О программных пакетах быстрого автоматического дифференцирования[1]

А.Ю. Горчаков

Аннотация. В статье проводится сравнительный анализ пакетов быстрого автоматического дифференцирования Adept, CoDiPack и FADBAD. На примере обратной коэффициентной задачи показано преимущество прямого метода быстрого автоматического дифференцирования, реализованного в пакете CoDiPack, над прямым и обратным методами, реализованными в пакете Adept, и над прямым методом, реализованным в пакете FADBAD. В качестве численных методов оптимизации использовались метод градиентного спуска и метод Левенберга-Марквардта. Даны рекомендации по выбору методов оптимизации и быстрого автоматического дифференцирования в зависимости от размерности задачи.

Ключевые слова: оптимальные управление, быстрое автоматическое дифференцирование, градиент, матрица Якоби, метод Левенберга-Марквардта.

Введение

При численном решении задач оптимального управления часто используются градиентные методы минимизации функций, которые требуют умения вычислять значения не только 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, [1-5]). Непосредственное применение данной методологии дает прекрасные результаты как по точности, так и по скорости вычисления градиента [6], но в ряде случаев может оказаться слишком трудоемким, требующим большой аналитической работы по выводу необходимых формул. С целью уменьшения трудозатрат предлагается использовать уже существующие пакеты быстрого автоматического дифференцирования. В статье рассматривается применение пакетов Adept [7], CodiPack [8], FADBAD [9], на примере обратной коэффициентной задачи для уравнения теплопроводности [10].

1. Методология быстрого автоматического дифференцирования

Пусть $z \in R^n$ и $u \in R^r$ - векторы. Дифференцируемые функции *W(z, u)* и *Φ(z, u)* определяют отображения $W: R^n \times R^r \to R^1$, $\Phi: R^n \times R^r \to R^n$. Векторы z и u удовлетворяют следующей системе из n нелинейных скалярных уравнений *Φ(z, u)* = 0. Если матрица $\Phi_z^T(z,u)$ невырождена, то сложная функция Ω(u) = W(z(u), u) дифференцируема и ее градиент

[1] Работа выполнена в Вычислительном центре им. А.А. Дородницына ФИЦ ИУ РАН при финансовой поддержке проекта НШ-8860.2016.1 программы государственной поддержки ведущих научных школ Российской Федерации, программы Президиума РАН I.33 "Математические модели и средства для исследования экономических и физических процессов с использованием высокопроизводительных вычислений", гранта РФФИ 17-07-00493 А.

относительно переменных u вычисляется по формулам (1)-(2) для прямого метода дифференцирования:

$$\frac{d\Omega}{\mathrm{du}} = W_u(\mathrm{z(u)}, \mathrm{u}) + N(u)W_{z^T}(z(u), u), \quad (1)$$

где прямоугольная $r \times m$ матрица $N(u)$ находится из решения линейной алгебраической системы:

$$W_{z^T}(\mathrm{z}, \mathrm{u}) + N(\mathrm{u})\Phi_z^T(z, u) = 0, \quad (2)$$

и (3)-(4) для обратного метода дифференцирования:

$$\frac{d\Omega}{\mathrm{du}} = W_u(\mathrm{z(u)}, \mathrm{u}) + \Phi_u^T(z(u), u) \cdot p(u), \quad (3)$$

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

$$W_z(\mathrm{z}, \mathrm{u}) + \Phi_z^T(z, u) \cdot p = 0, \quad (4)$$

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.

Прямой метод позволяет вычислить градиент функции за время $T\big(grad(f)\big)$, не превышающее:

$$T\big(grad(f)\big) \le C_{forward} * r * T(f), \quad (5)$$

а обратный метод за время

$$T\big(grad(f)\big) \le C_{reverse} * T(f), \quad (6)$$

где $T(f)$ - время вычисления функции и $C_{forward}, C_{reverse}$ - некоторые константы. Теоретические оценки данных констант (без учета времени доступа к оперативной памяти и возможности параллельных вычислений) приведены в работе [1]: $C_{forward} = 3$, $C_{reverse} = 3$. Практически достижимые оценки для $C_{reverse}$ приведены в работе [7]: 2-4 для ручного способа кодирования производных и 2.7-4 для пакета Adept. Для прямого метода дифференцирования практически достижимые оценки $C_{forward}$, как правило, не превышают теоретических.

