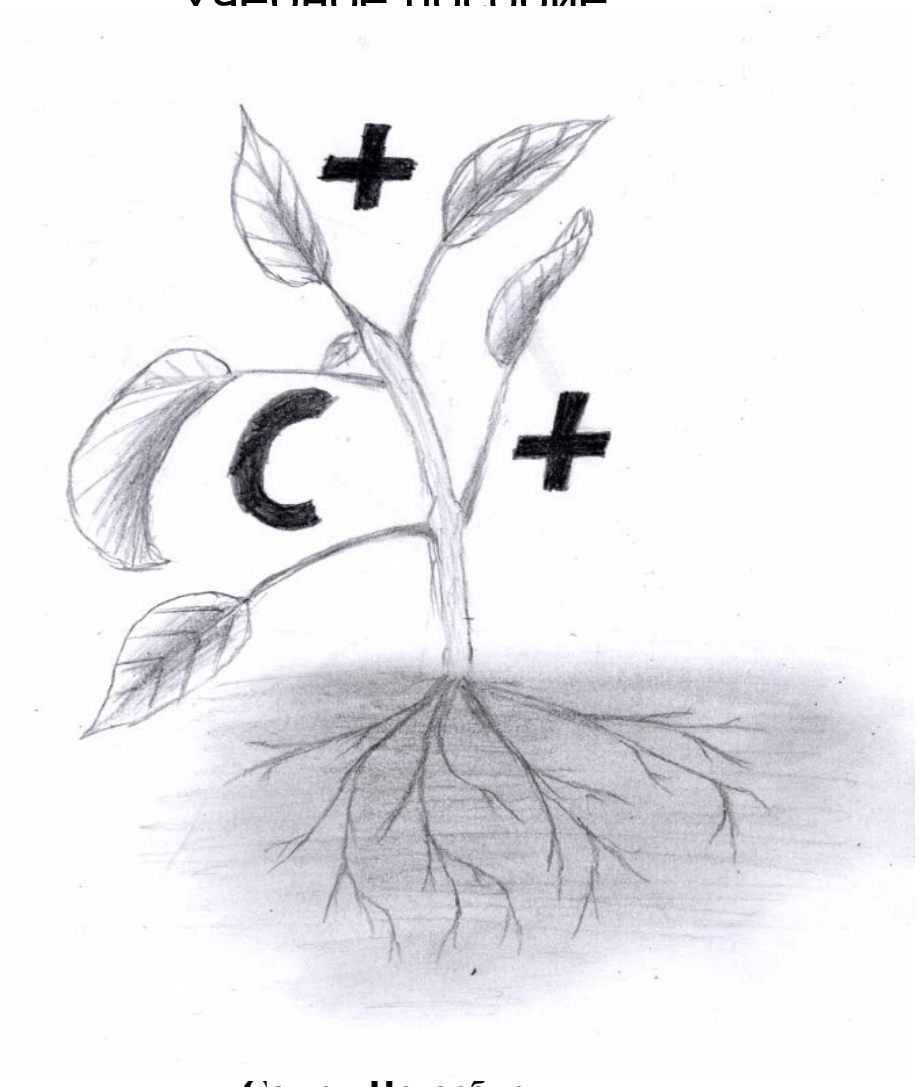
УНИВЕРСИТЕТ ИТМО

А.А. Никехин

Моделирование на C++. Приложение I. Библиотеки научных расчетов

Учебное пособие



Санкт–Петербург
2015

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

А.А. Никехин

Моделирование на C++. Приложение I. Библиотеки научных расчетов

Учебное пособие

УНИВЕРСИТЕТ ИТМО

Санкт–Петербург
2015

Никехин А.А. Основы C++ для моделирования и расчетов. Часть 2. Библиотеки для научных вычислений: Учебное пособие. – СПб: Университет ИТМО, 2016. – 64 с.

Пособие адресовано для студентов, обучающихся по направлениям 16.04.01 «Техническая физика», 18.04.02 «Энерго- и ресурсосберегающие процессы в химической технологии, нефтехимии и биотехнологии»

Содержит общие сведения по установке и применению вычислительных библиотек линейной алгебры, компьютерного зрения и машинного обучения в проектах на C++.

Рекомендовано к печати Ученым советом факультета лазерной и световой инженерии, протокол № 3 от 10.03.2015.

Университет ИТМО – ведущий вуз России в области информационных и фотонных технологий, один из немногих российских вузов, получивших в 2009 году статус национального исследовательского университета. С 2013 года Университет ИТМО – участник программы повышения конкурентоспособности российских университетов среди ведущих мировых научно-образовательных центров, известной как проект «5 в 100». Цель Университета ИТМО – становление исследовательского университета мирового уровня, предпринимательского по типу, ориентированного на интернационализацию всех направлений деятельности.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В первой части методического пособия были максимально кратко рассмотрены основы языка C++ и технологические способы, позволяющие осуществлять производительные вычисления на массовых компьютерах: параллельные вычисления с помощью OpenMP, использование векторных вычислений, вычисления с использованием графических карт (GPU).

Во второй части методического пособия будут разобраны некоторые конкретные примеры использования языка C++ и связанных с ним технологий программирования для моделирования различных объектов и систем. Примеры и описания составлены на основе и с использованием примеров и документации к соответствующим библиотекам. В полном объеме на английском языке примеры и документация доступны на страницах сайтов разработчиков по ссылкам:

- boost (набор различных библиотек) http://www.boost.org/
- boost.compute (ускорение вычислений) https://boostorg.github.io/compute/index.html
- Eigen (линейная алгебра) http://eigen.tuxfamily.org/
- Armadillo (линейная алгебра) http://arma.sourceforge.net/
- Shark (машинное обучение) http://image.diku.dk/shark/
- OpenCV (компьютерное зрение) http://opencv.org/

Также все коды и готовые для использования проекты в среде Code::Blocks можно найти в репозитории автора: https://github.com/nikekhin

Ввиду важности, частично повторим здесь необходимую информацию для начала работы. А именно, для начала работы необходимо установить и настроить среду разработки. На момент публикации разработку можно вести в различных операционных системах (Linux, Windows, MacOS). Из очень большого списка интегрированных сред разработки необходимо выделить следующие бесплатные варианты:

- Visual Studio Community 2015 (https://www.visualstudio.com)
- Qt Creator (http://www.qt.io/download-open-source/)
- Code::Blocks (http://www.codeblocks.org/)
- Eclipse CDT (https://eclipse.org/cdt/)
- NetBeans IDE (https://netbeans.org/features/cpp/)

Независимо от того, что вы предпочтете установить, загруженный дистрибутив будет включать в себя текстовый редактор для редактирования кода, компилятор C++, отладчик. В каждом из IDE есть генератор проекта по умолчанию. В примерах, которые будут приводиться в данном документе, чтобы не погружаться в особенности построения графических приложений (GUI), все приложения будут собираться как консольные, т.е. использующие текстовое окно.

При необходимости можно установить и использовать различные компиляторы C++ совместно с перечисленными выше IDE. Это потребует

дополнительных настроек от пользователя и не обязательно для работы с примерами, описанными в этом документе. Примеры таких компиляторов и SDK (Software Development Kit):

- Visual Studio Community 2015 с SDK, который будет установлен в "C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\bin\" или отдельный Windwows 10 SDK
- MinGW - минимальная среда разработки для Windows
- TDM-GCC - набор компиляторов для Windows
- Clang - C++ в инфраструктуре компиляторов LLVM

Для работы примеров будет необходима библиотека boost. Ниже описаны шаги по установке и настройке Code::Blocks и boost в Windows. При работе с MSVC можно воспользоваться имеющимися готовыми сборками и загрузить бинарные файлы отсюда: http://sourceforge.net/projects/boost/files/boost-binaries/.

Загрузка и установка необходимого программного обеспечения

Загрузка Boost

Загрузить Boost можно со следующей страницы: «http://www.boost.org/users/history/version_1_59_0.html»

Загружен файл «boost_1_59_0.7z».

В нашем случае для проекта будет выбрана следующая папка: C:_lib\boost_1_59_0

Но можно для собственного проекта выбрать любую другую.

Исправление ошибок в Boost

К сожалению, возможны ошибки в конкретной версии Boost при сборке определенными компиляторами и в определенной операционной системе. Во всех версиях старше 1.55, включая последнюю на данный момент версию 1.59, имеются ошибки в библиотеке сериализации (смотрите ссылки):

- http://permalink.gmane.org/gmane.comp.lib.boost.user/82831
- http://sourceforge.net/p/shark-project/mailman/message/32882318/
- https://svn.boost.org/trac/boost/ticket/11548
- https://github.com/boostorg/serialization/pull/19/files

Поэтому после загрузки boost необходимо обновить библиотеку serialization. Поступим следующим образом: Загружаем эту суб-библиотеку отсюда: «https://github.com/boostorg/serialization». Из полученного архива "serialization-master.zip" копировать все содержимое \serialization-master\include\boost\

в C:_lib\boost_1_59_0\boost\, а все остальное из \serialization-master\ копировать в C:_lib\boost_1_59_0\libs\serialization\.

Хочется надеяться, что все недоразумения будут исправлены к моменту чтения настоящего документа. Если они останутся, ответы можно найти в Интернет.

Кроме всего, потребуется, еще не включенная в официальный релиз 1.59 библиотека boost.compute. Загрузить ее можно отсюда: «https://github.com/boostorg/compute». Из загруженного архива compute-master.zip содержимое папки \compute-master\include\boost\ необходимо скопировать в папку, в которой установлен boost: «C:_lib_1_59_0».

В папке C:_lib\boost_1_59_0\libs\ создать каталог "compute" и перенести туда все остальное содержимое \compute-master\ кроме уже скопированного ранее "compute".

Загрузка MASM

Теперь можно загрузить и установить MASM (требуется для сборки Boost) http://www.masm32.com/index.htm. Загружен архив "masm32v11r.zip" и программа установлена в "C:/masm32". В PATH вручную добавлен путь "C:/masm32". Если уже установлена Visual Studio, то можно использовать SDK, устанавливаемый вместе с ней, убедившись, что путь к ассемблеру ml.exe указан в переменной окружения Path (Например, это путь C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\bin\).

Загрузка MinGW

Загрузить и установить компилятор GCC http://tdm-gcc.tdragon.net/ Загружена 64-битная версия tdm64-gcc-5.1.0-2.exe Установлена по умолчанию в C:\TDM-GCC-64. В процессе инсталляции подтвердить опцию изменить PATH (будет добавлен путь C:\TDM-GCC-64\bin). Добавить опцию поддержки OpenMP.

Альтернативой, в частности, работающей без ошибок при сборке OpenCV в статическом режиме, будет дистрибутив.

Загрузка Python

И да, нужно загрузить и установить CPython. Для проверки была выбрана сборка WinPython (http://winpython.sourceforge.net). После загрузки дистрибутива установка произведена в папку "C:_bin\\WinPython-64bit-3.4.3.5".

Сборка Boost

Для сборки Boost все готово. В каталоге "C:_lib\boost_1_59_0" нужно набрать команду >bootstrap mingw. Консольной командой "where gcc" убедитесь, что другого gcc кроме установленного в TDM-GCC у вас в путях нет, иначе bootstrap не сработает.

После того, как отработает bootstrap нужно вручную внести изменения в файл "C:_lib\boost_1_59_0\project-config.jam". Он должен выглядеть так:

import option ; using gcc ; option.set keep-going : false ; using python : 3.4 : C:_bin\WinPython-64bit-3.4.3.5\python-3.4.3.amd64 ;

Путь к Python должен быть добавлен в jam-файле выше.

Предыдущая команда (bootstrap mingw) должна создать в каталоге файл "b2.exe". Нужно запустить его со следующим параметром: >b2.exe --toolset=gcc или с такими, если собирать с поддержкой OpenMP >b2.exe --toolset=gcc --threading=multi --cxxflags=fopenmp --linkflags=fopenmp architecture=x86 address-model=64

Полный набор параметров можно получить здесь: «http://www.boost.org/doc/libs/1_59_0/doc/html/bbv2/reference.html»

Для компилятора MCVC достаточно выполнить две программы. Перед этим нужно только не забыть добавить путь к Python и указать исползование msvc вместо gcc в конфигурационном файле проекта project-config.jam. >bootstrap >b2.exe --toolset=msvc --threading=multi --cxxflags=openmp --linkflags=openmp architecture=x86 address-model=64

Пример конфигурационного файла для сборки проекта:

```
import option ;
using msvc ;
option.set keep-going : false ;
using python : 3.4 : C:\\_bin\\WinPython-64bit-3.4.3.5\\python-3.4.3.amd64 ;
```

Построение Boost займет заметное время. После успешной сборки будет выведено сообщение:

```
The Boost C++ Libraries were successfully built!
The following directory should be added to compiler include paths:
   C:\_lib\boost_1_59_0
The following directory should be added to linker library paths:
   C:\_lib\boost_1_59_0\stage\lib
```

Вручную необходимо добавить соответствующую переменную окружения в систему: BOOST_ROOT=C:_lib\boost_1_59_0

Сборка boost в Unix-подобных системах

Сборка описана на официальном сайте библиотеки и происходит без особых сложностей. Смотрите:
http://www.boost.org/doc/libs/1_59_0/more/getting_started/unix-variants.html.

Установка GIT

Загрузить с ресурса https://git-scm.com и установить последнюю версию GIT для вашей системы. После установки проверить, что он добавлен в PATH. C:\Program Files\Git\cmd, GIT понадобится один раз для того, чтобы выгрузить исходные тексты Shark, OpenCV, Vowpal Wabbit и других библиотек.

Установка CMake

Загрузить и установить Cmake. https://cmake.org/ Загружен файл: «**cmake-3.3.2-win32-x86.zip»**. Куда он будет установлен не так важно - будем пользоваться GUI-версией, запуская приложение cmake-gui.exe.

