University of Pittsburgh
Department of Electrical and Computer Engineering
ECE 0301 – Spring 2021

In-Class Assignment #6

Programming concepts:

- 1-D and 2-D arrays, accessing and updating elements
- Arrays as function arguments
- Copying arrays

ECE concepts:

- Linear algebra, matrix-vector multiplication
- Solving systems of linear equations as $\mathbf{Ax} = \mathbf{b}$ problems.
- Determinants, Cramer's rule

Reference: [1] *Advanced Engineering Mathematics, 10th ed.*, Erwin Kreyszig, Wiley, 2011, chapters 7, 20.

Background: There are a variety of applications in Electrical Engineering where we may apply tools from linear algebra to solve problems, including the analysis of linear circuits and state-space representation of linear time-invariant systems. In these problems, the impedances or admittances of the circuit components, or the coefficients of the differential equations that describe the system, are collected in a square matrix that describes the relationship between the variables of interest and an equal number of known quantities:

$$\mathbf{Ax} = \mathbf{b}. \qquad (1)$$

For the case of solving DC resistive circuits, as in Assignment #4, we can write a set of linearly independent KVL or KCL equations as a single matrix equation in the form of equation (1), where **x** represents a column vector of voltages or currents in the circuit, and the square matrix **A** contains resistances or conductances, as appropriate, and additional terms representing the gains of any dependent sources. The column vector **b** is of the same dimension as **x**, and contains terms representing the independent sources in the circuit, and resistances or conductances.

Let a_{ij} be the element in row *i* and column *j* of the matrix **A**, and let x_i and b_i be the elements in row *i* of the column vectors **x** and **b**, respectively. Let *n* be the dimension of the system, so that **A** is $n \times n$, while **x** and **b** are $n \times 1$. We note that the row and column indices may take their values in the range $1 \le i, j \le n$, as is common in textbooks on linear algebra [1], or $0 \le i, j \le n-1$, as is standard for arrays in C++, as needed. We choose the latter here.

Suppose for the moment that **A** and **x** are known, and we wish to compute **b**. Then we simply compute the matrix-vector product; for $0 \le i \le n-1$, element *i* of the product is

$$b_i = \sum_{j=0}^{n-1} a_{ij} x_j . \qquad (2)$$

It should be clear how to implement equation (2) with a loop.

For circuit analysis and related problems, we want to solve the so-called inverse problem, to find **x** given **A** and **b**. There are a variety of ways of doing so, and they are covered extensively in the Kreyszig textbook for ECE 0401. For this assignment, we will implement *Cramer's rule*, a technique that is used only for systems of low dimension, due to the computational complexity of the algorithm [1].

We will break down the steps of Cramer's rule into a series of tasks that build on one another until the algorithm is complete. The first step is simple: imagine that you have a $n \times n$ matrix, and you select an element, and cross out the row and column containing that element, to produce a $(n-1)\times(n-1)$ matrix. For example, suppose we have a 4×4 matrix **A**, and we select element a_{12}. Then we should form the sub-matrix by deleting row 1 and column 2,

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & \boxed{a_{12}} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \rightarrow \begin{bmatrix} a_{00} & a_{01} & a_{03} \\ & & \\ a_{20} & a_{21} & a_{23} \\ a_{30} & a_{31} & a_{33} \end{bmatrix} \rightarrow \begin{bmatrix} a_{00} & a_{01} & a_{03} \\ a_{20} & a_{21} & a_{23} \\ a_{30} & a_{31} & a_{33} \end{bmatrix}.$$

You will implement this in C++ by copying the first *n* rows and columns of a 2-D array, except for the row and column to be deleted, to the first $n-1$ rows and columns of another 2-D array.

The next step is to compute the determinant of a square matrix, which we will denote as

$$|\mathbf{A}|.$$

If *n* = 1, then the matrix is just a scalar, and the determinant is equal to this value,

$$\mathbf{A} = [a_{00}] \qquad \Rightarrow \qquad |\mathbf{A}| = a_{00}.$$

If *n* > 1, the algorithm to compute the determinant requires step 1, deleting the row or column of **A** containing any specified element.

