Московский физико-технический институт
(государственный университет)

Факультет аэромеханики и летательной техники

Программирование на языке C++

TEXнические материалы семинаров

2 сентября 2023 г.

Жуковский, *2023–2024 уч. гг.*

Оглавление

Введение

Цели и задачи

Усвоить основы разработки, тестирования и отладки программ, базовые конструкции языка C++ и элементы стандартной библиотеки и получить практические навыки их использования при решении ряда простых и более сложных задач.

Структура курса.

- О курсе в целом.
- Взаимосвязь с другими курсами.

- Контрольно-проверочные мероприятия.

Основная литература

Страуструп Б. Программирование: принципы и практика с использованием C++. М. : ООО «И.Д. Вильямс», 2016. 1328 с.

Материалы и задания

Значительная часть обучающих упражнений и заданий содержится в книге Бьярне Страуструпа. Дополнительные материалы размещены на яндекс-диске[1]:

- **`books`** — основная и дополнительная литература;
- **`cpp-lectures`** — лекционные материалы;
- **`cpp-seminars`** — семинарские материалы;
  - **`/libs`** — библиотеки, которые используются на занятиях;
  - **`/program.pdf`** — программа курса (темы для беседы на зачёте);
  - **`/progress.pdf`** — планы, успеваемость и проверочные мероприятия;
  - **`/seminars.pdf`** — вспомогательная методичка по материалам занятий.

Исходный код примеров из данного учебного пособия доступен в репозитории[2] на Git Hub, каталог `projects`.

Программное обеспечение

Интегрированная среда разработки (IDE — Integrated Development Environment):

- текстовый редактор,
- компилятор языка C++ (с поддержкой стандарта C++14 или выше),
- редактор связей,
- средства сборки,
- средства отладки.

Установка и настройка рабочей среды

Мы настоятельно рекомендуем в качестве IDE установить `VS Code`[3] от Microsoft. Эта программа активно развивается, имеет полноценный современный редактор кода и поддерживает интеграцию со множеством полезных инструментов (системой контроля версий, средствами форматирования кода, консолью). Распространяется свободно, кроссплатформенная, то есть доступна не только под Windows, но и под Unix: GNU/Linux, Mac OS X. А также доступна на территории России.

Запустите `VS Code`. Перейдите во вкладку «Расширения» на панели слева. Установите плагин C/C++ for Visual Studio Code.

Далее необходимо установить `Smartgit`[4] (или, в крайнем случае, просто `Git`[5]), оставляя рекомендуемые (по умолчанию) параметры в тех местах, где вы сомневаетесь, что выбрать. `Smartgit` включает в себя дистрибутив `Git`-а, поэтому последний не нужно ставить отдельно.

`Smartgit` — это удобный и мощный графический пользовательский интерфейс, то есть окошки и кнопочки, для работы с системой контроля версий `Git`. Это коммерческий продукт, однако, разработчик предоставляет некоммерческую лицензию для академических заведений, которую можно запросить, заполнив форму[6] на сайте.

[1] Материалы на Яндекс-диске: `https://yadi.sk/d/XiUj9ZsNf3xHJ`
[2] Репозиторий данного пособия: `https://github.com/vpodaruev/programming-with-cpp`
[3] Microsoft VS Code: `https://code.visualstudio.com/download`
[4] Syntevo Smartgit: `https://www.syntevo.com/smartgit/download`
[5] Система контроля версий Git: `https://git-scm.com/downloads`
[6] Запрос лицензии: `https://www.syntevo.com/register-non-commercial/#academic`

В сочетании с дополнительными плагинами, VS Code частично предоставляет подобные возможности. Но, увы, это пока не так удобно.

Система контроля версий — неотъемлемый инструмент разработки сложных программ. Также он весьма полезен при разработке небольших программ.

ОС Windows. Совместно с VS Code используйте MinGW-w64[7] (компилятор, компоновщик, отладчик и прочее).

Для этого:

– Распакуйте[8] архив MinGW-w64 в каталог c:\mingw-64 или любой другой, куда вы обычно устанавливаете программы.
– Добавьте[9] путь к компилятору (bin\g++.exe) в список стандартных системных путей (системная переменная PATH).

Удобная Unix-подобная командная среда Git Bash идёт в комплекте с системой контроля версий Git (а также в комплекте со Smartgit-ом). Простые программы мы будем собирать непосредственно в командной среде. Помимо прочего, она облегчит тестирование программ в автоматическом и полуавтоматическом режимах.

Добавьте[10] Git Bash в список доступных терминалов VS Code и выберите его в качестве терминала по умолчанию. (Файл с именем bash.exe нужно найти в каталоге установки Smartgit-а.)

ОС Unix. Пользователи Unix и подобных ей операционных систем могут установить средства сборки программ GNU/GCC при помощи менеджера системы управления пакетами (например, apt в Ubuntu или pamac в Manjaro). Командная среда bash, или подобная ей zsh, обычно доступна из коробки и не требует отдельной установки.

Если что-то пошло не так, повторите процедуру **спокойно**, выполняя **в точности** все указанные выше действия. Попробуйте использовать только латиницу для каталогов установки и проектов, иногда русские буквы в путях или пробелы могут вызывать ошибки.

И теперь не получилось? Хм-м... Тогда обратитесь за помощью к товарищам в группе, к студентам старших курсов или, в конце концов, к преподавателю.

