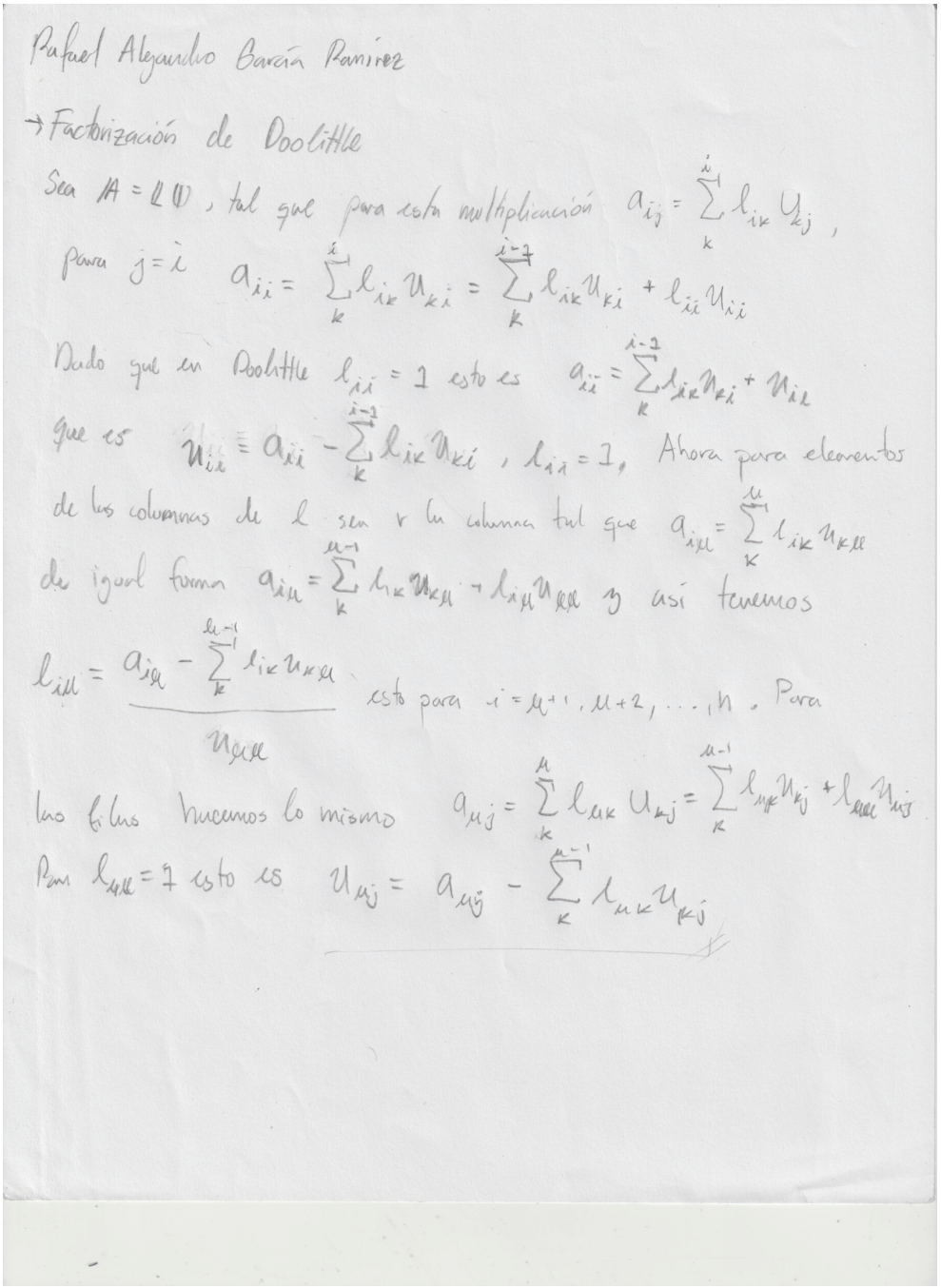
Tarea 4: Métodos Numéricos

Rafael Alejandro García Ramírez

17 de septiembre de 2023

1. Entregar a mano método de Factorización LU - Doolittle (poner 1 en la diagonal de la triangular inferior).

   ■ A continuación se presenta la deducción del método de Doolitttle:



   Este método fue programado y añadido dentro de un menú de opciones del archivo `main.c` (cuyas instrucciones de ejecución se encuentra al final de este reporte). Fundamentalemente el código utiliza para este método la siguiente función:

```c
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <math.h>

void doolittle(int N, double **matriz, double **L, double **U) {
    /* Realiza la descomposicion de Doolittle de una matriz A en las matrices triangular inferior L y triangular superior U.
    *
    * @ N       El tamano de la matriz A y las matrices L y U (N x N).
    * @ matriz  La matriz de entrada A que se descompone.
    * @ L       La matriz triangular inferior L resultante.
    * @ U       La matriz triangular superior U resultante.
    *
    * Esta funcion implementa el metodo de descomposici n de Doolittle para factorizar una matriz A en L y U
    * de manera que A = L * U. Es importante que la matriz A sea no singular y que su elemento diagonal superior no sea cero.
    * Si A no satisface estas condiciones, la funcion imprimira un mensaje de error con informacion de la iteracion de error
    * y terminara el programa.
    */

    if (fabs(matriz[0][0]) <= 1e-16) {
        printf("Error: matriz imposible. Error Doolittle 0: A[0][0] = 0\n");
        exit(1);
    }

    for (int d = 0; d < N; d++) for (int d2 = 0; d2 < N; d2++) L[d][d2] = U[d][d2] = 0;
    for (int i = 0; i < N; i++) L[i][i] = 1.0;

    U[0][0] = matriz[0][0];

    for (int j = 0; j < N; j++) {
        U[0][j] = matriz[0][j] / L[0][0];
        L[j][0] = matriz[j][0] / U[0][0];
    }

    double acumulador, acumulador1, acumulador2, acumulador3;
    for (int k = 0; k < N - 1; k++) {
        acumulador = 0;
        for (int x = 0; x < k; x++) {
            acumulador += L[k][x] * U[x][k];
        }
        U[k][k] = matriz[k][k] - acumulador;

        if (fabs(U[k][k] * L[k][k]) <= 1e-16) {
            printf("Error: matriz imposible. Error Doolittle 1: U[%d][%d]*L[%d][%d] = 0\n", k, k, k, k);
            exit(1);
        }

        for (int x = k + 1; x < N; x++) {
            acumulador1 = acumulador2 = 0;

            for (int n = 0; n < k; n++) {
                acumulador1 += L[k][n] * U[n][x];
                acumulador2 += L[x][n] * U[n][k];
            }

            U[k][x] = matriz[k][x] - acumulador1;
            L[x][k] = (matriz[x][k] - acumulador2) / U[k][k];
        }

        acumulador3 = 0;
        for (int n = 0; n < N - 1; n++) acumulador3 += L[N - 1][n] * U[n][N - 1];
        U[N - 1][N - 1] = matriz[N - 1][N - 1] - acumulador3;

        if (fabs(L[N - 1][N - 1] * U[N - 1][N - 1]) <= 1e-16) {
            printf("Error: matriz imposible. Error Doolittle 2: L[%d][%d]*U[%d][%d] = 0\n", N - 1, N - 1, N - 1, N - 1);
            exit(1);
        }
    }
}
```

2. Programa de método de Factorización LU - Crout.

   ■ El método factorización de Crout también viene incorporado en el menú de ejecución de `main.c`. La parte principal de este método consiste en la función:

```c
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <math.h>

void crout(int N, double **matriz, double **L, double **U) {
    /* Descompone una matriz A en las matrices L y U usando el metodo de Crout.
     *
     * @ N       El tamano de la matriz A y las matrices L y U.
     * @ matriz  La matriz de entrada A de tamano N x N.
     * @ L       La matriz triangular inferior L de tama o N x N.
     * @ U       La matriz triangular superior U de tama o N x N.
     *
     * Esta funcion implementa el metodo de descomposicion de Crout para factorizar una matriz A en L y U
     * de manera que A = L * U. Es importante que la matriz A sea no singular y que su elemento diagonal superior no sea cero.
     * Si A no satisface estas condiciones, la funcion imprimira un mensaje de error con informacion de la iteracion de error
     * y terminara el programa.
     */

    if (fabs(matriz[0][0]) <= 1e-16) {
        printf("Error: matriz imposible. Error Crout 0: A[0][0] = 0\n");
        exit(1);
    }
    for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) U[i][j] = L[i][j] = 0;

    for (int i = 0; i < N; i++) U[i][i] = 1.0;

    for (int i = 0; i < N; i++) {
        for (int j = i; j < N; j++) {
            double acumulador = 0;
            for (int k = 0; k < i; k++) {
                acumulador += L[j][k] * U[k][i];
            }
            L[j][i] = matriz[j][i] - acumulador;
        }
        if (fabs(L[i][i]) <= 1e-16) {
            printf("Error: matriz imposible. Error Crout 1: L[%d][%d] = 0\n", i, i);
            exit(1);
        }

        for (int j = i + 1; j < N; j++) {
            double acumulador = 0;
            for (int k = 0; k < i; k++) {
                acumulador += L[i][k] * U[k][j];
            }
            U[i][j] = (matriz[i][j] - acumulador) / L[i][i];
        }
    }
}
```

