ПРОТОТИПИРОВАНИЕ ПРОГРАММНЫХ КОМПЛЕКСОВ

И.В. Оселедец

Разработка сложных программных комплексов требует больших программистких затрат. Также, растет алгоритмическая сложность разработки вычислительных алгоритмов — большая часть времени уходит на отладку программного кода. Использование среды MATLAB позволяет существенно сократить время на реализацию и проверку алгоритмов, выбрать из них наиболее удачный, а также провести исследование применимости новых подходов к различным прикладным задачам. Это — так называемый этап *прототипирования*. Такой программный комплекс можно использовать для проведения численных экспериментов, анализа данных, подготовке данных для различных академических исследований. Однако подход, основанный на языке MATLAB, нельзя назвать самодостаточным в силу целого ряда понятных причин.

Среди них:

- MATLAB является платным, закрытым программным продуктом. При разработке собственного программного комплекса необходимо использовать саму среду MATLAB, поэтому программный комплекс, основанный на MATLAB, требует дополнительных наклады
- Код на языке MATLAB не является оптимальным, возможно существенное ускорение за счет использования компилируемых языков (в первую очередь, это языки C и Fortran).
- Написание параллельной версии на MATLAB затруднительно
- Включение различных библиотек, написанных на C или Fortran, достаточно трудоемко.

Поэтому, при написании прототипа на языке MATLAB процесс создание высокопроизводительного программного комплекса выглядит следующим образом.

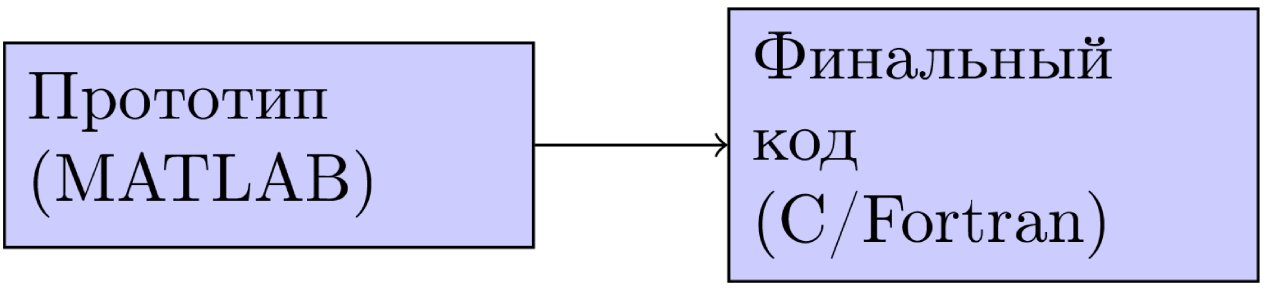

Рис. 1

Недостаток такого подхода состоит в том, что необходимо для создания высокоэффективного программного комплекса фактически переписывать код заново, при этом необходимо быть полностью уверенным в качестве предложенных алгоритмов. Если по ходу разработки возникнут алгоритмические сложности, снова придется возвращаться к процессу прототипирования, что приводит к дополнительным затратам. Поэтому возникает вопрос о том, как осуществлять процесс прототипирования программного комплекса вместе с написанием параллельной версии, а также создания удобной оболочки для работы с комплексом, визуализации результатов, при этом сохраняя удобство работы с интерпретатором, как это сделано в среде MATLAB.

Требования к среде разработки

Попытаемся сформулировать требования к среде разработке, которые позволили бы существенно сократить затраты на написание прототипов и высокоэффективных программных комплексов.

- Интерпретируемый, динамически типизированный язык
- Наличие большого количества прикладных библиотек (операции с матрицами, быстрые преобразования)
- Расширяемость: возможность реализовывать наиболее узкие места в качестве отдельных модулей на низкоуровневых языках, а также использовать имеющиеся наработки на других языках программирования без необходимости полного переписывания программного кода
- Наличие удобной системы визуализации, системы создания графических интерфейсов
- Возможность работы части модулей на системах с распределенной памятью (кластерах) а также на графических картах

Как уже упоминалось выше, MATLAB не удовлетворяет части этих требований, поэтому возникает необходимость искать альтернативы. В качестве программной платформы было принято решение начать использовать Python.

Прототипирование на Python

В научных вычислениях в последние годы все большее внимание обращается на язык Python (www.python.org) как основу для написания простых в использовании и при этом высокоэффективных программных комплексов. Опишем кратко некоторые отличительные особенности Python.

