1 GSW… Sets of Systems

Often, we have to solve a whole series of sets of simultaneous equations of the form **y** = **Ax**, all of which have the same matrix **A**, but each of which has a different known vector **y**, and a different unknown vector **x**. Many problems in communications have this form: for example, if we can represent a received waveform in terms of the elements of a vector **y** and the communications channel in terms of a matrix **A**, we'll want to work out what the transmitted data **x** was. If the channel doesn't change, or at least changes only slowly, then we'll have a whole series of calculations to do, with the same **A** (channel), but different **y** (received waveforms).

Obviously one way to solve this problem is to work out the matrix inverse $\mathbf{A}^{-1}$, and then we can just pre-multiply each new known vector **y** by $\mathbf{A}^{-1}$. However, working out inverse matrices is a time-consuming process, and as usual, we're trying to avoid it. Techniques that side-step the need to invert the matrix include some that represent the matrix **A** in terms of the product of two or more other matrices. Such techniques are called *decompositions*.

To really see the benefit of these techniques, it's useful to have a look at just how painful the more obvious techniques are.

1.1 The Cost of Solving Systems of Simultaneous Equations

Consider the general problem of solving a square system[1] of linear simultaneous equations, such as:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,n} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n,1} & A_{n,2} & \cdots & A_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{0.1}$$

Suppose we do this by the usual method of Gaussian elimination until the matrix is upper-triangular and we can write the equation in the form **z** = **Ux**, and then back-substitution to find the elements of **x**. The number of operations required is:

1. Gaussian elimination to find **U** and **z**: $(2n^3 + 3n^2 - 5n) / 6$ MACs and $(n - 1)$ inversions.
2. Back-substitution to find **x**: $n(n + 1) / 2$ MACs and n inversions.

Except that it's not as bad as that: $(n - 1)$ of the n inversions required for the back-substitution stage have already been done as part of the Gaussian elimination stage. We already know the inverses of the all the terms on the main diagonal of the upper-triangular matrix (except the last one: $U_{n,n}$). That gives a grand total of: $(n^3 + 3n^2 - n) / 3$ MACs and n inversions.

If we were going to do another solution of these equations using the same matrix **A** but different vectors **y** by the same method, we'd need another $(n^3 + 3n^2 - n) / 3$ MACs and n inversions. This is not so good… we can do better. How about taking the matrix inverse?

[1] Quick reminder: a square system has a square matrix **A**, so that the vectors **x** and **y** have the same number of elements. Provided the matrix **A** is invertible (i.e. $\mathbf{A}^{-1}$ exists), this system has one unique solution.

1.1.1 Multiple Systems of Simultaneous Equations using Matrix Inverse

To calculate the solution to **y** = **Ax** by first taking the matrix inverse requires:

1. Inverting **A**: $n^3 + n^2 - 3n + 2$ MACs and n inversions.
2. Calculation of **x** from $\mathbf{A}^{-1}\mathbf{y}$: n^2 MACs.

That's a total of $n^3 + 2n^2 - 3n + 2$ MACs and n inversions. If we then need to do another calculation with the same **A** but a different **y**, we only need to do another n^2 MACs. It's taken a lot more calculations to solve the first system of equations, but any further systems with the same **A** are now much quicker.

However, we can do a lot better than this.

1.1.2 Reducing the Cost for Multiple Systems of Simultaneous Equations

If we solve the equations by first converting the original matrix equation **y** = **Ax** into the form:

$$\mathbf{Ly} = \mathbf{Ux} \qquad (0.2)$$

then we've got to:

1. Find **L** and **U** from **A**: $(n^3 - 3n + 2) / 2$ MACs and $(n - 1)$ inversions.
2. Pre-multiply **y** by **L**: $n(n - 1) / 2$ MACs.
3. Solve (**Ly**) = **Ux** by back-substitution: $n(n + 1) / 2$ MAC and n inversions.

however, again, it's not as bad as that, as $(n - 1)$ of the n inversions required in step 3 have already been done in step 1. We actually need only $(n^3 + 2n^2 - 3n + 1) / 2$ MACs and n inversions to do the whole thing.

If we're then given another vector **y** with the same **A**, we only have to do steps 2 and 3 again. That's another n^2 MACs, but no new inversions. That's the same number as we needed after we'd calculated the full matrix inverse, but this time we've had to do about half the number of calculations the first time. Can we do still better?

