УДК 519.688+519.653

Туркин А.В.

Вычислительный центр им. А.А. Дородницына ФИЦ ИУ РАН, г. Москва, Россия; Национальный исследовательский университет «МИЭТ», Зеленоград, Россия

АВТОМАТИЧЕСКОЕ ДИФФЕРЕНЦИРОВАНИЕ В PYTHON

АННОТАЦИЯ

В статье рассматриваются вопросы применения современных программных средств автоматического дифференцирования в Python. Кратко приводятся основные характеристики некоторых инструментов и оценивается возможность их использования для решения задач большой размерности на примере проблемы нахождения оптимальной геометрической структуры кластера атомов. Оценивается время нахождения градиента целевой функции.

КЛЮЧЕВЫЕ СЛОВА

Автоматическое дифференцирование; программные библиотеки Python.

Turkin A.V.

Dorodnicyn Computing Centre, FRC CSC RAS, Moscow, Russia; National Research University of Electronic Technology, Zelenograd, Russia

AUTOMATIC DIFFERENTIATION IN PYTHON

ABSTRACT

The article discusses several tools for automatic differentiation in Python and its applicability to the task of finding an optimal geometric structure for a cluster with identical atoms. Such main characteristic of the tools as performance is assessed by measuring the time spent on calculation of the objective function gradient.

KEYWORDS

Automatic differentiation; Python tools.

Введение

Для численного решения многих задач оптимизации [1] необходимо получать значения производных, что в настоящее время может быть сделано с использованием библиотек автоматического дифференцирования [2]. Они доступны для большинства языков программирования, используемых на практике в настоящее время, в частности, такие средства созданы для такого языка как Python, который получил широкое распространение в научной среде [3][4][5][6]. Стоит отметить, что принципы, лежащие в основе методики автоматического дифференцирования [1,7,8], не относятся ни к численному, ни к символьному дифференцированию [9], а включают в себя набор приемов, которые приводят к вычислению производной с предопределенной точностью.

Выделяют два способа построения программ автоматического дифференцирования: с использованием перегрузки операторов и с применением методики преобразования исходного кода. Перегрузка операторов – это техника, которая предполагает переопределение таких элементарных операций как суммирование, умножение и деление для того, чтобы построить процедуру изменения градиента функции в соответствии с правилами дифференцирования. Второй подход – преобразование исходного кода – предполагает автоматическое переписывание части программы так, чтобы выполнить вычисление градиента для указанного участка. Несмотря на то, что реализация второго подхода сложнее, с его помощью возможно получение программ автоматического дифференцирования, производительность которых выше, чем при применении первого [10,11].

В настоящее время, многие исследователи используют Python для проведения научных расчетов, используя при этом множество свободно распространяемых библиотек. Некоторые из них могут быть использованы для осуществления автоматического дифференцирования программ, написанных с использованием конструкций этого языка. К таковым можно отнести следующие

библиотеки: PyADOL-C [12], PyCppAD [13], CasADi [14], Computation Graph Toolkit (CGT) [15], Theano [16,17] и AD [18]. Эти инструменты могут быть использованы как для вычисления первых, так и вторых производных, используя при этом перегрузку операторов (PyADOL-C, PyCppAD, Computation Graph Toolkit (CGT), Theano и AD) или преобразование исходного кода (CasADi) в качестве подхода к реализации методологии автоматического дифференцирования. Далее в статье будут рассмотрены все приведенные инструменты с целью оценки их производительности с точки зрения времени вычисления градиента рассматриваемой целевой функции.

Автоматическое дифференцирование в Python

Перед непосредственным использованием указанных инструментов, все они должны быть установлены надлежащим образом. В случае с Theano и AD, для этих целей может быть использован менеджер пакетов *pip*. Такие инструменты как PyCppAD, CasADi или CGT должны быть установлены вручную, используя информацию с сайтов разработчиков. Установка PyADOL-C может быть осуществлена с использованием, например, Homebrew в MacOS.