- Интерпретируемый, динамически типизированный язык (как в MATLAB)
- Объектно-ориентированный язык высокого уровня, направленный на простоту написания сложных программ
- Наличие большого числа прикладных библиотек (модулей), в первую очередь библиотека Numpy (numpy.scipy.org) для работы с многомерными массивами и матрицами и библиотека Scipy (scipy.org0, которая содержит, например, методы оптимизации, интегрирования, решения линейных систем.

Использование связки Python+Numpy+Scipy позволяет почти полностью заменить функциональность языка MATLAB (см, например, http://www.scipy.org/NumPy_for_Matlab_Users). Среди преимуществ Python — его распространение под свободной лицензией, а среди недостатков — отсутствие полностью интегрированной среды разработки такого же качества, как и MATLAB (хотя есть уже достаточно интересные среды, такие как Ipython (www.ipython.org).

Написание программы на чистом Python, в особенности при использовании циклов, приводит к очень медленному выполнению (аналогично ситуации с MATLAB). С этим можно бороться, сводя задачу к операциям с матрицами (векторизация), однако далеко не во всех случаях это можно сделать. Для этого предложены и активно используются целый ряд программных средств, направленных на соединение Python с кодами, написанными на компилируемых языках, таких как Fortran и C. Более того, эти средства позволяют использовать уже готовые наработки без существенного изменения программного кода. Python превращается в своего рода склейку между различнми языками, когда низкоуровневая обработка осуществляется оптимизированными библиотеками, а взаимодействие между программными блоками и управление заданиями осуществляется на более высоком уровне. Процесс написания прототипа программного комплекса можно представить следующим образом (см. Рис.). Основные блоки (алгоритмы, тесты, графический интерфейс, ввод данных) можно реализовать на Python достаточно быстро.

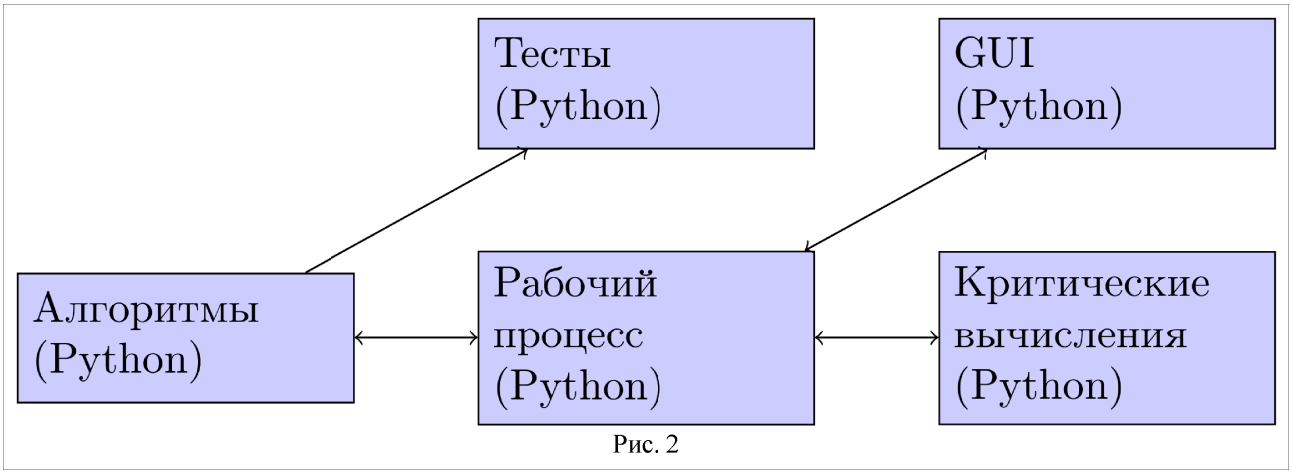

Рис. 2

После этого, вторым шагом является ускорение узких мест работы программы. Многие части программы не нужно переписывать, а только те, которые сильно влияют на производительность, при этом работа с самой программой остается такой же!

