BRYN MAWR COLLEGE

Department of Physics

Computational Module 6 – Solution of Linear Equations

Prerequisite modules: Module 0; Module 1
Estimated completion time: 3-6 hours
Learning objectives: Understand the application of matrix methods to the solution of simultaneous, linear algebraic equations; apply these methods to least-squares fitting of data
Note: This module requires some familiarity with linear algebra

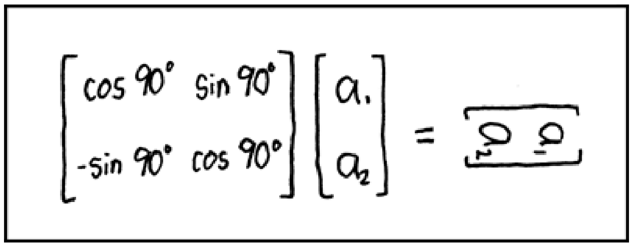

Image credit: xkcd.com

Applications that require the solution of simultaneous linear equations are common in science, engineering, and other quantitative fields. Examples include solving for unknown circuit elements, determining forces on parts of a structure, analyzing a system of coupled oscillators, and others. This module will focus on a few of the many techniques for solving systems of linear equations. These techniques, based heavily on matrix manipulations, also can be used for least-squares fitting of experimental data.

6.1 Gaussian elimination with back-substitution

We start by considering a relatively simple scenario involving just three unknown variables (x_1, x_2, x_3) whose relationships are represented by a set of simultaneous linear equations[1] of the form

$$a_{1,1}\,x_1 + a_{1,2}\,x_2 + a_{1,3}\,x_3 = r_1, \tag{1}$$

$$a_{2,1}\,x_1 + a_{2,2}\,x_2 + a_{2,3}\,x_3 = r_2, \tag{2}$$

$$a_{3,1}\,x_1 + a_{3,2}\,x_2 + a_{3,3}\,x_3 = r_3. \tag{3}$$

The set of coefficients $a_{i,j}$ can be represented as a matrix $\mathbf{A}$, and the constants r_i as a vector $\mathbf{r}$; all of these constants are assumed known, and we wish to solve for the set of unknown variables x_i $(i = 1, 2, 3)$ which can be represented as another vector, $\mathbf{x}$. This set of equations thus can be written in matrix form as $\mathbf{Ax} = \mathbf{r}$.

The standard way to solve such a matrix equation, at least in principle, is to invert the matrix $\mathbf{A}$, and then compute the x_i from $\mathbf{x} = \mathbf{A}^{-1}\mathbf{r}$. It turns out, though, that numerical matrix inversion is inefficient, and that there are superior methods for solving such an equation, at least in the case of a square matrix $\mathbf{A}$. (Rectangular $\mathbf{A}$'s can arise, e.g., in the case of fitting a function to a set of data, in which there may be more equations than unknowns. This situation will be considered in the next module.) A common method of solving the matrix equation is ***Gaussian elimination***, which you may be familiar with from a linear algebra course. Gaussian elimination makes use of two operations that do not change the solutions x_i: (i) Any of the equations can be multiplied by any constant; (ii) Any linear combination of two of the equations can be made into another valid equation. (See any linear algebra text for proofs of these facts.)

The Gaussian elimination procedure uses these two facts to change the matrix $\mathbf{A}$ into one having ***upper triangular*** form, such that all elements below the diagonal are zero (additionally, in the case of Gaussian elimination the diagonal elements are 1). For conciseness, the procedure will be described below in terms of the generic parameters $a_{i,j}$, x_i and r_i. (A specific example is presented below.) Note that in this procedure the values x_i are not altered; only the values of the parameters $a_{i,j}$ and r_i are modified. Also keep in mind that while the problem presented above involves three equations, and therefore 3-element vectors and a 3×3 matrix, the procedure below can be applied to any number of equations (N) for the same number of unknowns.