Загрузка Shark

Загрузить библиотеку машинного обучения Shark.Главная страница библиотеки размещена по адресу

«http://image.diku.dk/shark/sphinx_pages/build/html/index.html»

Чтобы загрузить Shark необходимо в каталоге "C:_lib" сделать клонирование репозитория, набрав команду `>git clone https://github.com/Shark-ML/Shark/`

либо воспользоваться пунктом "Git GUI Here" из контекстного меню explorer. В результате будет создана подпапка с проектом: C:_lib\Shark\ И снова, папку можно выбрать любую.

Генерация проекта Shark

Запустить Cmake GUI C:_bin\cmake-3.3.2-win32-x86\bin\cmake-gui.exe

Указать в путях к исходникам и в пути к бинарному результату один и тот-же путь: C:/_pjt/Shark

При первом запуске проверить отсутствие и впоследствии перед каждым конфигурированием удалять этот файл: **C:/_pjt/Shark/CMakeCache.txt**

В CMake-GUI поставить галочку "Advanced". Нажать "Configure". Выбрать генератор "CodeBlocks-MinGW makefiles", use default native compiler.

Установить отметку Boost_USE_STATIC_LIBS т.к. ранее boost был собран именно в статическом варианте. Снять отметку у опции BUILD_SHARED_LIBS, если библиотека Shark также нужна в статическом варианте.

Снять отметки напротив BUILD_TESTING и BUILD_EXAMPLES. Изменить CMAKE_INSTALL_PREFIX на "C:_lib" (это изменить путь инсталляции библиотеки).

Нажать "Generate". Убедиться в отсутствии ошибок. В итоге будет создан проект в CodeBlocks C:_pjt\Shark\shark.cbp

Сборка Shark

Открыть .cbp в Code::Blocks и начать сборку конфигурации "all". Все должно пройти без ошибок. После сборки всего проекта необходимо собрать конфигурацию "install".

При сборке Shark в Visual Studio все происходит аналогично сборке с использованием GCC. ЕСли необходимо собрать проект в для архитектуры x64, то в меню "Сборка->Диспетчер конфигураций..." в списке "Активная платформа решения" выбрать пункт "" и выбрать новую платформу "x64", скопировав параметры из "Win32". В процессе сборки для корректного построения тестов и примеров нужно будет вручную скопировать библиотеку из "..\x64\Release\shark.lib" в "..\lib\Release\shark.lib" или сразу настроить новый выходной каталог для новой конфигурации.

Сборка Shark в Unix-подобных системах

Этот вариант является основным для библиотеки и хорошо описан на официальном сайте

«http://image.diku.dk/shark/sphinx_pages/build/html/rest_sources/getting_started/installation.html»

Установка Eigen

На странице проекта (http://eigen.tuxfamily.org) найдется ссылка для загрузки библиотеки. Загрузить можно или архив или воспользоваться командой клонирования системы контроля версий Mercurial (https://www.mercurial-scm.org/).

Загрузите и установите в папке библиотек (Например, в впапку "C:_lib\eigen3"). Для этого потребуется аналогично тому, как это было сделано с проектом Shark, произвести конфигурацию с помощью CMake, изменив CMAKE_INSTALL_PREFIX на "C:_lib" и затем собрать конфигурацию "all" (В томже CodeBlocks). После сборки всего проекта необходимо собрать конфигурацию "install".

Установка Armadillo

Загружается библиотека архивом со страниц проекта (http://arma.sourceforge.net/) и устанавливается с помощью CMake. В составе архива есть предкомпилированные бинарные библиотеки, собранные в конфигурации MSVC x64.

BLAS_LIBRARY=C:/_lib/armadillo.src/examples/lib_win64/blas_win64_MT.lib
LAPACK_LIBRARY=C:/_lib/armadillo.src/examples/lib_win64/lapack_win64_MT.lib

Снять флажок с BUILD_SHARED_LIBS. Настроить переменную CMAKE_INSTALL_PREFIX на каталог, куда будет установлена библиотека. Сгенерировать проект. Собрать конфигурацию "all" и затем "install".

Установка OpenCV

Для для загрузки библиотеки необходимо клонировать репозиторий кода в папке "C:_lib\" командой git clone https://github.com/Itseez/opencv.git. Запустить CMake, указать путь к распакованным исходным файлам библиотеки. Нажать Configure. Изменить CMAKE_INSTALL_PREFIX на "C:_lib". Деактивировать опцию BUILD_SHARED_LIBS, если не намерены использовать OpenCV как динамическую библиотеку (В этом случае TDM-GCC нужно заменить на другой компилятор т.к. имеющаяся в текущей версии ошибка не позволит собрать статическую библиотеку). Деактивировать BUILD_TESTS, BUILD_PERF_TESTS, BUILD_EXAMPLES. Нажать Generate. На ошибки можно не обращать внимания. Как и в предыдущих примерах необходимо выпонить сборку конфигурации "all" и затем "install".

Другие библиотеки

Библиотеки линейной алгебры

Также интерес могут представлять другие библиотеки линейной алгебры, не вошедшие в документ:

- MTL4 - Matrix Template Library 4
- IT++
- Newmat C++ matrix library

Отдельно соит упомянуть библиотеку больших разреженных несимметричных систем линейных уравнений:

- SuperLU

Реализации BLAS/LAPACK

BLAS (Basic Linear Algebra Subprograms) реализует набор функций для выполнения основных векторных и матричных операций. Реализации BLAS эффективны и широкодоступны, поэтому BLAS используется в основе различных библиотек линейной алгебры, например, библиотеки LAPACK.

LAPACK предоставляет средства решения систем линейных уравнений, метод наименьших квадратов для систем линейных уравнений, вычисление сингулярных чисел, вычисление собственных чисел матрицы. Предоставляет методы разложения матриц (LU, Cholesky, QR, SVD, Schur, generalized Schur). Поддерживаются вещественные и комплексные числа с одинарной и двойной точностью.

Различные реализации BLAS и LAPACK:

- LAPACK на netlib (http://www.netlib.org/lapack/faq.html)
- BLAS на netlib (http://www.netlib.org/blas/faq.html)
- OpenBLAS (включает LAPACK)
- Intel Math Kernel Library (MKL)
- AMD Core Math Library (ACML)
- LAPACK 3.4 for Windows
- Бинарные файлы BLAS & LAPACK для Windows x32
- Бинарные файлы BLAS & LAPACK для Windows x64
- HP's Mathematical Software Library (MLIB)
- ATLAS (Automatically Tuned Linear Algebra Software)

Библиотеки обработки данных, машинного обучения и научных вычислений

На данный момент существует несколько библиотек, которые могут быть чрезвычайно интересны для решения задач анализа больших объемов данных и научных вычислений.

- **mlpack - scalable machine learning library**

В библиотеке реализованы следующие алгоритмы и модели машинного обучения:

  - Collaborative filtering (with many decomposition techniques)
  - Decision stumps (one-level decision trees)
  - Density estimation trees
  - Euclidean minimum spanning tree calculation
  - Gaussian mixture models
  - Hidden Markov models
  - Kernel Principal Components Analysis (optionally with sampling)

- k-Means clustering (with several accelerated algorithms)
- Least-angle regression (LARS/LASSO)
- Linear regression (simple least-squares)
- Local coordinate coding
- Locality-sensitive hashing for approximate nearest neighbor search
- Logistic regression
- Max-kernel search
- Naive Bayes classifier
- Nearest neighbor search with dual-tree algorithms
- Neighborhood components analysis
- Non-negative matrix factorization
- Perceptrons
- Principal components analysis (PCA)
- RADICAL (independent components analysis)
- Range search with dual-tree algorithms
- Rank-approximate nearest neighbor search
- Sparse coding with dictionary learning

• **GSL - GNU Scientific Library**

В состав этой большой библиотеки включены следующие разделы или модули:

- Комплексные числа - Complex Numbers
- Корни многочленов - Roots of Polynomials
- Специальные функции - Special Functions
- Векторы и матрицы - Vectors and Matrices
- Перестановки - Permutations
- Сортировка
- Поддержка BLAS
- Линейная алгебра - Linear Algebra
- Функции вычисления собственных чисел и векторов матриц - Eigensystems
- Быстрое преобразование Фурье - Fast Fourier Transforms
- Генерпция случайных чисел - Random Numbers
- Псевдо-случайные последовательности - Quasi-Random Sequences
- Распределения случайных чисел - Random Distributions
- Статистика - Statistics
- Гистограммы – Histograms
- Кортежи - N-Tuples
- Интегрирование методом Монте-Карло - Monte Carlo Integration
- Стохастические методы оптимизации - Simulated Annealing
- Дифференциальные уравнения - Differential Equations
- Интерполяция - Interpolation

- Численное дифференцирование - Numerical Differentiation
- Чебышевская аппроксимация функций - Chebyshev Approximation
- Алгоритмы ускорения сходимости рядов - Series Acceleration
- Дискретное преобразование Ханкеля - Discrete Hankel Transforms
- Поиск корней одномерных и многомерных функций - Root-Finding
- Минимизация - Minimization
- Аппроксимация методом наименьших квадратов (МНК) - Least-Squares Fitting
- Физические константы - Physical Constants
- Операции и форматы вещественных чисел - IEEE Floating-Point
- Дискретное вейвлет-преобразование - Discrete Wavelet Transforms
- Интерполяция B-сплацнами - Basis splines
- Накопительная статистика - Running Statistics
- Разреженные матрицы и линейная алгебра - Sparse Matrices and Linear Algebra

- **Ceres Solver**

Библиотека предназначена для моделирования и решения сложных оптимизационных проблем.

- Нелинейная оптимизация МНК с граничными условиями (non-linear Least Squares problems with bounds constraints)
- Задачи нелинейного программирования и общей оптимизации без граничных условий (general unconstrained optimization problems)

- **ROOT - объектно ориентированный каркас анализа данных**

Некоторые библиотеки из полного списка, входящие в состав ROOT

- libAsImage - обработка изображений
- libCling - интерпретатор C++ (Cling)
- libCore – ядро
- libGeom, libGpad, libGraf, libGraf3d - библиотеки 2D и 3D графики
- libGui пользовательски й графический интерфейс
- libMatrix - матричная и векторная алгебра
- libMathCore - основная математическая библиотека
- libMathMore - дополнительная математическая библиотека, использующая GSL
- libNet - сетевое взаимодействие
- libNew - управление динамической памятью
- libPhysics - физические абстракции

- libRGL - интерфейс к OpenGL.
- libRIO - библиотека ввода/вывода
- libSpectrum - спектральный анализ
- libThread - поддержка многопоточности
- libTMVA - тулкит мультивариантного анализа

- **TensorFlow - Library for Machine Intelligence**

Вычислительная библиотека, использующая для программирования вычислений графы. Узлы графов представляют собой вычислитеьные операции, ребра графов обозначают многомерные массивы данных (тензоры), передаваемые между узлами

- **Distributed Machine Learning Toolkit**

Поддерживается параллелизация данных и выстраивание конвейеров обработки

- **Vowpal Wabbit**

Реализация быстрого алгоритма машинного обучения на C++.

Расчеты на C++

Язык C++ позволяет писать производительные программы, осуществляющие математические расчеты и реализующие вычислительные алгоритмы. Для математических расчетов созданы многочисленные библиотеки для расчетов в линейной алгебре, решения дифференциальных уравнений и т.д. Для примера рассмотрим использование библиотеки Eigen для решения вычислительных задач линейной алгебры.

Расчеты с использованием библиотеки Eigen

Класс Matrix

Для начала использования библиотеки не требуется ничего, кроме ее заголовочных файлов. Для старта необходимо создать консольный проект и в настройках проекта указать путь к инсталляционному каталогу библиотеки "C:_lib3" для всех конфигураций (Release и Debug). В меню CodeBlocks это Project options->вкладка Search directories->вкладка Compiler, кнопка Add.

Для использования возможностей Eigen в исходный файл необходимо включить заголовок библиотеки. Пример с сайта библиотеки:

```
#include <iostream>
#include <Eigen/Dense>

using Eigen::MatrixXd;
using namespace std;
```

```
int main()
{
    MatrixXd m(2,2);
    m(0,0) = 3;
    m(1,0) = 2.5;
    m(0,1) = -1;
    m(1,1) = m(1,0) + m(0,1);
    cout << m << endl;
    return 0;
}
```

Классы библиотеки подключены в заголовке

```
#include <Eigen/Dense>
```

В примере использован класс MatrixXd. В Eigen предопределены следующие матричные и векторные классы:

- MatrixNt для шаблона Matrix. Например, typedef MatrixXi разворачивается в Matrix.
- VectorNt для шаблона Matrix. Например, typedef Vector2f разворачивается в Matrix.
- RowVectorNt для шаблона Matrix. Например, typedef RowVector3d разворачивается в Matrix.

где `N` и `t` могут принимать следующие значения: `N` - может быть равен 2, 3, 4, или X (означает динамический тип, Dynamic). `t` - может быть равен i (соответствует int), f (соответствует float), d (соответствует double), cf (соответствует complex), или cd (соответствует complex).