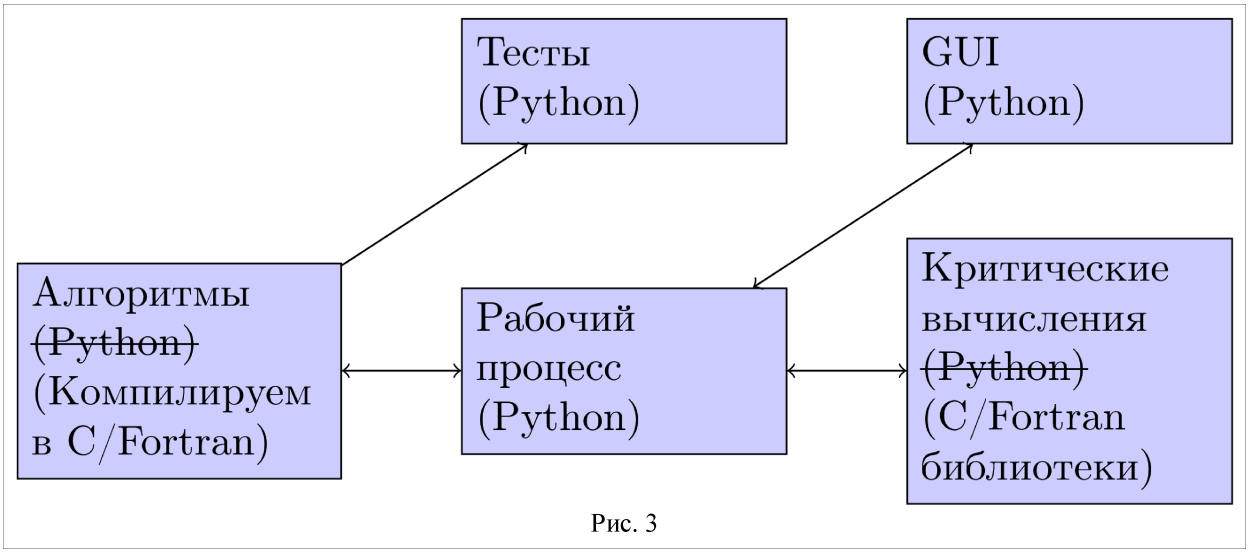

Рис. 3

Анализ инструментов для написания эффективного написания программ на Python

Опишем кратко инструменты, которые требуются для эффективного написания программного комплекса на языке и в среде Python. Нам потребуются средства для создания модулей на основе кода, написанного на компилируемых языках (в первую очередь, C и Fortran), а также (в дальнейшем) системы визуализации данных, создания графических интерфейсов, работы на параллельных машинах.

F2PY

Огромное количество вычислительного кода написано на языке Fortran. Он достаточно удобен для работы с многомерными массивами, а современные стандарты языка позволяют использовать различные конструкции, мало уступающие другим языкам высокого уровня (например, C++). Тем не менее, нас интересует в основном некоторые простые программы, написанные на Fortran, основной задачей которых является обработка массивов в некоторых циклах, так как циклы на Python являются одним из самых узких и медленных мест языка.

Пакет F2py, входящий в состав Scipy (http://www.scipy.org/F2py), позволяет очень просто создавать Python-модули, основанные на Fortran-коде, если входные и выходные данные для требуемой функции имеют простой тип данных: либо скаляры, либо многомерные массивы. Рассмотрим работу F2PY на простейшем примере. Сразу отметим, что новые процедуры будут писаться на Fortran90. Для подключения старых модулей синтаксис немножко меняется, но тем не менее это не очень критично. Итак, пусть у нас есть некоторая простая Fortran-подпрограмма (пусть она находится в файле test.f90)

```
subroutine test(a,n,b,c)
integer, intent(in) :: n
double precision, intent(in) :: a(n), b(n)
double precision, intent(out) :: c(n)
integer i
do i=1,n
  c(i) = a(i)+b(i)
end do
end subroutine test
```

Тогда, создание Python-модуля происходит командой

```
f2py -c -m test test.f90
```

Эта команда создает модуль test.so, который можно напрямую подключать к Python. Это делается следующим образом

```
import numpy as np
from numpy.random import randn
import test
```

```
n=128
a=randn(n)
b=randn(n)
c=test.test(a,b)
print test.__doc__
print test.test.__doc__
This module 'test' is auto-generated with f2py (version:2).
Functions:
 c = test(a,b,n=len(a))
.
test - Function signature:
 c = test(a,b,[n])
Required arguments:
 a : input rank-1 array('d') with bounds (n)
 b : input rank-1 array('d') with bounds (n)
Optional arguments:
 n := len(a) input int
Return objects:
 c : rank-1 array('d') with bounds (n)
```

Как видно, F2PY-сгенерированный модуль позволяет передавать в качестве входных данных Numpy-массивы (де-факто стандарт по численному представлению массивов в Python), и при этом размеры этих массивов получаются автоматически (т.е. нет необходимости явно передавать в процедуру параметр d). Также автоматическим образом генерится документация к каждой функции. Также можно передавать в качестве аргументов в Fortran Python-функции, т.е. такие объекты Python, которые допускают простые (скаляры или массивы) входные данные. Таким образом можно реализовывать обратную связь. Это полезно, например при работе крестового метода аппроксимации многомерных массивов.