В случае векторной функции $f(u): R^r \to R^m$. Матрица первых производных функции (матрица Якоби) вычисляется за время, не превышающее:

$$T\big(grad(f)\big) \le C_{forward} * r * T(f), \quad (7)$$

а обратный метод за время

$$T\big(grad(f)\big) \le C_{reverse} * m * T(f) \quad (8)$$

Как видно из (7) и (8), обратный метод дифференцирования эффективен при r >> m.

Наиболее простой и распространенный способ реализации методов быстрого автоматического дифференцирования - это использование механизма перегрузки операторов (англ. - operator overloading), доступного во многих современных языках программирования. Прямой метод дифференцирования реализуется достаточно прямолинейно – вводится новый тип данных, который содержит не только значение переменной, но и все значения ее производных по одной или нескольким входным переменным [12]. Обратный метод более сложен в реализации, так как при вычислении функции необходимо записать в память результаты вычисления каждой основной элементарной функции, создавая «информационный граф», и затем пройти этот граф в обратном порядке, вычисляя производные [11].

2. Пакеты быстрого автоматического дифференцирования

Наиболее простой способ применить методы автоматического дифференцирования к уже существующей программе, написанной на C/C++, - использовать один из следующих пакетов: Adept [7], CoDiPack [8], FADBAD [9].

Программа модифицируется с помощью формальных преобразований – тип данных double заменяется на специальный тип adouble (Adept), F<double> или B<double> (FADBAD), codi:: RealForward или codi::RealReverse (CoDiPack – прим.: всего 23 различных типа данных, реализующих разновидности того или иного метода дифференцирования); затем добавляется несколько строк инициализации и расчета градиента.

Пример такого преобразования:

Исходная программа

```
double W(const double x[2]) {
  double y = 4.0;
  double s = 2.0*x[0] + 3.0*x[1]*x[1];
  y *= sin(s); return y;
}
```

Модифицированная программа

```
adouble W(const adouble x[2]) {
  adouble y = 4.0;
  adouble s = 2.0*x[0] + 3.0*x[1]*x[1];
  y *= sin(s); return y;
 }

using namespace adept;
Stack stack;
adouble x[2] = {x_val[0], x_val[1]};
stack.new_recording();
adouble Y = W(x);
Y.set_gradient(*Y_ad);
stack.reverse();
x_ad[0] = x[0].get_gradient();
x_ad[1] = x[1].get_gradient();
*Y_ad = Y.get_gradient();
```

3. Постановка обратной коэффициентной задачи

В качестве тестовой была выбрана обратная коэффициентная задача для уравнения теплопроводности [10]. Рассматривается слой материала ширины L. Температура этого слоя в начальный момент времени известна. Также известно, как изменяется со временем температура на краях этого слоя. Распределение температурного поля в слое в каждый момент времени описывается решением следующей начально-краевой (смешанной) задачи:

$$\rho C\frac{\partial T(x,t)}{\partial t} - \frac{\partial}{\partial x}\left(K(T)\frac{\partial T(x,t)}{\partial x}\right) = 0, (x,t) \in Q, \quad (9)$$

$$T(x,0) = w_0(x), (0 \le x \le L), \quad (10)$$

$$T(0,t) = w_1(t), T(L,t) = w_2(t), \quad (0 \le t \le \Theta), \quad (11)$$

где x – декартова координата точки в слое, t – время, $Q = \{(0 < x < L) \times (0 < t \le \Theta)\}$, *T(x, t)* – температура вещества в точке с координатой x в момент времени t, ρ и C – плотность вещества и его теплоемкость соответственно, *K(T)* – коэффициент конвективной теплопроводности, $w_0(x)$ - заданная температура слоя в начальный момент времени, $w_1(t)$ - заданная температура на левом краю слоя, $w_2(t)$ - заданная температура на правом краю слоя.