Элементы матрицы можно определить несколькими способами. Один, с помощью круглых скобок, использовался в примере выше. Кроме того, есть способ инициализировать с помощью оператора `<<`

```
Matrix3f m;
m << 1, 2, 3,
     4, 5, 6,
     7, 8, 9;
cout << m;
```

Таким способом можно также объединять несколько матриц в одну:

```
MatrixXf matA(2, 2);
matA << 1, 2, 3, 4;
MatrixXf matB(4, 2);
matB << matA, matA;
cout << matB << endl;
```

Существует несколько видов предзаданных матриц (Zero, Identity, Random, Constant, LinSpaced), используя которые можно инициализировать собственные данные.

```
#include <Eigen/Dense>
#include <iostream>
using namespace Eigen;
using namespace std;
int main()
{
    const int size = 6;
```

```
MatrixXd mat1(size, size);
mat1.topLeftCorner(size/2, size/2)    = MatrixXd::Zero(size/2, size/2);
mat1.topRightCorner(size/2, size/2)   = MatrixXd::Random(size/2,size/2);
mat1.bottomLeftCorner(size/2, size/2) = MatrixXd::Identity(size/2,size/2);
mat1.bottomRightCorner(size/2, size/2)=
                                    MatrixXd::Constant(size/2,size/2,3.14);
cout << mat1 << endl << endl;
MatrixXd mat2(size, size);
mat2.topLeftCorner(size/2, size/2).setZero();
mat2.topRightCorner(size/2, size/2).setIdentity();
mat2.bottomLeftCorner(size/2, size/2).setIdentity();
mat2.bottomRightCorner(size/2, size/2).setZero();
cout << mat2 << endl << endl;
MatrixXd mat3(size, size);
mat3 << MatrixXd::Zero(size/2, size/2), MatrixXd::Identity(size/2,size/2),
MatrixXd::Identity(size/2, size/2), MatrixXd::Zero(size/2, size/2);
cout << mat3 << endl;
}
```

Операции с матрицами

Eigen реализует матричную и векторную арифметику через перегруженные арифметические операторы, такие как +, -, *, и с помощью отдельных методов, таких как dot(), cross(), и других.

Сложение и вычитание

Для корректного сложения и вычитания объектов матричного типа необходимо, чтобы матрицы были одинакового размера, а данные были одного и того же типа. Поддерживаемые операторы:

- бинарное сложение a+b
- бинарное вычитание a-b
- унарный минус -a
- составной оператор сложения и присваивания a+=b
- составной оператор вычитания и присваивания a-=b

Пример использования операций сложения/вычитания.

```
#include <iostream>
#include <Eigen/Dense>

using namespace Eigen;
using namespace std;

int main()
{
    Matrix2d a;
    a << 1, 2,
         3, 4;
    MatrixXd b(2,2);
    b << 2, 3,
         1, 4;
    cout << "a + b =\n" << a + b << endl;
```

```
    cout << "a - b =\n" << a - b << endl;
    cout << "Doing a += b;" << endl;
    a += b;
    cout << "Now a =\n" << a << endl;
    Vector3d v(1,2,3);
    Vector3d w(1,0,0);
    cout << "-v + w - v =\n" << -v + w - v << endl;
}
```

Скалярное умножение и деление

Поддерживаемые операторы:

- бинарный оператор умножения matrixscalar или scalarmatrix
- бинарный оператор деления matrix/scalar
- составной оператор умножения и присваивания matrix*=scalar
- составной оператор деления и присваивания matrix/=scalar

Пример использования операций скалярного умножения/деления.

```
#include <iostream>
#include <Eigen/Dense>

using namespace Eigen;
using namespace std;

int main()
{
  Matrix2d a;
  a << 1, 2,
       3, 4;
  Vector3d v(1,2,3);
  cout << "a * 2.5 =\n" << a * 2.5 << endl;
  cout << "0.1 * v =\n" << 0.1 * v << endl;
  cout << "Doing v *= 2;" << endl;
  v *= 2;
  cout << "Now v =\n" << v << endl;
}
```

Транспонирование и сопряжение

Операции транспонирования, сопряжения и сопряжения-транспонирования осуществляются с помощью методов матричных классов.

Поддерживаемые операторы: * транспонирование transpose() * сопряжение conjugate() * сопряжение-транспонирование adjoint()

Использование этих методов так же просто как и использование арифметических операторов

```
#include <iostream>
#include <Eigen/Dense>

using namespace Eigen;
using namespace std;
```

```
int main()
{
    MatrixXcf a = MatrixXcf::Random(2,2);
    cout << "Here is the matrix a\n" << a << endl;
    cout << "Here is the matrix a^T\n" << a.transpose() << endl;
    cout << "Here is the conjugate of a\n" << a.conjugate() << endl;
    cout << "Here is the matrix a^*\n" << a.adjoint() << endl;
}
```

Замечание: если написать оператор так `a = a.transpose()` то Eigen начнет записывать элементы матрицы `a` не дожидаясь завершения вычисления работы метода транспонирования. Это приведет к неверному результату. Для корректного транспонирования необходимо использовать метод `a.transposeInPlace()`. В аналогичной ситуации следует использовать `a.adjointInPlace()` для комплексных матриц.

Матричное умножение

Поддерживаемые операторы:

- бинарный оператор умножения matrix*matrix
- составной оператор умножения и присваивания matrix*=matrix

Пример использования операций матричного умножения

```
#include <iostream>
#include <Eigen/Dense>

using namespace Eigen;
using namespace std;

int main()
{
  Matrix2d mat;
  mat << 1, 2,
         3, 4;
  Vector2d u(-1,1), v(2,0);
  cout << "mat*mat:\n" << mat*mat << endl;
  cout << "mat*u:\n" << mat*u << endl;
  cout << "u^T*mat:\n" << u.transpose()*mat << endl;
  cout << "u^T*v:\n" << u.transpose()*v << endl;
  cout << "u*v^T:\n" << u*v.transpose() << endl;
  cout << "Умножение на саму себя..." << endl;
  mat = mat*mat;
  cout << "mat:\n" << mat << endl;
}
```

Замечание: конструкция `m=m*m`, которая не работала при транспонировании, здесь будет делать именно то, что от нее ожидают.

Скалярное и векторное произведение

Поддерживаемые операторы:

- скалярное произведение dot()
- векторное произведение cross()

Пример использования скалярного и векторного произведения

```
#include <iostream>
#include <Eigen/Dense>
using namespace Eigen;
using namespace std;
int main()
{
  Vector3d v(1,2,3);
  Vector3d w(0,1,2);
  cout << "Скалярное произведение: " << v.dot(w) << endl;
  double dp = v.adjoint()*w; // автопреобразование в скаляр
  cout << "Скалярное произведение перемножением матриц: " << dp << endl;
  cout << "Векторное произведение:\n" << v.cross(w) << endl;
}
```

Класс Array

Класс Array реализует массивы однотипных данных в отличие от класса Matrix, предназначенного специально для линейной алгебры. Класс Array предоставляет методы поэлементной манипуляции со своими значениями, не имеющими смысла в рамках линейной алгебры, например добавление константы к каждому элементу массива или поэлементное умножение двух массивов. Аналогично классу Matrix для Array есть предопределенные классы:

```
using ArrayXf   = Array<float,Dynamic,1>;
using Array3f   = Array<float,3,1>;
using ArrayXXd  = Array<double,Dynamic,Dynamic>;
using Array33d  = Array<double,3,3>;
```

Доступ к элементам массива осуществляется, аналогично тому, как это реализовано в классе Matrix, с помощью круглых скобок [привет, `operator()`]. Оператор `<<` используется для инициализации массивов или для его печати.

```
#include <Eigen/Dense>
#include <iostream>
using namespace Eigen;
using namespace std;
int main()
{
  ArrayXXf  m(2,2);

  // присвоение начальных значений поэлементно
  m(0,0) = 1.0; m(0,1) = 2.0;
  m(1,0) = 3.0; m(1,1) = m(0,1) + m(1,0);
```

```
  // печать значений на консоль
  cout << m << endl << endl;

  // использование формы инициализации через запятую (comma-initializer)
  m << 1.0,2.0,
       3.0,4.0;

  // печать новых значений на консоль
  cout << m << endl;
}
```

Поэлементные операции с Array

Поэлементное сложение двух массивов, сложение со скаляром, поэлементное произведение, умножение на константу иллюстрирует пример ниже:

```
#include <Eigen/Dense>
#include <iostream>

using namespace Eigen;
using namespace std;

int main()
{
  ArrayXXf a(3,3);
  ArrayXXf b(3,3);
  a << 1,2,3,
       4,5,6,
       7,8,9;
  b << 1,2,3,
       1,2,3,
       1,2,3;

    // Поэлементное сложение двух массивов
    cout << "a + b = " << endl << a + b << endl << endl;
    // Вычитание скалярного значения из элементов массива
    cout << "a - 2 = " << endl << a - 2 << endl;
    // Поэлементное умножение двух массивов
    cout << "a * b = " << endl << a * b << endl;
    // Умножение элементов массива на скалярное значение
    cout << "a * 3 = " << endl << a * 3 << endl;}
```

Преобразования между Array и Matrix

В случае, когда необходимо применять операции линейной алгебры, следует в программе использовать тип Matrix. Если требуются поэлементные операции, то для таких вычислений лучше подходит тип Array. Иногда приходится применять и алгебраические и поэлементные вычисления. В этом случае будет необходимо преобразовывать данные из одного типа в другой. На этот случай в классах реализованы методы

преобразования данных в обе стороны. Примеры преобразований в исходном коде ниже:

```
#include <Eigen/Dense>
#include <iostream>
using namespace Eigen;
using namespace std;
int main()
{
   MatrixXf m(2,2);
   MatrixXf n(2,2);
   MatrixXf result(2,2);
   m << 1,2,
   3,4;
   n << 5,6,
   7,8;
   result = m * n;
   cout << "Матрица m*n:" << endl << result << endl << endl;
   result = m.array() * n.array();
   cout << "Массив m*n:" << endl << result << endl << endl;
   result = m.cwiseProduct(n);
   cout << "Поэлементное произведение:" << endl << result << endl << endl;
   result = m.array() + 4;
   cout << "Сумма с константой m + 4:" << endl << result << endl << endl;

   result = (m.array() + 4).matrix() * m;
   cout << "Комбинация 1:" << endl << result << endl << endl;
   result = (m.array() * n.array()).matrix() * m;
   cout << "Комбинация 2:" << endl << result << endl << endl;
}
```

Реализация метода главных компонент с использованием Eigen

Метод главных компонент (Principal Component Analysis, PCA), также называемый преобразованием Кархунена — Лоэва (Karhunen–Loève transform, KLT) или преобразованием Хотеллинга (Hotelling transform), являетсчя способом понижения размерности в избыточных данных.

Его суть заключается в поиске подпространства в многомерном пространстве данных, в котором расположены все точки набора данных. Например, если точки плоскости (двумерные данные) расположены на одной прямой, то размерность пространства можно понизить до одного измерения, описав расположение точек их единственной координатой на этой прямой. Единичный вектор, задающий эту прямую, называется главной компонентой.

Метод главных компонент является показательным для оценки возможностей библиотек линейной алгебры, так как требует как элементарных векторных и матричных операций при подготовке исходных и выходных данных, так и решения задачи вычисления собственных чисел и собственных векторов.

Решение задачи вычисления главных компонент для данных, представляющих собой ИК-спектры двенадцати смесей двух жидкостей в различных пропорциях, с использованием Eigen приведено ниже

```
#include <Eigen/Dense>
#include <iostream>
#include <fstream>
using namespace Eigen;
using namespace std;
int main()
{
    // Матрица 12 спектров, 1866 точек.
    MatrixXd mat = MatrixXd::Random(12,1866);

    // Загрузка исходных данных из файла
    ifstream csv("data.csv");
    if (csv.is_open())
    {
        string line;
        int row=0;
        while ( getline(csv, line, '\n') )
        {
            string num;
            istringstream iss(line);
            int col=0;
            while ( getline( iss, num, ',') )
            {
                //cout << row << " " << col << "=" << stod(num) << endl;
                mat(row, col) = stod(num);
                col++;
            }
            row++;
        }
        csv.close();
    }
    else
    {
            cout << "Unable to open file";
    }

    // Вычисление ковариационной матрицы
    cout << "Covariance matrix" << endl;
    mat.transposeInPlace();
    MatrixXd centered = mat.colwise() - mat.rowwise().mean();
    MatrixXd cov = centered * centered.adjoint();
    cout << cov.rows() << "x" << cov.cols() << endl;

    // Расчет главных компонент
    SelfAdjointEigenSolver<MatrixXd> eig(cov, false);

    // Вывод результатов
    cout << "Eigenvector" << endl;
    auto x = eig.eigenvalues().rightCols(1).array().tail(8);
    cout << x/x.sum() << endl;
}
```

Расчеты с использованием boost.uBLAS и boost.odeint

Библиотека boost.uBLAS

Эффективные алгоритмы линейной алгебры реализованы в проекте uBLAS, предоставляющем алгоритмы BLAS – Basic Linear Algebra Subprograms.

Поддерживаются стандартные операции сложения, вычитания, умножения и деления на скаляр.

```
C = A + B; C = A - B; C = -A;
w = u + v; w = u - v; w = -u;
C = t * A; C = A * t; C = A / t;
w = t * u; w = u * t; w = u / t;
```

Операторы вычисления и присваивания

```
C += A; C -= A;
w += u; w -= u;
C *= t; C /= t;
w *= t; w /= t;
```

Скалярные, векторные и поэлементные произведения:

```
t = inner_prod(u, v);
C = outer_prod(u, v);
w = prod(A, u); w = prod(u, A); w = prec_prod(A, u); w = prec_prod(u, A);
C = prod(A, B); C = prec_prod(A, B);
w = element_prod(u, v); w = element_div(u, v);
C = element_prod(A, B); C = element_div(A, B);
```

Преобразования:

```
w = conj(u); w = real(u); w = imag(u);
C = trans(A); C = conj(A); C = herm(A); C = real(A); C = imag(A);
```

Пример программы, использующей некоторые перечисленные операции:

```
#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/io.hpp>

using namespace std;


int main () {
    using namespace boost::numeric::ublas;
    matrix<complex<double> > m (3, 3);
    matrix<double> m1 (3, 3), m2 (3, 3);
    for (unsigned i = 0; i < m.size1 (); ++ i)
        for (unsigned j = 0; j < m.size2 (); ++ j)
        {
            m (i, j) = complex<double> (3 * i + j, 3 * i + j);
            m1 (i, j) = m2 (i, j) = 3 * i + j;
        }

    cout << - m      << endl;
```

```
    cout << conj (m) << endl;
    cout << real (m) << endl;
    cout << imag (m) << endl;
    cout << trans(m) << endl;
    cout << herm (m) << endl;

    cout << m1 + m2  << endl;
    cout << m1 - m2  << endl;
}
```

В библиотеке реализованы более сложные операции, включающие вычисление нормы вектора или матрицы

```
t = norm_inf(v); i = index_norm_inf(v);
t = norm_1(v);   t = norm_2(v);
t = norm_inf(A); i = index_norm_inf(A);
t = norm_1(A);   t = norm_frobenius(A);
```

а также эффективные аналоги алгоритмов вычисления произведений

```
axpy_prod(A, u, w, true);  // w = A * u
axpy_prod(A, u, w, false); // w += A * u
axpy_prod(u, A, w, true);  // w = trans(A) * u
axpy_prod(u, A, w, false); // w += trans(A) * u
axpy_prod(A, B, C, true);  // C = A * B
axpy_prod(A, B, C, false); // C += A * B

// блочное вычисление произведений, обеспечивающее выигрыш
// в архитектурах с SIMD
w = block_prod<matrix_type, 8> (A, u); // w = A * u
w = block_prod<matrix_type, 8> (u, A); // w = trans(A) * u
C = block_prod<matrix_type, 8> (A, B); // C = A * B

// вычисление дает выигрыш для матриц с числом столбцов
// меньшим числа строк
opb_prod(A, B, C, true);  // C = A * B
opb_prod(A, B, C, false); // C += A * B
```

Также реализованы операции доступа к частям матриц и векторов: к отдельным строкам, столбцам, выборкам строк или столбцов, диапазонам.