1. Divide the first row of $\mathbf{A}$, and of $\mathbf{r}$, by the value $a_{1,1}$ (known as the ***pivot*** for this step), so as to make the new first coefficient, $a'_{1,1}$, equal to 1. The other first-row parameters then become $a'_{1,2} = \dfrac{a_{1,2}}{a_{1,1}}$, $a'_{1,3} = \dfrac{a_{1,3}}{a_{1,1}}$, etc., and $r'_1 = \dfrac{r_1}{a_{1,1}}$.

[1]Most of this module is adapted from *Computational Physics* by Newman, with small contributions from *Scientific Computing* 2e by Heath.

2. Multiply the first row by $a_{2,1}$ and subtract the resulting row from the second row to get a new second row with a leading coefficient $a'_{2,1} = 0$. Apply this procedure to each of the rows below the second one, so that the first column of the new matrix ends up as (1, 0, 0, ...).
3. Now basically repeat step 1, but starting with the *second* row: divide the second row of parameters by the value in its *second* column (the new pivot, in the position of what had been $a_{2,2}$) so that the value at this position becomes 1.
4. As in step 2, multiply the second row in turn by the value in the second position in each of the subsequent rows and subtract the corresponding result from each of those rows; this leaves each of them with the value zero in the second position.
5. Repeat steps 3 and 4 until the matrix has been converted to upper triangular form (with 1's along the diagonal).

The iterative procedure above can be expressed more concisely in a loop of just three steps (which almost serves as pseudocode for a program to implement Gaussian elimination):

1. Starting with $i = 1$ (`i = 0` in Python code), divide the i^{th} row of $\mathbf{A}$ and the i^{th} element of $\mathbf{r}$ by the value $a_{i,i}$ to make the new i^{th} coefficient, $a'_{i,i}$, equal to 1.
2. Looping over $j > i$, multiply the i^{th} row of $\mathbf{A}$ and also $\mathbf{r}_i$ by $a_{j,i}$ and subtract the resulting row from the j^{th} row (and the resulting r from r_j) to get a new j^{th} row with new coefficient $a'_{j,i} = 0$ (and corresponding new r_j value).
3. While $i < N$ (the number of rows in the matrix), increment i by 1 and return to step 1.

As a particular example, consider the equation

$$\begin{pmatrix} 1 & 2 & 2 \\ 4 & 4 & 2 \\ 4 & 6 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \\ 10 \end{pmatrix}.$$

The first element of the first row of the matrix, $a_{1,1}$, already is 1, so we don't need to divide by it. The next step, then, is to multiply the first row (of both the matrix and the numerical vector) by 4 and subtract the result from the second row (of both matrix and vector). The results are

$$\begin{pmatrix} 1 & 2 & 2 \\ 0 & -4 & -6 \\ 4 & 6 & 4 \end{pmatrix} \qquad \text{and} \qquad \begin{pmatrix} 3 \\ -6 \\ 10 \end{pmatrix}.$$

Next, we again multiply the first row by 4 and subtract it from the *third* row, to make *its* first element a zero. We get

$$\begin{pmatrix}1 & 2 & 2\\ 0 & -4 & -6\\ 0 & -2 & -4\end{pmatrix} \qquad \text{and} \qquad \begin{pmatrix}3\\ -6\\ -2\end{pmatrix}.$$

We next move to the second row and divide it by the leading non-zero term, -4, obtaining

$$\begin{pmatrix}1 & 2 & 2\\ 0 & 1 & 1.5\\ 0 & -2 & -4\end{pmatrix} \qquad \text{and} \qquad \begin{pmatrix}3\\ 1.5\\ -2\end{pmatrix}.$$

Breakpoint 1: Complete the Gaussian elimination process for this example.