Что читать

Страуструп Б.:[11] **главы 0, 1 и 2**

[7] MinGW-w64: https://github.com/niXman/mingw-builds-binaries/releases/download/13.1.0-rt_v11-rev1/x86_64-13.1.0-release-posix-seh-ucrt-rt_v11-rev1.7z

[8] Архиватор 7-Zip: https://www.7-zip.org/download.html

[9] Adding Path to MinGW: https://youtu.be/mQra00mT3Dg

[10] VS Code — Integrate Git Bash as Default Terminal: https://youtu.be/PzJCwfYfIzY?t=107

[11] *Страуструп Б.* Программирование: принципы и практика с использованием C++. М. : ООО «И.Д. Вильямс», 2016. 1328 с.

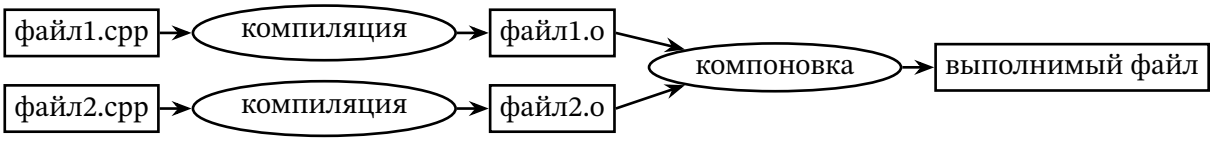
1 Hello, World!

1.1. Архитектура Фон-Неймана, уровни абстракции

Схематическое устройство компьютера. Алгоритмы и языки программирования. Абстракция от реального «железа».

1.2. Программы

C++[1] является компилируемым языком. Для работы программы её исходный текст должен быть обработан с помощью компилятора, который создаёт объектные файлы, объединяемые компоновщиком в выполнимую программу. Обычно программы на языке C++ состоят из многих файлов с исходными текстами (именуемыми просто *исходными файлами*).



Выполнимая программа создаётся для определённой комбинации аппаратного обеспечения и операционной системы; её нельзя просто перенести, скажем, из компьютера Mac в компьютер с Windows. Говоря о переносимости программ C++, мы обычно имеем в виду переносимость исходного кода, т. е. исходный код может быть успешно скомпилирован и выполняться в разных системах.

Стандарт ISO C++ определяет два типа сущностей.

- *Фундаментальные возможности языка*, такие как встроенные типы (например, `char` и `int`) или циклы (например, инструкции `for` и `while`).
- *Компоненты стандартных библиотек*, такие как контейнеры (например, `vector`, и `map`) или операции ввода–вывода (например, `<<` и `getline()`).

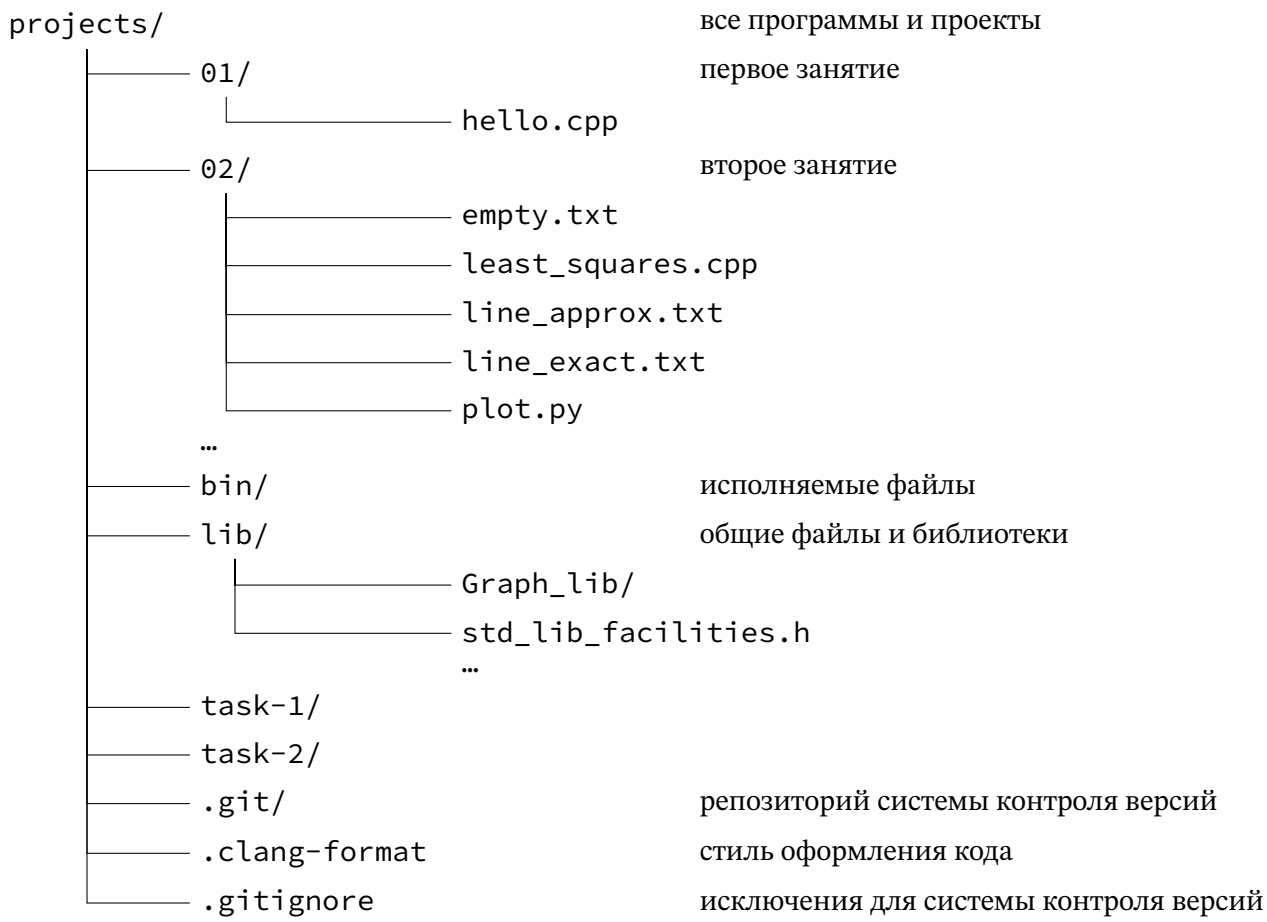
Компоненты стандартной библиотеки представляют собой совершенно обычный код C++, предоставляемый каждой реализацией языка. То есть стандартная библиотека C++ может быть реализована в самом C++ (и реализуется — с очень небольшим использованием машинного кода для таких вещей, как переключение контекста потока). Это означает, что C++ достаточно выразителен и эффективен для самых сложных задач системного программирования.