Cython

Часто, наиболее узкие места в программе на Python представляют собой некоторые циклы. Основной сложностью при работе с такими циклами является то, что даже в простейшем цикле по целочисленной переменной, каждая переменная хранится как объект Python, что приводит к огромному замедлению. Для решения этой проблемы было предложено в наиболее критически важных местах давать дополнительную информацию на этапе компиляции модуля о типе конкретных переменных. На этом основан Cython (www.cython.org), который, фактически является компилятором для Python. Язык Cython содержит некоторые дополнительные директивы, позволяющие объявить тип конкретной переменной. При этом основной целью является то, что готовый, работающий код на Python модифицируется минимально. Приведем пример работы Cython (на примере умножения ленточной матрицы на вектор). Код на Python может выглядеть следующим образом.

```
def band_matvec(A,u):
  result = zeros(u.shape[0],dtype=u.dtype)

  for i in xrange(A.shape[1]):
     result[i]=A[0,i]*u[i]

  for j in xrange(1,A.shape[0]):
     for i in xrange(A.shape[1]-j):
        result[i] += A[j,i]*u[i+j]
        result[i+j] += A[j,i]*u[i]
  return result
```

А код на Cython:

```
import numpy,scipy
cimport numpy as cnumpy

ctypedef cnumpy.float64_t reals
def matvec_help(cnumpy.ndarray[reals,ndim=2] A,
```

```
            cnumpy.ndarray[reals,ndim=1] u):
    cdef Py_ssize_t i,j
    cdef cnumpy.ndarray[reals,ndim=1] result = \
      numpy.zeros(A.shape[1],dtype=A.dtype)
    for i in xrange(A.shape[1]):
        result[i]=A[0,i]*u[i]
    for j in xrange(1,A.shape[0]):
        for i in xrange(A.shape[1]-j):
            result[i]=result[i]+A[j,i]*u[i+j]
            result[i+j]=result[i+j]+A[j,i]*u[i]
```

Такой достаточно простой подход дает ускорение в сотни раз по сравнению с чистым Python-кодом при сравнительно небольшом объеме дополнительного кода. При этом, Cython может использовать как обертка для более сложного C-кода. Для этого достаточно собрать библиотеку из C-функций и объявить соответствующую функцию как extern в Cython-файле (который имеет расширение .pyx).

Интерактивные среды

Одним из достоинств MATLAB является наличие единой системы запуска программ, визуализации, отладки, профилировке, редактирования кодов. В экосистеме Python в качестве среды разработки основным стал проект IPython (Interactive Python) (http://ipython.org/) который позволяет работать как с различными системами визуализации, предоставляет мощные средства по поиску документации, автодополнению, истории команд и многое другое. Также есть веб-интерфейс для среды разработки, который позволяет создавать встроенные графики, рисовать математические выражения, и многое другое.

Системы визуализации

Визуализация результатов — важный элемент вычислительной экосистемы. В MATLAB существует прекрасная система для построения разнообразных двумерных и трехмерных графиков. В экосистеме Python+Scipy основным средством для построения графиков является система Matplotlib (http://matplotlib.sourceforge.net/). Ее синтаксис во многом повторяет MATLAB. Достоинством Matplotlib является то, что он не зависит от конкретного пакета, осуществляющего рисования, а предоставляет единый интерфейс к каждому. Это делает возможным встраивать Matplotlib-графики в промышленные пакеты. Пример использования Matplotlib.