After going through this procedure, the matrix equation (for three simultaneous equations) will have the general form

$$\begin{pmatrix}1 & c_{1,2} & c_{1,3}\\ 0 & 1 & c_{2,3}\\ 0 & 0 & 1\end{pmatrix}\begin{pmatrix}x_1\\ x_2\\ x_3\end{pmatrix} = \begin{pmatrix}v_1\\ v_2\\ v_3\end{pmatrix}, \tag{4}$$

where the v_i are produced from the r_i as a result of the transformations outlined above. This equation could be written in matrix form as $\mathbf{Ux} = \mathbf{v}$ (where $\mathbf{U}$ is for upper-triangular).

Given a matrix in upper-triangular form with 1's on the diagonal, the rest of the solution now is straightforward. The method is known as ***back-substitution***, and the key is to work upward from the last row. We can see immediately that the third row of Eq. (4) indicates that $x_3 = v_3$. Next, matrix multiplication using the second row of the matrix gives $x_2 + c_{2,3}\, x_3 = v_2$; substituting v_3 for x_3 we find $x_2 = v_2 - c_{2,3}\, v_3$. Finally, the first row of the matrix equation implies $x_1 + c_{1,2}\, x_2 + c_{1,2}\, x_3 = v_1$, so $x_1 = v_1 - c_{1,2}\, x_2 - c_{1,3}\, x_3$, and we can solve for x_1 by substituting the values for x_2 and x_3 that we just found. The process in the case of more than three equations is the same; there simply are more substitutions to perform. In fact, we note that we can write each result for x_i generally as

$$x_i = v_i - \sum_{j>i} c_{i,j}\, x_j. \tag{5}$$

Breakpoint 2: Use back-substitution and your result for *Breakpoint 1* to solve for the x_i values in the example above.

A potential problem may have occurred to the attentive reader: if the first element of the first row of the matrix of values, $a_{0,0}$, is zero, then the Gaussian elimination process fails at the start, since it would require dividing by zero. An easy fix is to swap the first row with some other row that does not have this shortcoming. This process is known as ***pivoting***. However, since the matrix elements change as the Gaussian elimination procedure progresses, similar problems can arise throughout the process. A general solution is ***partial pivoting***: when processing the i^{th} row (in step 1 of the three-step procedure outlined above), first compare the absolute value of its i^{th} element with the absolute value of the i^{th} element in each lower row; if the largest such value is not the one in the i^{th} row, then swap the i^{th} row with the row that has the largest i^{th} absolute value. (The reason for using the row with the largest absolute value is to avoid, as much as possible, dividing by a small element, which could lead to numerical errors.) The rest of the procedure then follows as before.

Breakpoint 3: Consider the two matrices shown below. For each one, say whether partial pivoting should be used, based on the discussion above. If so, which rows should be interchanged? If not, explain why not.

$$\begin{pmatrix} 1 & -1 & 3 \\ 0 & 3 & 6 \\ 0 & 1 & -2 \end{pmatrix} \qquad \text{and} \qquad \begin{pmatrix} 1 & -1 & 3 \\ 0 & 3 & 2 \\ 0 & -6 & 5 \end{pmatrix}$$

6.2 LU decomposition

In physics applications, we often want to solve sets of equations of the form $\mathbf{Ax} = \mathbf{r}$ in which $\mathbf{A}$ remains the same but $\mathbf{r}$ changes. An example might be changing one or more of the batteries in a circuit, but keeping the resistors the same. In these situations, there is no need to repeat the entire Gaussian elimination procedure each time, as $\mathbf{A}$ would be transformed in the same way in each case. A more efficient solution method in such situations, called ***LU decomposition***, makes use of the fact that the steps of the earlier procedure all can be represented by matrix multiplications. (A reader who is not interested in the details of this approach can skip to the paragraph immediately above the small bit of Python code on page 7.)