Для рассказа о возможностях того или иного инструмента автоматического дифференцирования необходимо поставить задачу нахождения производной функции. Для этой цели была выбрана задача поиска геометрической структуры кластера атомов с минимальной энергией их взаимодействия. Энергия взаимодействия двух частиц задается парным потенциалом $v(r)$, который определяет зависимость энергии взаимодействия от расстояния между частицами [24,25]:

$$E(\boldsymbol{x}) = \sum_{i<j} v\left(\rho(x_i - x_j)\right), \tag{1}$$

где $\rho(\cdot)$ – это расстояние между частицами x_i и x_j, $\boldsymbol{x}$ – вектор размера $3N$, определяющий положение N атомов, входящих в кластер.

Требуется найти производную этой функции, учитывая, что парный потенциал задан следующим образом (потенциал Леннарда-Джонса):

$$v(r) = r^{-12} - 2r^{-6}. \tag{2}$$

PyADOL-C

Библиотека ADOL-C [19], с помощью которой можно осуществлять автоматической дифференцирование программ, написанных на C++, реализована на основе перегрузки операторов. Весь функционал библиотеки может быть перенесен в Python с использованием программной оболочки PyADOL-C. Она использует тот же подход к написанию функций, которые должны быть продифференцированы, что и ADOL-C. Функции *trace_on* и *trace_off* используются для выделения участков кода, которые требуют дифференцирования. Для объявления так называемых активных переменных используется функция *adouble*, для определения переменных дифференцирования служат функции *independent* и *dependent*. Результат может быть получен с использованием функции *gradient* (см. рис. 1).

```
D = 3
def Adolc_vLJ(x):
  N = len(x)/D
  vLJ = 0
  for j in range(1,N):
    for i in range(j):
      rho = ((x[i*D:i*D+D] - x[j*D:j*D+D])**2).sum()
      vLJ += rho**(-6.0)-2*(rho**(-3.0))
  return vLJ
adolc.trace_on(0)
ad_x = adolc.adouble(np.zeros(np.shape(x),dtype=float))
adolc.independent(ad_x)
ad_y = Adolc_vLJ(ad_x)
adolc.dependent(ad_y)
adolc.trace_off()
g = adolc.gradient(0, x)
```

Рис.1. Программная реализация вычисления градиента g функции (1) для вектора x с использованием библиотеки PyADOL-C

PyCppAD

Библиотека PyCppAD [13] – это еще одна оболочка для C++ библиотеки, которая называется CppAD [20]. Для реализации функционала библиотеки в Python разработчиками был использован

тот же подход, что и при реализации PyADOL-C: они использовали Boost.Python [21], чтобы осуществить переход от C++ реализации. Библиотека PyCppAD использует функцию *independent*, чтобы определить переменные дифференцирования и функцию *adfun*, чтобы выделить интересующий участок кода. Результат может быть получен с использованием функции *jacobian* (см. рис. 2).

```
D = 3
ix = np.zeros(np.shape(x),dtype=float)
ad_x = independent(ix)
vLJt = 0
for j in range(1,N):
  for i in range(j):
    rho = ((ad_x[i*D:i*D+D] - ad_x[j*D:j*D+D])**2).sum()
    vLJt += rho**(-6.0)-2*(rho**(-3.0))
f = adfun(ad_x, np.array([vLJt]))
g = f.jacobian(x)
```

Рис.2. Программная реализация вычисления градиента g функции (1) для вектора x с использованием библиотеки CppAD

CasADi

Помимо библиотек, ориентированных только на решение задачи дифференцирования, существуют инструменты, которые призваны решать некоторые практические задачи оптимизации, основываясь на методике автоматического дифференцирования. К таким инструментам можно отнести библиотеку автоматического дифференцирования и численной оптимизации CasADi [14]. В ней используется подход на основе преобразования исходного кода. Как и все инструменты, приведенные выше, CasADi использует эффективность языка C++ для реализации всех процедур автоматического дифференцирования. Чтобы использовать этот функционал в Python, разработчики CasADi используют SWIG [23] вместо Boost.Python, как это было сделано для ADOL-C и CppAD.

