Supplementary Chapter 3 material: Operation Counts

This is FYI - you don't have to do anything with it. But, it explains what I mean when I say that one method of solving a system takes more work than another, and that Cramer's Rule is the "worst" way to solve.

The basic operations that a computer performs are addition and multiplication (and multiplication is really addition and bit shifting). When we construct an algorithm to solve a process, we are interested not only in if it works, but how efficient it is - does it do what it's supposed to do with the least possible amount of effort? Operation counts give us an idea of the 'size' of an algorithm, and allow us to compare different algorithms that solve the same problem.

Count the operations:

How many operations (additions and multiplications) does it take to multiply a 2 by 2 matrix with another 2 by 2 matrix?

Two approaches: (1) 
\[
\begin{bmatrix}
    a_{11} a_{12} \\
    b_{11} b_{12}
\end{bmatrix}
\begin{bmatrix}
    c_{11} \\
    c_{12}
\end{bmatrix}
= 
\begin{bmatrix}
    a_{11} c_{11} + a_{12} c_{12} \\
    b_{11} c_{11} + b_{12} c_{12}
\end{bmatrix}
\]

(2) better, b/c we can generalize -

typical element for \( C \) is \( c_{ij} \)

is \( c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \)

each element has 2 multiplies, 1 add. There are \( 4 \) elements total.

4 (2 + 1) = 8 + 4 = 12

How many operations does it take to multiply an \( n \) by \( n \) matrix with another \( n \) by \( n \) matrix?

Typical element \( c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \) multiply = \( n \)

Add = \( n - 1 \)

How many elements? \( n^2 \)

Total: \( n^2 (n + n - 1) = n^2 (2n - 1) = 2n^3 - n^2 \)

Using the formula we just derived, how many operations to multiply two 100 by 100 matrices? Two 1000 by 1000 matrices?

\[
2(100)^3 - (100)^2 = 1,990,000
\]

\[
2(1000)^3 - (1000)^2 = 1,999,000,000
\]

The key idea here is that you can work out an operation count formula for a process in general, based on the size of the matrix (or matrices) involved.
O notation

In the expression for the operation count of $n$ by $n$, the dominant term is $n^3$, in the sense that, for large values of $n$ especially, the fact that you're cubing the $n$ does a great deal more to affect up the total than the fact that you're also subtracting an $n^2$. In fact,

$$2n^3 - n^2 \leq 2n^3 \quad n \geq 1$$

In general, if our polynomial expression $P(n)$ can be bounded above:

$$P(n) \leq kn^n \quad n \geq c \text{ for some } c$$

we say that $P(n)$ is $O(n^p)$ [read "oh n to the p"]; i.e. on the order of $n^p$. So matrix multiplication of two equally sized $n$ by $n$ matrices is an $O(n^3)$ process.

We'd be interested if we could find another algorithm for multiplication that was $O(n^2)$. That would be a fantastic improvement; instead of needing to perform some multiple of $1000^3 = 1,000,000,000$ operations to multiply two 1000 by 1000 matrices, we'd only need a multiple of $1000^2 = 1,000,000$ operations. Barring that, an algorithm that was still $O(n^2)$ but had a smaller value for the constant multiplier would be at least a slight improvement. And finally, there's other ways to speed up a process besides reducing the op count (although reducing the op count generally doesn't hurt). We'll come back around to that in a little bit.

Question: on what order is the multiplication of an $n$ by $n$ matrix with an $n$ by 1 matrix?

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j} \quad n \text{ mult } n-1 \text{ add}$$

Entries in $c$? $n$.

Total $n(n+n-1) = 2n^2 - n \quad O(n^2)$

Math with $O$: If I perform an $O(n^3)$ process followed by an $O(n^3)$ process, on what order is the total process?

$$O(n^3)$$

If I perform an $O(n^3)$ process followed by an $O(n^4)$ process, on what order is the total process?

$$O(n^4) \quad (the \ n^4 \ term \ dominates)$$
Gaussian Elimination

Gaussian Elimination and all its variations (Gauss Jordan elimination, finding an inverse through Gaussian Elimination) are \( O(n^3) \) processes as well. In fact, it has been proven that there is no process for solving a general system of \( n \) equations \( n \) unknowns that is less than \( O(n^3) \) (for systems with special properties, such as ones where the coefficient matrix is symmetric, we may be able to improve on this).

Let's perform an operation count on the number of operations needed to reduce a matrix to upper triangular form using Gaussian elimination:

Assume \( n \times (n+1) \) augmented matrix.

