Реализация 3-х проходного метода быстрого автоматического дифференцирования

А.Ю. Горчаков

***Аннотация*— В статье рассматривается реализация 3-х проходного метода быстрого автоматического дифференцирования. Добавление к стандартному 2-х проходному методу еще одного прохода приводит к существенному уменьшению потребления оперативной памяти, при незначительном (25%) увеличении количества вычислений. Приведены оценки производительности и требований к объему оперативной памяти 2-х и 3-х проходного методов. Проведен численный эксперимент на примере обратной задачи восстановления начальных данных для уравнения переноса.**

***Ключевые слова*— оптимальные управление, быстрое автоматическое дифференцирование, уравнение переноса, обратная задача восстановления начальных данных.**

I. Введение

При численном решении задач оптимизации часто используются градиентные методы минимизации функций, которые требуют умения вычислять не только значения оптимизируемых функций, но и их градиентов. При этом крайне важно вычислять точные значения градиентов за минимально возможное время.

Вычисление градиентов методом конечных разностей иногда связано с большими трудностями, и занимает достаточно много времени в случае задач большой размерности.

Более перспективным представляется использование методологии быстрого автоматического дифференцирования (обобщенная БАД-методология, см. [1]-[4]). Но, при очевидных преимуществах БАД-методологии, таких как, точное вычисление градиента, независимость времени расчета от размерности вектора управлений, имеется и определенный недостаток, а именно – высокие требования к объему оперативной памяти. Указанный недостаток начинает широко проявляться при решении задач оптимального управления на сетках большой размерности.

Работа выполнена при выполнена при финансовой поддержке РФФИ, проект 16-07-00458 А, программы Президиума РАН I.5 "Интеллектуальные информационные технологии и системы".

А.Ю. Горчаков – ведущий математик Вычислительного центра им. А.А. Дородницина Федерального исследовательского центра «Информатика и управление» Российской академии наук. andrgor12@gmail.com

В данной статье рассматривается методика уменьшения используемого объема оперативной памяти, за счет незначительного увеличения объема вычислений. Методика реализована автором в модифицированной версии пакета быстрого автоматического дифференцирования Adept [8]. Рассмотрение проводится на примере обратной задачи восстановления начальных данных для уравнения переноса. Необходимо отметить, что идея уменьшения требований к объему оперативной памяти не является абсолютно новой, в частности похожий метод (называемый revolve) можно найти в пакете ADOL-C [9].

II. Методология быстрого автоматического дифференцирования

Пусть $z \in R^n$ и $u \in R^r$ векторы. Дифференцируемые функции $W(z, u)$ и $\Phi(z, u)$ определяют отображения $W: R^n \times R^r \to R^1$, $\Phi: R^n \times R^r \to R^r$. Вектора z и u удовлетворяют следующей системе из n нелинейных скалярных уравнений $\Phi(z, u) = 0$. Если матрица $\Phi_z^T(z,u)$ невырожденна, то сложная функция $\Omega(u) = W(z(u), u)$ дифференцируема и ее градиент относительно переменных u вычисляется по формуле:

$$\frac{d\Omega}{\mathrm{du}} = W_u(\mathrm{z(u)}, \mathrm{u}) + \Phi_u^T(z(u), u) \cdot p, \qquad (1)$$

где вектор $p \in R^n$ находится из решения линейной алгебраической системы:

$$W_z(\mathrm{z}, \mathrm{u}) + \Phi_z^T(z, u) \cdot p = 0, \qquad (2)$$

Здесь и далее индекс T обозначает транспонирование, нижние индексы z, u обозначают частные производные функций по векторам z и u: $W_u = \frac{\partial W}{\partial u}, W_z = \frac{\partial W}{\partial z}, \Phi_u^T = \frac{\partial \Phi^T}{\partial u}, \Phi_z^T = \frac{\partial \Phi^T}{\partial z}$, так же будем обозначать i-е, j-е компоненты векторов z и u как z_i, z_j, u_i, u_j.

Предположим, что каждый компонент вектора z_i последовательно выражается только через компоненты вектора u и предыдущие z_j, т.е. $1 \leq j < i$., тогда формулы (1)-(2) можно записать в следующем виде:

$$p_i = \sum_{j=i+1}^{n} \Phi_{z_j}^T(z, u) p_j + W_{z_i}(\mathrm{z}, \mathrm{u}) \qquad (3)$$

$$\frac{d\Omega}{\mathrm{du}_i} = W_{u_i}(\mathrm{z},\mathrm{u}) + \sum_{j=1}^{n} \Phi_{u_j}^{T}(z,u)\cdot p_j \qquad (4)$$

Системы (3)-(4) называют явными [1], и возникают они, например, в задачах оптимального управления, если интегрирование проводится по явным схемам и по некоторым неявным схемам.