Библиотека boost.odeint

Библиотека boost.odeint позволяет решать обыкновенные дифференциальные уравнения с использованием разнообразных методов численного интегрирования. Пример решения уравнения `x' = 3/(2t^2) + x/(2t)` дан ниже. Библиотека интегрирована с другми числовыми (numeric) библиотеками boost, и в частности, с `boost.uBLAS`, которая используется для представления вектора состояния.

```
/* Boost libs/numeric/odeint/examples/simple1d.cpp
 Copyright 2012-2013 Mario Mulansky
 Copyright 2012 Karsten Ahnert
 example for a simple one-dimensional 1st order ODE
 Distributed under the Boost Software License, Version 1.0.
```

```
(See accompanying file LICENSE_1_0.txt or
 copy at http://www.boost.org/LICENSE_1_0.txt)
 */


#include <iostream>
#include <boost/numeric/odeint.hpp>

using namespace std;
using namespace boost::numeric::odeint;


/* решение ОДУ x' = 3/(2t^2) + x/(2t)
 * с начальными услуовиями x(1) = 0.
 * аналитическое решение: x(t) = sqrt(t) - 1/t
 */

void rhs( const double x , double &dxdt , const double t )
{
    dxdt = 3.0/(2.0*t*t) + x/(2.0*t);
}

void write_cout( const double &x , const double t )
{
    cout << t << '\t' << x << endl;
}

// использовать метод Dormand-Prince 5, state_type = double
typedef runge_kutta_dopri5< double > stepper_type;

int main()
{
    double x = 0.0; // начальные условия x(1) = 0
    // допустимая ошибкоа 10^-12, интегрировать t=1...10
    integrate_adaptive( make_controlled( 1E-12 , 1E-12 , stepper_type() )
, rhs , x , 1.0 , 10.0 , 0.1 , write_cout );
}
```

Расчеты с использованием библиотеки Armadillo

Настройка проекта

Armadillo это библиотека линейной алгебры, написанная на C++, сочетающая вычислительную скорость и читаемость кода, максимально приближенного к синтаксису MATLAB. Простой пример:

```
#include <iostream>
#include <armadillo>

using namespace std;
using namespace arma;

int main(int argc, char** argv)
```

```
{
mat A = randu<mat>(4,5);
mat B = randu<mat>(4,5);

cout << A*B.t() << endl;

return 0;
}
```

Для того чтобы собрать и запустить этот пример необходимо связать его с файлом библиотеки и указать путь к включаемым файлам. Если библиотека установлена в директории `C:\_lib\armadillo\`, в проекте в разделе `Search directries` указать для компилятора `C:\_lib\armadillo\include`, а для компоновщика `C:\_lib\armadillo\lib`.

В разделе `Linker settings`, `Linker libraries` перечислить следующие библиотеки:

- C:_lib\armadillo\lib\libarmadillo.a (или указать опцию -armadillo)
- C:_lib\armadillo.src\examples\lib_win64\blas_win64_MT.lib
- C:_lib\armadillo.src\examples\lib_win64\lapack_win64_MT.lib

Для запуска собранного приложения необходимо, чтобы динамические библиотеки BLAS и LAPACK находились в путях приложения. Например, их можно скопировать в текущую папку проекта:

- C:_lib\armadillo.src\examples\lib_win64\blas_win64_MT.dll
- C:_lib\armadillo.src\examples\lib_win64\lapack_win64_MT.dll

Для компоновки приложения со статическими библиотеками необходимо их собрать в статическом варианте, воспользовавшись рекомендациями на странице `http://icl.cs.utk.edu/lapack-for-windows/lapack/`.

Для использования HDF5 необходимо в пути компилятора добавить путь к заголовочным файлам HDF5: `C:\_lib\HDF5\include`, указать для линкера путь к библиотеке HDF5 `C:\_lib\HDF5\lib\libhdf5.a` (или добавить в `Search directries` для линкера путь `C:\_lib\HDF5\lib\` и указать опцию компоновщика `-lhdg5`).

В файле `C:\_lib\armadillo\include\armadillo_bits\config.hpp` раскоментировать определение макро `ARMA_USE_HDF5`.

Матричные и векторные типы данных

Основным матричным типом данных является класс `Mat<type>`. Для удобства предопределены следующие синонимы типов:

- mat = Mat<double>
- fmat = Mat<float>
- cx_mat = Mat<cx_double>

- cx_fmat= Mat<cx_float>
- umat = Mat<uword>
- imat = Mat<sword>

где `cx_float`, `cx_double` - комплексные типы данных; `uword` и `sword` - беззнаковый и знаковый целые типы данных, размер которых составляет 32 бита при использовании старых стандартов C++ (до C++11), и 64 бита для C++11 (`-std=c++11`) и более новых версий стандарта.

Примеры объяаления объектов матричного типа и их использования:

```
mat A(5, 5, fill::randu);
double x = A(1,2);

mat B = A + A;
mat C = A * B;
mat D = A % B;

cx_mat X(A,B);

B.zeros();
B.set_size(10,10);
B.ones(5,6);

B.print("B:");

mat::fixed<5,6> F;
```

Аналогичным образом могут быть объявлены векторы-столбцы и векторы-строки, реализованные как классы `Col<type>` и `Row<type>`, наследующие от `Mat<type>`, и представляющие собой разновидность матрицы с одним столбцом/строкой. Имеются соответствующие синонимы типов:

Для столбцов

- `vec = colvec = Col<double>`
- `fvec = fcolvec = Col<float>`
- `cx_vec = cx_colvec = Col<cx_double>`
- `cx_fvec = cx_fcolvec = Col<cx_float>`
- `uvec = ucolvec = Col<uword>`
- `ivec = icolvec = Col<sword>`

Для строк:

- `rowvec = Row<double>`
- `frowvec = Row<float>`
- `cx_rowvec = Row<cx_double>`
- `cx_frowvec = Row<cx_float>`
- `urowvec = Row<uword>`
- `irowvec = Row<sword>`

Пример использования:

```
// столбцы
vec x(10);
vec y = zeros<vec>(10);
mat A = randu<mat>(10,10);
vec z = A.col(5); // извлечь столбец
// строки
rowvec x(10);
rowvec y = zeros<rowvec>(10);
mat    A = randu<mat>(10,10);
rowvec z = A.row(5); // извлечь строку
```

Есть класс трехмерной, 3D-матрицы (тензора 3-го порядка), названный в библиотеке `Cube<type>` с соответствующими типами.

- `cube` = `Cube<double>`
- `fcube` = `Cube<float>`
- `cx_cube` = `Cube<cx_double>`
- `cx_fcube` = `Cube<cx_float>`
- `ucube` = `Cube<uword>`
- `icube` = `Cube<sword>`

Класс `field<object_type>` позволяет в матричной форме хранить объекты любых, не обязательно числовых, типов. Класс `SpMat<type>` реализует разреженные матрицы.

Еще несколько примеров объявления матриц и операций с ними

```
mat A = randu<mat>(5,10); // равномерное распределение чисел генератора
mat B = randu<mat>(5,10); // для нормального распределения - "randn"
mat C = randu<mat>(10,5);

mat P = A + B;
mat Q = A - B;
mat R = -B;
mat S = A / 123.0;
mat T = A % B;
mat U = A * C;

// конструирование V без промежуточных матриц
mat V = A + B + A + B;

imat AA = "1 2 3; 4 5 6; 7 8 9;";
imat BB = "3 2 1; 6 5 4; 9 8 7;";

// сравнение элементов
umat ZZ = (AA >= BB);

// C++11
vec v = { 1, 2, 3 };
mat A = { {1, 3, 5},
          {2, 4, 6} };
```

```
// C++98
mat B;
B << 1 << 3 << 5 << endr
  << 2 << 4 << 6 << endr;
```

Для доступа к элементам матриц и массивов реализован ряд функций для последовательной выборки соседних элементов и для произвольной выборки

- Доступ к элементам матрицы, вектора, или куба X:

X(n) - n-й элемент для столбца или строки. Для матрицыили куба это доступ к n-му элементу при последовательном их хранении в памяти "по столбцам". X.at(n) или X[n] - то же самое, без проверки выхода за границы массива

X(i,j) для матрицы доступ к элементу в i-й строке и j-м столбце. X.at(i,j) - то же самое, без проверки выхода за границы массива

X(i,j,k) - доступ к элементу куба в i-й строке и j-м столбце и k-м срезе. X.at(i,j,k) - то же самое, без проверки выхода за границы массива.

```
mat A = randu<mat>(10,10);
A(9,9) = 123.0;
double x = A.at(9,9);
double y = A[99];

vec p = randu<vec>(10,1);
p(9) = 123.0;
double z = p[9];
```

- Последовательная выборка для матрицы X:

X.col(col_number) X.row(row_number)

X.cols(first_col, last_col) X.rows(first_row, last_row)

X.submat(first_row, first_col, last_row, last_col)

X(span(first_row, last_row), span(first_col, last_col))

X(first_row, first_col, size(n_rows, n_cols)) X(first_row, first_col, size(Y)) (Y is a mat)

X(span(first_row, last_row), col_number) X(row_number, span(first_col, last_col))

X.head_cols(number_of_cols) X.head_rows(number_of_rows)

X.tail_cols(number_of_cols) X.tail_rows(number_of_rows)

X.unsafe_col(col_number)

- Последовательная выборака для вектора V:

V(span(first_index, last_index)) V.subvec(first_index, last_index)

V.head(number_of_elements) V.tail(number_of_elements)

- Произвольная выборка для вектора или матрицы X:

X.elem(vector_of_indices) X(vector_of_indices)

X.cols(vector_of_column_indices) X.rows(vector_of_row_indices)

X.submat(vector_of_row_indices, vector_of_column_indices) X(vector_of_row_indices, vector_of_column_indices)

- Дополнительные выборки и итераторы:
  X.diag() X.each_row() X.each_col()

```
mat A = zeros<mat>(5,10);

A.submat( 0,1, 2,3 )      = randu<mat>(3,3);
A( span(0,2), span(1,3) ) = randu<mat>(3,3);
A( 0,1, size(3,3) )       = randu<mat>(3,3);

mat B = A.submat( 0,1, 2,3 );
mat C = A( span(0,2), span(1,3) );
mat D = A( 0,1, size(3,3) );

A.col(1)        = randu<mat>(5,1);
A(span::all, 1) = randu<mat>(5,1);

mat X = randu<mat>(5,5);

// все элементы больше 0.5
vec q = X.elem( find(X > 0.5) );

// прибавить 123 ко всем элементам большим 0.5
X.elem( find(X > 0.5) ) += 123.0;

// присваивание 1 элемнтам по списку индексов
uvec indices;
indices << 2 << 3 << 6 << 8;
X.elem(indices) = ones<vec>(4);

// прибавить 123 к последним 5 элементам вектора
vec a(10, fill::randu);
a.tail(5) += 123.0;

// прибавить 123 к первым 3 элементам 2-го столбца X
X.col(2).head(3) += 123;
```

Аналоги методов Armadillo в MATLAB

MATLAB	Armadillo	Notes
`A(1, 1)`	`A(0, 0)`	элемент матрицы A
`A(k, k)`	`A(k-1, k-1)`	индексы отсчитываются от 0
`size(A,1)`	`A.n_rows`	число строк
`size(A,2)`	`A.n_cols`	число столбцов
`size(Q,3)`	`Q.n_slices`	Число слоев, где Q - куб (3D-массив)
`numel(A)`	`A.n_elem`	число элементов в матрице
`A(:, k)`	`A.col(k)`	k-й столбец (индексы отсчитываются от 0)