Yes, we can.

1.2 The LU Decomposition

I'll start this one with an example. Suppose we started with a matrix **A** given by:

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & -1 \\ 2 & 5 & 1 \\ 1 & 0 & 2 \end{bmatrix} \qquad (0.3)$$

and we decide we want to convert this into upper-triangular form, using the technique of reducing the equation **y** = **Ax** to **My** = **Ux** where **M** is unit lower-triangular[2] and **U** is upper-triangular. Going through the Gaussian elimination process on the extended matrix:

[2] Normally, I would use **L** to represent a lower-triangular matrix, but since I want to use **L** to represent the inverse of this matrix, I'll use **M** here. (The inverse of a lower-triangular matrix is also a lower-triangular matrix.)

$$\mathbf{A} = \left[\begin{array}{ccc|ccc} 1 & 3 & -1 & 1 & 0 & 0 \\ 2 & 5 & 1 & 0 & 1 & 0 \\ 1 & 0 & 2 & 0 & 0 & 1 \end{array}\right] \tag{0.4}$$

means first subtracting twice the first row from the second row to give:

$$\mathbf{A} = \left[\begin{array}{ccc|ccc} 1 & 3 & -1 & 1 & 0 & 0 \\ 0 & -1 & 3 & -2 & 1 & 0 \\ 1 & 0 & 2 & 0 & 0 & 1 \end{array}\right] \tag{0.5}$$

then subtracting the first row from the third row:

$$\mathbf{A} = \left[\begin{array}{ccc|ccc} 1 & 3 & -1 & 1 & 0 & 0 \\ 0 & -1 & 3 & -2 & 1 & 0 \\ 0 & -3 & 3 & -1 & 0 & 1 \end{array}\right] \tag{0.6}$$

and finally subtracting three times the second row from the third row:

$$\mathbf{A} = \left[\begin{array}{ccc|ccc} 1 & 3 & -1 & 1 & 0 & 0 \\ 0 & -1 & 3 & -2 & 1 & 0 \\ 0 & 0 & -6 & 5 & -3 & 1 \end{array}\right] \tag{0.7}$$

so we've transformed our original equation into the form:

$$\begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 5 & -3 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 & 3 & -1 \\ 0 & -1 & 3 \\ 0 & 0 & -6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{0.8}$$

Now, suppose we wanted to invert the lower-triangular matrix, we'd find that:

$$\begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 5 & -3 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 3 & 1 \end{bmatrix} \tag{0.9}$$

so we could pre-multiply both sides by this inverse, and write the original equations in the form:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 3 & 1 \end{bmatrix} \begin{bmatrix} 1 & 3 & -1 \\ 0 & -1 & 3 \\ 0 & 0 & -6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{0.10}$$

We've *decomposed* the matrix **A** into the product of a lower-triangular matrix and an upper-triangular matrix. This is the *LU-decomposition*. What's the point of this? Well, if we can get the matrix into this form, then we can solve the system of equations quite easily, first by writing:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 3 & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} \qquad (0.11)$$

and solving for **z** using forward-substitution; and then writing:

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 1 & 3 & -1 \\ 0 & -1 & 3 \\ 0 & 0 & -6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad (0.12)$$

and solving for **x** using back-substitution.

“All very well”, you might be saying, “but doesn’t decomposing into a lower-triangular and an upper-triangular, and then inverting the lower-triangular matrix use a lot of MACs?”

Well, no. It doesn’t. That’s the clever bit.

Think back to the stages of the Gaussian elimination we’ve just done to convert the matrix:

$$\begin{bmatrix} 1 & 3 & -1 \\ 2 & 5 & 1 \\ 1 & 0 & 2 \end{bmatrix} \qquad (0.13)$$

into upper triangular form:

1. Subtract **2** times the **first** row from the **second** row;
2. Subtract **1** times the **first** row from the **third** row;
3. Subtract **3** times the **second** row from the **third** row.

Now look at the unit-lower triangular matrix we ended up with:

$$\mathbf{M}^{-1} = \mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ \mathbf{2} & 1 & 0 \\ \mathbf{1} & \mathbf{3} & 1 \end{bmatrix} \qquad (0.14)$$

Notice anything interesting? The factors by which we multiplied the n^{th} row before subtracting it from the m^{th} row are identical to the value in the n^{th} column and m^{th} row of the unit-lower triangular matrix![3] There’s no need to calculate **L**, or $\mathbf{L}^{-1}$ separately, all you have to do is keep a note of how much of the various rows you are subtracting from other rows in the first matrix, and put these numbers into the corresponding place in the first matrix. That’s it. No additional calculations are required to get **L**, it comes ‘for free’.

