Solving Square Triangular Systems

When you were taught Gaussian Elimination, you were probably taught to stop the elimination process when you achieved an "upper triangular form", at which point you could solve the system by backward substitution. By *upper triangular* we mean that all matrix entries below the main diagonal are 0's.

Let's teach a computer to solve the upper triangular system of linear equations $Uy = b$, where $U \in \mathbb{R}^{n \times n}$. It is called backward substitution because the last equation has the form $u_{nn}y_n = b_n$, and can be solved as

$$y_n = b_n/u_{nn}.$$

Now since we know y_n, the penultimate equation $u_{n-1,n-1}y_{n-1} + u_{n-1,n}y_n = b_{n-1}$ can be solved as

$$y_{n-1} = (b_{n-1} - u_{n-1,n}y_n)/u_{n-1,n-1}.$$

And so we count backward, from n down to 1, finding one solution coefficient at each equation. In pseudocode:

```
for j = n : -1 : 1 do the following:
    find y(j)
end for j
```

So let's see how to implement "find y(j)". The j^{th} equation is

$$u_{jj}\ y_j + u_{j,j+1}\ y_{j+1} + \ldots + u_{jn}\ y_n = b_j,$$

and since at this point we know $y_{j+1}, \ldots, y_n$, we can solve this for y_j:

$$y_j \ = \ (b_j - \sum_{i=j+1}^{n} u_{ji}y_i) \ / \ u_{jj}.$$

The pseudocode now might be

```
for j = n : -1 : 1 do the following:
    y(j) = ( b(j) - u(j,j+1:n) * y(j+1:n) ) / u(j,j)
end for j
```

That's it! It is that simple: y(j) = (scalar - dot product) / scalar

Of course this only works when the diagonal of U has no zero elements; but that is precisely when U is nonsingular. Therefore (in exact arithmetic, at least), this method always returns the solution when a unique solution exists.

What about finite precision arithmetic? It does remarkably well, in fact if $\bar{y}$ is the *computed* solution, then there exists a matrix E such that

$$(U + E)\bar{y} = b, \quad \text{where } \ |E| \leq n\boldsymbol{\mu}|U| + \mathrm{O}(\boldsymbol{\mu}^2)$$