Computing the Permanent

The permanent of an N-by-N square matrix $A = (a_{i,j})$ is defined as

$$\operatorname{per}(A) = \sum_{\sigma \in P(N)} \prod_{i=1}^{N} a_{i,\sigma(i)} \tag{1}$$

where $P(N)$ is the set of permutations of the N-set $\{1, \ldots, N\}$ [1]. The definition can be extended to M-by-N rectangular matrices with $M \leq N$ [1],

$$\operatorname{per}(A) = \sum_{\sigma \in P(N,M)} \prod_{i=1}^{M} a_{i,\sigma(i)}. \tag{2}$$

Since $\operatorname{per}(A) = \operatorname{per}(A^T)$, this is sufficient for computing the permanent of all rectangular matrices.

Equation (1) is similar to the definition of the determinant,

$$\det(A) = \sum_{\sigma \in P(N)} \operatorname{sgn}(\sigma) \prod_{i=1}^{N} a_{i,\sigma(i)}, \tag{3}$$

except that the signatures of the permutations are not taken into account; this exclusion notably makes the permanent much more difficult to compute than the determinant, because it means that decomposition methods can no longer be used [2]. In fact, Valiant proved in 1979 that the computation of the permanent is in the complexity class #P-complete, i.e. neither in P nor in NP [3].

The permanent commonly appears in problems related to quantum mechanics, notably in our recent Coupled Cluster and quasi-particle ansatz for N-electron systems [4, 5, 6], so it is still worthwhile to pursue more efficient algorithms than the naive one suggested by Equation (1), which has time complexity $\mathcal{O}(N!N)$.

To date, there are two classes of algorithm considered to be the fastest; one due to Ryser in 1963 [7], and one due to Glynn in 2010 (although it was found independently by others prior to this) [2, 8]. The Ryser algorithm is based on the inclusion-exclusion principle and is given by the formula (for N-by-N square matrices)

$$\operatorname{per}(A) = (-1)^N \sum_{k=1}^{N} \sum_{\sigma \in P(N,k)} (-1)^k \prod_{i=1}^{N} \sum_{j=1}^{k} a_{i,\sigma(j)} \tag{4}$$

where $P(N,k)$ is the set of k-permutations of the N-set $\{1, \ldots, N\}$ [1]. Equation (4) is a special case of Ryser's original formula, which he had defined for M-by-N rectangular matrices with $M \leq N$ [7]:

$$\operatorname{per}(A) = \sum_{k=0}^{M-1} \sum_{\sigma \in P(N,M-k)} (-1)^k \binom{N-M+k}{k} \prod_{i=1}^{M} \sum_{j=1}^{M-k} a_{i,\sigma(j)}. \tag{5}$$

The Glynn algorithm is based on invariant theory, derived from the polarization identity for a symmetric tensor, and is given by the formula (for N-by-N square matrices)

$$\operatorname{per}(A) = \frac{1}{2^{N-1}} \cdot \sum_{\delta} \left(\sum_{k=1}^{N} \delta_k \right) \prod_{j=1}^{N} \sum_{i=1}^{N} \delta_i a_{i,j} \tag{6}$$

where the outer sum is over all 2^{N-1} vectors $\delta = \{+1_1, \pm 1_2, \ldots, \pm 1_N\}$. The Glynn algorithm can be generalized to work with M-by-N rectangular permanents with $M \leq N$ by use of the following identity (shown here for $M \geq N$):

$$\operatorname{per}\begin{pmatrix} a_{1,1} & \cdots & a_{1,N} \\ \vdots & \ddots & \vdots \\ a_{M,1} & \cdots & a_{M,N} \end{pmatrix} = \frac{1}{(M-N+1)!} \cdot \operatorname{per}\begin{pmatrix} a_{1,1} & \cdots & a_{1,N} & 1_{1,N+1} & \cdots & 1_{1,M} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{M,1} & \cdots & a_{M,N} & 1_{M,N+1} & \cdots & 1_{M,M} \end{pmatrix}. \tag{7}$$

This can be neatly fit into Equation (6) by extending the inner sums over δ from $[1, M]$ to $[1, N]$:

$$\operatorname{per}(A) = \frac{1}{2^{N-1}} \cdot \frac{1}{(N-M+1)!} \cdot \sum_{\delta} \left(\sum_{k=1}^{N} \delta_k \right) \prod_{j=1}^{N} \left(\sum_{i=1}^{M} \delta_i a_{i,j} + \sum_{i=M+1}^{N} \delta_i \right). \tag{8}$$

It is still not clear which algorithm is faster, but for square matrices, they both scale in time with $\mathcal{O}(2^N N^2)$ if implemented naively, and with $\mathcal{O}(2^N N)$ if the sets $P(N, k)$ in Equation (4) and δ in Equation (6) are iterated over in Gray code order [2, 9]. At small values of N, the naive algorithm is actually the fastest, but it is quickly outpaced by the other two algorithms, and the difference in time between the algorithms at large values of N is several orders of magnitude. Our goal is to determine which algorithm for square and rectangular matrices is fastest for each value of M and N, and to write a function which deploys the appropriate algorithm for the input matrix. It seems to me that for large rectangular matrices with M and N close in value, the Glynn algorithm should be fastest; and for large highly rectangular matrices where the difference between M and N is large, the Ryser algorithm should be fastest.