Задача оптимального управления состоит в определении оптимального управления *K(T)* и соответствующего оптимального решения *T(x,t)* задачи (9)-(11), при котором функционал $W\big(K(T)\big) = \int_0^{\Theta}\int_0^{L}[T(x,t) - Y(x,t)]^2 dxdt$ достигает минимального значения. Здесь $Y(x,t), (x,t) \in Q$- заданная функция.

Для численного эксперимента рассматривалась задача нахождения эффективного коэффициента конвективной теплопроводности при следующих входных данных:

$$\begin{array}{ll} L = 1, \Theta = 1, & Q = (0,1) \times (0,1), \\ \rho(T,x) \equiv C(T,x) \equiv 1, & (0 \le x \le 1), \\ w_0(x) = 2x, & (0 \le x \le 1), \\ w_1(t) = 0, & (0 \le t \le 1), \\ w_2(t) = 2, & (0 \le t \le 1) \end{array} \quad (12)$$

Функция Y(x,t) была получена как решение смешанной задачи (9)-(11) с указанными выше параметрами при $K(T) = \begin{cases} 1, & T \le 1 \\ 3, & T > 1 \end{cases}$

При решении прямой задачи использовалась равномерная сетка с параметрами *I1*=300 - количество интервалов вдоль оси *x*, *I2*=6000 (3000) - количество интервалов по времени, искомая функция *K(T)* аппроксимировалась непрерывной кусочно-линейной функцией, количество интервалов по температуре 19 (соответственно размерность вектора управления равна 20).

4. Метод Левенберга-Марквардта

Вышеупомянутая задача относится к классу задач о наименьших квадратах:

$$F(x) = \frac{1}{2}\sum_{i=1}^{m} f_i^2(x) = \frac{1}{2}\|f(x)\|_2^2 \to \min, \quad (13)$$

где *f(x)* – нелинейная вектор-функция с m компонентами $f_i(x)$, $x \in R^n$, $\|f(x)\|_2$ - евклидова норма.

Хотя для решения задач (13) можно использовать универсальные методы, более эффективным будет применение специальных алгоритмов, разработанных именно для задач о наименьших квадратах. Примерами таких алгоритмов являются метод Гаусса-Ньютона и Левенберга-Марквардта [13]. Оба метода используют особую структуру градиента функции и её матрицы Гессе.

Пусть *J(x)* - матрица Якоби для *f(x)* и $H_i(x)$ - матрица Гессе для $f_i(x)$. Тогда градиент *g(x)* и матрицу Гессе *H(x)* для *F(x)* можно записать как:

$$g(x) = J(x)^T f(x), \quad (14)$$

$$H(x) = J(x)^T J(x) + Q(x), \quad (15)$$

где $Q(x) = \sum_{i=1}^{m} f_i(x) H_i(x)$. Сделав предположения о том, что слагаемое $J(x)^T J(x)$ доминирует над *Q(x),* получаем следующие итерационные схемы методов.

Для метода Гаусса-Ньютона:

$$x_{k+1} = x_k + \alpha_k p_k, \quad (16)$$

где $0 < \alpha_k \leq 1$ - шаг метода, p_k- направление поиска, определяемое из решения системы уравнений

$$J_k^T J_k p_k = -J_k^T f_k \quad (17)$$

Для метода Левенберга-Марквардта:

$$x_{k+1} = x_k + p_k, \quad (18)$$

где p_k- направление поиска, определяемое из решения системы уравнений

$$\left(J_k^T J_k + \lambda_k I\right) p_k = -J_k^T f_k, \quad (19)$$

I - единичная матрица, λ_k – некоторая неотрицательная константа, своя для каждого шага.

5. Численный эксперимент

Эксперименты проводились на компьютере с процессором Intel Core-i7 4770, 8 Гб оперативной памяти. Необходимо отметить, кроме того, что процессор содержит 4 ядра с поддержкой технологии Hyper-Trading (8 логических ядер), в нем поддерживается технология AVX2, позволяющая оперировать с 256 битными векторными регистрами (относительно рассматриваемой задачи одновременно обрабатывается 4 числа двойной точности). Исходный код был написан на C/C++ и компилировался с помощью gcc версии 5.4.0. При этом использовались флаги оптимизации -O2, -O3,-Ofast и поддержки многопоточности -O2 -fopenmp.