A good way to see how this works is to start by expressing the original matrix in terms of the product of a unit matrix and the original matrix:

[3] I’m not prone to using explanation marks, but this fact is so remarkable and useful I think it deserves one.

$$\begin{bmatrix} 1 & 3 & -1 \\ 2 & 5 & 1 \\ 1 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 3 & -1 \\ 2 & 5 & 1 \\ 1 & 0 & 2 \end{bmatrix} \tag{0.15}$$

and then, step by step, to add in the relevant entries to the lower-triangular matrix as the Gaussian elimination process proceeds: the product of the two matrices remains constant all the way. (For a proof that this algorithm works, see the problems and solutions.)

$$\begin{aligned} \begin{bmatrix} 1 & 3 & -1 \\ 2 & 5 & 1 \\ 1 & 0 & 2 \end{bmatrix} &= \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 3 & -1 \\ 0 & -1 & 3 \\ 1 & 0 & 2 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 3 & -1 \\ 0 & -1 & 3 \\ 0 & -3 & 3 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 3 & 1 \end{bmatrix} \begin{bmatrix} 1 & 3 & -1 \\ 0 & -1 & 3 \\ 0 & 0 & -6 \end{bmatrix} \end{aligned} \tag{0.16}$$

As we've just seem, having reduced the matrix **A** to LU-form, we can then solve any set of simultaneous equations **y** = **Ax** by splitting up the calculation into **y** = **Lz** and **z** = **Ux**. **y** = **Lz** can be solved by forward substitution, and **z** = **Ux** by back-substitution.

For example, suppose we wanted to solve:

$$\begin{bmatrix} 2 \\ 3 \\ 5 \end{bmatrix} = \begin{bmatrix} 1 & 3 & -1 \\ 2 & 5 & 1 \\ 1 & 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{0.17}$$

we'd first do the LU-decomposition, to get:

$$\begin{bmatrix} 2 \\ 3 \\ 5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 3 & 1 \end{bmatrix} \begin{bmatrix} 1 & 3 & -1 \\ 0 & -1 & 3 \\ 0 & 0 & -6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{0.18}$$

then, solve:

$$\begin{bmatrix} 2 \\ 3 \\ 5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 3 & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} \tag{0.19}$$

by forward substitution, which gives:

$$\begin{aligned} 2 &= z_1 && \Rightarrow\ z_1 = 2 \\ 3 &= 2z_1 + z_2 && \Rightarrow\ z_2 = 3 - 4 = -1 \\ 5 &= z_1 + 3z_2 + z_3 && \Rightarrow\ z_3 = 5 - 2 + 3 = 6 \end{aligned} \tag{0.20}$$

and finally solve:

$$\begin{bmatrix} 2 \\ -1 \\ 6 \end{bmatrix} = \begin{bmatrix} 1 & 3 & -1 \\ 0 & -1 & 3 \\ 0 & 0 & -6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{0.21}$$

by back-substitution, which gives:

$$\begin{aligned} 6 &= -6x_3 & &\Rightarrow\ x_3 = -1 \\ -1 &= -x_2 + 3x_3 & &\Rightarrow\ x_2 = 1 - 3 = -2 \\ 2 &= x_1 + 3x_2 - x_3 & &\Rightarrow\ x_1 = 2 + 6 - 1 = 7 \end{aligned} \tag{0.22}$$

giving the final answer:

$$x = \begin{bmatrix} 7 \\ -2 \\ -1 \end{bmatrix} \tag{0.23}$$

1.2.1 The Cost of the LU-Decomposition

The LU-decomposition itself requires the simplest possible form of Gaussian elimination, and stores the factors (by which rows in the original matrix **A** multiplied before subtracting from the rows below) in another matrix **L**. The whole process takes $(n^3 - n) / 3$ MACs and $(n - 1)$ inversions. (That's about one-third of the number of MACs required to invert the matrix.)

Then, to solve any particular system of equations **y** = **LUx** requires a forward-substitution with the lower-triangular matrix **L** to find **z** where **y** = **Lz**, and since **L** is a unit lower-triangular matrix, this only takes $n(n - 1) / 2$ MACs and no inversions.