For example, recall that in the first step we divide the first row by $a_{1,1}$; we then subtract $a_{2,2}$ times the new first row from row 2, subtract $a_{3,3}$ times the new first row from row 3, etc. This part of the process can be implemented for a 4×4 matrix $\mathbf{A}$ (with elements $a_{i,j}$) by the matrix multiplication indicated below

$$\frac{1}{a_{1,1}} \begin{pmatrix} 1 & 0 & 0 & 0 \\ -a_{2,1} & a_{1,1} & 0 & 0 \\ -a_{3,1} & 0 & a_{1,1} & 0 \\ -a_{4,1} & 0 & 0 & a_{1,1} \end{pmatrix} \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{pmatrix} = \begin{pmatrix} 1 & b_{1,2} & b_{1,3} & b_{1,4} \\ 0 & b_{2,2} & b_{2,3} & b_{2,4} \\ 0 & b_{3,2} & b_{3,3} & b_{3,4} \\ 0 & b_{4,2} & b_{4,3} & b_{4,4} \end{pmatrix}, \tag{6}$$

where the $b_{i,j}$ are new matrix elements determined by the multiplication. We see that the first column has been converted into the desired form by multiplication by the first transformation matrix, which (including the initial factor of $1/a_{1,1}$) will be denoted by $\mathbf{T}_1$. Thus, we can write the previous expression as $\mathbf{T}_1\mathbf{A} = \mathbf{B}$.

Another transformation matrix can be constructed to represent the next step of the Gaussian elimination process, in which row 2 of the new matrix, $\mathbf{B}$, is divided by $b_{2,2}$ and then appropriate multiples of the resulting row are subtracted from each of the lower rows. This second elimination step is implemented by a second transformation matrix, $\mathbf{T}_2$, which appears here as the first matrix (together with the preceding factor $1/b_{2,2}$):

$$\frac{1}{b_{2,2}}\begin{pmatrix} b_{2,2} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -b_{3,2} & b_{2,2} & 0 \\ 0 & -b_{4,2} & 0 & b_{2,2} \end{pmatrix}\begin{pmatrix} 1 & b_{1,2} & b_{1,3} & b_{1,4} \\ 0 & b_{2,2} & b_{2,3} & b_{2,4} \\ 0 & b_{3,2} & b_{3,3} & b_{3,4} \\ 0 & b_{4,2} & b_{4,3} & b_{4,4} \end{pmatrix} = \begin{pmatrix} 1 & c_{1,2} & c_{1,3} & c_{1,4} \\ 0 & 1 & c_{2,3} & c_{2,4} \\ 0 & 0 & c_{3,3} & c_{3,4} \\ 0 & 0 & c_{4,3} & c_{4,4} \end{pmatrix}, \tag{7}$$

or $\mathbf{T}_2\mathbf{B} = \mathbf{C}$. In matrix notation we so far have $\mathbf{T}_2\mathbf{T}_1\mathbf{A} = \mathbf{C}$. A similar procedure applied to the third row of matrix $\mathbf{C}$, and then to the fourth row of the matrix that it is transformed into ($\mathbf{D}$), requires two additional transformation matrices, $\mathbf{T}_3$ and $\mathbf{T}_4$, shown below:

$$\mathbf{T}_3 = \frac{1}{c_{3,3}}\begin{pmatrix} c_{3,3} & 0 & 0 & 0 \\ 0 & c_{3,3} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -c_{4,3} & c_{3,3} \end{pmatrix} \quad ; \quad \mathbf{T}_4 = \frac{1}{d_{4,4}}\begin{pmatrix} d_{4,4} & 0 & 0 & 0 \\ 0 & d_{4,4} & 0 & 0 \\ 0 & 0 & d_{4,4} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{8}$$

The complete Gaussian elimination process applied to matrix $\mathbf{A}$ is represented by the matrix multiplication $\mathbf{T}_4\mathbf{T}_3\mathbf{T}_2\mathbf{T}_1\mathbf{A}$, which will result in $\mathbf{A}$ being converted into upper-triangular form. (Note that all of the $\mathbf{T}$ matrices are lower-triangular: they have only zeros above their diagonals.) The original equation that we wanted to solve, $\mathbf{Ax} = \mathbf{r}$, now can be modified to $\mathbf{T}_4\mathbf{T}_3\mathbf{T}_2\mathbf{T}_1\mathbf{Ax} = \mathbf{T}_4\mathbf{T}_3\mathbf{T}_2\mathbf{T}_1\mathbf{r}$. The matrix multiplying $\mathbf{x}$ on the left will be upper-triangular, and on the right we would get a vector, so this equation is of the form of Eq. (4), and it can be solved by back-substitution. The point of this long process is that we now know how to put $\mathbf{A}$ into upper-triangular form regardless of what $\mathbf{r}$ is, so if that vector changes then only the back-substitution part of the analysis must be done from scratch.