Assume 1st stage, using \( R_1 \) as pivot.
- How many multiplies? \( n+1 \)
- How many adds? \( n+1 \)
- Total: \( 2(n+1) \)
- How many rows do you need to zero out? \( n-1 \)
- Grand total: \( 2(n+1)(n-1) \)

Second stage, \( R_2 \) as pivot:
- Multiplies: \( n \)
- Adds: \( n \)
- Total: \( 2n \)
- Rows: \( n-2 \)
- Grand total: \( 2n(n-2) \)

Keep going. Down to the last stage.
- Multiplies: \( 3 \)
- Adds: \( 3 \)
- Total: \( 2(3) \)
- Rows: \( 1 \)
- Grand total: \( 2(3)(1) \)

Now, add them all up:

\[
2(3)(1) + 2(4)(2) + \ldots + 2(n)(n-2) + 2(n+1)(n-1)
\]

Use summation notation:
\[
2 \sum_{i=2}^{n} (i+1)(i-1) \\
= 2 \sum_{i=2}^{n} (i^2 - 1) \\
= 2 \left[ \left( \sum_{i=1}^{n} i^2 - 1 \right) - (1^2 - 1) \right] \quad \text{* shift index} \\
= 2 \sum_{i=1}^{n} (i^2 - 1) \\
= 2 \left[ \left( \sum_{i=1}^{n} i^2 - \sum_{i=1}^{n} 1 \right) \right] \quad \text{* summation formulas} \\
= 2 \left[ \frac{n(n+1)(2n+1)}{6} - n \right] \\
= 2 \left[ \frac{2n^3 + 3n^2 + n}{6} - n \right] \\
= 2 \left[ \frac{1}{3} n^3 + \frac{1}{2} n^2 - \frac{5}{6} n \right] \\
= \frac{2}{3} n^3 + n^2 - \frac{5}{3} n \\
O(n^3) \text{ with leading coeff. } \frac{2}{3}
\]
Now, you will always see the $O(n^3)$ figure, with the $\frac{2}{3}$ coefficient, reported for Gaussian Elimination. However, different books will show different values for the lower terms? Why? Depends on your assumptions. What else could we take into consideration?

- The first main thing is obtaining the zero – if you know you’re going to get that zero, you don’t really need to perform the multiply and add on this position! A GE algorithm on a machine generally won’t take the step to explicitly compute the zero it knows is there. This reduces the operation count by one less multiplication on each stage, and one less addition on each step in the stage.

- Not all counts use the augmented matrix. Some count only the operations to reduce the coefficient matrix (and count the right hand side separately). This gives one less element to mult and add each time.

- We still haven’t solved the system. We’d need back substitution to continue solving for the variables. Turns out back sub is an $O(n^2)$ process, so it won’t affect the dominant term, but it will change the lower order terms. Some counts include the back sub process along with the elimination.

All the GE related routines are $O(n^3)$. However, for Gaussian Elimination with back substitution, the dominant term is $\frac{2}{3}n^3$. For Gauss - Jordan Elimination, it’s $1n^3$. For computing an inverse matrix, and then multiplying by the inverse, it’s $3n^3$. For computing an inverse matrix, and then multiplying by the inverse, it’s $3n^3$. So in practice, we use GE with back substitution to solve systems.

Cramer’s Rule? Depends on how you compute the determinant. If you use Gaussian Elimination to get zeros, that’s $O(n^3)$. You’d then need to multiply down the diagonal. That adds only $n$ multiplications, so still only $O(n^3)$. But, you have to repeat that for each variable. That’s an $O(n^3)$ process performed $n$ times. So now, you’re up to $O(n^4)$. Compare the operations for solving a 1000 by 1000 system, GE vs Cramer, and you’ll see the impact.

Oh, and if you get your determinants by cofactor expansion ... that’s an $O(n!)$ process repeated $n$ times. Go figure out what $n n!$ is for $n = 1000$!

**Operations and time**

You’d think there’d be a direct correlation, right? Well, there is, and there isn’t. Those of you who are computing experts, what else can speed up an algorithm? Parallel processing is the main thing - if you can break up your algorithm into chunks, and feed it to different computers (or run the processes at the same time on one computer) where they can run the parts *simultaneously*, you might be doing more operations ... but get your result faster. GE as we perform it doesn’t chunk well; you need to be operating on the entire matrix from start to finish. Cramer’s Rule would chunk ... but each individual determinant would still involve GE, so no improvement there. There are folks who do research in this area, though, coming up with algorithms that have better performance (although possibly higher op counts). For a general comparison though, looking at op count is a good way to rank algorithms in terms of efficiency.