Для реализации автоматического дифференцирования в Python, необходимо определить активные переменные путем использования двух возможных подходов: с использованием SX или MX объектов. На рис. 3 приводится реализация с использованием первого из приведенных объектов.

```
D = 3
N = len(x)
xc = SX.sym('xc',1,x.size)
vLJ = 0.0
for j in range(1,N):
  for i in range(j):
    rho = 0.0
    for d in range(D):
      rho += (xc[i*D+d] - xc[j*D+d])**2
   vLJ += rho**(-6.0)-2*(rho**(-3.0))
F = Function('F',[xc],[vLJ])
J = F.jacobian()
g = J.call(x)
```

Рис.3. Программная реализация вычисления градиента g функции (1) для вектора x с использованием библиотеки CasADi

Theano

Библиотека Theano [16,17], широко используемая в области машинного обучения, может быть применена для решения задач автоматического дифференцирования. Для работы с данными используется Numpy, а также имеется возможность применения GPU для вычислений, использующих большой объем данных. Theano использует специальные макросы из модуля *theano.tensor*, чтобы выполнять дифференцирование. Для получения первой производной функции необходимо использовать макрос grad из указанного модуля (см. рис. 4).

```
D = 3
N = len(x)
xt = T.dvector('xt')
vLJt = theano.shared(0.0, name='vLJt')
for j in range(1,N):
```

```
  for i in range(j):
    rho = ((xt[i*D:i*D+D] - xt[j*D:j*D+D])**2).sum()
    vLJt += rho**(-6.0)-2*(rho**(-3.0))
J = theano.function([xt], T.grad(vLJt, xt))
g = J(x)
```

Рис.4. Программная реализация вычисления градиента g функции (1) для вектора x с использованием библиотеки Theano

Computation Graph Toolkit

Разработчики этой библиотеки переписали функционал Theano с целью повышения вычислительной эффективности автоматического дифференцирования. В настоящий момент библиотека CGT [15] находится на стадии разработки, однако уже сейчас можно судить о ее возможностях. На рис. 5 приводится код, который используется для вычисления градиента функции (1). Как видно из этого рисунка, отличия в реализации заключаются в использовании типа данных без указания точности вычислений (vector вместо dvector), которая определяется с использованием функции *set_precision(.)*, что избавляет от использования нескольких типов данных.

```
D = 3
N = len(x)
xt = cgt.vector('xt')
vLJt = 0
for j in range(1,N):
  for i in range(j):
    rho = ((xt[i*D:i*D+D] - xt[j*D:j*D+D])**2).sum()
    vLJt += rho**(-6.0)-2*(rho**(-3.0))
J = cgt.function([xt], cgt.grad(vLJt, xt))
g = J(x)
```

Рис.5. Программная реализация вычисления градиента g функции (1) для вектора x с использованием библиотеки CGT

AD

Отличительной особенностью библиотеки AD [18] является возможность использования одной единственной функции *gh* для вычисления производной, написанной на Python. Рис. 6 демонстрирует использование такого подхода к вычислению производной функции (1).

```
def AD_vLJ(x):
  N = len(x)/D
  vLJ = 0.0
  for j in range(1,N):
    for i in range(j):
      rho = ((x[i*D:i*D+D] - x[j*D:j*D+D])**2).sum()
      vLJ += rho**(-6.0)-2*(rho**(-3.0))
  return vLJ
J = gh(AD_vLJ)[0]
g = J(x)
```

Рис.6. Программная реализация вычисления градиента g функции (1) для вектора x с использованием библиотеки AD

Экспериментальные данные

Учитывая, что вычисление производной может быть основной процедурой для некоторых приложений, важно использовать инструменты, показывающие высокую скорость получения производных. Для этой цели в работе была проведена оценка времени вычисления производной для каждого из представленных инструментов. Все эксперименты проводились на компьютере, имеющем процессор Intel Core i7 с 2.2 Гц и 16 ГБ памяти. Для оценки производительности использовался набор 1610 кластеров с различным числом атомов [25,26,27,28,29]. Для каждого элемента этого набора производился трехкратный подсчет значения производной с последующим усреднением результатов, чтобы оценить среднюю скорость вычисления градиента функции, что важно, например, для использования в процедурах локальной оптимизации. Стоит отметить, что время оценивалось только для процедуры подсчета градиента функции, без учета инициализации, т.е. в расчет бралась последняя строка каждой из приведенных ранее программ. Такой подход позволяет выяснить действительное время вычисления производной для некоторой конфигурации, необходимое для решения какой-либо оптимизационной задачи, т.е. когда требуется многократное повторение одной и той же процедуры для заданной функции. Все эксперименты,

данные о которых приводятся в Табл. 1 и на Рис.7, могут быть повторены с использованием программы, которую можно загрузить с GitHub: https://github.com/andreiturkin/MSU2016.ADTools.git.