C++ является статически типизированным языком, т.е. тип каждой сущности (например, объекта, значения, имени или выражения) должен быть известен компилятору в точке использования. Тип объекта определяет набор применимых к нему операций.

[1] *Страуструп Б.* Язык программирования C++. Краткий курс : пер. с англ. СПб. : ООО «Диалектика», 2019. 320 с.

1.3. Структура каталогов

Настоятельно рекомендуем упорядочивать файлы. Например, для семинарских программ и проектов можно использовать следующую структуру:



1.4. Классическая первая программа

Наберите код классической первой программы (`hello.cpp`):

```
#include <std_lib_facilities.h>

int main ()
{
  cout << "Hello, World!\n";
  return 0;
}
```

Чтобы скомпилировать код, необходимо перейти (команда `cd`) в каталог всех проектов:

```
$ cd /path/to/your/projects
```

и уже оттуда выполнить команду:

```
$ g++ -Og -Wall -Wextra -pedantic -Ilib -o p 01/hello.cpp
```

Здесь мы использовали следующие ключики (флаги) компилятора `g++`:

`-Og`	Задаёт разумный уровень оптимизации при сохранении скорости компиляции и возможности отладки
`-Wall`	Включает все предупреждения относительно конструкций, которые многие пользователи считают сомнительными и которые легко исправить
`-Wextra`	Включает дополнительные предупреждения, которые не включает `-Wall`
`-pedantic`	Сообщает обо всех нарушениях стандарта языка C++. Отвергает программы, которые используют запрещённые расширения
`-Ilib`	Добавляет каталог `lib` в стандартные пути поиска компилятора. Таким образом, компилятор находит наш заголовок `std_lib_facilities.h` в стандартных путях (когда, используя директиву `#include`, мы указываем имя файла в угловых скобках, а не в двойных кавычках)
`-o p`	Даёт исполняемому файлу имя `p`. В `Windows` к имени автоматически добавляется расширение `exe`

Следом за ключиками мы перечисляем все файлы с исходным кодом (только `*.cpp` файлы). На первых порах у нас будут программы, в которых весь код размещается в одном файле. Немного позже мы начнём распределять код на несколько модулей.

Если компиляция прошла успешно, то на экране не появится никаких сообщений. Можно вывести список файлов в текущем каталоге, чтобы убедиться, что исполняемый файл с именем `p` действительно появился.

```
$ ls
01  02  lib  p
```

Теперь можно запустить программу на исполнение:

```
$ ./p
Hello, World!
```

Текущий каталог (обозначается символом «`.`») не входит в системные пути Unix, поэтому мы обязаны написать `./p`. Напротив, утилита `ls` расположена в системных путях, поэтому не требует уточнения пути.

В дальнейшем рекомендуем сохранять бинарный (исполняемый) файл в каталоге `bin`:

```
$ rm p
$ mkdir bin
$ ls
01  02  bin  lib
```

Мы удалили (команда `rm`) файл `p` и затем создали (команда `mkdir`) каталог `bin`.

```
$ g++ -Og -Wall -Wextra -pedantic -Ilib -o bin/p 01/hello.cpp
$ bin/p
Hello, World!
```

Совет: чтобы избежать повторного набора команд в терминале, используйте стрелки ↑ и ↓ для навигации по истории команд.

1.5. Редактирование текста

Набор исходного текста — неотъемлемая часть разработки программ. Используйте «горячие» клавиши для более быстрого редактирования. Краткий список наиболее популярных комбинаций, поддерживаемых даже простыми редакторами:

Ctrl + A	Выделить всё
Ctrl + C	Скопировать выделение
Ctrl + V	Вставить
Ctrl + Z	Отменить последнее действие
Home	Переместить курсор в начало/конец текущей строки
End	
Shift + Home	Выделить символы с текущей позиции и до начала/конца строки
Shift + End	
Ctrl + Home	Переместить курсор в начало/конец файла
Ctrl + End	

Дополнительные полезные комбинации в VS Code:

Ctrl + /	Комментировать/раскомментировать текущую строку или выделение
Alt + ↑	Сместить текущую строку или выделение вверх/вниз
Alt + ↓	
Alt + Shift + ↑	Дублировать текущую строку или выделение вверх/вниз
Alt + Shift + ↓	
Ctrl + Alt + ↑	Редактировать столбец
Ctrl + Alt + ↓	

Полезно научиться набирать текст вслепую, то есть не глядя на клавиатуру. В сочетании с использованием горячих клавиш, это позволяет достичь существенного ускорения при наборе исходного кода. Таким образом, остаётся больше времени для размышлений над структурой и логикой самой программы. В качестве примера он-лайн клавиатурного тренажёра приведём эту[2] ссылку.