- Choose any row of the matrix. If we choose row *i*, the elements in that row are
$$a_{i0}, a_{i1}, \ldots, a_{i(n-1)}.$$
- For each element a_{ij} in the selected row, delete the corresponding row and column, to yield a $(n-1)\times(n-1)$ sub-matrix containing the remaining elements of **A**. Compute the determinant of the sub-matrix, which is called the minor of a_{ij}, and is denoted by M_{ij}.
- The determinant of **A** is found by multiplying each element a_{ij} by the corresponding minor M_{ij}, and taking the sum of these terms, with alternating signs,
$$|\mathbf{A}| = \sum_{j=0}^{n-1} (-1)^{i+j} a_{ij} M_{ij}. \qquad (3)$$

We can also find the determinant by applying the same procedure with the elements in any selected column. Since the choice of row or column is arbitrary, it is common to use the uppermost row, $i = 0$, so that the algorithm becomes

$$|\mathbf{A}| = \sum_{j=0}^{n-1} (-1)^j \, a_{0j} M_{0j}. \qquad (4)$$

Note that the determinant of the original $n \times n$ matrix requires us to compute n determinants of $(n-1)\times(n-1)$ sub-matrices. Therefore, the algorithm is *recursive*, which means that when we write a `C++` function to implement it, this function will have to call itself to compute the determinants of the sub-matrices. This recursion is also the reason for the high computational complexity of this approach to solving **Ax** = **b** problems.

We can illustrate the algorithm with some examples. Let $n = 2$, and

$$\mathbf{A} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix}.$$

Working across the upper row, we quickly obtain the familiar formula for the determinant of a 2×2 matrix,

$$|\mathbf{A}| = \begin{vmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{vmatrix} = (-1)^0 \cdot a_{00} \cdot a_{11} + (-1)^1 \cdot a_{01} \cdot a_{10} = a_{00}a_{11} - a_{01}a_{10}.$$

We would obtain the same result working through any row or column. For $n = 3$,

$$\begin{vmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{vmatrix} = a_{00} \cdot \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} - a_{01} \cdot \begin{vmatrix} a_{10} & a_{12} \\ a_{20} & a_{22} \end{vmatrix} + a_{02} \cdot \begin{vmatrix} a_{10} & a_{11} \\ a_{20} & a_{21} \end{vmatrix},$$

and we can obtain each of the 2×2 determinants as illustrated above.

Once we have implemented steps 1 and 2 in `C++`, we will be able to compute the determinant of the square matrix **A** for any **Ax** = **b** problem. If $|\mathbf{A}| = 0$, then this means that we can write any row (or column) of **A** as a weighted sum of the other rows (columns), which in turn means that the original set of n equations in n unknowns are not all linearly independent, and there are infinitely many solutions, and Cramer's rule will not find them.

Assuming $|\mathbf{A}| \neq 0$, the third step is another manipulation of matrix elements: replacing a selected column of **A** with **b**. This does not change the dimension of the matrix. For example, if $n = 4$, and we wish to replace column $j = 2$,

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \rightarrow \begin{bmatrix} a_{00} & a_{01} & b_0 & a_{03} \\ a_{10} & a_{11} & b_1 & a_{13} \\ a_{20} & a_{21} & b_2 & a_{23} \\ a_{30} & a_{31} & b_3 & a_{33} \end{bmatrix}$$

You will implement this in C++ by copying $n-1$ columns of a $n \times n$ 2-D array into the corresponding columns of another 2-D array, and then copying the elements of the 1-D array containing **b** into the remaining column.

The final step uses steps 2 and 3 to compute the solution vector **x**. Let $D = |\mathbf{A}|$, and let D_j be the determinant of the matrix formed by replacing column *j* of **A** with **b**. Then element *j* of the solution vector **x** is

$$x_j = \frac{D_j}{D}. \tag{5}$$

Examples: Use these **Ax** = **b** problems to test your programs.

$$\begin{bmatrix} 0.4 & 1.2 \\ 1.7 & -3.2 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \end{bmatrix} = \begin{bmatrix} -2 \\ 8.1 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 1 & 1 \\ 1 & -1 & -1 \\ 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 3 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 10 & 4 & -2 \\ -3 & -17 & 1 & 2 \\ 1 & 1 & 1 & 0 \\ 8 & -34 & 16 & -10 \end{bmatrix} \begin{bmatrix} 4 \\ 0 \\ 2 \\ 6 \end{bmatrix} = \begin{bmatrix} -4 \\ 2 \\ 6 \\ 4 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 300 & 0 & -300 & 0 \\ 0 & 0 & 150 & 0 & -150 \\ 45,000 & -22,500 & -45,000 & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{36}{7} \\ -\frac{81}{7} \\ \frac{153}{14} \\ \frac{129}{7} \\ -\frac{57}{14} \end{bmatrix} = \begin{bmatrix} 12 \\ 12 \\ -9000 \\ 2250 \\ 0 \end{bmatrix}$$

Instructions: Develop software according to the following specifications and submit whatever portion you have completed to the class repository before midnight on the due date.

