.

- Ejecución [**15 %**]
  El programa compila sin advertencias. El ejecutable devuelve resultados correctos.
  Si el compilador no produce un ejecutable (por errores de compilación), la nota será 1.

- Eficiencia [**75 %**]
  Los tiempos de calculo de la multiplicación iterativa completa comparados con una implementación "básica" hecha por el profesor que satisface lo solicitado.
  Rangos de notas para la eficiencia:

- 1: no produce un ejecutable o no devuelve resultados correctos
- 1.1–2.4: más de 20 veces más lenta
- 2.5–3.9: 5 a 20 veces más lenta
- 4–5.4: 2 a 5 veces más lenta
- 5.5–6.9: a lo más 2 veces más lenta
- 7: tiempos parecidos o mejores