Используйте моноширинный шрифт для кода, например, JetBrains Mono. По желанию, его можно скачать[3] и установить[4] в систему. Чтобы указать шрифт редактору, откройте файл с настройками VS Code и добавьте такие строки:

```
"editor.fontFamily": "JetBrains Mono",
"editor.fontLigatures": false
```

Если включить лигатуры, то некоторые составные операторы будут заменяться на более привычное начертание. Например, оператор != будет выглядеть как ≠.

Аккуратно оформляйте код, соблюдая правила хорошего стиля[5]. Современные средства позволяют автоматически форматировать код согласно заданному стилю. Описание стиля projects/.clang-format есть в репозитории. В настройки добавьте:

```
"editor.formatOnSave": true
```

Можно изменить стиль по вашему вкусу. Для этого отредактируйте загруженный файл, используя документацию[6] утилиты Clang-format.

[2] Он-лайн клавиатурный тренажёр: https://www.typingclub.com/sportal/program-3.game
[3] Шрифты JetBrains Mono: https://github.com/JetBrains/JetBrainsMono/releases
[4] Как установить шрифт в систему: https://youtu.be/LQlnsoaUSmY
[5] Стиль кода: https://lefticus.gitbooks.io/cpp-best-practices/content/03-Style.html
[6] Опции Clang-format: https://clang.llvm.org/docs/ClangFormatStyleOptions.html

1.6. Unix утилиты

Совместно с командной средой `bash` (**b**ourne **a**gain **sh**ell) поставляется широкий набор полезных программ, или утилит (utilities).

```
$ cd dir
```

change **d**irectory — перейти в каталог `dir`. Если каталог опущен, то перейти в домашний каталог. (Определяется значением переменной среды `HOME`.) Файловая система Unix представляет собой единое дерево, и любой абсолютный путь начинается с корня (`/`). Точка (`.`) означает текущий каталог, две точки (`..`) — родительский каталог.

```
$ cd ..
```

Перейти в родительский каталог.

```
$ cd /
```

Перейти в корневой каталог.

```
$ pwd
```

print **w**orking **d**irectory — вывести на экран абсолютный путь к текущему каталогу.

```
$ ls dir
```

lst — вывести на экран содержимое каталога `dir`. Если каталог опущен, то по умолчанию используется текущий каталог.

```
$ man cmd
```

manual — вывести на экран подробную справку о команде `cmd`.

```
$ cmd --help
```

Вывести на экран короткую справку о команде `cmd`. Полезно, если утилита `man` недоступна.

```
$ mkdir dir
```

ma**k**e **dir**ectory — создать каталог `dir`.

```
$ rmdir dir
```

re**m**ove **dir**ectory — удалить пустой каталог `dir`.

```
$ rm file
```

re**m**ove — удалить файл. Используя опцию `-r` можно рекурсивно удалить непустой каталог.

```
$ mv src dst
```

mo**v**e — перемещает/переименовывает файл `src` в `dst`. Имя `dst` может быть каталогом, тогда `mv` перемещает `src` туда.

```
$ cat file...
```

catenate — связывает файлы и выводит объединённое содержимое на экран.

```
$ less file
```

Более мощный аналог утилиты `more` — постраничного фильтра для просмотра файлов. Часто применяется для буферизации вывода на экран, используя конвейер:

```
$ odjdump -d prog | less
```

```
$ vim file
```

visual editor **im**proved — открыть файл для редактирования. Пройти вводный курс по использованию этого мощного редактора можно запустив команду:

```
$ vimtutor
```

1.7. Интерпретатор `shell`

Когда система (в терминале) выдаёт приглашение `$` и вы вводите команды для выполнения, вы имеете дело не с ядром самой системы, а с неким посредником, называемым интерпретатором команд, или `shell`[7]. Это обычная программа, но она может делать удивительные вещи. Применение программы–посредника обеспечивает три главных преимущества:

– Сокращённые имена файлов: можно задать целое множество файлов в качестве аргументов команде, указав шаблон для имён: `shell` будет искать файлы, имена которых соответствуют заданному шаблону.

  ```
  $ ls *.cpp *.c
  ```

  Вывести на печать имена всех файлов в текущем каталоге, которые оканчиваются на `.cpp` или `.c`. Символ `*` в шаблоне соответствует любой последовательности символов. Поддерживаются и другие специальные символы для задания шаблона.

– Переключение ввода–вывода: вывод любой программы можно направить в файл, а не на терминал, ввод можно получать из файла, а не с терминала. Ввод и вывод можно даже передать другим программам.

  ```
  $ ls *.cpp > cppfiles.txt
  ```

  Имена всех файлов, оканчивающихся на `.cpp`, направить в файл `cppfiles.txt`. Файл будет создан, если не существует, или перезаписан, если существует.

  ```
  $ wc -l < main.cpp
  ```

  Подсчитать количество строк (опция `-l`) в файле `main.cpp`.

  ```
  $ cat main.cpp | wc -l
  ```

  То же. Символ `|` обозначает конвейер. Вывод команды слева передаётся на ввод команде справа. Можно организовывать в конвейере цепочку любой длины.

  ```
  $ wc -l main.cpp
  ```

[7] Более подробно об интерпретаторе `shell` и других возможностях системы Unix излагается, например, в книге *Керниган Б. У.*, *Пайк Р.* UNIX — универсальная среда программирования / пер. с англ. А. Берёзко, В. Иващенко. М. : Финансы и статистика, 1992. 304 с.

То же, используя лишь возможности самой утилиты `wc` (**w**ord **c**ount).

– Создание собственной среды: можно определить свои собственные команды и правила сокращений.

1.8. Система контроля версий `Git`

Основы работы в `Git` изложены в книге *Chacon S.*, *Straub B.* Pro Git : пер. с англ. Вер. 2.1.113-3. Apress, 2023. 535 с.

