

Computation of matrix permanent with automatic differentiation *

Koichi KUBOTA †

Abstract

With a well-known formulation as a multiple variable polynomial, algorithms for matrix permanent are considered with automatic differentiation. Introducing a concept of "commutative quadratic nilpotent elements" with definitions of addition/multiplication operation, it is shown that the permanent can be computed straightforwardly as a variation of implementation of automatic differentiation.

Given several ways for transforming the multiple variable polynomial into univariable polynomial, an algorithm that computes the value of permanent with computation of residue, or digital Fourier transformation is introduced. Its computational complexity is $O(n2^n)$ which is as same as that of the best known Ryser's algorithm.

keywords: Automatic Differentiation, Permanent

1 Introduction

Since the computation of permanent of n -dimensional square matrix $A = (a_{ij})$, $\text{per}(A) \equiv \sum_{\sigma} \prod_{k=1}^n a_{k\sigma(k)} = \sum_{\sigma} a_{1\sigma(1)} a_{2\sigma(2)} \cdots a_{n\sigma(n)}$ was shown as #P-complete where σ runs all the permutations for $\{1, 2, \dots, n\}$, the direction of researches on permanent was moved to the computation of approximate values of permanents [1, 3, 4, 6, 8, 9, 10].

The Ryser's algorithm for exact value of permanent is well-known as the best algorithm of which complexity is $O(n2^n)$ [7], however, another formulation with multiple variable polynomial is also popular

*extended abstract for "AD2004, the forth international conference on automatic differentiation, Chicago, July, 2004."

†Department of Information and System Engineering, Faculty of Science and Engineering, Chuo University, Tokyo 112-8551, Japan.

which is simple and easy way to compute permanent with symbolic manipulation systems.

In this paper, without symbolic manipulation systems, we introduce a straightforward algorithm of permanent with automatic differentiation that can be used for differentiating multiple variable polynomials.

Then, providing several ways for transformation of the multiple variable polynomials into uni-variable polynomials, we consider numerical computations with the Taylor series. Giving several ways for the transformation, we show a uniqueness condition required for the transformation.

Furthermore, in order to represent naturally the sparsity of the coefficients of the differentiation, we introduce "commutative quadratic nilpotent elements" into multiple variable polynomials as indeterminates. Showing an implementation of the commutative quadratic nilpotent elements, it can be regarded as a kind of automatic differentiation and also regarded as an implementation of another method of permanent explained by D.E.Knuth[7].

1.1 Formulation with multi variable polynomial

According to the definition of permanent, it is well known that the permanent can be represented as the coefficient of the factor $x_1 x_2 \cdots x_n$ in the polynomial defined by

$$f(x_1, x_2, \dots, x_n) \equiv \prod_{i=1}^n (a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n). \quad (1)$$

Thus,

$$\text{per}(A) = \frac{\partial^n}{\partial x_1 \partial x_2 \cdots \partial x_n} f(x_1, x_2, \dots, x_n) \quad (2)$$

is a method for computing the value of the permanent and it can be naturally and easily computed with automatic differentiation.

On the other hand, in order to compute the same coefficient of $f(x_1, \dots, x_n)$, the method that evaluates f at 2^n points represented by $\{0, 1\}^n$ is well known as Ryser's form

$$\text{per}(A) = (-1)^n \sum_{s \in 2^{\{1, 2, \dots, n\}}} (-1)^{|s|} \prod_{i=1}^n \sum_{j \in s} a_{ij}.$$

It can be computed in $O(n2^n)$ with Gray-code technique and it is the best known algorithm for exact value of the permanent.

2 Formulation with transformation to uni-variable polynomial

Here, considering n non-negative integers $\alpha_1, \alpha_2, \dots, \alpha_n$, we substitute x^{α_i} for x_i and transform $f(x_1, \dots, x_n)$ into $f(x)$ defined by

$$f(x) \equiv \prod_{i=1}^n (a_{i1}x^{\alpha_1} + a_{i2}x^{\alpha_2} + \dots + a_{in}x^{\alpha_n}). \quad (3)$$

Note that the following condition is very important for choosing the integers α_i ($i = 1, \dots, n$).