`A(k, :)`	`A.row(k)`	k-я строка
`A(:, p:q)`	`A.cols(p, q)`	столбцы от p до q
`A(p:q, :)`	`A.rows(p, q)`	строки от p до q
`A(p:q, r:s)`	`A( span(p,q), span(r,s) )`	диапазон элементов со строками от p до q и столбцами от r до s
`Q(:, :, k)`	`Q.slice(k)`	заданный слой для куба Q
`Q(:, :, t:u)`	`Q.slices(t, u)`	диапазон слоев
`Q(p:q, r:s, t:u)`	`Q( span(p,q), span(r,s), span(t,u) )`	диапазон элементов, заданный диапазонами строк, столбцов и слоев
`A'`	`A.t() or trans(A)`	транспонирование / Эрмитово сопряжение для комплексных матриц
`A = zeros(size(A))`	`A.zeros()`	матрица нулей
`A = ones(size(A))`	`A.ones()`	матрица единиц
`A = zeros(k)`	`A = zeros(k,k)`	аналогично для размерности k*k
`A = ones(k)`	`A = ones(k,k)`	
`C = complex(A,B)`	`cx_mat C = cx_mat(A,B)`	комплексная матрица, составленная из действительной и мнимой
`A .* B`	`A % B`	поэлементное умножение
`A ./ B`	`A / B`	поэлементное деление
`A  B`	`solve(A,B)`	соответствует inv(A)*B
`A = A + 1;`	`A++`	поэлементный инкремент
`A = A - 1;`	`A--`	поэлементный декремент
`A = [ 1 2; 3 4; ]`	`A << 1 << 2 << endr << 3 << 4 << endr;`	инициализация элементов матрицы специальный элемент endr означает конец строки
`X = A(:)`	`X = vectorise(A)`	разворачивание элементов в вектор
`X = [ A B ]`	`X = join_horiz(A,B)`	горизонтальное объединение двух матриц
`X = [ A; B ]`	`X = join_vert(A,B)`	вертикальное объединение двух матриц
`A`	`cout << A << endl;` или `A.print("A =");`	печать содержимого в консоль
`save -ascii 'A.dat' A`	`A.save("A.dat", raw_ascii);`	сохраненные в формате ascii матрицы могут
`load -ascii 'A.dat'`	`A.load("A.dat", raw_ascii);`	быть прочитанны в MATLAB и наоборот)

`A = randn(2,3);`	`mat A = randn(2,3);`	случайные числа для элементов матрицы
`B = randn(4,5);`	`mat B = randn(4,5);`	

`F = { A; B }`	`field F(2,1);`
	`F(0,0) = A;`
	`F(1,0) = B;`
		поля объединяют произвольные объекты

Чтение и запись данных

За чтение и запись данных отвечают функции `load` и `save` соответственно. Если не указан тип хранилища, функция записи данных сохранит их в бинарном формате `arma_binary`. Функция чтения автоматически попытается распознать тип данных. Функции возвращают значение `true` в случае успеха операции и `false` в случае ошибки.

```
mat A;
// параметр "name" - имя файла
// параметр "file_type" - тип файла

A.save( name );
A.save( name, file_type );
A.load( stream );
A.load( stream, file_type );
```

Параметр `file_type`, тип файла, может принимать одно из следующих значений:

- auto_detect - автоматическое определение типа при загрузке
- raw_ascii - сохранение/загрузка в текстовом формате без метаданных и заголовков
- raw_binary - сохранение/загрузка в двоичном формате без метеданных
- arma_ascii - сохранение/загрузка в текстовом формате с заголовком, описывающим тип и размерность данных
- arma_binary - сохранение/загрузка в двоичном формате с заголовком, описывающим тип и размерность данных
- csv_ascii - сохранение/загрузка в текстовом формате CSV (значения, разделенные запятой)
- hdf5_binary - сохранение/загрузка в бинарном формате HDF5
- pgm_binary - сохранение/загрузка в 8-битовом формате PGM (Portable Gray Map)
- ppm_binary - сохранение/загрузка в 8-битовом формате PPM (Portable Pixel Map)

Пример использования функций чтения/записи:

```
#include <iostream>
#include <armadillo>

using namespace std;
using namespace arma;

int main(int argc, char** argv)
{
    mat A = randu<mat>(5,5);

    // сохранение в формате arma_binary по умолчанию
    A.save("A1.mat");

    // сохранение в формате arma_ascii
    A.save("A2.mat", arma_ascii);

    mat B;
    // автоопределение типа
    B.load("A1.mat");

    mat C;
    // чтение в формате arma_ascii
    C.load("A2.mat", arma_ascii);

    // операция чтения с проверкой результата
    mat D;
    bool status = D.load("A2.mat");

    if(status == true)
    {
        cout << "OK" << endl;
    }
    else
    {
        cout << "Ошибка" << endl;
    }
    return 0;
}
```

Методы декомпозиции, решение уравнений

Armadillo предоставляет методы вычисления обратных матриц, разложения на множители, вычисление преобразований Фурье для одномерных и двумерных данных.

- chol - разложение Холецкого (A=UTU)
- eig_sym - спектральное разложение симметричной/эрмитовой матрицы (A=PDP-1)
- eig_gen - спектральное разложение произвольной квадратной матрицы

- eig_pair - спектральное разложение пары произвольных квадратных матриц
- eigs_sym - спектральное разложение симметричной вещественной разреженной матрицы
- eigs_gen - спектральное разложение произвольной квадратной разреженной матрицы
- fft / ifft - одномерное преобразование Фурье
- fft2 / ifft2 - двумерное преобразование Фурье
- inv - обращение произвольной квадратной матрицы
- inv_sympd - обращение симметричной положительно-определенной квадратной матрицы
- lu - LU-разложение (A=LU)
- null - ортонормированный базис ядра матрицы
- orth - ортонормированный базис пространства строк/столбцов матрицы
- pinv - псевдо-обратная матрица
- princomp - метод главных компонент
- qr - QR-разложение (A=QR)
- qr_econ - экономное QR-разложение (для неквадратных матриц)
- qz - обобщенное разложение Шура (QZ-разложение, A=QSZ* & B=QTZ*)
- schur - разложение Шура (A=QUQ-1)
- solve - решение системы линейных уравнений
- spsolve - решение разреженной системы линейных уравнений
- svd - сингулярное разложение
- svd_econ - экономное сингулярное разложение (для неквадратных матриц)
- svds - сингулярное разложение разреженной матрицы
- syl - решение уравнения Сильвестра (AX+XB=C)

Другие возможности и более подробную информацию можно найти на сайте библиотеки в разделе документов:

`http://arma.sourceforge.net/docs.html`.

Реализация метода главных компонент в Armadillo

Благодаря лаконичному и понятному синтаксису библиотеки, имеющимся методам загрузки данный, эффективной реализации алгоритмов разложения матриц, результат получается быстрым, а код сжатым и понятным:

```
#include <iostream>
#include <armadillo>

using namespace std;
using namespace arma;
```

```
int main(int argc, char** argv)
{
    // Матрица со спектральными данными
    mat A;

    // Загрузка данных из файла в формате arma_ascii
    bool status = A.load("data.csv", csv_ascii);
    // Проверка успешности
    if(status == true)
    {
        cout << "loaded okay" << endl;
    }
    else
    {
        cout << "problem with loading" << endl;
    }
    cout << A.n_rows << "x"<< A.n_cols<< "=" << A.n_elem << endl;

    // Вычисление ковариационной матрицы
    mat Q = cov(A);
    cout << Q.n_rows << "x"<< Q.n_cols<< "=" << Q.n_elem << endl;

    // Расчет главных компонент
    mat coeff;
    mat score;
    vec latent;
    vec tsquared;

    princomp(coeff, score, latent, tsquared, A);

    // Вывод результатов расчета
    cout<<"coeff" << endl;
    cout<<coeff.n_rows<<"x"<< coeff.n_cols<< "=" << coeff.n_elem << endl;
    cout << "score" << endl;
    cout<<score.n_rows<<"x"<< score.n_cols<< "=" << score.n_elem << endl;
    cout << "latent" << endl;
    cout << latent.n_elem << endl;
    cout << latent.head(12)<< endl;
    cout << "tsquared" << endl;
    cout << tsquared.n_elem << endl;
    cout << tsquared.head(12)<< endl;

    return 0;
}
```

Расчеты с использованием библиотеки OpenCV

Настройка проекта

Для начала использования библиотеки необходимо ее подключить к проекту и указать пути к заголовочным файлам. Предполагая, что создание и установка библиотеки произведены успешно в соответствии сописанием

в начале этого документа, необходимо в создаваемом учебном проекте в его настройках указать путь к include-каталогу библиотеки "C:_lib" для всех конфигураций (Release и Debug). Проект должен создаваться с признаком "GUI application" так как примеры будут использовать оконные возможности Windows.В отличие от Eigen, OpenCV требует для использования в проекте не только включение заголовочных файлов, но и использование библиотек при компоновке. В CodeBlocks подключить библиотеки можно добавив все ранее собранные библиотеки из каталога `"C:\_lib\opencv\x64\mingw\staticlib"` в учебный проект (Build options->linker settings->link libraries, Add). Можно компоновать учебный проект с динамическими библиотеками OpenCV.В этом случае компоновку нужно осуществлять с библиотеками из `"C:\_lib\opencv\x64\mingw\lib"`, и при запуске программы убедиться, что динамические библиотеки (DLL), находящиеся в каталоге `"C:\_lib\opencv\x64\mingw\bin\"` добавлены в пути или скопированы в каталог, откуда запускается учебный проект.

Замечание: CodeBlocks по какой-то причине для успешной сборки требует включить набор библиотек в проект дважды, причем теряет дубликаты при перезапуске. Чтобы обойти эту неприятность, библиотеки можно включить в каждую из конфигураций (Debug, Release), а также в корневую конфигурацию. Для наглядности ниже напечатан снимок экрана, на котором указаны библиотеки OpenCV. Они должны быть включены и в Debug и в Release и в корневую конфигурацию, от которой они наследуют и которая носит имя проекта (на снимке экрана `snippets_opencv`)

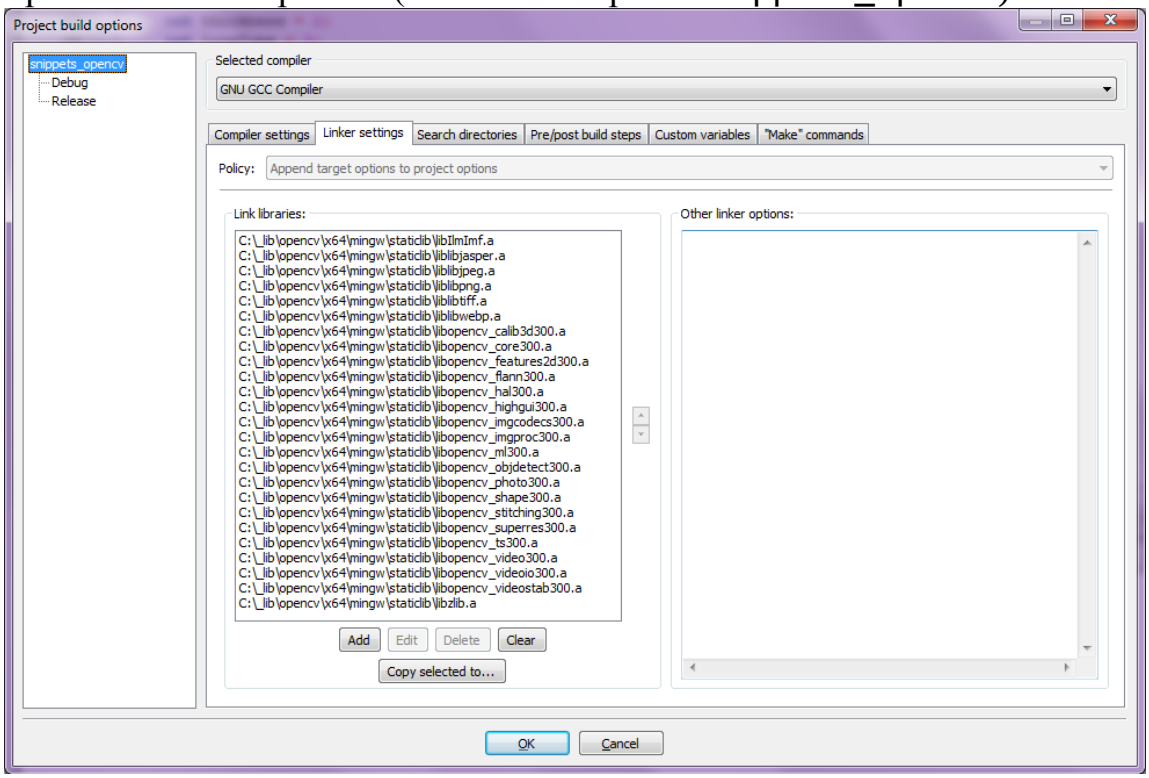


Рисунок 1 - Снимок экрана настроек проекта «`snippets_opencv`»

В дистрибутиве OpenCV можно найти большое количество примеров её использования. Они находятся в каталоге "samples" и его подкаталогах. Пример простой программы из набора примеров, загружающей изображение из файла, и показывающей его в графическом окне, прииведен ниже. Изображение "HappyFish.jpg" можно найти в папке "/samples/data/" дистрибутива OpenCV.

```
#include <opencv2/core/core.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui/highgui.hpp>

#include <iostream>
#include <string>

using namespace cv;
using namespace std;

int main( int argc, char** argv )
{
    // файл изображения по умолчанию
    string imageName("HappyFish.jpg");
    if( argc > 1)
    {
        imageName = argv[1];
    }
    // изображение
    Mat image;

    // чтение файла
    image = imread(imageName.c_str(), IMREAD_COLOR);
    // проверка результата чтения
    if( image.empty() )
    {
        cout <<  "Невозможно открыть файл изображения" << endl ;
        return -1;
    }
    // создание окна для изображения
    namedWindow( "Display window", WINDOW_AUTOSIZE );
    // показать изображение в созданном только что окне
    imshow( "Display window", image );
    // ожидание нажатия любой кнопки
    waitKey(0);
    return 0;
}
```

В этом примере изображение зачитывается из файла в память с помощью функции `imread`. Структурой, отвечающей за хранение данных изображения в памяти, является класс `Mat`. Класс может быть конвертирован в класс `Matrix` или `Array` библиотеки Eigen. Таким образом