В табл. 1 приводятся выборочные результаты этого эксперимента. Из них видно, что тремя самыми эффективными с вычислительной точки зрения являются инструменты: CasaADi, PyADOL-C и PyCppAD. Последний заметно проигрывает в скорости первым двум для кластеров большой размерности.

Табл.1. Выборочные значения времени вычисления градиента функции (1) с использованием таких инструментов автоматического дифференцирования в Python как PyADOL-C, PyCppAD, CasAD, CGT, AD и Theano. Знаком x *помечены те поля таблицы, для которых значения не были получены*

Кол-во атомов в кластере	Время, затраченное на вычисление градиента, сек.					
	PyADOL-C	PyCppAD	CasADi	CGT	AD	Theano
4	7.20024E-05	3.7988E-05	2.82923E-05	0.000101328	0.027191639	6.69956E-05
8	5.96046E-05	2.43187E-05	2.43187E-05	0.000361681	0.223882357	0.000200351
16	0.000106335	7.00156E-05	5.07037E-05	0.001779636	2.241400639	0.000754356
24	0.000132322	0.000132322	7.73271E-05	0.005251646	10.23096474	0.002060652
32	0.000179052	0.000165383	8.19365E-05	0.008975347	x	x
60	0.000407378	0.000645955	0.000238736	0.031063	x	x
120	0.001571337	0.00230368	0.00151666	0.140996059	x	x
250	0.006930033	0.010178725	0.004883687	1.553861698	x	x
550	0.034822305	0.050853332	0.025175015	x	x	x
700	0.060145696	0.083074649	0.047762632	x	x	x
900	0.085181634	0.135694027	0.081611713	x	x	x
1100	0.128189643	0.199071646	0.118741274	x	x	x
1300	0.203603347	0.303058386	0.161732992	x	x	x
1350	0.202304999	0.315208673	0.169876973	x	x	x
1400	0.226418654	0.388677041	0.189475377	x	x	x
1450	0.239901622	0.40497462	0.198199352	x	x	x
1550	0.270102978	0.481769005	0.242792606	x	x	x
1600	0.297964652	0.474971294	0.238883336	x	x	x

Полные данные о трех наиболее «быстрых» инструментах приведены в графическом виде на рис. 6. Они показывают, что CasADi показывает наилучшее время вычисления производной для большинства рассмотренных кластеров. Вторым и третьим по скорости вычислений являются такие инструменты как PyADOL-C и PyCppAD соответственно. Стоит отметить, что несмотря на удобный функционал библиотека AD она показала наихудший результат из всех рассмотренных инструментов.