В первом эксперименте сравнивается время расчета градиента с помощью пакетов Adept (прямой-forward и обратный-reverse методы БАД), CoDiPack (forward), FADBAD (forward) и различными флагами оптимизации. В Табл. 1 приведено время расчетов функции и ее градиента для расчетной сетки 300x3000 (300 интервалов по оси x и 3000 интервалов по оси времени). Данная сетка была подобрана таким образом, чтобы обратный метод БАД пакета Adept при расчетах использовал только оперативную память.

Как видно из таблицы, при включенном флаге оптимизации –O2 метод показывает быстродействие, близкое к теоретической оценке. Приведем теоретическую оценку времени вычисления функции и градиента для данной задачи: если расчет функции занимает 1 секунду, то для обратного метода быстрого автоматического дифференцирования время вычисления функции и градиента не должно превышать 4 секунд, а для прямого метода 1+3*20=61 секунды. Прямой метод БАД для пакетов Adept, CodiPack, FADBAD показывает худший результат, так как оценка быстродействия линейно зависит от размерности задачи (размерности вектора управления). Исключение составляет пакет CoDiPack с ключем оптимизации –Ofast, где за счет эффективной реализации прямого метода и использования технологии AVX2 удается получить незначительное преимущество.

В Табл. 2 приведено время расчетов функции и ее градиента для расчетной сетки 300x6000 (300 интервалов по оси x и 6000 интервалов по оси времени). На данной расчетной

Табл. 1. Время расчета функции и градиента при I2=3000

	Пакет	Метод	-O2	-O3	-Ofast	-O2 -fopenmp
Функция	-	-	1.06	1.02	0.27	-
Функция и градиент	Adept	обратный	3.78	5.66	4.93	-
Функция и градиент	Adept	прямой	12.22	14.77	13.71	7.87
Функция и градиент	CoDiPack	прямой	11.57	6.72	2.78	-
Функция и градиент	FADBAD	прямой	51.35	52.53	51.40	-

Табл. 2. Время расчета функции и градиента при I2=6000

	Пакет	Метод	-O2	-O3	-Ofast	-O2 -fopenmp
Функция	-	-	1.85	1.80	0.44	1.85
Функция и градиент	Adept	обратный	106.20	193.49	170.77	106.20
Функция и градиент	CoDiPack	прямой	20.02	12.05	5.00	20.02
Функция и градиент	FADBAD	прямой	86.02	88.79	88.97	86.02

Табл. 3. Сходимость метода градиентного спуска и метода Левенберга-Марквардта

Номер шага/значение найденного минимума	Метод градиентного спуска	Метод Левенберга-Марквардта
Начальная точка	9.526582e-03	9.526582e-03
1	5.489041e-03	1.254659e-03
2	3.349420e-03	9.806420e-05
3	2.306810e-03	2.254612e-06
4	1.451139e-03	1.623696e-08
5	1.292372e-03	2.172197e-10
6	9.722118e-04	1.751552e-11
7	8.386604e-04	1.394123e-12
8	6.620138e-04	1.124777e-13
9	6.620138e-04	9.093626e-15
10	6.620138e-04	7.356306e-16
Время расчета	85 секунд	105 секунд

сетке пакету Adept требуется более 8Гб для хранения «информационного графа» и при расчетах используется виртуальная память.

Как видно из Табл. 2, нехватка оперативной памяти привела к существенной деградации производительности обратного метода пакета Adept. На основании Табл. 1 и Табл. 2 можно сделать вывод, что реализация прямого метода в пакете FADBAD существенно хуже, чем в пакете CoDiPack.

Во втором эксперименте сравниваются метод градиентного спуска и метод Левенберга-Марквардта. Для расчета градиента и матрицы Якоби используется наиболее эффективный для этой задачи пакет CoDiPack (прямой метод БАД). Расчеты производятся на сетке 300x6000 (300 интервалов по оси x и 6000 интервалов по оси времени).

Как мы видим, метод Левенберга-Марквардта обладает существенно более высокой скоростью сходимости. А так как время вычисления градиента и матрицы Якоби при прямом методе БАД практически совпадают, и матричное произведение $J(x)^T J(x)$ (*J(x)* матрица Якоби размерности 20x1 800 000 (размерность вектора управлений на количество экспериментальных данных) не требует значительного времени, то метод Левенберга-Марквардта оказывается вне конкуренции. На Рис. 1 и Рис. 2 приведены распределение управления *K(T)* для метода градиентного спуска и метода Левенберга-Марквардта соответственно.