Computing the Derivatives of Permanents

We can compute the derivative of a permanent with respect to a general parameter t by using a formula analogous to Jacobi's formula [10],

$$\frac{d}{dt}\operatorname{per}(A(t)) = \operatorname{tr}\left(\operatorname{adj}_+(A(t)) \frac{dA(t)}{dt} \right). \tag{9}$$

The function $\operatorname{adj}_+(A)$ in Equation (9) is the permanental adjugate, wherein the i, j-minors of A are computed with permanents instead of determinants. We can also compute the derivative of a permanent with respect to a matrix element $a_{i,j}$ by reducing Equation (9) to a special case:

$$\frac{d}{da_{i,j}}\operatorname{per}(A) = \operatorname{adj}_+(A)_{j,i}. \tag{10}$$

Carvalho's 2014 paper also gives a more general formula for the k-th derivative of the permanent [10]. As Paul says, derivatives of permanents are a "stretch goal".

Code Structure

Implementing the discussed permanent algorithms will require the following combinatorial generation algorithms:

- All k-permutations of the N-set $\{1, \ldots, N\}$ for Equations (1) and (2)
- All subsets of the N-set $\{1, \ldots, N\}$ for Equations (4) and (5)
- Gray code generation for Equations (6) and (8)

The first two should also be implemented in some minimal-change or Gray code ordering. Jörg Arndt's book *Matters Computational* contains explanations, C++ code, and benchmarks for all of these [11]. The relevant chapters are 8.2, 10.5–10.7, and 12.2. All of the code from the book is also available as a library on Arndt's website (`https://jjj.de/fxt/fxtpage.html`). We should probably use it. My example code for the Glynn algorithm also shows how the Gray code generation can work.

Ideally, our code should work with an input matrix in the form of either a raw pointer/std::array, an STL container, a Python/NumPy buffer, or whatever you prefer. We should leverage C++ abstractions to reduce code duplication where possible, while still allowing for efficient enough output to properly measure which algorithm is best at each value of M and N. Here is a basic plan we can follow:

1. Write template C++ functions for permanents; maybe write first implementations for some STL container?
2. Test the code against the example Python code, and benchmark the code
3. Optimize the code and find the best algorithm for each value of M and N
4. Write a C++ function which automatically calls the best algorithm based on the size of the input matrix
5. Ensure the code works with Python or NumPy buffers (maybe)
6. Write functions for the derivative of the permanent with respect to a parameter (maybe)

We should prioritize the combinatoric algorithm and the Glynn algorithm since I have good Python code we can follow. The Ryser algorithm would be nice to have, though, and David's code linked in the readme is a good starting point (in the case of square matrices, at least).

References

[1] Wikipedia, "Permanent (mathematics)." `http://en.wikipedia.org/w/index.php?title=Permanent\%20(mathematics)&oldid=1046043878`.

[2] Wikipedia, "Computing the permanent." `http://en.wikipedia.org/w/index.php?title=Computing\%20the\%20permanent&oldid=1037768944`.

[3] L. G. Valiant, "The complexity of computing the permanent," *Theoretical computer science*, vol. 8, no. 2, pp. 189–201, 1979.

[4] P. A. Limacher, P. W. Ayers, P. A. Johnson, S. De Baerdemacker, D. Van Neck, and P. Bultinck, "A new mean-field method suitable for strongly correlated electrons: Computationally facile antisymmetric products of nonorthogonal geminals," *Journal of chemical theory and computation*, vol. 9, no. 3, pp. 1394–1401, 2013.

[5] P. A. Johnson, P. A. Limacher, T. D. Kim, M. Richer, R. A. Miranda-Quintana, F. Heidar-Zadeh, P. W. Ayers, P. Bultinck, S. De Baerdemacker, and D. Van Neck, "Strategies for extending geminal-based wavefunctions: Open shells and beyond," *Computational and Theoretical Chemistry*, vol. 1116, pp. 207–219, 2017.

[6] T. D. Kim, R. A. Miranda-Quintana, M. Richer, and P. W. Ayers, "Flexible ansatz for n-body configuration interaction," *Computational and Theoretical Chemistry*, vol. 1202, p. 113187, 2021.

[7] H. J. Ryser, *Combinatorial mathematics*, vol. 14. American Mathematical Soc., 1963.

[8] D. G. Glynn, "The permanent of a square matrix," *European Journal of Combinatorics*, vol. 31, no. 7, pp. 1887–1891, 2010.

[9] D. E. Knuth, *The Art of Computer Programming, Volume 4, Fascicle 2: Generating All Tuples and Permutations*. Addison-Wesley Professional, 2005.

[10] S. Carvalho and P. Freitas, "The k-th derivatives of the immanent and the chi-symmetric power of an operator," *The Electronic Journal of Linear Algebra*, vol. 27, 2014.

[11] J. Arndt, *Matters Computational: ideas, algorithms, source code*. Springer Science & Business Media, 2010.