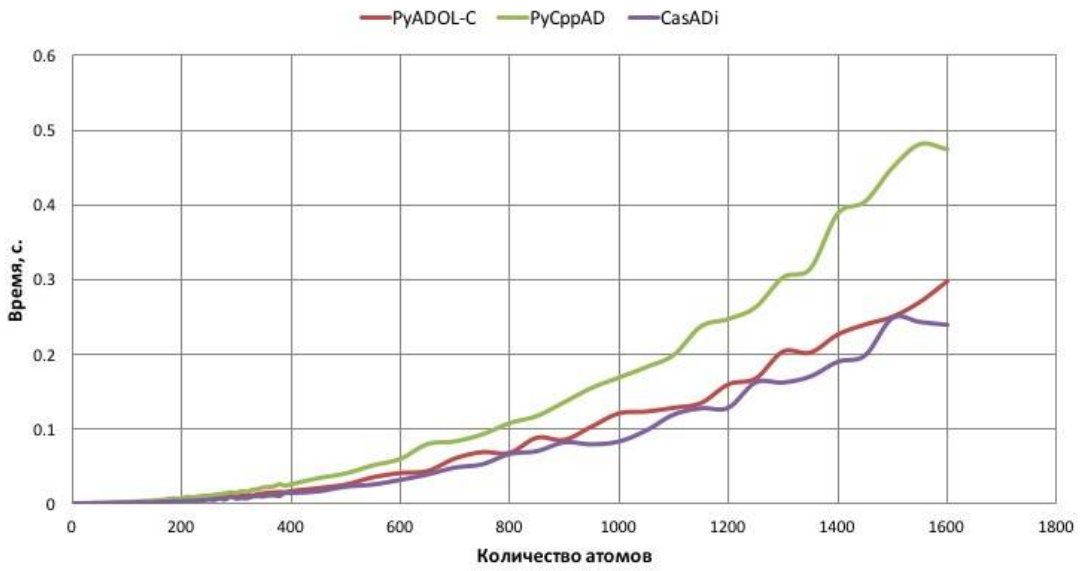

Рис.7. График зависимости времени расчета градиента функции (1) для заданной конфигурации кластера от количества атомов в нем. Приведены дынные только для трех инструментов автоматического дифференцирования: CasADi, PyCppAD и PyADOL-C, которые вычисляют градиенты функции (1) за кратчайшее время

Заключение

В этой статье были рассмотрены несколько инструментов автоматического дифференцирования и оценена скорость их работы без учета времени на инициализацию необходимых для осуществления дифференцирования структур данных. Было показано, что такой инструмент как CasADi обладает наибольшей производительностью среди рассмотренных инструментов для кластеров с различным количеством атомов. Второе и третье время показывают PyADOL-C и PyCppAD соответственно, что связано, по всей видимости, с подходом к реализации автоматического дифференцирования: CasADi использует преобразование кода, а PyADOL-C и PyCppAD – перегрузку операторов. Для кластеров больших размеров эта разница оказывается существенной, давая возможность сократить время расчетов, что важно для многократного использования этой процедуры в задачах оптимизации.

Литература

1. Евтушенко Ю. Г. Оптимизация и быстрое автоматическое дифференцирование //М.: Научное издание ВЦ РАН. – 2013. http://www.ccas.ru/personal/evtush/p/198.pdf
2. Community Portal for Automatic Differentiation. http://www.autodiff.org/
3. H. Koepke. Why Python Rocks for Research. Hacker Monthly, Issue 8, January 2011. https://www.stat.washington.edu/~hoytak/_static/papers/why-python.pdf
4. Filguiera R. et al. dispel4py: a Python framework for data-intensive scientific computing //International Journal of High Performance Computing Applications. – 2016. – С. 1094342016649766.
5. Mortensen M., Langtangen H. P. High performance Python for direct numerical simulations of turbulent flows //Computer Physics Communications. – 2016. – Т. 203. – С. 53-65.
6. Shukla X. U., Parmar D. J. Python–A comprehensive yet free programming language for statisticians //Journal of Statistics and Management Systems. – 2016. – Т. 19. – №. 2. – С. 277-284.
7. Griewank A., Walther A. Evaluating derivatives: principles and techniques of algorithmic differentiation. – Siam, 2008.
8. Griewank A. A mathematical view of automatic differentiation //Acta Numerica. – 2003. – Т. 12. – С. 321-398.
9. Baydin A. G., Pearlmutter B. A. Automatic differentiation of algorithms for machine learning //arXiv preprint arXiv:1404.7456. – 2014. http://arxiv.org/pdf/1404.7456.pdf
10. Weinstein M. J., Rao A. V. A source transformation via operator overloading method for the automatic differentiation of mathematical functions in MATLAB //ACM Transactions on Mathematical Software (TOMS). – 2016. – Т. 42. – №. 2. – С. 11.
11. Bischof C. H., Bücker H. M. Computing derivatives of computer programs //Modern Methods and Algorithms of Quantum Chemistry: Proceedings, Second Edition, NIC Series. – 2000. – Т. 3. – С. 315-327.
12. S. F. Walter. PyADOL-C: a python module to differentiate complex algorithms written in python. www.github.com/b45ch1/pyadolc/
13. B. M. Bell and S.F. Walter. Pycppad: Python algorithmic differentiation using cppad. Available: http://www.seanet.com/~bradbell/pycppad/pycppad.htm
14. Andersson J. A general-purpose software framework for dynamic optimization //Arenberg Doctoral School, KU Leuven. – 2013.
15. B. Stadie, Z. Xie, P. Moritz, J. Schulman, J. Ho. Computational graph toolkit: a library for evaluation and differentiation of functions of multidimensional arrays. https://github.com/joschu/cgt