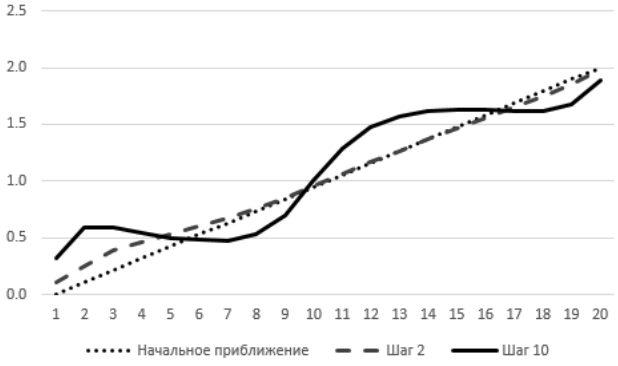

Рис. 1. Сходимость метода градиентного спуска

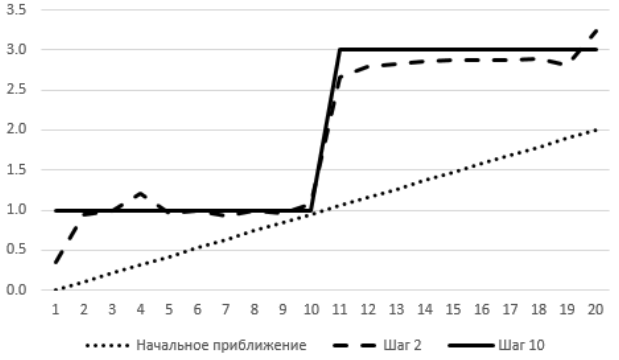

Рис. 2. Сходимость метода Левенберга-Марквардта

Заключение

Сравнительный анализ показал, что при решении задач оптимального управления, сводящихся к задаче о наименьших квадратах и имеющих небольшие размерности вектора управлений, для расчета градиента и матрицы Якоби целесообразно использовать прямой метод быстрого автоматического дифференцирования (например, пакет CoDiPack) и метод Левенберга-Марквардта (при условии его применимости). Исходя из опыта работы рекомендуется применять прямой метод при расчетах на персональных компьютерах без графических ускорителей для размерностей вектора управлений - 10^2, с графическими ускорителями - 10^3-10^4. В случае больших размерностей необходимо использовать обратный метод быстрого автоматического дифференцирования. При этом выбор используемого метода зависит от количества экспериментальных данных.

Литература

1. Айда-Заде К.Р., Евтушенко Ю.Г. Быстрое автоматическое дифференцирование // Математическое моделирование, 1989. Т. 1, С. 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. Евтушенко Ю.Г., Зубов В.И. Об обобщённой методологии быстрого автоматического дифференцирования // Ж. вычисл. матем. и матем. физ. 2016. Т. 56. № 11, С. 1847–1862.
6. Албу А. Ф., Зубов В. И. Об эффективности решения задач оптимального управления с помощью методологии быстрого автоматического дифференцирования //Труды Института математики и механики УрО РАН. – 2015. – Т. 21. – №. 4. – С. 20-29.
7. Hogan R. J. Fast reverse-mode automatic differentiation using expression templates in C++ //ACM Transactions on Mathematical Software (TOMS). – 2014. – Т. 40. – №. 4. – С. 26.
8. Albring T. et al. An aerodynamic design framework based on algorithmic differentiation //ERCOFTAC Bull. – 2015. – Т. 102. – С. 10-16.
9. Bendtsen C., Stauning O. FADBAD, a flexible C++ package for automatic differentiation //Department of Mathematical Modelling, Technical University of Denmark. – 1996.
10. Зубов В. И., “Применение методологии быстрого автоматического дифференцирования к решению обратной коэффициентной задачи для уравнения теплопроводности”, Ж. вычисл. матем. и матем. физ., 56:10 (2016), 1760–1774; Comput. Math. Math. Phys., 56:10 (2016), 1743–1757
11. Евтушенко Ю. Г. Оптимизация и быстрое автоматическое дифференцирование //М.: Научное издание ВЦ РАН. – 2013.
12. Griewank and A. Walther. 2008. Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation 2nd Ed. SIAM.
13. Гилл Ф., Мюррей У., Райт М. Практическая оптимизация. – 1985.