Condition 2.1 uniqueness condition on $\alpha_1, \dots, \alpha_n$: *Chose one value from $\{\alpha_1, \dots, \alpha_n\}$ and repeat n times. Denote the k -th chosen value by β_k . Note that β_k may be equal to β_ℓ ($\ell \neq k$). Let $\alpha_* \equiv \sum_{i=1}^n \alpha_i$.*

If and only if $\sum_{k=1}^n \beta_k = \alpha_$, the sequence of β_1, \dots, β_n is a permutation of the sequence of $\alpha_1, \dots, \alpha_n$. In other words, when $\sum_{k=1}^n \beta_k = \alpha_*$, there are no indices k and ℓ ($\ell \neq k$) such that $\beta_k = \beta_\ell$*

When the above uniqueness condition is satisfied on $\alpha_1, \dots, \alpha_n$, the coefficient of x^{α_*} of $f(x)$ gives the value of $\text{per}(A)$.

In the following, we show some sets of α_i 's that satisfy the uniqueness condition.

Lemma 2.1 *Without loss of generality, we can assume $\alpha_1 < \alpha_2 < \dots < \alpha_n$. The set of α_i defined by $\alpha_i = 2^{i-1}$ satisfies the uniqueness condition.*

Corollary 2.1 *The set of α_i 's defined by $\alpha_i = 2^{i-1} - 1$ ($i = 1, \dots, n$) satisfies the uniqueness condition 2.1.*

Corollary 2.2 *For arbitrary integer p , the set of α_i defined by $\alpha_i = p^{i-1}$ satisfies the uniqueness condition 2.1.*

Note that the highest degree of the polynomial for the computation of the permanent is equal to $n\alpha_n$.

3 Methods

3.1 Automatic differentiation

On automatic differentiation [2, 5], the following is a brief note related to this report.

Given a program that computes a value of a function, automatic differentiation is a method for generating/transforming programs that compute the values of the derivatives of the function. There are two styles for implementation, one is precompiler and the other is library program that provides a kind of operator overloading.

There are also several methods to differentiate higher order derivatives with many variables. One of the simplest ways is repeated applications of pre-compilers to the derived programs. Designing a class for operator overloading with mechanism like C++ template, we can define a class for differentiation like `ad<double>`, then define another class for the second differentiation `ad<ad<double>>`, and so on.

For uni-variable polynomial, the computation of Taylor series is the method of the very automatic differentiation.

3.2 Commutative quadratic nilpotent element

The target value as the permanent is only the coefficient of a factor $x_1 x_2 \dots x_n$ in the multiple variable polynomial defined by eq.(1). In the intermediate

computation of this coefficient, any factor that consists of a quadratic of some x_i is not needed. Thus, we can consider an algebra where every factor that consists of a quadratic of some indeterminate is replaced to 0.

That is, introducing a set of n indeterminates $\xi_1, \xi_2, \dots, \xi_n$ such that, they are commutative $\xi_i \xi_j = \xi_j \xi_i$ ($i = 1, \dots, n, j = 1, \dots, n$), and quadratic nilpotent $\xi_i^2 = 0$ ($i = 1, \dots, n$), we consider a kind of polynomial of these ξ_i 's.

With this algebra, we can eliminate the terms and factors in the computation and expansion of the polynomial eq.(1) and we can save the computational time and space. That is, with an implementation of this algebra, we can represent the expanded term of $(a_1 \xi_1 + a_2 \xi_2 + \dots + a_n \xi_n)^k$ by the $\binom{n}{k}$ non-zero elements. Note that the number of non-zero elements is $\binom{n+k-1}{k}$ with the conventional symbolic manipulators.

3.3 Residue, DFT

In this report, the main problem is to compute p_β of $p(x) = p_0 + p_1 x + p_2 x^2 + \dots + p_\ell x^\ell$ ($\beta = 2^n - 1, \ell = n2^{n-1}$). Therefore, it can be computed as the residue of the regular function defined by $p(z)/z^{\beta+1}$, that is $p_\beta = \frac{1}{2\pi i} \oint \frac{p(z)}{z^{\beta+1}} dz$.