Falla del algoritmo en las matrices dadas para 1 y 2

♦ Cuando se realiza la prueba para las matrices `SmallMatrix.txt` y `BigMatrix.txt` y los vectores `SmallVector.txt` y `BigVector.txt` en ambos casos de llega a una falla de distinto tipo. Como se puede ver en los algoritmos para Doolittle, Crout y (más abajo) de Cholesky, se tienen establecidos diferentes errores para cada distinto caso; en la tabla siguiente se muestra un resumen para cada método según el conjunto de datos:

Datos	Método Doolittle	Método Crout	Método Cholesky
Small	`Error 1: U[1][1]*L[1][1] = 0`	`Error 1: L[1][1] = 0`	Error: Matriz no simétrica
Big	`Error de comprobación: Ax - b = 0`	`Error de comprobación: Ax - b = 0`	`Error de comprobación: Ax - b = 0`

Podemos notar que para el cálculo de Doolittle uno de los elementos de la diagonal se hace cero, en particular el elemento `U[1][1]` dado que `L[1][0] = 0.500000` y`U[0][1] = 4.425232` mientras que `A[1][1] = 2.212616`, es decir que `A[1][1] = L[1][0]*U[0][1]`. Más formalmente esto se debe a que si miramos la submatriz 2×2 de la esquina superior izquierda

$$A \supset A^* = \begin{bmatrix} 2.402822 & 4.425232 \\ 1.201411 & 2.212616 \end{bmatrix}$$

Podemos notar que la fila 1 es un múltiplo entero de la fila 2, y en general toda la fila 1 de la matriz original es un múltiplo entero de la fila 2 (más precisamente el doble), Por lo que realmente contamos con 3 ecuaciones para 4 variables, lo que

hace al sistema de ecuaciones imposible de resolver. Este mismo problema se presenta a momento de realizar el cálculo para el método de Crout. Po otro lado, el método de Cholesky por definición descarta este tipo de matriz dado que no se cumple la condición de simetría.

Por otra parte se tiene que para los tres métodos los datos *"Big"*se tienen un error dentro de la comprobación dado que el método no es lo suficientemente preciso, para ser más exactos, para una tolerancia de `1e-13` o mayor la comprobación resulta ser exitosa para los tres casos, sin embargo para una tolerancia menor se tiene un resultado casi siempre fuera del valor en los tres casos. Esto se puede deber tanto a la acumulación de errores al tratarse de una matriz mucho más grande (y también dado que se comparan y se realizan operaciones varias veces con valores muy cercanos entre sí). Asimismo, la necesitad de una tolerancia mayor en este punto se puede deber a la condición de la matriz, tal como el escalamiento o proporción de los elementos de la diagonal con respecto a los demás elementos en la fila. Es probable que para alcanzar una precisión mucho mayor que `1e-13` se requiera del uso de métodos iterativos.

3 Programar un método de solución para matrices simétricas: ya sea LLt (Cholesky) ó LDLt. Adicionalmente, dada la matriz de tamaño $n = 4$, 50, y 100, en donde la entrada ij será 2 si $i = j$, -1 si $i = j + 1$ o $i = j - 1$, y 0 en otro caso. Probar el método de Cholesky guardando las matrices LL^T (descomposición usual) o LDL^T (descomposición clásica) de cada tamaño n.