Для примера рассмотрим подробно дифференцирование скалярной функции двух переменных:

$f(u_1,u_2) = u_2\sin(u_1) + u_1 * u_2,$ (5)

в точке $u_1 = 0.3, u_2 = 0.5.$

Вычисление значения этой функции представляем, как последовательность формул, содержащих арифметические действия и основные элементарные функции:

$i=1$	$z_1 = \sin(u_1)$	$\sin(0.3) = 0.296$
$i=2$	$z_2 = u_2 * z_1$	$0.5 * 0.296 = 0.148$
$i=3$	$z_3 = u_1 * u_2$	$0.3 * 0.5 = 0.15$
$i=4$	$z_4 = z_2 + z_3 = f(u_1,u_2)$	$0.148 + 0.15 = 0.298$

Переходя от одной формулы к другой в сторону увеличения индекса i, вычисляем значение функции. Далее в обратном порядке определяем компоненты вектора p и все частные производные функции по независимым переменным u.

$$p_4 = 1.0,\ p_3 = p_2 = p_1 = \frac{\partial f}{\partial u_2} = \frac{\partial f}{\partial u_1} = 0$$

$i=4$ $p_3 = p_3 + p_4 = 1.0$

$p_2 = p_2 + p_4 = 1.0$

$i=3$ $\frac{\partial f}{\partial u_1} = \frac{\partial f}{\partial u_1} + u_2 * p_3 = 0.5 * 1.0 = 0.5$

$\frac{\partial f}{\partial u_2} = \frac{\partial f}{\partial u_2} + u_1 * p_3 = 0.3 * 1.0 = 0.3$

$i=2$ $\frac{\partial f}{\partial u_2} = \frac{\partial f}{\partial u_2} + z_1 * p_2 = 0.596$

$p_1 = p_1 + u_2 * p_2 = 0.5 * 1.0 = 0.5$

$i=1$ $\frac{\partial f}{\partial u_1} = \frac{\partial f}{\partial u_1} + \cos(u_1) * p_1 = 0.978$

Можно представить процесс вычислений в виде информационного графа (см. рис.1), в таком случае говорят, что сначала расчеты ведутся слева-направо, а затем справа-налево.

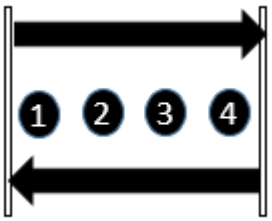

Рис.1 графическое представление процесса вычислений

В работе [10] дана оценка временной сложности вычисления градиента $T_g/T_0 \le 3$, где T_0 полное время, требуемое для вычисления значения функции, T_g дополнительное время, требуемое для вычисления всех частных производных функции; оценка дана без учета времени, необходимого для работы с памятью компьютера. Напомним, что в случае численного дифференцирования для расчета градиента потребуется времени не менее $(1+n)T_0$, где n число необходимых нам частных производных.

Необходимо отметить, то что при вычислениях требуется сохранять в памяти компьютера значения компонентов векторов z и p. Для многих задач оптимального управления, решаемых конечно-разностными методами, это означает, что необходимый объем памяти прямо пропорционален количеству узлов расчетной сетки. Например, для расчетной сетки с $4.8 * 10^{11}$ узлами и вычислениями с двойной точностью (8 байт на число) потребовалось бы порядка 7 терабайт памяти.

III. Постановка задачи восстановления начальных данных

Рассмотрим задачу восстановления начальных данных для одномерного уравнения переноса пассивной примеси в жидкости движущейся с постоянным значением скорости:

$$\frac{\partial\varphi}{\partial t} + c\frac{\partial\varphi}{\partial x} = 0, c = 1, \quad (x,t) \in Q, \qquad (6)$$

