Ok, so first things first - cubic spline interpolation. You're probably fine with this bit, but I want to write it out to solidify the notation system I'm using, otherwise it could be confusing.

- We have a set of points (x_i, y_i) such that

$$S_i(x_i) = y_i, i = 0, ..., n \tag{1}$$

  S is our set of unknown spline functions - we know the set of $n + 1$ fixed points they must pass through, but not what the functions are.

- Since we have $n + 1$ points, we have n intervals, so n spline functions. For each cubic spline, we will have four unknown coefficients, so in total we get $4n$ unknown coefficients.

- We can set a certain number of conditions on our problem, to help eliminate some of the unknowns. For example, with natural splines, we know the second derivatives at the very end points must be zero, ie

$$S''(x_0) = S''(x_n) = 0 \tag{2}$$

- And we also know, because each spline needs to join with those preceding and following it, that

$$S''_{i-1}(x_i) = S''_i(x_i) \tag{3}$$

  (ie the function is not discontinuous in the second derivative at any point).

- So for each spline equation, let the coefficients be (a, b, c, d). Then

$$S_i(x) = a_i + b_i x + c_i x^2 + d_i x^3 \tag{4}$$

  and obviously when $x = x_i$

$$a_i + b_i x_i + c_i x_i^2 + d_i x_i^3 = y_i \tag{5}$$

  Now, we also know $S_i(x_{i+1}) = y_{i+1}$, ie

$$a_i + b_i x_{i+1} + c_i x_{i+1}^2 + d_i x_{i+1}^3 = y_{i+1} \tag{6}$$

  So this gives us two equations for the coefficients. (Note that these equations are simplified if we use the parameterisation seen in wolfram, because then x_i is equivalent to zero and x_{i+1} is equivalent to 1, so both become simple linear equations). We need two more equations to solve it properly, though!

- These next two come from our inflection conditions - the second derivative of the spline equations must match at each join point. So

$$S_i''(x_i) = 2c_i + 6d_i x_i \tag{7}$$

and

$$S_i''(x_{i+1}) = 2c_i + 6d_i x_{i+1} \tag{8}$$

Now we have four equations for every segment except for the first and last segments, and these two missing equations we obtain using our conditions for a natural spline.

- So we arrange all these equations into a linear matrix equation. We know the matrix on the left-hand-side (it's the basic tridiagonal arrangement). The right hand side will be a function of the known solution points y_i. The vector on the left hand side contains our unknowns. In the non-parameterized version, the unknowns d_i give us the four coefficients in the following way:

$$\begin{aligned} a_i &= \frac{d_{i+1} - d_i}{6h} && (9)\\ b_i &= \frac{1}{2} d_i && (10)\\ c_i &= \frac{y_{i_1} - y_i}{h} - \frac{d_{i+1} + 2d_i}{6} h && (11)\\ d_i &= y_i && (12) \end{aligned}$$

where $x_i = x_{i-1}$. The parameterised version will look a little simpler (which is one reason wolfram uses it), but is mathematically equivalent since parameterising is equivalent to mucking around with h in the manner above. If you use the parameterised version of the method, then the relationship between the coefficients and the solutions to the linear equation is as given in Wolfram. In any case, we find these unknowns by now inverting the matrix, either directly or by using gaussian elimination (and hence the streamlined gaussian elimination applicable to tridiagonal matrices).

We can write the system as

$$T\mathbf{d} = \mathbf{m} \tag{13}$$

where T is the tri-diagonal matrix which I can't be bothered writing out, but consists of known integers ie see the bottom of this page http://mathworld.wolfram.com/CubicSpline.html . **m** is a vector of knowns which is derived from the fixed points y_i, and **d** is our vector of unknowns, which corresponds to the values of the first derivatives of the spline at the junction points between the splines (so **d** is not actually the solution to the problem, but it is very close to it - once you have **d** then getting the spline coefficients is trivial). This notation clashes a bit with the page you linked me to, sorry about that. Hopefully it's clear.

This is a simple linear system, which can easily be solved using the standard gaussian elimination procedure, as I'm sure you know. The 'efficient' method described here is just an algebraic formalization of the division and subtraction procedures you would make normally - basically because the matrix is regular and predictable, we can create an algorithm for the division, subtraction and back-substitution. So it's actually exactly the same as a standard elimination method - you divide a row through by the first co-efficient so your first unknown has a unity coefficient, and then you select a suitable row to subtract from, etc., repeat. If you like I can go into more detail on traditional Gaussian elimination, this might make things clearer. The algorithm here is just filtering several elimination steps down into one, and it can do that because the matrix has some predictable properties.