Это довольно обширный материал и представляет собой отдельный курс. Поэтому здесь мы рекомендуем прочитать главы «Введение» и «Основы `Git`» для первого знакомства с системой контроля версий.

Затем, полезно получить некоторой опыт работы с данной системой. И далее, перечитать эти главы, а также прочитать главу «Ветвление в `Git`». Этих возможностей вам будет вполне достаточно для успешного освоения данного курса.

Поняв суть и возможности системы контроля версий `Git`, очень просто освоить `Smartgit` — графическую обёртку. Каждая кнопка интерфейса выполняет в консоли одну или несколько команд `Git`-а. Эти команды печатаются в окне Output. Попутно в разных окнах отображаются состояние файлов, изменения от версии к версии, а также полное дерево истории коммитов. Встроенные средства позволяют не только сравнивать, принимать/отклонять, но и редактировать изменения.

Что читать

Страуструп Б.:[8] **глава 3**
Chacon S., *Straub B.*:[9] **глава «Основы `Git`»**

Упражнения

1. Напишите программу, которая по заданному n находит n-е число Фибоначчи F_n. Числа Фибоначчи определяются соотношениями:

$$F_n = F_{n-1} + F_{n-2}, \quad F_1 = F_2 = 1.$$

   Необходимо ли хранить в памяти все вычисленные числа Фибоначчи?

2. Напишите программу, которая принимает на входе число и печатает на экран его зеркальную копию, то есть, в «перевёрнутом» виде.

3. Напишите функцию, которая принимает на входе целое число и возвращает ответ, является ли оно простым. Протестируйте работу вашей функции.

4. Вычислите и сравните левую и правую части тождеств:

$$1)\ \sum_{k=0}^{\infty} \frac{1}{(2k)!} = \frac{e + e^{-1}}{2}; \qquad 2)\ \prod_{k=1}^{\infty} \frac{4k^2}{4k^2 - 1} = \frac{\pi}{2}; \qquad 3)\ e = \lim_{n\to\infty} \frac{n}{\sqrt[n]{n!}};$$

$$4)\ \frac{2}{1} \cdot \left(\frac{4}{3}\right)^{\frac{1}{2}} \cdot \left(\frac{6}{5}\frac{8}{7}\right)^{\frac{1}{4}} \cdot \left(\frac{10}{9}\frac{12}{11}\frac{14}{13}\frac{16}{15}\right)^{\frac{1}{8}} \cdot \ldots = e.$$

[8] *Страуструп Б.* Программирование: принципы и практика с использованием C++.
[9] *Chacon S.*, *Straub B.* Pro Git : пер. с англ. Вер. 2.1.113-3. Apress, 2023. 535 с.

2 Работа в IDE

В ходе студенческой лабораторной работы по физике получены данные измерений некоторой величины y, которая описывается линейной зависимостью $y = a + bx$, где a и b — какие-то константы. Полученные точки записаны в файл, следуя простому формату:

$$\begin{matrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_N & y_N \end{matrix}$$

Количество пар координат не указано и может быть произвольным. Несмотря на линейную зависимость, точки не ложатся строго на прямую из-за различного рода погрешностей измерений.

Давайте напишем программу, которая считывает данные эксперимента из файла и методом наименьших квадратов вычисляет коэффициенты a и b, а также доверительные интервалы $\pm\Delta a$ и $\pm\Delta b$ соответственно.

2.1. Метод наименьших квадратов

Постановка задачи. Рассмотрим регрессию (зависимость) следующего вида:

$$y_i = a + bx_i + \varepsilon_i, \quad i = \overline{1, N}.$$

Среди всевозможных значений $\{a, b\}$ будем искать такие, которые приводят к минимальной сумме квадратов отклонений (ошибок):

$$S_\varepsilon = \sum_{i=1}^{N} \varepsilon_i^2 = \sum_{i=1}^{N} (y_i - a - bx_i)^2 \quad \to \quad \min_{a,b}.$$

Запишем условие существования экстремума:

$$\begin{aligned} \frac{\partial}{\partial a} S_\varepsilon &= -2 \sum_{i=1}^{N} (y_i - a - bx_i) = 0, \\ \frac{\partial}{\partial b} S_\varepsilon &= -2 \sum_{i=1}^{N} (y_i - a - bx_i) x_i = 0, \end{aligned}$$

откуда получим систему линейных уравнений:

$$\begin{cases} \sum\limits_{i=1}^{N} y_i - Na - b \sum\limits_{i=1}^{N} x_i = 0, \\ \sum\limits_{i=1}^{N} x_i y_i - a \sum\limits_{i=1}^{N} x_i - b \sum\limits_{i=1}^{N} x_i^2 = 0. \end{cases}$$

Вводя обозначение для среднего арифметического множества значений некоторой величины f:

$$\bar{f} = \frac{1}{N}\sum_{i=1}^{N} f_i,$$

перепишем систему в виде:

$$\begin{cases} \bar{y} - a - b\bar{x} = 0, \\ \overline{xy} - a\bar{x} - b\overline{x^2} = 0, \end{cases}$$

и получим искомое решение:

$$\boxed{\begin{aligned} a &= \bar{y} - b\bar{x}, \\ b &= \frac{\overline{xy} - \bar{x}\bar{y}}{\overline{x^2} - \bar{x}^2}. \end{aligned}}$$