With $z = e^{i\theta}$, we have $p_\beta = \frac{1}{2\pi i} \oint \frac{p(z)}{z^{\beta+1}} dz = \frac{1}{2\pi} \int_0^{2\pi} p(e^{i\theta}) \cdot e^{-i\beta\theta} d\theta$. Defining $g(\theta) \equiv p(e^{i\theta}) \cdot e^{-i\beta\theta}$, we have $p_\beta = \frac{1}{2\pi} \int_0^{2\pi} g(\theta) d\theta$.

Approximation of this integration with N -point trapezoidal quadrature is equivalent to the N -point Discrete Fourier Transformation. Thus, when N is larger than ℓ , it is equivalent to the DFT of N coefficients of $p(x)$ so that we have $p_\beta = \frac{1}{2\pi} \sum_{k=0}^{N-1} g(\frac{2\pi}{N}k)$. $\frac{2\pi}{N} = \frac{1}{N} \sum_{k=0}^{N-1} g(\frac{2\pi}{N}k)$. (The value of N may be reduced according to the property of $p(x)$ and β .)

4 Algorithm

In the following, algorithms for permanent are explained, where $A = (a_{ij})$ is an n -dimensional square matrix.

Algorithm 4.1 (multiple variable polynomial) Differentiate eq.(1) with automatic differentiation straightforwardly.

Algorithm 4.2 (uni-variable polynomial) With Taylor series on eq.(3), compute the coefficient of x^{α_*} for $\alpha_i = 2^{i-1}$, $\alpha_* = 2^n - 1$.

Algorithm 4.3 (commutative quadratic nilpotent element)

We use n commutative quadratic nilpotent elements ξ_1, \dots, ξ_n instead of x_1, \dots, x_n , respectively, according to §3.2. The commutative quadratic nilpotent elements are implemented as instances of a class with overloaded multiplication and addition.

Algorithm 4.4 (Computation of residues)

Defining a function $g(\theta) \equiv f(e^{i\theta})e^{-i(2^n-1)\theta}$, compute $p_{\alpha_*} = \frac{1}{N} \sum_{k=0}^{N-1} g(\frac{2\pi}{N}k)$ (N is given as follows).

Lemma 4.1 (Appropriate N) Assume that α_i 's are given in lemma 2.1. Although degree of the original polynomial is $n2^{n-1}$, $N = 2^n$ gives the exact value of the permanent: $\text{per}(A) = \frac{1}{N} \sum_{k=0}^N g(\frac{2\pi}{N}k)$, i.e., $N = 2^n$ does not cause of the alias on the coefficient of x^{2^n-1} with DFT/FFT.

Algorithm 4.5 (Mixed algorithm) This is a mixed algorithm for eq.(3) using commutative quadratic nilpotent element and FFT. Let $n_1 = \lfloor (n+1)/3 \rfloor$. First, compute three products s_1, s_2, s_3 : $s_1 = \prod_{i=1}^{n_1} (\sum_{j=1}^n a_{ij} \xi_j)$, $s_2 = \prod_{i=n_1+1}^{2n_1} (\sum_{j=1}^n a_{ij} \xi_j)$, $s_3 = \prod_{i=2n_1+1}^n (\sum_{j=1}^n a_{ij} \xi_j)$.¹ Secondly, transform $s_i(\xi_1, \dots, \xi_n)$ into $s_i(x)$ by replacing ξ_i with $x^{2^{i-1}}$. Thirdly, compute 2^n -dimensional vectors \hat{s}_i corresponding to DFT of $s_i(x)$ with FFT. Finally, computing a vector \hat{s} as the component-wise product of \hat{s}_1, \hat{s}_2 and \hat{s}_3 , compute the reverse FFT of \hat{s} and get the coefficient of x^{2^n-1} .

¹The permanent can be computed as $f(x_1, x_2, \dots, x_n) = s_1 \cdot s_2 \cdot s_3$, but it needs $O(\binom{n}{n/3}^3)$ operations.

5 Comments and conclusion

5.1 Higher order differentiation with multiple variables

The automatic differentiation for multiple variables can be easily implemented with operator overloading features in C++, however, it does not seem so flexible for changing the number of variables. (With a kind of symbolic manipulators it is quite easy to derive the permanent, of course.)