16. F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio. Theano: new features and speed improvements. //Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.
17. J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: A CPU and GPU math expression compiler. //Proceedings of the Python for Scientific Computing Conference (SciPy), June 2010. Oral Presentation.
18. A.D. Lee AD: python package for first- and second-order automatic differentiation. http://pythonhosted.org/ad
19. Walther A., Griewank A. Getting started with ADOL-C //Combinatorial scientific computing. – 2012. – Т. 20121684.
20. B.M. Bell. Cppad: A package for differentiation of c++ algorithms. http://www.coin-or.org/CppAD
21. Abrahams D., Grosse-Kunstleve R. W. Building hybrid systems with Boost. Python //CC Plus Plus Users Journal. – 2003. – Т. 21. – №. 7. – С. 29-36.
22. Andersson J., Åkesson J., Diehl M. CasADi: A symbolic package for automatic differentiation and optimal control //Recent Advances in Algorithmic Differentiation. – Springer Berlin Heidelberg, 2012. – С. 297-307.
23. Beazley D. M. Automated scientific software scripting with SWIG //Future Generation Computer Systems. – 2003. – Т. 19. – №. 5. – С. 599-609.
24. Посыпкин М. А. Методы и распределенная программная инфраструктура для численного решения задачи поиска молекулярных кластеров с минимальной энергией //Вестник нижегородского университета им. НИ Лобачевского. – 2010. – №. 1.
25. Wales D. J., Doye J. P. K. Global optimization by basin-hopping and the lowest energy structures of Lennard-Jones clusters containing up to 110 atoms //The Journal of Physical Chemistry A. – 1997. – Т. 101. – №. 28. – С. 5111-5116.
26. Northby J. A. Structure and binding of Lennard-Jones clusters: 13≤ N≤ 147 //The Journal of chemical physics. – 1987. – Т. 87. – №. 10. – С. 6166-6177.
27. Shao X., Jiang H., Cai W. Parallel random tunneling algorithm for structural optimization of Lennard-Jones clusters up to n= 330 //Journal of chemical information and computer sciences. – 2004. – Т. 44. – №. 1. – С. 193-199.
28. Xiang Y. et al. Structural distribution of Lennard-Jones clusters containing 562 to 1000 atoms //The Journal of Physical Chemistry A. – 2004. – Т. 108. – №. 44. – С. 9516-9520.
29. Shao X., Xiang Y., Cai W. Structural transition from icosahedra to decahedra of large Lennard-Jones clusters //The Journal of Physical Chemistry A. – 2005. – Т. 109. – №. 23. – С. 5193-5197.