может быть обеспечено применение алгоритмов линейной алгебры для обработки изображений и видео. Для иллюстрации приводим дополненный операциями конвертации пример из библиотеки OpenCV.

```
#include "Eigen/Core" // заголовок библиотеки Eigen. Должен быть включен
перед eigen.hpp
#include "opencv2/core/core.hpp"
#include "opencv2/core/eigen.hpp"
#include <iostream>

using namespace std;
using namespace cv;
using namespace Eigen;

int main(int,char**)
{
    // создание изображения конструктором инициализации
    Mat M(2,2, CV_8UC3, Scalar(0,0,255));
    cout << "M = " << endl << " " << M << endl << endl;

    // создание изображения методом create()
    M.create(4,4, CV_8UC(2));
    cout << "M = "<< endl << " "  << M << endl << endl;

    // создание многомерных матриц
    int sz[3] = {2,2,2};
    Mat L(3,sz, CV_8UC(1), Scalar::all(0));
    // оператор печати на консоль для них не определен

    // создание изображения в стиле MATLAB и Eigen:
    Mat E = Mat::eye(4, 4, CV_64F);
    cout << "E = " << endl << " " << E << endl << endl;
    Mat O = Mat::ones(2, 2, CV_32F);
    cout << "O = " << endl << " " << O << endl << endl;
    Mat Z = Mat::zeros(3,3, CV_8UC1);
    cout << "Z = " << endl << " " << Z << endl << endl;

    // создание матрицы 3x3 с элементами двойной точности (double)
    Mat C = (Mat_<double>(3,3) << 0, -1, 0, -1, 5, -1, 0, -1, 0);
    cout << "C = " << endl << " " << C << endl << endl;

    Mat RowClone = C.row(1).clone();
    cout << "RowClone = " << endl << " " << RowClone << endl << endl;

    // заполнение матрицы случайными значениями
    Mat R = Mat(3, 2, CV_8UC3);
    randu(R, Scalar::all(0), Scalar::all(255));

    // преобразование в типы данных Eigen и обратно
    Mat_<float> cv_mat = Mat_<float>::ones(2,2);
    MatrixXf eigen_matrix;
    cv2eigen(cv_mat,eigen_matrix);
    eigen2cv(eigen_matrix,cv_mat);
```

```
    // возможности форматирования консольного вывода
    cout << "R (default) = " << endl <<        R
<< endl << endl;
    cout << "R (python)  = " << endl << format(R, Formatter::FMT_PYTHON)
<< endl << endl;
    cout << "R (numpy)   = " << endl << format(R, Formatter::FMT_NUMPY )
<< endl << endl;
    cout << "R (csv)     = " << endl << format(R, Formatter::FMT_CSV   )
<< endl << endl;
    cout << "R (c)       = " << endl << format(R, Formatter::FMT_C     )
<< endl << endl;

    Point2f P(5, 1);
    cout << "Двумерная точка, Point (2D) = " << P << endl << endl;

    Point3f P3f(2, 6, 7);
    cout << "Трехмерная точка, Point (3D) = " << P3f << endl << endl;

    vector<float> v;
    v.push_back( (float)CV_PI);
    v.push_back(2);
    v.push_back(3.01f);
    cout << "Вектор вещественных чисел с использованием Mat = " << Mat(v)
<< endl << endl;

    vector<Point2f> vPoints(20);
    for (size_t i = 0; i < vPoints.size(); ++i)
        vPoints[i] = Point2f((float)(i * 5), (float)(i % 7));
    cout << "Вектор 2D Points = " << vPoints << endl << endl;
    return 0;
}
```

Кроме чтения и записи файлов в разнообразных графических форматах OpenCV позволяет создавать графику с помощью встроенных графических примитивов. Следующий пример использует функции `ellipse` и `circle` для рисования. Набор графических примитивов этими двумя функциями не ограничивается, можно рисовать линии, стрелки, прямоугольники, полигоны, текст.

```
#include <opencv2/core.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>

// ширина и высота изображения
#define w 400

using namespace cv;

void MyEllipse( Mat img, double angle );
void MyFilledCircle( Mat img, Point center );

int main( void )
```

```
{
    // название окна
    char atom_window[] = "Drawing 1: Atom";

    // изображение изначально черное
    Mat atom_image = Mat::zeros( w, w, CV_8UC3 );

    // a. рисуем эллипсы
    MyEllipse( atom_image, 90 );
    MyEllipse( atom_image, 0 );
    MyEllipse( atom_image, 45 );
    MyEllipse( atom_image, -45 );

    // b. рисуем круги
    MyFilledCircle( atom_image, Point( w/2, w/2) );

    // показываем нарисованное в окне и ожидаем нажатия любой клавиши
    imshow( atom_window, atom_image );
    moveWindow( atom_window, 0, 200 );
    waitKey( 0 );
    return(0);
}

void MyEllipse( Mat img, double angle )
{
    int thickness = 2;
    int lineType = 8;

    // примитив рисования эллипса в изображениии `img`
    ellipse( img,
             Point( w/2, w/2 ),
             Size( w/4, w/16 ),
             angle,
             0,
             360,
             Scalar( 255, 0, 0 ),
             thickness,
             lineType );
}

void MyFilledCircle( Mat img, Point center )
{
    int thickness = -1;
    int lineType = 8;

    // примитив рисования круга в изображениии `img`
    circle( img,
            center,
            w/32,
            Scalar( 0, 0, 255 ),
            thickness,
            lineType );
}
```

OpenCV позволяет работать с видео в различных форматах. В примере работы с видео, видеофайл зачитывается в память кадр за кадром, и каждый кадр отдельно обрабатывается. В примере применяются алгоритмы удаления фона.

```
#include "opencv2/imgcodecs.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/videoio.hpp"
#include <opencv2/highgui.hpp>
#include <opencv2/video.hpp>
#include <stdio.h>
#include <iostream>
#include <sstream>

using namespace cv;
using namespace std;

Mat frame; // текущий кадр видео
Mat fgMaskMOG2; // выделение (маска) переднего плана, создаваемое методом
MOG2
Ptr<BackgroundSubtractor> pMOG2; // удалитель фона MOG2
int keyboard; //введенный с клавиатуры символ

void processVideo(char* videoFilename);

int main(int argc, char* argv[])
{
    // проверка входных параметров
    if(argc != 3) {
        cerr <<"Ошибка параметров программы." << endl;
        return EXIT_FAILURE;
    }

    // создание окон графического интерфейса
    namedWindow("Frame");
    namedWindow("FG Mask MOG 2");

    // создание объекта удаления фона
    pMOG2 = createBackgroundSubtractorMOG2(); // метод MOG2

    if(strcmp(argv[1], "-vid") == 0) {
        // если входные данные - видео, передаем их на обработку
        processVideo(argv[2]);
    }
    else {
        // ошибка параметра - отсутствие параметра `-vid`
        cerr <<"Ошибка параметров программы." << endl;
        return EXIT_FAILURE;
    }
    // закрытие окон графического интерфейса
    destroyAllWindows();
    return EXIT_SUCCESS;
}
```

```
// функция обработки видео
void processVideo(char* videoFilename) {
    // создание объекта захвата видео
    VideoCapture capture(videoFilename);
    if(!capture.isOpened()){
        // ошибка открытия устройства видеозахвата
        cerr << "Невозможно открыть видеофайл: " << videoFilename << endl;
        exit(EXIT_FAILURE);
    }
    // покадровое чтение входных данных. Для выхода нажать ESC или 'q'
    while( (char)keyboard != 'q' && (char)keyboard != 27 ){
        // чтение текущего кадра
        if(!capture.read(frame)) {
            cerr << "Невозможно прочитать кадр." << endl;
            exit(EXIT_FAILURE);
        }
        // обновление фона с помощью MOG2
        pMOG2->apply(frame, fgMaskMOG2);
        // получение номера кадра и запись результата в текущий кадр
        stringstream ss;
        rectangle(frame, cv::Point(10, 2), cv::Point(100,20),
                  cv::Scalar(255,255,255), -1);
        ss << capture.get(CAP_PROP_POS_FRAMES);
        string frameNumberString = ss.str();
        putText(frame, frameNumberString.c_str(), cv::Point(15, 15),
                FONT_HERSHEY_SIMPLEX, 0.5 , cv::Scalar(0,0,0));
        // показать текущий кадр и выделение переднего плана
        imshow("Frame", frame);
        imshow("FG Mask MOG 2", fgMaskMOG2);
        // проверить нажатие клавиши
        keyboard = waitKey( 30 );
    }
    // удалить объект видеозахвата
    capture.release();
}
```

Библиотека содержит разнообразные классы и методы, разделенные на следующие функциональные модули: * core: ядро, основные структуры данных * imgproc: обработка изображений * highgui: интерфейсы пользователя (GUI) и ввод-вывод * video: анализ видео * calib3d: калибровка и 3D-реконструкция * features2d: 2D выделение признаков * objdetect: детектирование объектов * ml: машинное обучение * flann: кластеризация и поиск в многомерном пространстве * gpu: ускорение с использованием GPU * photo: вычислительная фотография * stitching: соединение изображений * ocl: ускорение с использованием OpenCL * superres: сверх-разрешение * viz: 3D-визуализация

Реализация метода главных компонент с использованием OpenCV

Реализация PCA на OpenCV получилась наиболее лаконичной:

```
#include "Eigen/Core" // should be included ahead of eigen.hpp
#include "opencv2/opencv.hpp"
#include "opencv2/ml.hpp"
#include <iostream>

using namespace std;
using namespace cv;
using namespace Eigen;


int main(int,char**)
{
    // Чтение данных из CSV
    cv::Ptr<cv::ml::TrainData> raw_data =
                        cv::ml::TrainData::loadFromCSV("data.csv", 0);
    cv::Mat data = raw_data->getSamples();
    cout << data.rows << "x" << data.cols << endl;


    // Расчет метода главных компонент
    PCA pca(data, Mat(), CV_PCA_DATA_AS_ROW);

    // Вывод результата
    cout << pca.eigenvalues / sum(pca.eigenvalues)[0] << endl;

    return 0;
}
```

Расчеты с использованием библиотеки boost.compute

Библиотека `boost.compute` является частью набора библиотек `boost`, одобренной организационным комитетом к включению в состав набора, но на момент написания документа не включенная в последний релиз 1.59. Boost.compute позволяет использовать возможности графической карты для ускорения вычислений.

```
/**
  Возвращает результат применения функции `function` ко всем элементам в
  диапазоне [first, last) и значению `init`.
 */
T accumulate(InputIterator first, InputIterator last, T init,
  BinaryFunction function)

/**
  Возвращает true если value будет найдена в отсортированном диапазоне
[first, last).
 */
bool binary_search(InputIterator first, InputIterator last,
```

```
  const T & value)

/**
  Вычисление префиксной суммы без текущего элемента (exclusive)
  для диапазона [first, last) и сохранение результата начиная
  с итератора result.
 */
OutputIterator exclusive_scan(InputIterator first,
  InputIterator last, OutputIterator result)

/**
  Вызов функции `function` для каждого элемента в диапазоне [first, last).
 */
UnaryFunction for_each(InputIterator first, InputIterator last,
  UnaryFunction function)

/**
  Копирование элементов, использующее индексы в диапазоне [first, last)
  в диапазон, начинающийся с итератора result
  из диапазона, начинающегося с input.
 */
void gather(MapIterator first, MapIterator last,
  InputIterator input, OutputIterator result)

/**
  Вычисление префиксной суммы с текущим элементом (inclusive)
  для диапазона [first, last) и сохранение результата начиная
  с итератора result.
 */
OutputIterator inclusive_scan(InputIterator first,
  InputIterator last, OutputIterator result)

/**
  Возвращает скалярное произведение элементов в диапазоне
  [first1, last1) с элементами во втором диапазоне,
  начинающимся с first2.
 */
T inner_product(InputIterator1 first1, InputIterator1 last1,
  InputIterator2 first2, T init,
  BinaryAccumulateFunction accumulate_function,
  BinaryTransformFunction transform_function)

/**
  Объединяет множества отсортированных значений в диапазоне [first1,
last1) с
  множеством отсортированных значений в диапазоне [first2, last2)
  и сохраняет результат в диапазоне начинающемся в result.
  Значения сравниваются функцией comp.
 */
OutputIterator merge(InputIterator1 first1, InputIterator1 last1,
  InputIterator2 first2, InputIterator2 last2,
  OutputIterator result, Compare comp)

/**
```

```
  Вычисляет накопительную сумму элементов в диапазоне [first, last)
  и сохраняет результат в диапазоне, начинающимся с итератора result.
 */
OutputIterator partial_sum(InputIterator first,
  InputIterator last, OutputIterator result)

/**
  Случайным образом перемешивает элементы в диапазоне [first, last).
 */
void random_shuffle(Iterator first, Iterator last)

/**
  Возвращает результат последовательного применения оператора к
  элементам в диапазоне [first, last).
  Бинарный оператор должен быть коммутативным.
 */
void reduce(InputIterator first, InputIterator last,
  OutputIterator result, BinaryFunction function)

/**
  Осуществляет вращение элементов таким образом чтобы элемент n_first
  переместился в начало.
 */
void rotate(InputIterator first, InputIterator n_first,
  InputIterator last)

/**
  Копирует элементы из диапазона [first, last) в диапазон
  начинающийся с итератора result используя индексы из map.
 */
void scatter(InputIterator first, InputIterator last,
  MapIterator map, OutputIterator result)

/**
  Сортирует элементы в диапазоне [first, last) используя оператор compare.
 */
void sort(Iterator first, Iterator last, Compare compare)

/**
  Осуществляет сортировку по ключу в диапазоне
  [keys_first, keys_last) по значениям ключей в диапазоне
  [values_first, values_first + (keys_last - keys_first))
  используя оператор compare.
 */
void sort_by_key(KeyIterator keys_first, KeyIterator keys_last,
  ValueIterator values_first, Compare compare)

/**
  Сортирует элементы в диапазоне [first, last) используя оператор compare.
  Относительный порядок объектов с равнозначными значениями сохраняется.
 */
void stable_sort(Iterator first, Iterator last, Compare compare)

/**
```

```
  Преобразует элементы в диапазоне [first, last) используя функцию
transform
  и сохраняет результат в диапазоне, наинающимся с result.
 */
OutputIterator transform(InputIterator1 first1, InputIterator1 last1,
  InputIterator2 first2, OutputIterator result, BinaryOperator op)

/**
  Преобразует элементы в диапазоне [first, last) с помощью унарной функции
  transform_function и затем накапливает полученные значения с помощью
  функции reduce_function.
 */
void transform_reduce(InputIterator first, InputIterator last,
  OutputIterator result, UnaryTransformFunction transform_function,
  BinaryReduceFunction reduce_function)
```