Finally, there is the back-substitution to find **x** from **z** = **Ux**. **U** is not a unit upper-triangular matrix, so this takes $n(n + 1) / 2$ MACs and n inversions, but $(n - 1)$ of these have already been done in the first stage.

That's a grand total of:

$$\frac{n^3 - n}{3} + \frac{n(n-1)}{2} + \frac{n(n+1)}{2} = \frac{n^3 + 3n^2 - n}{3} \tag{0.24}$$

MACs and n inversions. Which is exactly the same number as we needed to solve the equations in the most straightforward way, back in section 1.1; but this time, for any subsequent systems with the same matrix **A**, we just need to do another:

$$\frac{n(n-1)}{2} + \frac{n(n+1)}{2} = n^2 \tag{0.25}$$

MACs and no new inversions. That's the best so far.

Can we do better? Well, sometimes, yes. But before that, there's a slight problem that I've been ignoring for this entire chapter so far…

1.3 PLU Decomposition

You might have spotted the problem: we can't express just any matrix **A** in terms of the product of a lower-triangular and an upper-triangular matrix. This decomposition only works if we can turn **A** into an upper-triangular matrix without any pivoting. If we have to pivot, then we can still do a decomposition, but we're no longer going to get the product of a lower-triangular matrix times an upper-triangular matrix.

For example, consider the following matrix, which needs a pivot:

$$\begin{bmatrix} 4 & 1 & 5 & -8 \\ 8 & 2 & 8 & -15 \\ 2 & -1 & 3 & -2 \\ 6 & 0 & 4 & -2 \end{bmatrix} \tag{0.26}$$

and going through the Gaussian elimination into the form **Ly** = **Ux** gives:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -0.5 & 0 & 1 & 0 \\ -2 & 1 & 0 & 0 \\ 3 & -2 & -1 & 1 \end{bmatrix}\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} 4 & 1 & 5 & -8 \\ 0 & -1.5 & 0.5 & 2 \\ 0 & 0 & -2 & 1 \\ 0 & 0 & 0 & 6 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \tag{0.27}$$

and since:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -0.5 & 0 & 1 & 0 \\ -2 & 1 & 0 & 0 \\ 3 & -2 & -1 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 \\ 0.5 & 1 & 0 & 0 \\ 1.5 & 1 & 2 & 1 \end{bmatrix} \tag{0.28}$$

we could write the original equations as:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 \\ 0.5 & 1 & 0 & 0 \\ 1.5 & 1 & 2 & 1 \end{bmatrix}\begin{bmatrix} 4 & 1 & 5 & -8 \\ 0 & -1.5 & 0.5 & 2 \\ 0 & 0 & -2 & 1 \\ 0 & 0 & 0 & 6 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \tag{0.29}$$

This is a decomposition, but it's not an LU-decomposition, since the first matrix is not lower triangular. The first matrix can, however, be made lower-triangular by swapping the second and third rows. Such matrices (ones that can be made lower-triangular by swapping the order of the rows) are sometimes known as *psychologically lower-triangular matrices*. Bit of a strange name, perhaps.

This decomposition can still be used to calculate the results of other systems of equations with the same **A** but different **y**, but the terms of the intermediate vector **z** can't be found from forward-substitution any more, the elements of **z** will have to be found in some different order. It doesn't take any more MACs, but the programming can be a bit awkward, keeping track of the right order to use, and of which terms in the 'psychologically lower-triangular matrix' are zero. One simple way to keep track results in the *PLU-decomposition*.

Ideally, we'd swap the rows of the matrix round before we started, so that no pivoting was then required. Swapping rows around in matrices can be done using a *permutation matrix* (a square matrix that has exactly one '1' in each row and column, and zeros elsewhere). For example:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix} = \begin{bmatrix} A_{2,1} & A_{2,2} & A_{2,3} \\ A_{1,1} & A_{1,2} & A_{1,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix} \tag{0.30}$$

swaps over the first and second row in the matrix **A**.

If the matrix is first multiplied by a suitable permutation matrix **P** so that no pivoting is necessary, and then decomposed into a lower-triangular matrix **L** and an upper-triangular matrix **U**, then we get a decomposition of **A** into the product of three matrices: **P**, **L** and **U**. This is known as a *PLU-decomposition*.