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np

fig = plt.figure()
ax = fig.gca(projection='3d')
X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)
surf=ax.plot_surface(X, Y, Z, rstride=1, cstride=1, color='0.8',
    alpha=0.85, linewidth=1)
fig.savefig("test1.png")
fig.clf()
```

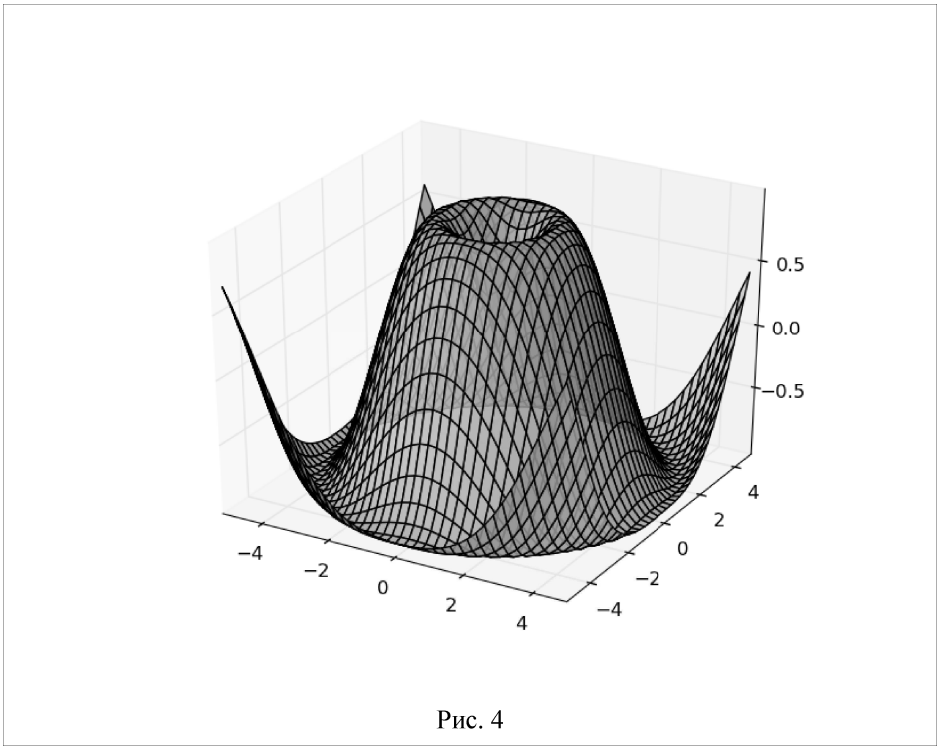

Рис. 4

Для визуализации в реальном времени (например, в задачах аэродинамики) можно использовать профессиональные пакеты, основанные на OpenGL. Это, в первую очередь, VTK (www.vtk.org) и Mayavi (http://code.enthought.com/projects/mayavi/), которые представляют собой более высокий уровень по сравнению со стандартными операциями OpenGL. Их использование позволяет также осуществлять взаимодействие со сложными пакетами для 3D моделирования.

Работа на высокопроизводительных архитектурах

Одним из достоинств Python является то, что для него доступны модули практически всех важных библиотек. Не является исключением и параллельное программирование с использованием технологии MPI. Для MPI в Python существует несколько оболочек, но наиболее известной и широко используемой из них является mpi4py (http://mpi4py.scipy.org)yyy. Он обладает простым синтаксисом и позволяет выполнять базовые операции (пересылка данных и т.п.) на кластере с установленным MPI. Приведем пример использования mpi4py.

```
from mpi4py import MPI
import numpy

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

# pass explicit MPI datatypes
if rank == 0:
  data = numpy.arange(1000, dtype='i')
  comm.Send([data, MPI.INT], dest=1, tag=77)
elif rank == 1:
  data = numpy.empty(1000, dtype='i')
  comm.Recv([data, MPI.INT], source=0, tag=77)

# automatic MPI datatype discovery
if rank == 0:
  data = numpy.arange(100, dtype=numpy.float64)
  comm.Send(data, dest=1, tag=13)
elif rank == 1:
  data = numpy.empty(100, dtype=numpy.float64)
  comm.Recv(data, source=0, tag=13)
```

Простейший вариант параллельного матрично-векторного произведения осуществляется с помощью программы