OpenCV, расматривавшийся раньше, отлично может быть интегрированн в программы, написанные для использования библиотеки boost.compute. Пример с сайта `boost` показывает возможности интеграции методов ввода данных из файла или из камеры с помощью OpenCV с дальнейшей их обработкой на графической карте (GPU) методами boost.compute. Большая библиотека примеров использования других технологий может быть найдена по ссылке: https://github.com/boostorg/compute/tree/master/example

```
//---------------------------------------------------------------------
---//
// Copyright (c) 2013-2014 Mageswaran.D <mageswaran1989@gmail.com>
//
// Distributed under the Boost Software License, Version 1.0
// See accompanying file LICENSE_1_0.txt or copy at
// http://www.boost.org/LICENSE_1_0.txt
//
// See http://boostorg.github.com/compute for more information.
//---------------------------------------------------------------------
---//

#include <iostream>
#include <string>

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>

#include <boost/compute/system.hpp>
#include <boost/compute/interop/opencv/core.hpp>
#include <boost/compute/interop/opencv/highgui.hpp>
#include <boost/compute/utility/source.hpp>

#include <boost/program_options.hpp>

namespace compute = boost::compute;
namespace po = boost::program_options;
```

```
// Функция свертки на OpenCL
const char source[] = BOOST_COMPUTE_STRINGIZE_SOURCE (
    __kernel void convolution(__read_only  image2d_t  sourceImage,
                              __write_only image2d_t  outputImage,
                              __constant float* filter,
                              int filterWidth)
    {
        const sampler_t sampler = CLK_NORMALIZED_COORDS_FALSE |
                                  CLK_ADDRESS_CLAMP_TO_EDGE   |
                                  CLK_FILTER_NEAREST;

        // Уникальные ряд и столбец для данного work-item
        int x   = get_global_id(0);
        int y   = get_global_id(1);

        // Пловина ширины фильтра для вычислений
        int halfWidth = (int)(filterWidth/2);

        // Обращение к данным изображения возвращает ветор вещественных
чисел
        // одинарной точности (float4).
        float4 sum = {0.0f, 0.0f, 0.0f, 0.0f};

        // Счетчик для фильтра
        int filterIdx = 0;

        // Каждый work-item обрабатывает свою собственную область
        // на глубину ширины фильтра
        int2 coords;  // Координаты точки изображения

        // Обход по строкам фильтра
        for(int i = -halfWidth; i <= halfWidth; i++)
        {
            coords.y = y + i;

            // Обход по столбцам фильтра
            for(int j = -halfWidth; j <= halfWidth; j++)
            {
                coords.x = x + j;

                float4 pixel;

                // Чтение пикселя из изображения.Обработка результата.
                pixel = read_imagef(sourceImage, sampler, coords);
                sum.x += pixel.x * filter[filterIdx++];
                //sum.y += pixel.y * filter[filterIdx++];
                //sum.z += pixel.z * filter[filterIdx++];
            }
        }

         barrier(CLK_GLOBAL_MEM_FENCE);
        // Сохранение результата в выходное изображение если точка
        // находится в его пределах
```

```
        if(y < get_image_height(sourceImage) &&
                x < get_image_width(sourceImage))
        {
            coords.x = x;
            coords.y = y;

            write_imagef(outputImage, coords, sum);
        }
    }
);

// В примере изображения загружаются из файла или камеры
// с помощью OpenCV, передаются в память GPU,
// и обрабатываются с помощью сверточной функции, реализованной на OpenCL
int main(int argc, char *argv[])
{
    // установка параметров кмандной строки, обрабатываемых программой
    po::options_description desc;
    desc.add_options()
            ("help",  "show available options")
            ("camera", po::value<int>()->default_value(-1),
                            "if not default camera, specify a camera id")
            ("image", po::value<std::string>(), "path to image file");

    // разбор командной строки
    po::variables_map vm;
    po::store(po::parse_command_line(argc, argv, desc), vm);
    po::notify(vm);

    // проверка аргументов командной строки
    if(vm.count("help"))
    {
        std::cout << desc << std::endl;
        return 0;
    }

////////////////////////////////////////////////////////////////////////////
/

    // переменные OpenCV
    cv::Mat cv_mat;
    cv::VideoCapture cap; //камера OpenCV.

    // Коэффициенты фильтра
    float filter[9] =  {
                -1.0,      0.0,      1.0,
                -2.0,      0.0,      2.0,
                -1.0,      0.0,      1.0,
                };

    // Размерность фильтра 3x3
    int filterWidth = 3;

    // получение основного устройства GPU и настройка контекста
```

```
    compute::device gpu = compute::system::default_device();
    compute::context context(gpu);
    compute::command_queue queue(context, gpu);
    compute::buffer dev_filter(context, sizeof(filter),
                               compute::memory_object::read_only |
                               compute::memory_object::copy_host_ptr,
                               filter);

    compute::program filter_program =
            compute::program::create_with_source(source, context);

    try
    {
        filter_program.build();
    }
    catch(compute::opencl_error e)
    {
        std::cout<<"Build Error: "<<std::endl
                 <<filter_program.build_log();
    return -1;
    }

    // создание вычислительного ядра фильтра и передача параметров
    compute::kernel filter_kernel(filter_program, "convolution");

////////////////////////////////////////////////////////////////////////
/

    // проверка аргументов и чтение изображения
    if(vm.count("image"))
    {
        // Read image with OpenCV
        cv_mat = cv::imread(vm["image"].as<std::string>(),
                                       CV_LOAD_IMAGE_COLOR);
        if(!cv_mat.data){
            std::cerr << "Failed to load image" << std::endl;
            return -1;
        }
    }
    else // при отстутсвии аргументов использовать изображение с камеры
    {
        // открыть камеру
        cap.open(vm["camera"].as<int>());
        // прочитать первый кадр
        cap >> cv_mat;
        if(!cv_mat.data){
            std::cerr << "failed to capture frame" << std::endl;
            return -1;
        }
    }


  // преобразовать в формат цвета BGRA
    // (OpenCL требует 16-байтного выравнивания)
```

```
    cv::cvtColor(cv_mat, cv_mat, CV_BGR2BGRA);

    // передача изображения/кадра в память GPU
    compute::image2d dev_input_image =
            compute::opencv_create_image2d_with_mat(
                cv_mat, compute::image2d::read_write, queue
                );

    // создание выходного изображения заданного размера
    compute::image2d dev_output_image(
                context,
                dev_input_image.width(),
                dev_input_image.height(),
                dev_input_image.format(),
                compute::image2d::write_only
                );

    filter_kernel.set_arg(0, dev_input_image);
    filter_kernel.set_arg(1, dev_output_image);
    filter_kernel.set_arg(2, dev_filter);
    filter_kernel.set_arg(3, filterWidth);

    // запуск вычислительного ядра
    size_t origin[2] = { 0, 0 };
    size_t region[2] = { dev_input_image.width(),
                         dev_input_image.height() };

    queue.enqueue_nd_range_kernel(filter_kernel, 2, origin, region, 0);

    // проверка аргументов и показ результата
    if(vm.count("image"))
    {
        // показать исходное изображение
        cv::imshow("Original Image", cv_mat);

        // показать отфильтрованное изображение
        compute::opencv_imshow("Convoluted Image", dev_output_image,
queue);

        // ожидание нажатия на клавиатуре
        cv::waitKey(0);
    }
    else
    {
        char key = '\0';
        while(key != 27) // проверка нажатия клавиши ESC
        {
            cap >> cv_mat;

            // преобразовать в формат цвета BGRA
            // (OpenCL требует 16-байтного выравнивания)
            cv::cvtColor(cv_mat, cv_mat, CV_BGR2BGRA);

            // передача кадра в память GPU
```

```
            compute::opencv_copy_mat_to_image(cv_mat,
                                              dev_input_image,queue);

            // запуск вычислительного ядра
            queue.enqueue_nd_range_kernel(filter_kernel, 2,
                                          origin, region, 0);

            // показать исходное изображение (кадр)
            cv::imshow("Camera Frame", cv_mat);

            // показать обработанное фильтром на на GPU изображение
            compute::opencv_imshow("Convoluted Frame",
                                   dev_output_image, queue);


            // ожидание нажатия на клавиатуре
            key = cv::waitKey(10);
        }
    }
    return 0;
}
```

Расчеты с использованием библиотеки Shark

Shark [1] - многоплатформенная библиотека алгоритмов компьютерного обучения, достоинствами которой являются простота использования, вычислительная эффективность, расширяемость, универсальность. Библиотека лицензируется лицензией Lesser GPL и может быть использована в коммерческих приложениях и научной работе.

Для создания своего проекта, использующего возможности Shark, необходимо дать доступ компилятору к include-каталогу библиотеки и скомпоновать проект с `Shark.a`. В настройках проекта установить в пути компилятора (Build options->Search directories->Compiler) следующие каталоги:

- "C:_lib\shark\include\shark"
- "$(BOOST_ROOT)"

В пути компоновщика (Build options->Search directories->Linker) каталог "C:_lib\shark\lib".

В настройках компилятора на вкладке "#defines" добавить определение "_USE_MATH_DEFINES".

Линейная алгебра. Векторы и матрицы.

[1]Christian Igel, Verena Heidrich-Meisner, and Tobias Glasmachers. Shark. Journal of Machine Learning Research 9, pp. 993-996, 2008.

Линейная алгебра в Shark реализована модулем LinAlg, основанным на подмножестве алгоритмов пакета uBLAS, и подключается к проекту заголовком `#include <shark/LinAlg/Base.h>`. Аналогично `Eigen`, реализованы два основных класса - матрицы и векторы. Матричные и векторные выражения возвращают объект, вычисление значений которого происходит только в момент присваивания.

Создание объектов и обращение к их значениям производится следующими операциями:

Класс или операция	**Описание**
XVector, XMatrix	Создание вектора/матрицы типа X, где X может быть Real, Float, Int, UInt или Bool.
SparseXVector, SparseXMatrix	Создание разреженного вектора/матрицы типа X, где X может быть Real, Float, Int, UInt или Bool.
XVector x(n,t)	Создание вектора типа X с n элементами, инициализированными значением t (по умолчанию 0).
XMatrix A(m,n,t)	Создание матрицы типа X размерностью [m;n], со значениями t (по умолчанию 0).
x.size()	Размерность объекта x.
A.size1(),A.size2()	Число элементов по первому измерению объекта A (число строк) и число элементов по по второму измерению (число столбцов).
A(i,j)	Возвращает ij-й элемент матрицы A.
x(i)	Возвращает i-й element вектора x.
row(A,k)	Возвращает k-ю строку A.
column(A,k)	Возвращает k-й столбец A.
rows(A,k,l)	Возвращает строки матрицы A с номерами от k до l.
columns(A,k,l)	Возвращает столбцы матрицы A с номерами от k до l.
subrange(x,i,j)	Возвращает подвектор x с номерами элементов от i до j−1.
subrange(A,i,j,k,l)	Возвращает подматрицу A с элементами в строках от i до j-1 и столбцами от k до l-1.

Пример простой программы, использующей эти операции, приведен ниже.

```
#include <iostream>
#include <shark/LinAlg/Base.h>

using namespace shark;
using namespace std;
```

```
int main(int argc, char** argv)
{
    RealVector a(10);
    for( auto &e: a)
        e++;
    cout << a << endl;

    RealMatrix b(10,10), c = subrange(b,0,2,0,3);
    cout << c << endl;
}
```

Поддерживаются как поэлементные, так и матричные операции.

Поэлементные операции:

Операция	**Описание**
t*B, B*t	Скалярное произведение: t·Aij и Aij·t.
B/t	Скалярное деление: Aij/t.
A+B	Поэлементное сложение: Aij+Bij.
A-B	Поэлементное вычитание: Aij−Bij.
A*B, element_prod	Поэлементное умножение (произведение Адамара): Aij*Bij.
A/B, element_div(A,B)	Поэлементное деление: Aij/Bij.
safe_div(A,B,x)	Поэлементное деление с проверкой делителя на ноль. Если Bij=0 то результат равен x.
-A	Инверсия элементов A: −Aij.
exp(A), log(A),...	Применение одной из функций exp,log, abs, tanh, sqrt к каждому элементу.
pow(A,t)	Функция возведения в степень t каждого элемента A: pow(Aij,t)
sqr(A)	Возведение в квадрат каждого элемента A, эквивалентно A*A.
sigmoid(A)	Взятие функции sigmoid $f(x)=1/(1+e^{-x})$ для каждого элемента A.
softPlus(A)	Взятие функции softplus $f(x)=\log(1+e^{x})$ для каждого элемента A.
trans(A)	Транспонирование матрицы A.

Матричные операции

Операция	**Описание**
prod(A,B)	Произведение двух матриц, с корректной

	размерностью.
prod(A,x), prod(x,A)	Произведение матрица и вектора: Ax и xA.
inner_prod(x,y)	Скалярное произведение (также "dot product").
outer_prod(x,y)	Тензорное произведение, результат которого матрица C, где $C_{ij}=x_i y_j$.
fast_prod(A,B,C,b,t)	Вычислительно эффективное матричное произведение для матриц A, B и C. Вычисляет C += t*prod(A,B) если b истинно или C = t*prod(A,B) в противном случае. По умолчанию b = false и t = 1.
fast_prod(A,x,z,b,t)	То же самое для произведения матрица-вектор.
fast_prod(x,A,z,b,t)	То же самое для произведения вектор-матрица.
symmRankKUpdate(A,C,b,t)	Аналогично fast_prod(A,trans(A), C,b,t). Предполагается, что матрица C симметрическая.