$$\varphi(x,0) = w_1(x), \qquad x \in [0,1], \qquad (7)$$

$$\varphi(0,t) = w_2(t), \qquad t \in (0,1]. \qquad (8)$$

Здесь x – пространственная координата; t – время; c – скорость движения жидкости; $\varphi(x,t)$ – концентрация примеси в точке x в момент времени t; прямоугольник $Q = (0,1)\times(0,1)$; $w_2(x)$ – заданная функция, определяющая концентрацию примеси в точке x=0; $w_1(x)$ – искомая функция, определяющая концентрацию примеси в начальный момент времени. Пусть на прямой $x+t=1$ задана функция $\bar{\varphi}(x,t)$, где $(x,t \in Q)$. Ее можно рассматривать как концентрацию примеси, полученную в результате некоторых экспериментальных исследований. Поставим следующую задачу: требуется определить оптимальное управление $u_*(x) = w_{1_*}(x)$ и соответствующее решение $\varphi_*(x,t)$ системы (6)-(8), при котором интегральный функционал

$$W(u) = \frac{1}{2}\int_0^1 [\varphi(x,1-x) - \bar{\varphi}(x,1-x)]^2\,dx \qquad (9)$$

достигал бы минимально возможного значения.

Задачи, подобные этой, обычно решаются численно с помощью некоторого метода спуска, который требует знания градиента функционала (9).

Перейдем к дискретному варианту системы (6)-(8). Для этого покроем прямоугольник Q регулярной расчетной сеткой с шагами $h=1/J$ и $\tau=1/N$, где N – количество интервалов по времени, а J – количество интервалов по оси x (для удобства расчетов возьмем N кратным J, т.е. $N=k*J$). Используем хорошо известную схему «уголок», аппроксимирующую уравнение (6) на трехточечном шаблоне с первым порядком точности:

$$\frac{\varphi_j^{n+1} - \varphi_j^n}{\tau} + c\frac{\varphi_j^n - \varphi_{j-1}^n}{h} = 0\,, \qquad (10)$$

$$\varphi_j^0 = w_{1_j} = u_j, \qquad (11)$$

$$\varphi_0^n = {w_2}^n, \qquad (12)$$

$$W(u) = \frac{1}{2}\sum_{j=1}^{J}({\varphi_j}^{N-k*j} - {\bar{\varphi}_j}^{N-k*j})^2, \qquad (13)$$

где $\varphi_j^n = \varphi(x_j,t_n), \bar{\varphi}_j^n = \bar{\varphi}(x_j,t_n), x_j = jh, t_n = n\tau, \varphi_j^0 = w_{1_j} = u_j = u(x_j),$

$\varphi_0^n = w_2{}^n = w_2(t_n), 0 \le j \le J, 0 \le n \le N.$

Требуется найти вектор u, при котором сложная дискретизированная функция *W(u)* принимает минимальное значение.

IV. Описание методики вычисления градиента 3-х проходным методом

С целью снижения требований к объему памяти, предлагается использовать 3-х проходный метод автоматического дифференцирования. Идея метода основана на том, что для «явных» по времени разностных схем вычисления можно остановить, сохранив один временной слой Φ^n – значения $\varphi_j^n, 1 \le j \le J$ (или несколько временных слоев, в зависимости от используемой разностной схемы), а затем продолжить их в любой момент используя сохраненные значения. Иначе говоря, для таких разностных схем можно делать «контрольные точки». Используя эту особенность, разобьем расчетную сетку по оси времени на *M*-групп, с *K=N/M* временных слоев в каждой группе и будем хранить в памяти только последний временной слой в каждой группе и временные слои, относящиеся к текущей группе. Приведем описание метода.

МЕТОД 3Х

1. Разбить расчетную сетку по оси времени на *M* групп, с *K=N/M* временных слоев в группе.
2. В цикле по n от 1 до *(M-1)*K* выполнить
   a. вычислить Φ^n,
   b. для *n =K,2K,…,(M-2)*K* сохранить Φ^n в памяти
3. Для группы *M* вычислить Φ^n и частные производные $\partial W \Big/ \partial \varphi_j^{(M-1)K}, 1 \le j \le J$
4. В цикле по m от M-1 до 1 выполнить
   a. вычислить Φ^n,
   b. вычислить $\partial W \Big/ \partial \varphi_j^{(m-1)K}, 1 \le j \le J$
5. как градиент функционала W(u) вернуть $\partial W \Big/ \partial \varphi_j^0, 1 \le j \le J$

Графически этот метод можно представить следующим образом:

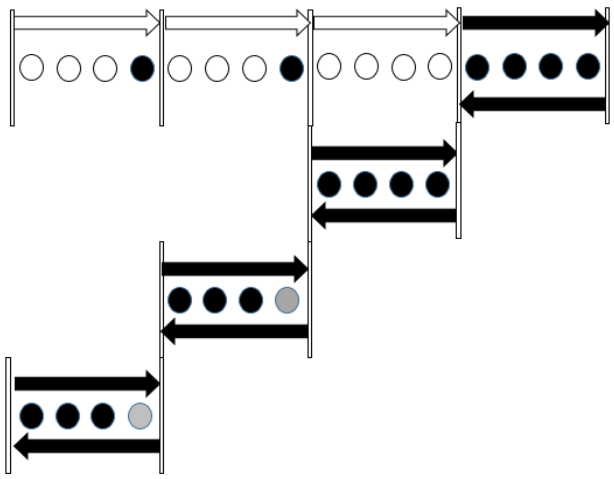

Рис.2 графическое представление метода 3Х.

Здесь черные стрелки представляют собой классические проходы слева-направо и справа-налево, а белые стрелки это дополнительный проход, при котором вычисляются и запоминаются промежуточные временные слои Φ^n. Белые шарики - временные слои рассчитанные, но не хранящиеся в памяти; черные – рассчитанные и сохраненные; серые – рассчитанные ранее.

Легко показать, что временная сложность вычисления градиента для 3-х проходного метода $T_g/T_0 \le 4$. При этом в оперативной памяти необходимо хранить значения φ_j^n в *J*(M-2+K)* и значения компонентов вектора p в *J*K* узлах расчетной сетки. При оптимальном выборе M требования по памяти уменьшаются с *J*N* до $J * 2 * \sqrt{2 * N}$. Для расчетной сетки 16000x48000 вместо 11.4 Гб потребуется 75.6 Мб оперативной памяти.

Метод 3Х реализован как дополнительный программный модуль в пакете быстрого автоматического дифференцирования Adept функцией класса *Stack*:

*compute_adjoint3X(double *x, double *g, int N, int M, adouble (*F_k)(const adouble *x, const int n1, const int n2, adouble *PHI))*, где *x* – вектор независимых переменных, *g* – вектор частных производных функции, *N* – количество интервалов по времени, *M* – количество групп, *F_k* – ссылка на пользовательскую функцию, которая принимая как параметры вектор x, n1 и n2 индексы начала и конца группы, возвращает значение функционала и PHI – вектор Φ^n.

V. Численный эксперимент

Задача (6)-(9) решалась при начальном условии:

$$w_1(x) = e^{-\left(\frac{x-0.2}{0.07}\right)^2} + e^{-\left(\frac{x-0.4}{0.07}\right)^2} \quad (14)$$

и $w_2(t) = 0$

Разностная схема и вычисление функционала были записаны на языке C и продифференцированы с помощью программного пакета Adept [8]. Вычисления производились на персональном компьютере с процессором Intel Core i7-4770, 3.4 GHz, 8Gb оперативной памяти. Значение функционала *W(u)* и его частных производных вычислялось в точке

$$u(t) = \bar{\varphi}\left(\frac{1+x}{2}\right), \quad (15)$$

на трех расчетных сетках различной размерности, стандартным методом и методом *3X*. Графики $w_1(x), u(x)$, приведены на рис.3

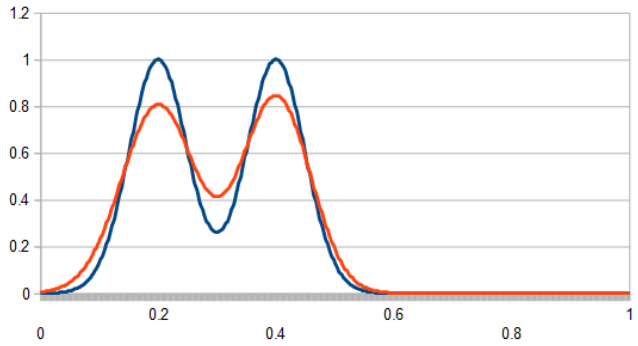

Рис. 3 Графики функций (14) – синяя линия и (15) – красная линия

Значение *M=N/600* (т.е. 600 временных слоев в группе). Значения частных производных функционала вычисленных обоими методами совпали друг с другом с машинной точностью. Для расчетной сетки 1 оба метода используют только оперативную память, при этом метод 3X работает немного медленнее чем стандартный метод. Для расчетной сетки 2 стандартный метод БАД начинает использовать более медленную виртуальную память компьютера, соответственно наблюдается уменьшение скорости его работы. При переходе к расчетной сетке 3 скорость работы стандартного метода БАД уже значительно меньше скорости работы метода 3X.