1. Create a `main` function. Define an `ofstream` object, and use it to open a text file for output named "`ECE0301_ICA06_Axeqb_solution.txt`". For the remainder of this assignment, all required output must be written to this file. Output that you use for temporary debugging purposes can be written to standard output if you choose.

   Write a computer program that writes this introductory message to the output file:
   ```
   ECE 0301 – Matrix-Vector Computations,
   Determinants and Cramer's Rule.
   ```

2. Create a new text file named "`array_dim.h`", and type the following statement as the only line in the file:
   ```
   const int DIM = 2;
   ```
   This statement defines a constant integer variable named `DIM` that will used to represent the dimension of all matrices and vectors in the program, and initializes it to the value 2.

   In your program file, insert the pre-processor directive
   ```
   #include "array_dim.h"
   ```
   above your `main` function. This will cause the definition and initialization of `DIM` to be included in your program when it is compiled, so that you can use it anywhere in the program that the array dimension is required.

   Modify your `main` function so that, after the introductory message has been written to the output file, a blank line is written, followed by one line that reports the array dimension:
   ```
   Global array dimension: DIM = 2
   ```

   Test your program. Change the header file to assign different integer values to `DIM`, then recompile and run the program, and confirm each time that the correct value is written to the output file.

3. Create a new text file named "`ECE0301_ICA06_Axeqb_problem.txt`".

   Download the file "`ECE0301_ICA06_Axeqb_problem_dim2.txt`" from the course website (on the same page as this assignment). Copy the contents of the file you downloaded into the empty file you created, and save it.

   The format of this text file is as follows; the first three lines are:
   ```
   ECE 0301: Ax = b Problem
   N = 2
   A =
   ```
   The second line indicates the dimension N for the system of equations. After line 3, the next N^2 lines contain the elements of the **A** matrix, one per line. Following the last element of **A**, the next line is:
   ```
   b =
   ```

and after that the next *N* lines hold the elements of the **b** vector, one per line.

Write a function that reads the value of *N* from the file, and returns the value as an `int` variable. The parameter for this function is an `ifstream` object for the input file, passed by reference.

Modify your `main` function so that it calls your new function to read the value of *N* from the input file, and then writes the value obtained to the output file. The required format for the output file at this step is:

```
ECE 0301 - Matrix-Vector Computations,
Determinants and Cramer's Rule.

Global array dimension: DIM = 2
Input file: N = 2
```

Test your program to ensure that the output is correct.

4. Add error-checking to your function. If the first line of the input file does not match:

   `ECE 0301: Ax = b Problem`

   or the first four characters of the second line are not "`N = `", then the function must print the following message to standard output, and exit with the failure termination code:

   `ERROR! Input file format error.`

   In addition, if the value of *N* that is read from the input file **is not equal to** the array dimension `DIM` as obtained from the header file, then the function must print the following message to standard output, and exit with the failure termination code:

   `ERROR! Dimension mismatch, N != DIM.`

   Test your program. Change the header file and input file to test all cases where errors should be reported, and ensure that the program does so.

5. Modify your function from steps 3 and 4 so that it also reads the matrix **A** and vector **b** from the input file, and performs all relevant error checking. The parameters for this function are the 2D array `A`, the 1D array `b`, and an `ifstream` object for the input file, passed by reference. The function must return `N` as an integer variable, and fill the `A` and `b` arrays with the values read from the file.

   The steps to be performed by this function are:

   - Read the first line from the input file. If it does not match the required format, exit with the error message and failure code, otherwise proceed.
   - Read the second line from the input file. If it does not match the required format, or the values of *N* and `DIM` are not equal, exit with the error message and failure code, otherwise proceed.
     Read the third line from the input file. If it does not match the required format ("`A = `"), exit with the error message and failure code, otherwise proceed.
   - Read the next N^2 lines and fill the `A` array with the values obtained.

- Read the next line from the input file. If it does not match the required format (“`b = `”), exit with the error message and failure code, otherwise proceed.
- Read the next *N* lines and fill the `b` array with the values obtained.
- Return *N*.