How do you know which permutation matrix will avoid any need to pivot before you start? Well, usually you don't. You have to keep track of things as you go. Fortunately, that's quite simple. Instead of starting with the product of a unit matrix and the initial matrix **A**:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 4 & 1 & 5 & -8 \\ 8 & 2 & 8 & -15 \\ 2 & -1 & 3 & -2 \\ 6 & 0 & 4 & -2 \end{bmatrix} \tag{0.31}$$

we put another unit matrix onto the start of our matrix product:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 4 & 1 & 5 & -8 \\ 8 & 2 & 8 & -15 \\ 2 & -1 & 3 & -2 \\ 6 & 0 & 4 & -2 \end{bmatrix} \tag{0.32}$$

and we use the new first matrix to keep track of the pivots. The first matrix then ends up as a permutation matrix (since all permutation matrices are just unit matrices with their rows swapped over), the second matrix ends up as a lower-triangular matrix, and the third matrix as an upper-triangular matrix.

This method works by applying the useful fact that when you have the product of two matrices **AB**, swapping any two rows of **B** and the corresponding two columns of **A** doesn't change the product **AB**. (Very useful result this.)

For example, starting with the matrix above, after the first Gaussian elimination step we'd get:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 4 & 1 & 5 & -8 \\ 0 & 0 & -2 & 1 \\ 2 & -1 & 3 & -2 \\ 6 & 0 & 4 & -2 \end{bmatrix} \tag{0.33}$$

and after the second and third steps, we get:

$$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\2&1&0&0\\0.5&0&1&0\\1.5&0&0&1\end{bmatrix}\begin{bmatrix}4&1&5&-8\\0&0&-2&1\\0&-1.5&0.5&2\\0&-1.5&-3.5&10\end{bmatrix} \tag{0.34}$$

and now we need to pivot: swapping the second and third rows of the first matrix. We can preserve the product by also swapping the second and third columns of the second matrix:

$$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\2&0&1&0\\0.5&1&0&0\\1.5&0&0&1\end{bmatrix}\begin{bmatrix}4&1&5&-8\\0&-1.5&0.5&2\\0&0&-2&1\\0&-1.5&-3.5&10\end{bmatrix} \tag{0.35}$$

but this means the middle matrix is no longer lower-triangular. So, we then swap the second and third rows of the middle matrix, and the second and third columns of the first matrix, to get the second matrix back into lower-triangular form. Once again, we don't change the value of the product. This gives:

$$\begin{bmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0.5&1&0&0\\2&0&1&0\\1.5&0&0&1\end{bmatrix}\begin{bmatrix}4&1&5&-8\\0&-1.5&0.5&2\\0&0&-2&1\\0&-1.5&-3.5&10\end{bmatrix} \tag{0.36}$$

and now we can carry on:

$$\begin{bmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0.5&1&0&0\\2&0&1&0\\1.5&1&0&1\end{bmatrix}\begin{bmatrix}4&1&5&-8\\0&-1.5&0.5&2\\0&0&-2&1\\0&0&-4&8\end{bmatrix} \tag{0.37}$$

and finally:

$$\begin{bmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0.5&1&0&0\\2&0&1&0\\1.5&1&2&1\end{bmatrix}\begin{bmatrix}4&1&5&-8\\0&-1.5&0.5&2\\0&0&-2&1\\0&0&0&6\end{bmatrix} \tag{0.38}$$

and that's it: a PLU decomposition.

Note for every pivot required swapping rows *n* and *m* in the last matrix (the one that will become upper-triangular), we need to:

1. swap rows *n* and *m* in all three matrices; then
2. swap columns *n* and *m* in the middle matrix,

in order to preserve the product of the three matrices. It's a bit more work than the LU-decomposition, but it doesn't require any more multiplications or divisions, it's just moving terms around[4]; and provided **A** is invertible, it's guaranteed to work.

Now, back to trying to do better than the LU-decomposition.

1.4 The Cholesky Decomposition

In many real problems[5] the matrix **A** in the expression **y** = **Ax** is Hermitian and positive semi-definite[6], and any Hermitian positive semi-definite matrix can be decomposed into the form $\mathbf{A} = \mathbf{L}\mathbf{L}^H$. There are an infinite number of ways to do this, but it turns out that it is always possible to choose **L** so that **L** is lower-triangular, and therefore $\mathbf{L}^H$ (the complex transpose of **L**) is upper-triangular.

