EE231002 Introduction to Programming

Lab07. Matrix Determinants

Due: Apr. 12, 2014

Given an $N \times N$ matrix $\mathbf{A}$ with entries $a_{ij}, 1 \leq i, j, \leq N$, the determinant can be found by the following formula:

$$\det(\mathbf{A}) = \sum_{j=1}^{N} (-1)^{j+1} \times a_{1j} \times \det(\mathbf{A}_{1j}) \tag{7.1}$$

where $\mathbf{A}_{1j}$ is an $(N-1) \times (N-1)$ submatrix of A with row 1 and column j removed.

For example, given

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \tag{7.2}$$

then

$$\det(\mathbf{A}) = a_{11} \times \det(\begin{bmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{bmatrix}) - a_{12} \times \det(\begin{bmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{bmatrix}) + a_{13} \times \det(\begin{bmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}) \tag{7.3}$$

$$= a_{11}(a_{22}a_{33} - a_{32}a_{23}) - a_{12}(a_{21}a_{33} - a_{31}a_{23}) + a_{13}(a_{21}a_{32} - a_{31}a_{22}) \tag{7.4}$$

This is an example of recursive definition in mathematics. Using the recursive definition of determinant, Eq. (7.1), please write a C program to

1. read in a matrix,
2. calculate it's determinant using a recursive function, `det`. The declaration of `det` function is as following.

   ```
   double det(double A[N][N],int dim);
   ```

   The `det` function returns the determinant of matrix `A` as a double precision number. And it has two arguments, the two-dimensional $N \times N$ array `A` and an integer `dim`, that specifies the real size of the matrix. Since the determinant is evaluated recursively, this `dim` may vary even though `A` can stay as a fixed-size array. The constant `N` should be defined as a macro.
3. Once the determinant is obtained, your program prints the answer.

To test out your program, 12 matrices with various dimensions are also given. They are: `mat1.in` – `mat12.in`. Since these matrices have different dimensions, your program needs to be recompiled with different `N` to be able to solve different input matrices. To facilitate such recompilation, please use the following C preprocessing directive to define `N`.

```
#if !defined(N)
#define N 3
#endif
```

If this is done, then we can recompile and execute your program without editing the file, as shown below.

```
$ gcc -DN=7 lab07.c
$ ./a.out < mat7.in
```

Note that the second command above executes the program `a.out` and read the file `mat7.in` as the standard input. Thus, we don't need to retype the matrix using keyboard.

Example of program execution is shown below.

```
$ gcc -DN=3 lab07.c
$ ./a.out < mat1.in
Matrix A is
  1  2  3
  4  5  6
  7  8  9
det(A) = 0
```

Notes.

1. Create a directory **`lab07`** and use it as the working directory.
2. Name your program source file as **`lab07.c`**.
3. The first few lines of your program should be comments as the following.

   ```
   /* EE231002 Lab07. Matrix Determinant
      ID, Name
      Date:
   */
   ```

4. After finishing editing your source file, you can execute the following command to compile it,

   ```
   $ gcc lab07.c
   ```

   If no compilation errors, the executable file, **`a.out`**, should be generated, and you can execute it by typing

   ```
   $ ./a.out < mat1.in
   ```

5. Since the matrices provided have different dimensions, please open the file to find out its dimension, then recompile with suitable `N` before solving for its determinant.
6. You can use **`unix time`** command to find the execution time of your program with different matrix sizes. It is a good idea to find how the CPU time changes with matrix dimension.

   ```
   $ time ./a.out < mat1.in
   ```

7. After you finish verifying your program, you can submit your source code by

   ```
   $ ~ee231002/bin/submit lab07 lab07.c
   ```

   If you see a "submitted successfully" message, then you are done. In case you want to check which file and at what time you submitted your labs, you can type in the following command:

   ```
   $ ~ee231002/bin/subrec
   ```

   It will show the last few submission records.

8. You should try to write the program as efficient as possible. The format of your program should be compact and easy to understand. These are part of the grading criteria.