The actual computation in the LU decomposition approach differs slightly from the procedure just described. It first involves defining two matrices:

$$\mathbf{L} = \mathbf{T}_1^{-1}\mathbf{T}_2^{-1}\mathbf{T}_3^{-1}\mathbf{T}_4^{-1}, \qquad \mathbf{U} = \mathbf{T}_4\mathbf{T}_3\mathbf{T}_2\mathbf{T}_1\mathbf{A}. \tag{9}$$

Note that $\mathbf{U}$ is just the upper-triangular matrix that will multiply $\mathbf{x}$, as described above. Furthermore, matrix multiplication reveals that $\mathbf{L}$ is a lower-triangular matrix. In fact, that matrix has the form

$$\mathbf{L} = \begin{pmatrix} a_{1,1} & 0 & 0 & 0 \\ a_{2,1} & b_{2,2} & 0 & 0 \\ a_{3,1} & b_{3,2} & c_{3,3} & 0 \\ a_{4,1} & b_{4,2} & c_{4,3} & d_{4,4} \end{pmatrix}. \tag{10}$$

If we multiply the two matrices just defined, we get

$$\mathbf{LU} = \mathbf{A}, \tag{11}$$

which is the LU decomposition of $\mathbf{A}$. With this result, the original set of equations $\mathbf{Ax} = \mathbf{r}$ can be rewritten as $\mathbf{LUx} = \mathbf{r}$. The trick now is to write $\mathbf{Ux} = \mathbf{y}$, so then $\mathbf{Ly} = \mathbf{r}$. The usefulness of these two expressions is that they both involve just one triangular matrix, as in Eq. (4), and therefore can be solved immediately by back-substitution. One first solves for $\mathbf{y}$ in the $\mathbf{Ux} = \mathbf{y}$ equation, and that vector then is substituted into the $\mathbf{Ly} = \mathbf{r}$ equation to obtain $\mathbf{r}$.

LU decomposition is the most common method for solving simultaneous equations based on Gaussian elimination. One can see how the LU decomposition process would go in general. First compute $\mathbf{T}_1\mathbf{A}$, where the form of $\mathbf{T}_1$ is clear (in the case of any number of dimensions) from Eq. (6). This gives the new matrix $\mathbf{B}$, from which the elements of $\mathbf{T}_2$ can be determined via generalization of Eq. (7). Use this to compute $\mathbf{T}_2\mathbf{T}_1\mathbf{C}$, from which the elements of $\mathbf{C}$ needed in $\mathbf{T}_3$ can be found, and so on. Keep in mind that there must be as many $\mathbf{T}$ matrices as there are equations to be solved. In addition, partial pivoting generally should be used as part of the process to avoid problems with zero-valued diagonal elements.

Fortunately, in Python (as in many other languages) there already exist functions for solving sets of simultaneous equations. The function that performs LU decomposition and back-substitution is called `solve`, and is part of the `linalg` module within the `numpy` package (more information on it and other Python functions for solving sets of equations is at www.scipy.org). It is called as follows:

```
from numpy.linalg import solve
x = solve(A,r)
```