Consider the general form of such a matrix:

$$\mathbf{A} = \begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} & \cdots & A_{1,n} \\ A_{2,1} & A_{2,2} & A_{2,3} & \cdots & A_{2,n} \\ A_{3,1} & A_{3,2} & A_{3,3} & \cdots & A_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{n,1} & A_{n,2} & A_{n,3} & \cdots & A_{n,n} \end{bmatrix} \tag{0.39}$$

The Cholesky decomposition expresses this matrix as the product of a lower-triangular matrix and the complex transpose of the lower-triangular matrix:

$$\begin{bmatrix} L_{1,1} & 0 & 0 & \cdots & 0 \\ L_{2,1} & L_{2,2} & 0 & \cdots & 0 \\ L_{3,1} & L_{3,2} & L_{3,3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ L_{n,1} & L_{n,2} & L_{n,3} & \cdots & L_{n,n} \end{bmatrix} \begin{bmatrix} L_{1,1} & L_{2,1}^* & L_{3,1}^* & \cdots & L_{n,1}^* \\ 0 & L_{2,2} & L_{3,2}^* & \cdots & L_{n,2}^* \\ 0 & 0 & L_{3,3} & \cdots & L_{n,3}^* \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & L_{n,n} \end{bmatrix} = \begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} & \cdots & A_{1,n} \\ A_{2,1} & A_{2,2} & A_{2,3} & \cdots & A_{2,n} \\ A_{3,1} & A_{3,2} & A_{3,3} & \cdots & A_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{n,1} & A_{n,2} & A_{n,3} & \cdots & A_{n,n} \end{bmatrix} \tag{0.40}$$

To calculate the terms of **L**, we can multiply out the terms in the matrix product one-by-one, starting with the first row, first column. That gives:

$$L_{1,1}^{\;2} = A_{1,1} \tag{0.41}$$

[4] I say 'just' moving terms around. In fact, moving terms around in memory can take just as long as doing MAC operations with some hardware. It's something to avoid if you can.

[5] For example the solution to the linear minimum least-square error problem, and the Wiener-Hopf auto-correlation matrix.

[6] Reminder: Hermitian means that $\mathbf{A}^H = \mathbf{A}$, and positive semi-definite means that $\mathbf{x}^H\mathbf{A}\mathbf{x}$ is positive or zero for all non-zero vectors **x**. For more about the properties of Hermitian positive semi-definite matrices, see the chapter on Matrix Properties.

and immediately we see a problem. $L_{1,1}$ is the square-root of $A_{1,1}$? What's the cost of that? Again, the answer depends on the architecture being used, but here, I'll assume it's broadly similar to taking an inverse[7]. We'll also need the inverse of $L_{1,1}$.

Then, all the terms below $L_{1,1}$ in the first column of **L** (and hence all the terms on the first row of $\mathbf{L}^H$) can be readily determined using one MAC each:

$$L_{i,1} = \frac{1}{L_{1,1}} A_{i,1} \tag{0.42}$$

So, that's the first column of **L** (and the first row of $\mathbf{L}^H$) calculated at a cost of one square-root, one inversion, and $(n - 1)$ MACs. Onto the second column, and since the term in the first row has already been calculated (it's just $L^*_{2,1}$), the next term is on the main diagonal, and looking at the product of the second row of **L** and second column of $\mathbf{L}^H$, we get:

$$L_{2,2}{}^2 = A_{2,2} - L_{2,1}L^*_{2,1} \tag{0.43}$$

that's one MAC and a square root. The terms underneath $\mathrm{L}_{2,2}$ are then calculated using:

$$L_{i,2} = \frac{1}{L_{2,2}}\left(A_{i,2} - L_{i,1}L^*_{2,1}\right) \tag{0.44}$$

which takes two MACs each (having done the inversion of $L_{2,2}$).