■ Se creó un programa llamado `cholesky.c` que genera y resuelve la matriz especificada para cualquier $n \in \mathbb{N}$ y después guarda las matrices en archivos. Se ejecutaron y se guardaron los archivos (adjuntos). Las funciones son las siguientes:

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void cholesky(int N, double **matriz, double **L) {
    /* Descompone una matriz A en la matriz triangular inferior L usando el m todo de Cholesky.
     *
     * @ N       El tama o de la matriz A y la matriz L.
     * @ matriz  La matriz de entrada A de tama o N x N.
     * @ L       La matriz triangular inferior L de tama o N x N.
     *
     * Esta funci n realiza la descomposici n de Cholesky de la matriz A en la matriz triangular inferior L.
     * Se revisa si la matriz es sim trica y definida positiva, asi como que su primer valor no sea cero.
     * En caso de cumplir con esto, se imprimir  un mensaje y se terminar  el programa.
     */
    if (fabs(matriz[0][0]) <= 1e-16) {
        printf("Error: matriz imposible. Error Cholesky 0: A[0][0] = 0\n");
        exit(1);
    }

    // revisamos si la matriz es sim trica
    double **matriz_t = (double **)malloc(N * sizeof(double *));
    for (int i = 0; i < N; i++) matriz_t[i] = (double *)malloc(N * sizeof(double));
    for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) matriz_t[i][j] = matriz[j][i];

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (fabs(matriz[i][j] - matriz_t[i][j]) > 1e-16) {
                printf("Error Cholesky: matriz no simetrica. Primer fallo en --> A[%d][%d] != A[%d][%d]\n", i, j, j, i);
                exit(1);
            }
        }
    }

    // Liberar memoria para matriz_t
    for (int i = 0; i < N; i++) {
        free(matriz_t[i]);
    }
    free(matriz_t);

    // Inicializamos la matriz con ceros
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            L[i][j] = 0.0;
        }
    }

    // comenzamos cholesky
    double suma, dummy;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j <= i; j++) {
```

```
            suma = 0.0;
            if (i == j) {
                for (int k = 0; k < j; k++) suma += L[j][k] * L[j][k];
                if ((dummy = matriz[j][j] - suma) < 0) {
                    printf("Error: matriz no definida positiva. Error Cholesky 1: A[%d][%d] - suma < 0\n", j, j);
                    exit(1);
                }
                L[j][j] = sqrt(dummy);
            } else {
                for (int k = 0; k < j; k++) suma += L[i][k] * L[j][k];
                if (fabs(L[j][j]) <= 1e-16) {
                    printf("Error: Matriz imposible. Error Cholesky 2: L[%d][%d] = 0\n", j, j);
                    exit(1);
                }
                L[i][j] = (matriz[i][j] - suma) / L[j][j];
            }
        }
    }
}
void construir_matriz_calor(int nodos, double **matriz) {
    /* Construye una matriz que describe mediante difetencias finitas el sistema de una barra 1D siendo calentada
     *
     * @ nodos   El numero de nodos en la barra.
     * @ matriz  La matriz de entrada A de tama o N x N.
     *
     * Las condiciones de fontera se asume que son de Dirichlet, es decir, que los extremos de la barra y que
     * estos ser n tomados en cuenta dentro del vector de terminos independientes. La matriz resultante es una
     * matriz tridiagonal simetrica con 2 en la diagonal y -1 en las diagonales superior e inferior. En todas
     * las dem s partes es cero.

    */
    // Llenamos la matriz con ceros
    for (int i = 0; i < nodos; i++) for (int j = 0; j < nodos - 1; j++) matriz[i][j] = 0.0;
    // Llenamos la matriz con los valores correspondientes
    for (int i = 0; i < nodos ; i++) {
        for (int j = 0; j < nodos; j++){
            if (i == j) matriz[i][j] = 2.0;
            else matriz[i][j] = 0.0;
            if (i == j + 1 || i == j - 1) matriz[i][j] = -1.0;
        }
    }
}
```

Ejecución de los programas

La creación de los códigos en esta tarea se realizaron dentro de la biblioteca `solvers.h` mientras que el programa principal que despliega el menú se encuentra en `main.c`. El formato de ejecución es el siguiente:

```
$ ./NombreEjecutable NombreArchivoMatriz.txt NombreArchivoVector.tx TOLERANCIA
```

Donde los nombres de los archivos son los correspondientes archivos de los cuales se extraerá los datos y después irá la tolerancia que uno deseé utilizar al momento de realizar la comparación final. Los archivos a leer deberán estar dentro de la misma carpeta donde está `main.c` así como también el archivo `solvers.h`. Si estos archivos contienen una matriz cuadrada de dimensión mayor a 10 estos se guardarán dentro de un archivo dentro de la misma carpeta.

Los programas se probaron los archivos `testMatrix.txt` y `testVector.txt`.