It's worth briefly mentioning that the solution approach discussed above can be streamlined in a certain common physics situation; namely, when the matrix $\mathbf{A}$ of coefficients has a ***banded*** structure consisting of a main diagonal and a limited number of additional diagonals immediately above and below, with the elements farther from the diagonal being zero. Here is a 4 × 4 example, with one diagonal above and below the main one (making a ***tridiagonal*** matrix):

$$\begin{pmatrix} a_{1,1} & a_{1,2} & 0 & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & 0 \\ 0 & a_{3,2} & a_{3,3} & a_{3,4} \\ 0 & 0 & a_{4,3} & a_{4,4} \end{pmatrix}. \tag{12}$$

This sort of situation arises, for example, in the case of a linear arrangement of masses connected by springs, and in similar scenarios in which an element of a system is affected by nearby neighbors but not more distant ones (e.g., the famous Ising model for a lattice of interacting spins). The streamlining possible in this case occurs in step 3 of the three-step outline of the general Gaussian elimination process. In the general case, each row needs to be multiplied by appropriate $a_{j,i}$'s and then subtracted from *each* of the rows below it. In the case of a banded matrix with n diagonals above and below the main one, each row needs to be appropriately multiplied and then subtracted only from the n rows immediately below it. For large matrices with narrow bands, this simplification can considerably reduce the computation time needed.

6.3 Other decompositions

The LU decomposition is not the only possible factorization of a matrix, and other decompositions can be more efficient or otherwise superior in certain situations. For example, in the case of a Hermitian, positive-definite matrix $\mathbf{A}$ (for a real matrix, this means it's symmetric, with all positive eigenvalues), the ***Cholesky decomposition*** will run in about half the time of the LU decomposition. This decomposition involves factoring the matrix $\mathbf{A}$ into the product of a lower-triangular matrix and its (conjugate) transpose, i.e. $\mathbf{A} = \mathbf{L}\mathbf{L}^T$, which sometimes is described as (sort of) taking the square root of a matrix. It turns out that the Cholesky method does not even require any pivoting. Another factorization in the same vein is the ***QR decomposition***, in which the matrix $\mathbf{A}$ is factored into the form $\mathbf{A} = \mathbf{QR}$, where $\mathbf{Q}$ is orthogonal ($\mathbf{Q}^T\mathbf{Q} = \mathbf{1}$, where $\mathbf{1}$ is the identity matrix) and $\mathbf{R}$ is upper-triangular (what we labeled as $\mathbf{U}$ earlier). This is used for factorizing real matrices in ***Hessenberg form***: matrices with zeros everywhere below the main diagonal except for the diagonal just below the main one. More information on these decompositions can be found in advanced texts on numerical methods.

Finally, a powerful method known as ***singular value decomposition*** may work even in the case of a matrix that is *singular* (having a zero determinant) or nearly so, when the Gaussian procedure fails. It also is the favored method for solving linear least-squares problems. This factorization method will be discussed again in Module 7.

Recap

- Gaussian elimination with back-substitution is commonly used for numerically solving the matrix equation $\mathbf{Ax} = \mathbf{r}$. Partial pivoting is used to avoid numerical errors associated with division by small (or zero) matrix entries.

- LU decomposition efficiently solves such problems in cases when $\mathbf{r}$ varies but $\mathbf{A}$ does not. Other decompositions may be preferred for matrices with specific forms.

Exercises

Exercise #1

Write a Python function to implement the Gaussian elimination algorithm presented in the three-step loop shown above. It should take a matrix of coefficients ($\mathbf{A}$) and vector of results ($\mathbf{r}$) as its inputs, and it should output an upper-triangular matrix ($\mathbf{U}$) and corresponding vector ($\mathbf{v}$). Apply your function to the matrix below, and the vector $[-4., 3., 9., 7.]$, and check whether it yields an upper-triangular matrix with 1's along the diagonal. Print out both the final matrix and final vector. (Python reminders: `A[i,j]` denotes the `i`-`j` element of array `A`, `A[i]` denotes the entire row with index `i`, and `A[:,j]` denotes the entire column with index `j`. Also remember that the first row/column has index `0`.)

$$\begin{pmatrix} 2. & 1. & 4. & 1. \\ 3. & 4. & -1. & -1. \\ 1. & -4. & 1. & 5. \\ 2. & -2. & 1. & 3. \end{pmatrix}.$$

(Note: It is strongly recommended that $\mathbf{A}$ and $\mathbf{r}$ be implemented as *arrays* rather than lists in your function. Answer: the output vector should be $[-2., 3.6, -2., 1.]$.)