For taking account of the sparsity of the derivatives, use of commutative quadratic nilpotent elements is better than the original differentiation of polynomial of multiple variables.

5.2 Use of Taylor series

The transformation into uni-variable polynomial is simple, so that the use of Taylor series is also simple and easy to compute the value of permanent. Furthermore, changing the data structure of the coefficient of Taylor series, for example use of `bool` data type, it can be used for the decision problem whether the value of the permanent is zero or not.

But there is a big weak point for the use of Taylor series. The multiplication of two N th order Taylor series requires $O(N^2)$ arithmetic operations with naive implementation, or $O(N \log N)$ arithmetic operations with FFT.

5.3 Transformation into uni-variable polynomial

We gave several sets of α_i 's for replacing x_i with x^{α_i} in order to transform the multiple variable polynomial eq.(1) into uni-variable polynomials. There are other sets satisfying the uniqueness condition 2.1. For some small n , sets of α_i 's that satisfy the condition are shown in Table 1

For the sets of α_i 's, the next lemma is curious.

Lemma 5.1 *For any n , if there exists a function (polynomial) $p(n)$ such that there is a set of α_i 's ($\alpha_1 < \alpha_2 < \dots < \alpha_n < p(n)$) satisfying the uniqueness condition 2.1, the value of the permanent can be computed with $O(n^3 p(n))$ arithmetic operations.*

Table 1: set of α_i 's satisfying the uniqueness condition

The sets whose largest number is the smallest in the same n .

n						
3	0	1	3			
4	0	1	4	6		
5	0	1	5	11	13	
6	0	2	10	21	22	26
7	0	1	5	21	43	45 53

5.4 Computation of Residue

Since the degree of the polynomial is very large, the integrand should be integrated along with $z = e^{i\theta}$. And the integration of cyclic function on a whole cycle should be computed with trapezoidal rule at equidistant points.

Although $g(\theta)$ in eq.(4.4) consists of components of very high frequencies, other numerical integration schemes may be considered for integration of $g(\theta)$ in order to reduce the number of the N -points. In this case, the resulting value is an approximation of the permanent with some truncation errors. More investigations may be needed along with this direction.

5.5 Conclusion

We show algorithms for computing exact value of the permanent with the technique of automatic differentiation. Introducing commutative quadratic nilpotent elements, we give another algorithm similar to that with automatic differentiation. Mixing the commutative quadratic nilpotent elements and FFT, we show an algorithm for exact permanent with $O(n2^n)$ arithmetic operations that is equal to the complexity of Ryser's algorithm known as the best algorithm of permanent.

References

- [1] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The design and analysis of*

- computer algorithms*. Addison-Wesley, Reading, Massachusetts, London, 1974.
- [2] M. Berz, C. Bischof, G. Corliss, and A. Griewank, editors. *Computational Differentiation: Techniques, Applications, and Tools*. SIAM, Philadelphia, Pennsylvania, 1996.
 - [3] Uriel Feige and Carsten Lund. On the hardness of computing the permanent of random matrices. *24th Annual ACM STOC*, pages 643–654, 1992.
 - [4] Martin Fürer. Approximating permanents of complex matrices. *STOC 2000*, pages 667–669, 2000.
 - [5] A. Griewank and G. F. Corliss, editors. *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*. SIAM, 1991. SIAM Workshop on the Automatic Differentiation of Algorithms at Breckenridge, Colorado, 1991.
 - [6] Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with non-negative entries. *STOC'01*, pages 712–721, 2001.
 - [7] D. E. Knuth. *The art of computer programming, 3rd ed.*, volume 2. Addison-Wesley, Reading, Massachusetts, London, 1998.
 - [8] Nathan Linial, Alex Samorodnitsky, and Avi Wigderson. A deterministic strongly polynomial algorithm for matrix scaling and approximate permanents. *STOC 98*, pages 644–652, 1998.
 - [9] Frank RUBIN. A search procedure for hamilton paths and circuits. *Journal of the Association for Computing Machinery*, 21(4):576–580, 1974.
 - [10] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM journal on computing*, 8(3):410–421, 1979.