References

1. Y. G. Evtushenko. Optimization and fast automatic differentiation. Dorodnicyn Computing Center of Russian Academy of Sciences, 2013. [Online]. Available: http://www.ccas.ru/personal/evtush/p/198.pdf.
2. Community Portal for Automatic Differentiation. http://www.autodiff.org/
3. H. Koepke. Why Python Rocks for Research. Hacker Monthly, Issue 8, January 2011. https://www.stat.washington.edu/~hoytak/_static/papers/why-python.pdf.
4. Filguiera R. et al. dispel4py: a Python framework for data-intensive scientific computing //International Journal of High Performance Computing Applications. – 2016. – С. 1094342016649766.
5. Mortensen M., Langtangen H. P. High performance Python for direct numerical simulations of turbulent flows //Computer Physics Communications. – 2016. – Т. 203. – С. 53-65.
6. Shukla X. U., Parmar D. J. Python–A comprehensive yet free programming language for statisticians //Journal of Statistics and Management Systems. – 2016. – Т. 19. – №. 2. – С. 277-284.
7. Griewank A., Walther A. Evaluating derivatives: principles and techniques of algorithmic differentiation. – Siam, 2008.
8. Griewank A. A mathematical view of automatic differentiation //Acta Numerica. – 2003. – Т. 12. – С. 321-398.
9. Baydin A. G., Pearlmutter B. A. Automatic differentiation of algorithms for machine learning //arXiv preprint arXiv:1404.7456. – 2014. http://arxiv.org/pdf/1404.7456.pdf.
10. Weinstein M. J., Rao A. V. A source transformation via operator overloading method for the automatic differentiation of mathematical functions in MATLAB //ACM Transactions on Mathematical Software (TOMS). – 2016. – Т. 42. – №. 2. – С. 11.
11. Bischof C. H., Bücker H. M. Computing derivatives of computer programs //Modern Methods and Algorithms of Quantum Chemistry: Proceedings, Second Edition, NIC Series. – 2000. – Т. 3. – С. 315-327.
12. S. F. Walter. PyADOL-C: a python module to differentiate complex algorithms written in python. www.github.com/b45ch1/pyadolc/
13. B. M. Bell and S.F. Walter. Pycppad: Python algorithmic differentiation using cppad. Available: http://www.seanet.com/~bradbell/pycppad/pycppad.htm
14. Andersson J. A general-purpose software framework for dynamic optimization //Arenberg Doctoral School, KU Leuven. – 2013.
15. B. Stadie, Z. Xie, P. Moritz, J. Schulman, J. Ho. Computational graph toolkit: a library for evaluation and differentiation of functions of multidimensional arrays. https://github.com/joschu/cgt
16. F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio. Theano: new features and speed improvements. //Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.
17. J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: A CPU and GPU math expression compiler. //Proceedings of the Python for Scientific Computing Conference (SciPy), June 2010. Oral Presentation.
18. A.D. Lee AD: python package for first- and second-order automatic differentiation. http://pythonhosted.org/ad
19. Walther A., Griewank A. Getting started with ADOL-C //Combinatorial scientific computing. – 2012. – Т. 20121684.
20. B.M. Bell. Cppad: A package for differentiation of c++ algorithms. http://www.coin-or.org/CppAD
21. Abrahams D., Grosse-Kunstleve R. W. Building hybrid systems with Boost. Python //CC Plus Plus Users Journal. – 2003. – Т. 21. – №. 7. – С. 29-36.
22. Andersson J., Åkesson J., Diehl M. CasADi: A symbolic package for automatic differentiation and optimal control //Recent Advances in Algorithmic Differentiation. – Springer Berlin Heidelberg, 2012. – С. 297-307.
23. Beazley D. M. Automated scientific software scripting with SWIG //Future Generation Computer Systems. – 2003. – Т. 19. – №. 5. – С. 599-609.
24. M.A. Posypkin. Searching for minimum energy molecular cluster: Methods and distributed software infrastructure for numerical solution of the problem. Vestnik of Lobachevsky University of Nizhni Novgorod, (1):210 – 219, 2010.
25. Wales D. J., Doye J. P. K. Global optimization by basin-hopping and the lowest energy structures of Lennard-Jones clusters

containing up to 110 atoms //The Journal of Physical Chemistry A. – 1997. – Т. 101. – №. 28. – С. 5111-5116.
26. Northby J. A. Structure and binding of Lennard-Jones clusters: 13≤ N≤ 147 //The Journal of chemical physics. – 1987. – Т. 87. – №. 10. – С. 6166-6177.
27. Shao X., Jiang H., Cai W. Parallel random tunneling algorithm for structural optimization of Lennard-Jones clusters up to n= 330 //Journal of chemical information and computer sciences. – 2004. – Т. 44. – №. 1. – С. 193-199.
28. Xiang Y. et al. Structural distribution of Lennard-Jones clusters containing 562 to 1000 atoms //The Journal of Physical Chemistry A. – 2004. – Т. 108. – №. 44. – С. 9516-9520.
29. Shao X., Xiang Y., Cai W. Structural transition from icosahedra to decahedra of large Lennard-Jones clusters //The Journal of Physical Chemistry A. – 2005. – Т. 109. – №. 23. – С. 5193-5197.

Поступила 15.10.2016

Об авторах:

Туркин Андрей Владимирович, младший научный сотрудник отдела прикладных проблем оптимизации Вычислительного центра им. А.А. Дородницына ФИЦ ИУ РАН, доцент кафедры Вычислительной техники Национального исследовательского университета “МИЭТ”, кандидат физико-математических наук, andrei_turkin@hotmail.com.