Exercise #2

Write a Python function to implement the back-substitution algorithm, Eq. (5). Its inputs should be an upper-triangular array and a vector, and it should output a vector of x_i values. Run it on the array ($\mathbf{U}$) and vector ($\mathbf{v}$) from Exercise #1 to find the unknown x_i values. (Python tip: to create a loop that runs backwards, use the syntax `range[max, min-1, -1]`, where `max` is the largest index desired for the loop and `min` is the smallest index; the last `-1` reverses the loop.) (Answer: $[2., -1., -2., 1.]$)

Exercise #3

(a) Incorporate partial pivoting into the function you constructed in Exercise #1 above. Print out the resulting matrix and vector. (Note: to swap the rows with indices 1 and 2 in an array, you can use this syntax: `A[[1,2]] = A[[2,1]]`.)

(b) Combine the Gaussian elimination (with partial pivoting) and back-substitution functions into a single function for solving simultaneous linear equations. (This is not complicated: just call those two functions sequentially within a new function to which you pass the original matrix and vector, and pass the output of the Gaussian elimination function to the back-substitution function.) Run it on the matrix of Exercise #1 and print out the solution vector to make sure the two parts work together properly.

Exercise #4

The simple circuit below (Figure 1) can be analyzed by hand, but we will use it to further test our code. The circuit contains two batteries (of 6 and 12 volts) and two resistors (of 6 and 9 ohms). The *assumed*

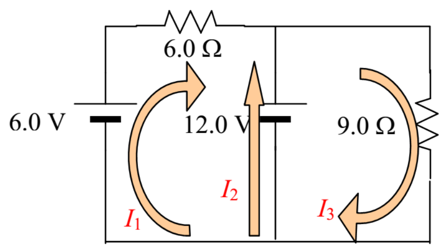

Figure 1: A simple circuit (for Exercise #4)

current directions in the three "branches" also are shown (as I_1, I_2, I_3). Kirchhoff's **loop rule** states that the sum of voltage changes encountered in tracing a closed loop around the circuit must be zero. That requirement gives the following equations for the left-hand and right-hand loops, respectively:

$$6\,I_1 + 0\,I_2 + 0\,I_3 = -6,$$
$$0\,I_1 + 0\,I_2 + 9\,I_3 = 12,$$

where the right-hand sides represent the battery voltages in each loop, and the current terms with zeros in front are included just to make the equation structure clear. [The sign of a battery voltage depends on the direction in which the loop containing it is traversed relative to the battery's orientation. The directions here have been chosen to make the current terms positive, but that's not necessary (or even always possible). The -6 on the right-hand side of the first equation comes from the combination of the 6-V and 12-V batteries.]

We need one more equation to solve for the three unknown currents: that equation is found from Kirchhoff's **junction rule**, which says that the net flow of current into a junction (where two or more branches meet) must be zero; i.e., the amount flowing in equals the amount flowing out. Looking at the junction at the top of the circuit, we see I_1 and I_2 flowing in, and I_3 flowing out, so the junction rule equation would be $I_1 + I_2 - I_3 = 0$. Combining this equation with the other two, we have

$$6\,I_1 + 0\,I_2 + 0\,I_3 = -6, \tag{13}$$
$$0\,I_1 + 0\,I_2 + 9\,I_3 = 12, \tag{14}$$
$$1\,I_1 + 1\,I_2 - 1\,I_3 = 0. \tag{15}$$

(a) Use your Gaussian elimination function (*without* partial pivoting) and your back-substitution function to analyze this problem to find the unknown currents. You should find that you get error messages and nonsense output [the symbol "nan" (often written as "NaN") means "not a number"]. Why is that?