Modify your `main` function so that it calls your new function. Test your program to ensure that it provides the same output shown in step 3, when the input file is formatted correctly. Then alter the input file to cause each of the errors that your function is designed to detect, and ensure that the proper error message and exit code are produced each time.

6. Create a function that writes information about an **Ax** = **b** problem to the output file. The parameters for this function are the integer variable `N`, the 2-D array `A`, the 1D array `b`, and an `ofstream` object for the output file, passed by reference.

   Move the statements that write the lines of text from step 3 from your `main` function to this new output function. Add statement to your `main` function that calls the output function

   Add lines of code to your output function to display the 2-D array `A` as a square matrix. Use the `setw(10)` manipulator in the `iomanip` library to write lines of text that show the matrix elements in right-justified columns. Print each row on a separate line, with square-bracket characters to give the appearance of matrix brackets, as shown below.

   Add lines of code to your output function to display the 1-D array `b` as a column vector. Print each row on a separate line, with square-bracket characters to give the appearance of matrix brackets, as shown below.

   The required format for the output file at this step, for the *N* = 2 example problem, is shown below. Test your program to ensure that the output is correct for this case.

```
ECE 0301 - Matrix-Vector Computations,
Determinants and Cramer's Rule.

Global array dimension: DIM = 2
Input file: N = 2

A =
[        0.4       1.2 ]
[        1.7      -3.2 ]

b =
[         -2 ]
[        8.1 ]
```

7. Download the input files for the example problems of dimension 3, 4 and 5 from the course website. Use each of these files to test your program, by following these steps:
   - Copy the contents of one of the files to the input file.
   - Change the value of `DIM` in the header file to match the value of *N* in the input file.
   - Compile and run your program.
   - Repeat for each of the input files, and ensure that the output is correct in each case.

   At this point, you should have a program that is capable of reading a properly-formatted input file representing an **Ax** = **b** problem of arbitrary dimension, and writing information about the problem to the output file. When you are certain your program is correct, save it in a file named:

   `ece0301_ICA06_step07.cpp`

   Submit this file to the class repository.

8. Write a function that implements matrix-vector multiplication. The parameters for this function are a 2-D array containing the **A** matrix, a 1-D array containing the **x** vector, and a third vector that will store the product vector **Ax**. Write your function with a loop that implements equation (2).

   Modify your `main` function so that, after reading the input file but before writing the output file, it defines a 1-D array `x`. Next, use a loop to prompt the user to enter the elements of **x**, and read the user's responses from standard input, one element at a time.

   Finally, call your matrix-vector multiplication function to compute the product **Ax**. The purpose of this step is to check whether the product **Ax** produced by your function is equal to the **b** vector. However, you will not be able to perform this check until you implement the next step.

9. Modify your output function so that:
   - It takes an additional parameter for the solution array `x`.
   - After writing the matrix **A** and vector **b** to the output file, it writes a matrix-vector product **Ax** = **b** to the file, by showing the elements in rows and columns. The formatting to use is shown below for the *N* = 2 example.

```
A * x = b
[        0.4        1.2    ] * [          1 ] = [         -2 ]
[        1.7       -3.2    ]   [         -2 ]   [        8.1 ]
```

   Modify your `main` function so that it performs the following steps:
   - Declares an o`fstream` object, uses it to open the output file.
   - Declares an `ifstream` object, uses it to open the input file.
   - Declares arrays for **A** and **b**, and calls the file-reading function to read *N*, **A** and **b** from the input file.
   - Defines the array `x`, allows the user to enter the elements of **x**, and reads them from standard input.

- Calls the matrix-vector multiplication function to compute **Ax**.
- Calls the matrix-vector output function, with the matrix-vector product **Ax** as the `b` parameter, to write *N*, **A**, **b**, and **Ax** = **b** to the output file, following the formatting specified in steps 4 and 6. All the write operations should be done from inside the output function.

Test your program with each of the example problems, and confirm that your program provides the correct product, i.e. that **Ax** = **b**, in each case. You can do this by comparing the product **Ax** in your output file with the corresponding vector **b** from the example problems. Each time you choose a new example, you will need to copy the contents of the appropriate example file into the input file, change the value of `DIM` in the header file, compile and build your program, then run it and enter the elements of the solution vector **x** when prompted.