```
from mpi4py import MPI
import numpy

def matvec(comm, A, x):
  m = A.shape[0] # local rows
  p = comm.Get_size()
  xg = numpy.zeros(m*p, dtype='d')
  comm.Allgather([x,  MPI.DOUBLE],
            [xg, MPI.DOUBLE])
  y = numpy.dot(A, xg)
  return y
```

Отметим, что накладные расходы достаточно невысоки. Также, можно использовать уже имеющийся код, с использованием F2PY или SWIG.

Также, для Python существуют библиотеки для работы на графических картах (PyCuda и PyOpenCL) (http://documen.tician.de/pycuda/).

Создание графических интерфейсов

Разработка графических интерфейсов является не последней задачей при создании программных комплексов. Практически все основные системы по созданию и прототипированию таких интерфейсов имеют Python-модули. Среди них:

- PyQt
- PySide
- Tk
- GTK
- и многие другие

Среди них наиболее интересным видится PySide, который почти совпадает по функциональности с PyQt, но распространяется по свободной лицензии.

Реализация основных процедур TT-Toolbox на Python

Была выполнена реализация части функциональности программного комплекса TT-Toolbox (MATLAB-версия доступна на сайте http://github.com/oseledets/TT-Toolbox) на языке Python. Комплекс позволяет выполнять базовые операции с многомерными массивами (тензорами), представленными в так называемом Tensor Train (TT) формате. Целью исследования было сравнить скорость работы связки Python+Fortran и MATLAB-прототипа. Основные вычислительные процедуры были реализованы на Fortran, а для них был создан интерфейс с помощью пакета F2PY, позволяющий вызывать их на языке Python. Следующие процедуры были реализованы на Fortran и могут быть вызваны из Python.

- Перевод полного тензора в TT-формат
- Перевод из TT-формата в полное представление
- Процедура TT-округления
- Сложение двух TT-тензоров
- Вычисление поэлементного произведения двух TT-тензоров
- Вычисление скалярных произведений и нормы вектора
- Умножение TT-матрицы на полный вектор

Несмотря на то, что эти процедуры в основном требуют лишь матрично-матричных умножений, версия Python+Fortran во многих случаях оказывается заметно быстрее, чем версия MATLAB.

Структура программы полностью повторяет структуру объектов TT-Toolbox с небольшими изменениями.

В таблицах ниже строчку Python следует понимать, как результат работы гибридной Python+Fortran версии, которая вызывается из Python.

Сравнение производительности

Был взят полный тензор, соответствующий квантизованному представлению функции

$$f(x)=\frac{1}{x},$$

Также, был проведен следующий эксперимент. Строились случайные тензоры в ТТ-формате и вычислялось время перехода от полного тензора к ТТ-формату

d	10	12	14	16	18	20
MATLAB	0.0002	0.0002	0.0004	0.0011	0.0243	0.075
Python	0.0002	0.0002	0.0006	0.0022	0.0123	0.0604

Для процедуры округления тензора в ТТ-формате был проведен следующий эксперимент. Был взят случайный ТТ-тензор с $n_k = 2$ и различными d, и рангом 10. После этого, вычислялась сумма (с рангами 20) после чего производилось округление (до рангов 10). Времена расчета приведены в таблице.

d	10	20	40	80	120
MATLAB	0.0021	0.0062	0.0142	0.0641	0.0645
Python	0.0008	0.0015	0.0035	0.0074	0.0114

Важной операций является сложение двух тензоров. Был проведен следующий эксперимент. Был взят случайный ТТ-тензор с $n_k = 2$ и различными d, и рангом 10. После этого, вычислялась сумма такого тензора с самим собой. Времена расчета приведены в таблице.

d	10	20	40	80	120
MATLAB	4.5e-04	7.5e-04	1.3e-03	2.6e-03	3.7e-03
Python	5.9e-05	3.7e-05	6.5e-05	3.0e-04	3.8e-04

Также, для разных порядков ТТ-тензоров были измерено время вычисления адамарового (поэлементного) произведения). ТТ-ранги брались равными 20. Результаты расчетов приведены в таблице.

d	10	20	40	80	120
MATLAB	5.2e-03	1.3e-02	2.4e-02	1.4e-01	1.7e-01
Python	1.2e-03	3.7e-03	8.5e-03	2.0e-02	3.3e-02

И, наконец, для разных порядков ТТ-тензоров были измерено время вычисления нормы тензора. ТТ-ранги брались равными 10. Результаты расчетов приведены в таблице.

d	10	20	40	80	120
MATLAB	4.4e-04	8.9e-04	1.8e-03	3.6e-03	5.4e-03
Python	3.5e-04	1.5e-04	3.1e-04	6.4e-04	1.0e-03

Заключение

В заключение отметим, что в данной работе технологии создания программных комплексов с использованием Python были успешно применены для улучшения имеющегося программного кода без ухудшения взаимодействия с ним.

ЛИТЕРАТУРА:

1. Python www.python.org
2. Cython www.cython.org
3. Oseledets I. V. Tensor-train decomposition // SIAM J. Sci. Comput. — Vol. 33, no. 5. — Pp. 2295–2317.