(b) Use your combined function (incorporating partial pivoting) from Exercise #3 part (b) to determine the currents. [Answer: $I_1 = -1.00$ amps, $I_2 = 2.33$ amps, $I_3 = 1.33$ amps; a negative current just means the actual direction is opposite what was assumed in drawing that current in the figure.]

Of course, something like the above approach can, and in fact must, be used in more complicated circuits that cannot be analyzed by hand.

Breakpoint Answers

Breakpoint 1

Multiply the second row by -2 and subtract it from the bottom row to get

$$\begin{pmatrix} 1 & 2 & 2 \\ 0 & 1 & 1.5 \\ 0 & 0 & -1 \end{pmatrix} \qquad \text{and} \qquad \begin{pmatrix} 3 \\ 1.5 \\ 1 \end{pmatrix}.$$

Finally, divide the last row by -1 to obtain

$$\begin{pmatrix} 1 & 2 & 2 \\ 0 & 1 & 1.5 \\ 0 & 0 & 1 \end{pmatrix} \qquad \text{and} \qquad \begin{pmatrix} 3 \\ 1.5 \\ -1 \end{pmatrix}.$$

Breakpoint 2

From the result above, we have $\begin{pmatrix} 1 & 2 & 2 \\ 0 & 1 & 1.5 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 1.5 \\ -1 \end{pmatrix}$.

Immediately, we get $x_3 = -1$ from the bottom row of the matrix equation. The second row gives $x_2 + 1.5\,x_3 = 1.5$; substituting $x_3 = -1$ we find $x_2 = 3$. Finally, the first row gives $x_1 + 2x_2 + 2x_3 = 3$, and substitution of the other x_i values yields $x_1 = -1$.

Breakpoint 3

$\begin{pmatrix} 1 & -1 & 3 \\ 0 & 3 & 6 \\ 0 & 1 & -2 \end{pmatrix}$ This matrix does not require pivoting: the element (1) in the first row, first column is larger (in absolute value) than the other elements in the first column, and the element in the second row, second column (3) is larger than the element in the same column in the third row.

$\begin{pmatrix} 1 & -1 & 3 \\ 0 & 3 & 2 \\ 0 & -6 & 5 \end{pmatrix}$ This matrix should be pivoted based on the discussion in the text. The element in the second row, second column (3) is smaller in absolute value than the element in the third row, second column (-6), so those two rows should be interchanged.

Scientist Profile

Dr. Ellen Ochoa is an American astronaut and, since 2012, the director of NASA's Johnson Space Center in Houston, Texas. Born in Los Angeles in 1958, she earned a Bachelor's degree in physics from San Diego State University in 1980, and a Masters and Ph.D. in electrical engineering from Stanford University (in 1981 and 1985, respectively). She is the first Hispanic director and second woman director of the Johnson facility, and the first Hispanic woman to have gone into space. Her first journey beyond Earth was a nine-day mission on the shuttle Discovery in 1993, which focused on observations to help understand the effect of solar activity on the Earth's climate and environment. She returned to space three times thereafter, in 1994, 1999, and 2002. As a Ph.D. student and later as a researcher at Sandia National Laboratories in New Mexico and at NASA Ames Research Center in California, Ochoa investigated optical systems for performing information processing, and she is the co-holder of three patents in the field. Dr. Ochoa has been awarded NASA's Distinguished Service Medal, the Exceptional Service Medal, the Outstanding Leadership Medal, and four Space Flight Medals. She also is a recipient of numerous other awards, including the Harvard Foundation Science Award, the Women in Aerospace Outstanding Achievement Award, The Hispanic Engineer Albert Baez Award for Outstanding Technical Contribution to Humanity, and the Hispanic Heritage Leadership Award. (This profile adapted from `http://www.nasa.gov/centers/johnson/about/people/orgs/bios/ochoa.html` and `http://www.jsc.nasa.gov/Bios/htmlbios/ochoa.html`.)