Необходимо отметить, что программный пакет Adept сохраняет в оперативной памяти не только значения в узлах расчетной сетки, но и дополнительные внутренние переменные, поэтому объем используемой оперативной памяти выше расчетной.

Таблица 1. Сравнение стандартного метода БАД и 3X

№№	Расчетная сетка (*JxN*)	Использованная память стандарт/3X (Мб)	Время расчетов стандарт/3X (сек)
1	4000x12000	2197.8/110.8	4.1/5.8
2	8000x24000	8790.5/222.8	53.3/29.6
3	16000x48000	35158.4/450.6	1429.2/127.0

VI. ЗАКЛЮЧЕНИЕ

В работе рассмотрена реализация 3-х проходного метода быстрого автоматического дифференцирования для задач оптимального управления. Метод реализован как дополнительный программный модуль в пакете быстрого автоматического дифференцирования Adept. По сравнению со стандартным методом БАД существенно снижены требования к оперативной памяти компьютера (вместо линейной зависимости объема оперативной от количества интервалов по времени N, получена зависимость $2\sqrt{2N}$), при незначительном (на 25%) увеличении объема вычислений. Особенно актуально использование метода при расчетах на GPU (графические ускорители), так как в отличие от CPU, которому можно предоставить 1Тб и более оперативной памяти, для видеокарт и профессиональных графических ускорителей существует ограничение 16/32Гб.

Метод применим для задач оптимального управления любой размерности по пространственным координатам. Единственное требование, предъявляемое к разностным схемам – возможность делать «контрольные точки». Метод совместим с большинством современных пакетов автоматического дифференцирования. В случае необходимости дополнительного снижения требований к оперативной памяти можно, добавляя вспомогательные проходы, получать методы 4X, 5X и т.д.

VII. БИБЛИОГРАФИЯ

[1] Айда-Заде К.Р., Евтушенко Ю.Г. Быстрое автоматическое дифференцирование // Математическое моделирование, 1989. V. 1, pp. 121-139.

[2] Евтушенко Ю.Г., Мазурик В.П. Математическое обеспечение оптимизации. М: Знание. 1989.

[3] Evtushenko Y.G. Automatic differentiation viewed from optimal control theory. Proceedings of the Workshop on Automatic Differentiation of Algorithms: Theory, Implementation and Application. Philadelphia: SIAM, 1991.

[4] Griewank A, Corliss G.F., eds. Automatic Differentiation of Algoritms. Theory, Implementation and Application. Philadelphia: SIAM, 1991.

[5] Горчаков А.Ю., Посыпкин М.А. Эффективность методов локального поиска в задаче минимизации энергии плоского кристалла. Международный научный журнал «современные информационные технологии и ИТ-образование». – 2017. – V. 13. – N. 2. – pp. 97-102.

[6] Евтушенко, Ю. Г., Лурье, С. А., Посыпкин, М. А., Соляев, Ю. О. (2016). Применение методов оптимизации для поиска равновесных состояний двумерных кристаллов. Журнал вычислительной математики и математической физики, 56(12), 2032-2041.

[7] Turkin A., Thu A. Benchmarking Python Tools for Automatic Differentiation. International Journal of Open Information Technologies. – 2016. – V. 4. – N. 9.

[8] Hogan R. J. Fast reverse-mode automatic differentiation using expression templates in C++. ACM Transactions on Mathematical Software (TOMS). – 2014. – V. 40. – N. 4. – pp. 26.

[9] Griewank A., Walther A. Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation. ACM Transactions on Mathematical Software (TOMS). 2000. V. 26. N. 1. pp. 19-45.

[10] Евтушенко Ю.Г. Оптимизация и быстрое автоматическое дифференцирование. Научное издание ВЦ РАН, 2013.

The implementation of a 3-pass method for fast automatic differentiation

Andrei Y. Gorchakov

***Abstract*— in this paper, we consider the implementation of the three-pass method of fast automatic differentiation. Adding one more pass to the standard 2-pass method leads to a significant decrease in the consumption of RAM, with an insignificant (25%) increase in the number of calculations. Estimates of performance and requirements for the amount of RAM of 2 and 3-pass methods are given. A numerical experiment is performed on the example of the inverse problem of reconstructing the initial data for the transport equation.**

***Keywords*— optimal control, fast automatic differentiation, transport equation, inverse problem of recovering initial data.**