Горчаков Андрей Юрьевич. Ведущий математик Вычислительного центра им. А.А. Дородницына ФИЦ ИУ РАН. Окончил Московский физико-технический институт (государственный университет) в 1993 году. Кандидат физико-математических наук. Количество печатных работ: 7. Область научных интересов: вычислительная математика, информационные технологии. E-mail: andrgor12@gmail.com

About software packages for fast automatic differentiation

A.Y. Gorchakov

In the article, a comparative analysis of the packets of fast automatic differentiation Adept, CoDiPack and FADBAD is carried out. The example of the inverse coefficient problem shows the advantage of the direct method of fast automatic differentiation implemented in the CoDiPack package over the direct and inverse methods implemented in the Adept package and over the direct method implemented in the FADBAD package. As numerical optimization methods, the gradient descent method and the Levenberg-Marquardt method were used. Recommendations are given on the choice of optimization methods and rapid automatic differentiation, depending on the dimension of the problem.

Keywords: optimal control, fast automatic differentiation, gradient, Jacobi matrix, Levenberg-Marquardt method.

References

1. Ajda-Zade K.R., Evtushenko Ju.G. Bystroe avtomaticheskoe differencirovanie // Matematicheskoe modelirovanie, 1989. T. 1, S. 121-139.
2. Evtushenko Y.G., Mazurik V.P. Matematicheskoe obespechenie optimizacii. M: Znanie. 1989.
3. Evtushenko Y.G. Automatic differentiation viewed from optimal control theory // Proceedings of the Workshop on Automatic
4. Differentiation of Algorithms: Theory, Implementation and Application. Philadelphia: SIAM, 1991.
5. Griewank A, Corliss G.F., eds. Automatic Differentiation of Algoritms. Theory, Implementation and Application. Philadelphia: SIAM, 1991.
6. Evtushenko J.G., Zubov V.I. Ob obobshhjonnoj metodologii bystrogo avtomaticheskogo differencirovanija // Zh. vychisl. matem. i matem. fiz. 2016. T. 56. № 11, S. 1847–1862.
7. Albu A. F., Zubov V. I. Ob jeffektivnosti reshenija zadach optimal'nogo upravlenija s pomoshh'ju metodologii bystrogo avtomaticheskogo differencirovanija //Trudy Instituta matematiki i mehaniki UrO RAN. – 2015. – T. 21. – №. 4. – S. 20-29.
8. Hogan R. J. Fast reverse-mode automatic differentiation using expression templates in C++ //ACM Transactions on Mathematical Software (TOMS). – 2014. – T. 40. – №. 4. – S. 26.
9. Albring T. et al. An aerodynamic design framework based on algorithmic differentiation //ERCOFTAC Bull. – 2015. – T. 102. – S. 10-16.
10. Bendtsen C., Stauning O. FADBAD, a flexible C++ package for automatic differentiation //Department of Mathematical Modelling, Technical University of Denmark. – 1996.
11. Zubov V. I., “Primenenie metodologii bystrogo avtomaticheskogo differencirovanija k resheniju obratnoj kojefficientnoj zadachi dlja uravnenija teploprovodnosti”, Zh. vychisl. matem. i matem. fiz., 56:10 (2016), 1760–1774; Comput. Math. Math. Phys., 56:10 (2016), 1743–1757
12. Evtushenko Y. G. Optimizacija i bystroe avtomaticheskoe differencirovanie //M.: Nauchnoe izdanie VC RAN. – 2013.
13. A. Griewank and A. Walther. 2008. Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation 2nd Ed. SIAM.
14. Gill F., Murray U., Rait M. Prakticheskaja optimizacija. – 1985.

Gorchakov A.Yu. Dorodnicyn Computing Centre, FRC CSC RAS, Moscow, Russia.
E-mail: andrgor12@gmail.com