Although you won't be submitting step 9, make sure to check the posted template on Courseweb to make sure that your output still follows the required format.

10. Write a new matrix-display function that writes only the first *n* rows and columns of a 2D array to the output file. The parameters for this function are the 2D array, an integer `n`, and an `ofstream` object for the output file, passed by reference. Use the same formatting that is shown in earlier steps.

    Modify your `main` function so that, after reading the input file but before doing anything else, it calls the new matrix-display function to write the first *n* rows and columns of `A`. Run the program several times with different values of *n*, to test that this feature works correctly. This will be useful for debugging the next two steps.

11. Write a function that copies the upper-left portion of one matrix to another, except for a specified row and column. The parameters for this function are a 2D array containing the matrix to be copied, a 2D array to receive the copied elements, and integer parameters *i*, *j* and *n*. Your function must copy the first *n* rows and columns of the first array, except for row *i* and column *j*, to the first $n-1$ rows and columns of the second array.

    Modify your `main` function so that, after reading the input file but before doing anything else, it performs the following steps:
    - Declare integer variables `i`, `j` and `n`, and initialize them to `i = 1`, `j = 2`, `n = 4`.
    - Call the function from step 10 to write the first `n` rows and columns of `A`.
    - Declare a new 2D array to receive the copied matrix.
    - Call the matrix-copy function to copy the first `n` rows and columns of `A` to the new array, except for row `i` and column `j`.
    - Call the function from step 10 to write the first `n-1` rows and columns of the copied matrix.

    Test your program using the $N = 5$ example, and ensure that the proper row and column have been deleted from the **A** matrix. Vary the integers `i`, `j` and `n`, and ensure that the program output is correct each time.

12. Write a *recursive function* that returns the determinant of a square matrix, following equation (4), which is reproduced here for convenience:

$$|\mathbf{A}| = \sum_{j=0}^{n-1} (-1)^j \, a_{0j} M_{0j}$$

The parameters for this function are a 2-D array containing the matrix **A**, and an integer parameter *n* specifying that the matrix of interest lies in the first *n* rows and columns of the array.

The form of equation (4) suggests the following pseudocode for the determinant function:

- If $n = 1$, then the array of interest contains only one element, and the determinant is equal to this element. Return this value and go back to the calling routine.
- Otherwise, initialize the determinant to 0, and define a new 2D array with at least $n-1$ rows and columns. We will refer to this new array as the sub-matrix.
- Loop over the elements in row 0 of **A**. For each element,
  - Call your function from step 11 to copy the first *n* rows and columns of **A**, except for row 0 and the column of the current element, to the first $n-1$ rows and columns of the sub-matrix.
  - Recursively call the determinant function with the array parameter set to the sub-matrix and the integer parameter set to $n-1$.
  - Multiply the value returned by the determinant function by $(-1)^j$, where *j* is the column index, and add the result to the determinant.
- Return the value of the determinant.

Modify your `main` function so that it performs the following actions:

- Declare an `ofstream` object, and uses it to open the output file.
- Declare an `ifstream` object, and uses it to open the input file.
- Declare arrays for **A** and **b**.
- Call your function to read the input file.
- Call the function from step 10, with $n = N$, so that all rows and columns of **A** are written to the output file.
- Call your new function to compute the determinant of the **A** matrix
- Write the determinant to the output file.

Test your program with each of the example problems. The determinant of the A matrix for each problem is listed in the table below.

n	det(**A**)
2	-3.32
3	3
4	-1,568
5	14,175,000,000

The required output formatting at this step is considerably simpler than for previous steps. For the $N = 4$ example, the correct output is

```
[           0         10          4         -2 ]
[          -3        -17          1          2 ]
[           1          1          1          0 ]
[           8        -34         16        -10 ]
det(A) = -1568
```

When you are certain your program is correct, save it in a file named:

`ece0301_ICA06_step12.cpp`

Submit this file to the class repository.

13. Write a function that will copy all but one of the columns of a matrix **A** to another matrix, and copy a specified vector **b** to the remaining column. The parameters for this function are a 2-D array representing **A**, a 1-D array representing **b**, a 2-D array to receive the copied columns, and an integer parameter indicating which column of **A** should be replaced by **b**.