Программная реализация. Решение этой задачи может быть выражено на языке C++ следующим образом:

```cpp
// Read data of an experiment from file given as command line argument and
// having the following format:
//   x1 y1
//   ...
//   xN yN
// Compute linear regression [y = a + b*x] coefficients using the least
// squares method.

#include <cmath>
#include <fstream>
#include <iostream>
#include <iterator>
#include <stdexcept>
#include <string>
#include <tuple>
#include <vector>

struct Point
{
  double x, y;

  Point() = default;

  Point(double xx, double yy) : x{xx}, y{yy}
  { /* empty body */
  }
};

std::istream& operator>> (std::istream& is, Point& rhs)
{
  return is >> rhs.x >> rhs.y;
}

std::ostream& operator<< (std::ostream& os, const Point& rhs)
{
  return os << rhs.x << " " << rhs.y;
}
```

```
38
39 auto read (const std::string& filename)
40 {
41   std::ifstream ifs{filename};
42   if (!ifs)
43     throw std::runtime_error{"can't open file '" + filename + "'"};
44
45   return std::vector<Point>{std::istream_iterator<Point>{ifs},
46                             std::istream_iterator<Point>{}};
47 }
48
49 struct Coeff
50 {
51   double value;  // coefficient estimate
52   double delta;  // confidence band
53
54   Coeff(double v, double d) : value{v}, delta{d}
55   { /* empty body */
56   }
57 };
58
59 auto least_squares (const std::vector<Point>& points)
60 {
61   // compute average values
62   size_t N = points.size();
63   double x_ave = 0., x2_ave = 0.;
64   double y_ave = 0., xy_ave = 0.;
65
66   for (const auto& p : points)
67   {
68     x_ave += p.x;
69     x2_ave += p.x * p.x;
70     y_ave += p.y;
71     xy_ave += p.x * p.y;
72   }
73   x_ave /= N;
74   x2_ave /= N;
75   y_ave /= N;
76   xy_ave /= N;
77
78   // compute linear coefficient estimate
79   double b = (xy_ave - x_ave * y_ave) / (x2_ave - x_ave * x_ave);
80
81   if (!std::isfinite(b))
82     throw std::overflow_error{"division by zero"};
83
84   // compute constant coefficient estimate
85   double a = y_ave - b * x_ave;
86
87   return std::make_tuple(Coeff{a, 0.}, Coeff{b, 0.});
88 }
89
90 int main (int argc, char* argv[])
91 {
```

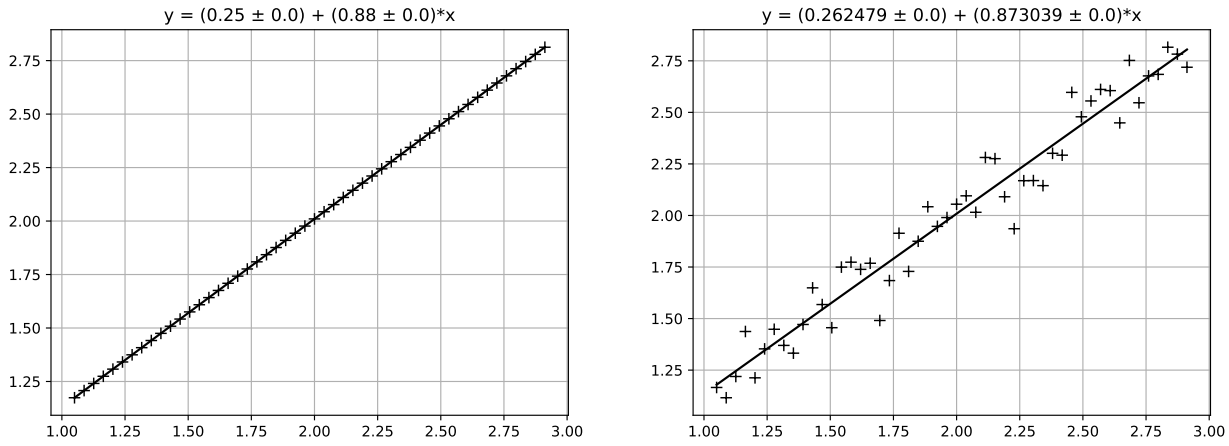
```
92  if (argc != 2)
93  {
94    std::cerr << "usage: " << argv[0] << "  file_with_data" << std::endl;
95    return 2;
96  }
97
98  try
99  {
100   std::string datafile{argv[1]};
101
102   auto [a, b] = least_squares(read(datafile));  // C++17
103
104   std::cout << datafile << "  " << a.value << " " << a.delta << "  "
105             << b.value << " " << b.delta << std::endl;
106 }
107 catch (std::exception& e)
108 {
109   std::cerr << "error: " << e.what() << std::endl;
110   return 1;
111 }
112 }
```

Сборка. Чтобы собрать исполняемый файл из командной среды, или «консоли», необходимо перейти в каталог с вашими проектами:

```
$ cd /path/to/your/projects
```

и вызвать GCC:

```
$ g++ -o bin/lsm -std=c++17 -pedantic -Wall -Wextra 02/least_squares.cpp
```



а) точки лежат строго на прямой б) точки не лежат на прямой

Рис. 2.1 — Линейная аппроксимация методом наименьших квадратов

Тестирование. Полученная программа невелика, и, тем не менее, даже она может содержать приличное количество ошибок. Поэтому её работоспособность необходимо как следует проверить, запустив несколько разнообразных тестов. Приведём примеры:

– забыли указать имя файла:

```
$ bin/lsm
usage: bin/lsm  file_with_data
```

– файл не существует:

```
$ bin/lsm  not_exist.txt
error: can't open file 'not_exist.txt'
```

– файл пуст:

```
$ bin/lsm  02/empty.txt
error: division by zero
```

– идеальный случай (рис. 2.1а), когда точки лежат строго на прямой:

```
$ bin/lsm  02/line_exact.txt
line_exact.txt  0.25 0  0.88 0
```

– реальный случай (рис. 2.1б), когда точки лежат вдоль прямой с некоторым разбросом:

```
$ bin/lsm  02/line_approx.txt
line_approx.txt  0.262479 0  0.873039 0
```

Что читать

Страуструп Б.:[1] **глава 4**

Упражнения

1. Доработайте программу, приведённую на странице 9. В функции `least_squares` необходимо дополнительно вычислить доверительные интервалы.
   Обязательно протестируйте вашу программу. Воспользуйтесь файлами, упомянутыми в подразделе на странице 11.