In general, we can calculate the terms on the main diagonal of **L** using:

$$L_{i,i}{}^2 = A_{i,i} - \sum_{j=1}^{i-1} L_{i,j}L^*_{i,j} \tag{0.45}$$

and each takes $(i - 1)$ MACs and a square root; and the terms under the main diagonal are given by:

$$L_{i,j} = \frac{1}{L_{j,j}}\left(A_{i,j} - \sum_{k=1}^{j-1} L_{i,k}L^*_{j,k}\right) \tag{0.46}$$

which takes j MACs each (and the inversion of $L_{j,j}$). Writing a matrix in terms of the number of MACs required to calculate each term in the decomposition produces:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & \cdots & 0 \\ 1 & 2 & 2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 2 & 3 & \cdots & n-1 \end{bmatrix} \tag{0.47}$$

and the total number of MACs required to calculate the terms on the main diagonal is then:

[7] It can be done in a similar way to taking the inverse: start with an approximate solution using a look-up table, and iterate to a more accurate solution by squaring the current estimate and comparing it to $A_{1,1}$.

$$0+1+2+3+...+(n-1)=\frac{n(n+1)}{2} \tag{0.48}$$

and to calculate the terms below the main diagonal requires:

$$(n-1)+2(n-2)+3(n-3)+...+(n-1)=\frac{n^3-n}{6} \tag{0.49}$$

which makes a grand total of:

$$\frac{n^3-n}{6}+\frac{n^2+n}{2}=\frac{n^3+3n^2+2n}{6} \tag{0.50}$$

MACs, plus of course the n square roots, and the $(n - 1)$ inversions. For large n, that's about half of the number of MACs that the LU-decomposition requires, and one-sixth the number required to invert the matrix.

The (slight) disadvantage is that now neither of the matrices that **A** has been decomposed into is a unit matrix (i.e. has ones on the main diagonal), so the process of forward and back substitution required to solve the equations **y** = **Ax** take a total of $n(n + 1)$ MACs, not just n^2.

If you've got a large, Hermitian positive-definite matrix **A**, this is a good technique to use.

1.5 Summary of the Costs of Solving y = Ax

In summary then, we've looked at five ways to solve a series of systems of simultaneous equations of the form **y** = **Ax**, with the same matrix **A**, but different **y** each time. The costs are:

		MACs	Inversions and Square-Roots
Just solve **y** = **Ax** from scratch each time		$(n^3 + 3n^2 - n) / 3$	n
Invert **A**	First time:	$n^3 + n^2 - 3n + 2$	n
	Subsequent:	n^2	0
Convert to **Ly** = **Ux**	First time:	$(n^3 + 2n^2 - 3n + 1) / 2$	n
	Subsequent:	n^2	0
LU-Decomposition	First time:	$(n^3 + 3n^2 - n) / 3$	n
	Subsequent:	n^2	0
Cholesky-Decomposition	First time:	$(n^3 + 3n^2 + 2n) / 6$	$2n$
	Subsequent:	$n(n + 1)$	0

1.6 Tutorial Questions

1) Decompose the following matrix into the product of the inverse of a known lower triangular matrix **L**, and an upper-triangular matrix **U**.

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

2) Prove that the product of two matrices **AB** remains constant if:

1. Any two rows of **B**, and the corresponding two columns of **A** change places;
2. A row of **B** is multiplied by a constant, and the corresponding column of **A** is divided by the same constant;
3. Row *n* of **B** is added to row *m* of **B**, and column *m* of **A** is subtracted from column *n* of **A**;

and hence show that the algorithm to produce the LU-decomposition works.

3) Is it possible to perform an LU-decomposition on the matrix $\begin{bmatrix} 1 & 3 & -1 \\ 2 & 6 & 1 \\ 1 & 1 & 2 \end{bmatrix}$?

If not, perform a PLU-decomposition on this matrix.

4) Consider the matrix A = $\begin{bmatrix} 6 & 5 & 1 \\ 5 & 10 & 1 \\ 1 & 1 & 14 \end{bmatrix}$. Perform a decomposition of this matrix into the form $\mathbf{LL}^H$ where **L** is a lower-triangular matrix.

5) If partial pivoting (i.e. swapping rows but not columns) in a matrix **A** is performed, we end up with a PLU decomposition, rather than an LU decomposition. What happens if full pivoting is allowed (i.e. both the rows and the columns of the matrix **A** can be swapped over to get the largest possible entry on the main diagonal)?

6) Suppose I tried to solve a series of systems of simultaneous equations **y** = **Ax** by first converting the matrix **A** to the form **By** = **Dx** by Gaussian elimination, where **D** is a diagonal matrix, and then for each new value of **y**, first compute **By**, then just divide each element in the vector **By** by the corresponding value in the matrix **D**.

How many MACs and inversions does that take? Is this better than calculating the full matrix inverse?