    Modify your `main` function so that it performs the following actions:

    - Define an `ofstream` object, and use it to open the output file.
    - Define an `ifstream` object, and use it to open the input file.
    - Define arrays for **A** and **b**.
    - Call the function to read the input file.
    - Call the function from step 10, with $n = N$, so that all rows and columns of **A** are written to the output file.
    - Call the function from step 12 to compute the determinant of the **A** matrix
    - Write the determinant of **A** to the output file.
    - Define a new array the same size as **A**. We will refer to this as the column-replaced matrix.
    - Loop over the columns of **A**, beginning with column 0. For each column,
        - Call your new function to copy **A** to the column-replaced matrix, but copy **b** to the current column.
        - Call the function from step 10, with $n = N$, so that all rows and columns of the column-replaced matrix are written to the output file.
        - Call the function from step 12 to compute the determinant of the column-replaced matrix.
        - Write the determinant to the output file.

    Test the resulting program for various example problems, and ensure in each case that the proper columns of **A** are replaced by **b**. For the example problems, the correct values for the determinants are:

    $N = 2$:

    $$D = -3.32, \quad D_0 = -3.32, \quad D_1 = 6.64$$

$N = 3$:

$$D = 3, \quad D_0 = 3, \quad D_1 = -6, \quad D_2 = 9$$

$N = 4$:

$$D = -1568, \quad D_0 = -6272, \quad D_1 = 0, \quad D_2 = -3136, \quad D_3 = -9408$$

$N = 5$:

$$D = 1.4175 \times 10^{10}, \quad D_0 = 7.29 \times 10^{10}, \quad D_1 = -1.64025 \times 10^{11},$$
$$D_2 = 1.549125 \times 10^{11}, \quad D_3 = 2.61225 \times 10^{11}, \quad D_4 = -5.77125 \times 10^{10}$$

14. The final step is to use all of the functions you have developed to read the input file defining an **Ax** = **b** problem, use Cramer's rule to solve for **x**, compute **Ax** and the error between **Ax** and **b**, and write information about the problem and results to the output file.

    Modify the file output function from step 9 so that it performs the following actions. The output file format is specified on the next page.
    - Write the introductory message to the output file.
    - Write the global array dimension to the output file.
    - Write the value of *N* to the output file.
    - Write the matrix **A** to the output file.
    - Write the column vector **b** to the output file.
    - Write the **Ax** = **b** problem (with solution vector unspecified) to the output file.
    - Write the solution vector **x** to the output file.
    - Define an array to store the matrix-vector product **Ax**.
    - Compute **Ax**.
    - Write the **Ax** = **b** problem solution to the output file, with the solution vector in place of **x**, and the product **Ax** in place of **b**.
    - Compute and report the error vector **Ax – b**.

    Modify your `main` function so that it performs the following actions:
    - Define an `ofstream` object, and uses it to open the output file.
    - Define an `ifstream` object, and uses it to open the input file.
    - Define arrays for **A** and **b**.
    - Call the function to read the input file.
    - Call the function from step 12 to compute the determinant of the **A** matrix
    - Define a new array the same size as **A**. We will refer to this as the column-replaced matrix.
    - Loop over the columns of **A**, beginning with column 0. For each column,
        - Call your new function to copy **A** to the column-replaced matrix, but copy **b** to the current column.
        - Call the function from step 12 to compute the determinant of the column-replaced matrix.
        - Compute the corresponding element of the solution vector.

- Call the file output function to report all information about this **Ax** = **b** problem and the solution to the output file.

If your program can solve **Ax** = **b** problems correctly, you should find that the error in the elements of **b** is zero or very small (less than 10^{-10}) for all of the example problems.

When you are certain your program is correct, save it in a file named:

`ece0301_ICA06_step14.cpp`

Submit this file to the class repository.

The output file format for the complete program, for the $N = 2$ example problem, is shown below:

```
ECE 0301 - Matrix-Vector Computations,
Determinants and Cramer's Rule.

Global array dimension: DIM = 2
Input file: N = 2

A =
[        0.4       1.2 ]
[        1.7      -3.2 ]

b =
[         -2 ]
[        8.1 ]

Problem: A * x = b
[        0.4       1.2   ] * [ x0 ] = [         -2 ]
[        1.7      -3.2   ]   [ x1 ]   [        8.1 ]

Solution: x =
[          1 ]
[         -2 ]

Checking Solution: A * x = b
[        0.4       1.2    ] * [          1 ] = [         -2 ]
[        1.7      -3.2    ]   [         -2 ]   [        8.1 ]

Error in RHS values:
[        0 ]
[        0 ]
```