2.★ Продолжая предыдущее упражнение, напишите на *любом* известном вам языке программу, которая рисует график, отражающий результаты лабораторной работы. На вход подаётся исходный файл с точками и оценки коэффициентов линейной регрессии.
   Организуйте взаимодействие с программой из предыдущего упражнения так, чтобы можно было автоматически передавать входные данные. На выходе необходимо получить изображение на экране или в файле одного из популярных форматов, например `PNG`, с графиком, на котором маркерами нанесены экспериментальные точки, и сплошной линией — аппроксимирующая прямая.
   Например, рисунок 2.1 получен в результате выполнения этого упражнения с использованием языка `Python`, пакета `Matplotlib` и командной среды (см. страницу 15).

[1] *Страуструп Б.* Указ. соч.

Что дальше

Станете ли вы профессиональным программистом или экспертом по языку C++, прочитав эту книгу?[2] Конечно, нет! Настоящее программирование — это тонкое, глубокое и очень сложное искусство, требующее знаний и технических навыков. Рассчитывать на то, что за четыре месяца вы станете экспертом по программированию, можно с таким же успехом, как и на то, что за полгода или даже год вы полностью изучите биологию, математику или иностранный язык (например, китайский, английский или датский) или научитесь играть на виолончели. Если подходить к изучению книги серьёзно, то можно ожидать, что вы сможете писать простые полезные программы, читать более сложные программы и получите хорошие теоретическую и практическую основы для дальнейшей работы.

Прослушав этот курс, лучше всего поработать над реальным проектом. Ещё лучше параллельно с работой над реальным проектом приступить к чтению какой-нибудь книги профессионального уровня (например, *Stroustrup B.*[3]), более специализированной книги, связанной с вашим проектом, например, документации по библиотеке `Qt` для разработки графического пользовательского интерфейса (GUI) или справочника по библиотеке `ACE` для параллельного программирования, или учебника, посвященного конкретному аспекту языка C++, например *Кёниг Э.*, *Му Б. Э.*, *Саттер Г.* и *Гамма Э.* [и др.].[4]

В конечном итоге вам придётся приступить к изучению некоторого другого языка программирования. Невозможно стать профессионалом в области программного обеспечения (даже если программирование не является вашей основной специальностью), зная только один язык программирования.

[2] Там же. Текст взят из подраздела 0.1.3.

[3] *Stroustrup B.* The C++ programming language. Addison-Wesley, 2013. 1366 p.

[4] *Кёниг Э.*, *Му Б. Э.* Эффективное программирование на C++. М. : Издательский дом «Вильямс», 2002. 384 с. ; *Саттер Г.* Решение сложных задач на C++. М. : Издательский дом «Вильямс», 2008. 400 с. ; Приёмы объектно-ориентированного проектирования. Паттерны проектирования.Пер. с англ. / Э. Гамма [и др.]. Спб. : Питер, 2004. 366 с.

Список литературы

1. *Страуструп Б.* Программирование: принципы и практика с использованием C++ / пер. с англ. И. Красикова. — 2-е изд. — М. : ООО «И.Д. Вильямс», 2016. — 1328 с.
2. *Керниган Б. У.*, *Пайк Р.* Практика программирования / пер. с англ. В. Бродового. — М. : Издательский дом «Вильямс», 2004. — 288 с. — (Программирование для профессионалов).
3. *Мэйерс С.* Эффективное использование C++. 55 верных способов улучшить структуру и код ваших программ : пер. с англ. — 3-е изд. — М. : ДМК Пресс, 2006. — 300 с. — (Профессиональная серия от Addison–Wesley).
4. *Мейерс С.* Наиболее эффективное использование C++. 35 новых рекомендаций по улучшению ваших программ и проектов : пер. с англ. — М. : ДМК Пресс, 2000. — 304 с. — (Для программистов).
5. *Мейерс С.* Эффективное использование STL : пер. с англ. — СПб. : Питер, 2002. — 224 с. — (Библиотека программиста).
6. *Мейерс С.* Эффективный и современный C++: 42 рекомендации по исполыованию C++11 и C++14 : пер. с англ. — ООО «И.Д. Вильямс», 2016. — 304 с.
7. *Джосаттис Н. М.* Стандартная библиотека C++: справочное руководство : пер. с англ. — 2-е изд. — М. : ООО «И.Д. Вильямс», 2014. — 1136 с.
8. *Страуструп Б.* Дизайн и эволюция C++ / пер. с англ. И. Красикова. — 2-е изд. — М. : ДМК Пресс, 2006. — 448 с.
9. *Stroustrup B.* The C++ programming language. — 4th ed. — Addison-Wesley, 2013. — 1366 p.
10. *Страуструп Б.* Язык программирования C++. Краткий курс : пер. с англ. — 2-е изд. — СПб. : ООО «Диалектика», 2019. — 320 с.
11. *Керниган Б. У.*, *Пайк Р.* UNIX — универсальная среда программирования / пер. с англ. А. Берёзко, В. Иващенко. — М. : Финансы и статистика, 1992. — 304 с.
12. *Chacon S.*, *Straub B.* Pro Git : пер. с англ. — Вер. 2.1.113-3. — Apress, 2023. — 535 с.
13. *Кёниг Э.*, *Му Б. Э.* Эффективное программирование на C++ / пер. с англ. Н. Ручко. — М. : Издательский дом «Вильямс», 2002. — 384 с. — (Серия C++ In-Depth).
14. *Саттер Г.* Решение сложных задач на C++ / пер. с англ. И. Красикова. — М. : Издательский дом «Вильямс», 2008. — 400 с. — (Серия C++ In-Depth).
15. Приёмы объектно-ориентированного проектирования. Паттерны проектирования.Пер. с англ. / Э. Гамма [и др.]. — Спб. : Питер, 2004. — 366 с.