Ускорение расчетов с использованием векторных инструкций

Использование векторных инструкций SIMD (Single Instruction Multiple Data) позволяет вручную оптимизировать исполнение кода. Компиляторы справляются с подобного рода оптимизациями автоматически, но в некоторых случаях имеет смысл написать критические части кода вручную. И для автоматической генерации кода и для использования векторных инструкций в коде для компилятора GCC необходимо включить флаг архитектуры целевого процессора `-march=`. Список возможных архитектур можно найти на странице `https://gcc.gnu.org/onlinedocs/gcc/x86-Options.html#x86-Options`. Можно включить используемый набор инструкций, явно выбрав нужные из набора опций:

- -mmmx
- -msse,-msse2,-msse3,-mssse3,-msse4,-msse4a,-msse4.1,-msse4.2
- -mavx,-mavx2,-mavx512f,-mavx512pf,-mavx512er,-mavx512cd
- -msha,-maes
- -mpclmul
- -mclfushopt
- -mfsgsbase
- -mrdrnd
- -mf16c
- -mfma

- -mfma4
- -mno-fma4
- -mprefetchwt1
- -mxop
- -mlwp
- -m3dnow
- -mpopcnt
- -mabm
- -mbmi
- -mbmi2
- -mlzcnt
- -mfxsr
- -mxsave
- -mxsaveopt
- -mxsavec
- -mxsaves
- -mrtm
- -mtbm
- -mmpx
- -mmwaitx

В в заголовочном файле описаны встроенные функции (intrinsics), поддерживающие данный расширенный набор SIMD команд. В результате для их использование не требуется ассемблерный код, достаточно использовать вызов встроенных функций, как обычных функций C/C++. В примере ниже с помощью встроенной SIMD-функции `__builtin_shuffle` за четыре вызова выполняется быстрое преобразование Уолш-Адамара для матрицы 16x16.

```
#include <x86intrin.h>

void fast_transform(SIMD &c)
{
    // Константы времени компиляции для преобразования Уолша-Адамара
    // (Fast Walsh–Hadamard Transform, FWHTh)
 constexpr SIMD_t shuffler_1{8,9,10,11,12,13,14,15, 0,1,2,3,4,5,6,7};
 constexpr SIMD_t shuffler_2{4,5,6,7,  0,1,2,3,  12,13,14,15,  8,9,10,11};
 constexpr SIMD_t shuffler_3{2,3, 0,1, 6,7, 4,5, 10,11, 8,9, 14,15, 2,13};
 constexpr SIMD_t shuffler_4{1,0, 3,2, 5,4, 7,6, 9,8, 11,10, 13,12, 5,14};
 constexpr SIMD_t inv_1{+1,+1,+1,+1,+1,+1,+1,+1,-1,-1,-1,-1,-1,-1,-1,-1};
 constexpr SIMD_t inv_2{+1,+1,+1,+1,-1,-1,-1,-1,+1,+1,+1,+1,-1,-1,-1,-1};
 constexpr SIMD_t inv_3{+1,+1,-1,-1,+1,+1,-1,-1,+1,+1,-1,-1,+1,+1,-1,-1};
 constexpr SIMD_t inv_4{+1,-1,+1,-1,+1,-1,+1,-1,+1,-1,+1,-1,+1,-1,+1,-1};

    // Преобразование матрицы 16x16 за четыре инструкции
    // Иллюстрация преобразования на Wikipedia:
    // http://en.wikipedia.org/wiki/Fast_Walsh-Hadamard_transform
    c.simd = c.simd * inv_1 + __builtin_shuffle(c.simd, shuffler_1 );
    c.simd = c.simd * inv_2 + __builtin_shuffle(c.simd, shuffler_2 );
```

```
    c.simd = c.simd * inv_3 + __builtin_shuffle(c.simd, shuffler_3 );
    c.simd = c.simd * inv_4 + __builtin_shuffle(c.simd, shuffler_4 );
}
```

Встроенные функции можно разгруппировать по нескольким категориям:

- чтение/запись
- побитовые операции: вращение, подсчет, логика, перемещение, манипулирование
- преобразования: округления, переформатирование
- арифметические функции: сложение, вычитание, умножение, деление, составные операции (умножить и сложеть)
- сравнения: арифметические, битовые, строковые
- манипулирование байтами: вставка, перестановка
- дополнительные: шифрование и служебные функции

Полное описание встроенных функций (intrinsics) доступно на сайте «https://software.intel.com/sites/landingpage/IntrinsicsGuide/»

Заключение

Во второй части методического пособия дан краткий обзор и примеры использования различных библиотек научных вычислений на C++. Синтаксис языка из первой части пособия и обзор имеющихся вычислительных средств, данный в этой части, обеспечивают базу для разработки самостоятельных проектов моделирования и научных расчётов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

При составлении пособия использовалась документация и примеры с сайтов следующих библиотек в сети Интернет:

- boost (набор различных библиотек) http://www.boost.org/
- boost.compute (ускорение вычислений) https://boostorg.github.io/compute/index.html
- Eigen (линейная алгебра) http://eigen.tuxfamily.org/
- Armadillo (линейная алгебра) http://arma.sourceforge.net/
- Shark (машинное обучение) http://image.diku.dk/shark/
- OpenCV (компьютерное зрение) http://opencv.org/

Миссия университета – генерация передовых знаний, внедрение инновационных разработок и подготовка элитных кадров, способных действовать в условиях быстро меняющегося мира и обеспечивать опережающее развитие науки, технологий и других областей для содействия решению актуальных задач.

КАФЕДРА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ ТОПЛИВНО-ЭНЕРГЕТИЧЕСКОГО КОМПЛЕКСА

Кафедра химии входила в состав первых 14 кафедр ЛИТМО, сформированных в 1930 году. В 1930–1960 годах кафедра работала в рамках факультета Точной механики; в период деятельности Инженерно-физического факультета (ИФФ) с 1946 года по 1954 год кафедра входила в состав ИФФ. С 1933 года – кафедру возглавлял известный специалист в области оптического стекла профессор В.Г. Воано, позже – известный русский ученый-химик профессор С.А. Щукарев. С 1954 по 1972 год кафедру возглавлял доцент Г.С. Кошурников.

С момента второго рождения инженерно-физического факультета в 1976 г. кафедра химии вошла в его состав. В это время на кафедре стали развиваться, в основном, три научно-технологических направления: создание новых композиционных оптических материалов; разработка химических сенсоров; технология оптического волокна.

В последующие годы сотрудники кафедры, прежде всего, профессора Новиков А.Ф. и Успенская М.В., существенно переработали методику преподавания курса химии, адаптировав ее к активно внедрявшейся тогда в Университете системе дистанционного обучения. В результате, преподавание курса химии в Университете ИТМО вышло на новый более высокий уровень.

В дальнейшем на кафедре под руководством профессора М.В. Успенской активно развивалось научно-техническое направление в области химии и физики сорбирующих полимерных материалов и нанокомпозитов. В частности, на основе акриловых супервлагоабсорбентов разработан ряд новых материалов многофункционального назначения: сенсоры, жидкие линзы, раневые повязки, искусственные почвы для сельского хозяйства, огнестойкие конструкционные элементы и др.

В связи с этим в 2011 году данная кафедра (исторически – кафедра химии) позиционировала себя как отдельное структурное подразделение Национального исследовательского университета ИТМО в качестве

кафедры “Информационных технологий топливно-энергетического комплекса”.

С переходом отечественных предприятий на международные стандарты продукции, повышением требований к охране окружающей среды и внедрением сложных аналитических автоматизированных систем контроля качества и мониторинга, с 2008 года в рамках направления «Техническая физика» кафедра проводит подготовку магистров и бакалавров по профилю «Физико-технические аспекты аналитического приборостроения».

Подготовка включает в себя следующие разделы:

- Компьютерные комплексы для автоматизированного контроля физических, химических, механических, термических, реологических и некоторых других свойств нефтяного сырья и продуктов нефтепереработки;
- Встроенные микропроцессорные комплексы для управления технологическими процессами и измерением широкого круга параметров энергетических установок и систем энергоснабжения;
- Физико-математическое моделирование технологических процессов нефтепереработки и топливно-энергетического комплекса;
- Информационно-аналитические системы и комплексы различного профиля, адаптированные под специфические условия работы на предприятиях ТЭК.

Уникальная программа обучения сочетает фундаментальную подготовку в области информационных систем, физической оптики, молекулярной спектроскопии, аналитической и физической химии, компьютерной метрологии, общехимической технологии и автоматики.

В рамках специальных дисциплин изучаются приборы и методы контроля качества продукции и принципы построения автоматизированных анализаторных систем для предприятий ТЭК, нефтяной и химической промышленности.

Такие системы как основа информационных технологий контроля качества и мониторинга безопасности могут успешно применяться практически на всех предприятиях и лабораториях химического и нефтехимического профиля, а также в металлургической, пищевой и фармацевтической промышленности.

Выпускники кафедры имеют широкие перспективы трудоустройства в современных крупных компаниях ТЭК, таких как Роснефть, ПТК, Газпром, Киришинефтеоргсинтез, Лукойл, ТНК-ВР, а также на предприятиях и лабораториях пищевой, фармацевтической и других отраслях промышленности.

Практика эксплуатации предприятий ТЭК подтверждает необходимость создания и применения эффективных систем контроля за безопасностью и систем экологического мониторинга.

В связи с этим с 2011 года были разработаны и открыты бакалаврская и магистерская программы по направлению подготовки 241000 " Энерго- и ресурсосберегающие процессы в химической технологии, нефтехимии и биотехнологии ". Основной целью образовательной магистерской программы "Информационные ресурсосберегающие технологии и экологические аспекты на предприятиях ТЭК" является подготовка высококвалифицированных специалистов, соответствующих современным требованиям к выпускникам вуза, с учетом потребностей рынка труда Санкт-Петербурга и регионов России. Будущие магистры будут способны использовать информационные технологии и математическое моделирование для описания различных физических и физико-химических процессов, для контроля качества продукции нефтепереработки, работать на современном оборудовании в научных, научно-производственных и производственных лабораториях по исследованию выпускаемой продукции и т.д.

Основными направлениями научной деятельности в рамках магистерской программы являются:

- Создание приборов и датчиков физических величин и физико-химических параметров углеводородного сырья и продуктов (в том числе на основе нанотехнологий);
- Разработка приборов для измерения параметров качества нефтепродуктов и пищевых продуктов на основе компьютерных технологий;
- Создание эффективных информационных систем контроля качества продукции и коммерческого учета на предприятиях ТЭК на основе приборов и устройств различного назначения;
- Создание эффективных информационных систем мониторинга безопасности эксплуатации объектов ТЭК.

Подготовка магистров ведется с участием ряда промышленных предприятий, научно-производственных объединений, научно-исследовательских институтов и вузов Санкт-Петербурга, что дает возможность получить отличные знания и неоценимый опыт в различных сферах деятельности: производственной, научно-исследовательской, административной и т.д.

Биотехнология и биоинженерия являются приоритетными направлениями современной науки и промышленного производства. Продукты биотехнологии и биоинженерии востребованы в медицине, фармации, биологии, и других высокотехнологичных отраслях народного хозяйства. Разработка новых источников энергии, создание биосовместимых материалов и синтез биологически активных веществ – главные составляющие этих двух наук и отраслей производства. В частности, интенсивно развиваются производство и применение ферментов в переработке различных видов сырья и в получении

биопрепаратов. Ферментные технологии имеют преимущества с экономической, технологической и экологической точек зрения, поэтому годовой оборот ферментных препаратов составляет десятки миллионов долларов США и он непрерывно растёт. По объёму производства ферментные препараты занимают третье место после аминокислот и антибиотиков. Ферментативные процессы, применяемые в технологиях, аналогичны природным, но они более безопасны и для здоровья человека и для окружающей среды.

Развитие этих отраслей сдерживается недостатком специалистов высшего уровня, подготовленных в области информационного обеспечения и средств измерения живых систем и биологических структур.

Для решения проблемы подготовки магистров на стыке информационных технологий, биологии и инженерии объединены усилия двух кафедр: Кафедра химии и молекулярной биологии ИХиБТ и кафедра ИТТЭК, имеющих опыт подготовки специалистов бакалавров и магистров в информационных технологиях и биотехнологии.

В учебный план предлагаемой программы включены, наряду с общеобразовательными, дисциплины по информационной, биологической, химической, технологической подготовке и ряду других отраслей знаний, необходимых в подготовке специалистов заявленного уровня.

В настоящее время на каф. ИТТЭК под руководством проф. Успенской М.В., ведутся работы по направлениям, связанных с созданием материалов для фармакологии и регенеративной медицины, предметов санитарно-гигиенического назначения, а также биосовместимых и биодеградируемых материалов.

Также на кафедре под руководством проф. Неелова И.М. активно развивается моделирование полимеров и биополимеров, начиная от структуры веществ и физико-химических процессов, протекающих в живых организмах до физико-механических и эксплуатационных характеристик материалов и биосистем.

Профессорско-преподавательский состав на кафедре насчитывает 18 человек, из них 6 профессоров и докторов наук.

В настоящее время на базе кафедр НИУ ИТМО создан Международный научно-исследовательский институт биоинженерии, возглавляемый проф. М.В. Успенской, что значительно расширяет экспериментальную базу и научный потенциал кафедр и способствует повышению уровня подготовки кадров высшей категории.

В настоящее время на кафедре трудятся 18 преподавателей, шестеро из них являются докторами наук, профессорами, признанными на международном уровне, членами ученых советов в России и за рубежом.

Никехин Алексей Алексеевич

Основы C++ для моделирования и расчетов.
Часть 2. Библиотеки для научных вычислений

Учебное пособие

В авторской редакции
Редакционно–издательский отдел Университета ИТМО
Зав. РИО Н.Ф. Гусарова
Подписано к печати
Заказ №
Тираж
Отпечатано на ризографе

Редакционно-издательский отдел
Университета ИТМО
197101, Санкт-Петербург, Кронверкский пр., 49