Приложение

Рисование графиков в Python

Каждый язык имеет свои достоинства и недостатки и, как правило, нацелен на эффективное решение определённого класса задач. Совместное использование разных языков часто помогает сократить усилия при разработке программ и повысить гибкость либо удобство создаваемых инструментов. Попробуем продемонстрировать это на примере визуализации результатов обработки данных лабораторной работы по физике, которые можно получить методом наименьших квадратов, рассмотренным в главе 2 (также см. упр. 2 на странице 12).

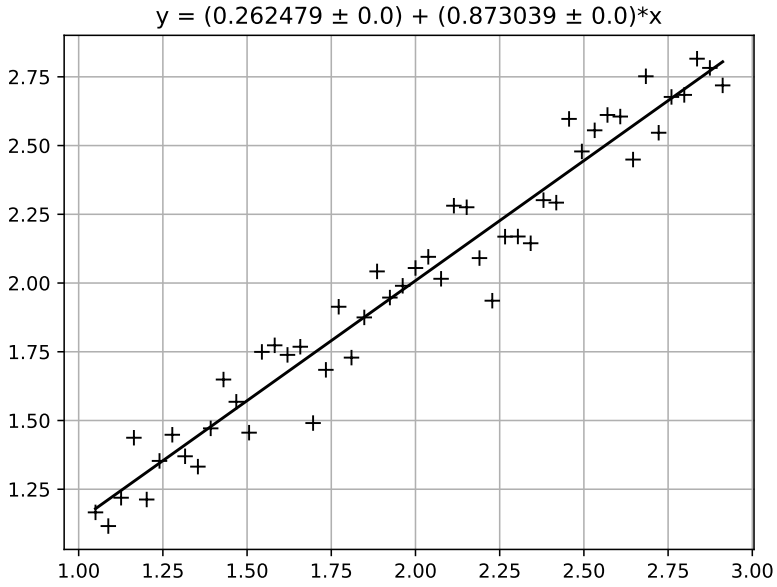


Рис. 1 — Визуализация при помощи `Matplotlib`

В языке C++ нет встроенных графических средств. Для этого необходимо использовать сторонние библиотеки. Работа с подобными инструментами не всегда настолько проста, как хотелось бы. А настройка внешнего вида координатных осей, изображаемых кривых и точек может потребовать перекомпиляции всей программы.

Одним из действительно удобных для этого случая решений является написание сценария (script) на интерпретируемом языке. Python относится к этому ряду языков и имеет мощную поддержку разнообразных средств практически прямо «из коробки». Ниже приведён вариант решения нашей задачи с использованием пакета Matplotlib.

```
#!/usr/bin/python3

"""Draw a curve from the experimental data.

Use experimental points and a computed linear regression coefficients
to make a plot. Save it in a PDF file.
```

```
"""

import argparse
import matplotlib.pyplot as plt
import pathlib


def read_data(infile):
    """Read an experimental data.

    Expected `infile` format is:
        x1 y1
        ...
        xN yN
    """
    with infile.open() as f:
        data = [[float(item) for item in line.split(maxsplit=1)]
                for line in f.readlines() if not line.isspace()]
        xx, yy = zip(*data)
        return xx, yy
    raise RuntimeError(f"Failed to read data from file ('{infile}')")


def compute_curve(xx, args):
    """Compute curve points using linear regression model."""
    def y(x):
        return args.a + args.b * x

    xe = [xx[0], xx[-1]]
    ye = [y(x) for x in xe]
    return xe, ye


def make_plot(points, curve, args):
    """Compose a figure."""
    plt.plot(*curve, "k")
    plt.plot(*points, "k+", markersize=8)
    plt.grid(True, which="both")
    plt.title(f"y = ({args.a} ± {args.da}) + ({args.b} ± {args.db})*x")


if __name__ == "__main__":
    parser = argparse.ArgumentParser(
        description="Make a plot of the linear regression y=a+b*x"
                    "built from the data of an experiment")

    parser.add_argument("file", type=pathlib.Path,
                        help="file with experiment data")

    parser.add_argument("a", type=float,
                        help="estimate of the constant coefficient")
    parser.add_argument("da", type=float,
                        help="confidence band for `a`")

```

```
61      parser.add_argument("b", type=float,
62                          help="estimate of the linear coefficient")
63      parser.add_argument("db", type=float,
64                          help="confidence band for `b`")
65
66      args = parser.parse_args()
67
68      xx, yy = read_data(args.file)
69      xe, ye = compute_curve(xx, args)
70      make_plot((xx, yy), (xe, ye), args)
71
72      plt.savefig(args.file.with_suffix(".pdf").name, bbox_inches="tight")
```

Совместить использование разработанных нами отдельных инструментов можно при помощи командной среды. Для этого достаточно всего одной строки (см. страницу 6):

```
$ bin/lsm 02/line_approx.txt | xargs python3 02/plot.py
```

Результат в виде графического файла `02/line_approx.pdf`, изображён на рисунке 1.