Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _____Информатика и системы управления_____

КАФЕДРА _____Программное обеспечение ЭВМ и информационные технологии_____

ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

Студент_____Мирзоян Сергей Азатович_____
фамилия, имя, отчество

Группа ИУ7-45Б

Тип практики_____Стационарная_____

Название предприятия_____АО НИИ «Точных приборов»_____

Студент _____ _____
подпись, дата *фамилия, и.о.*

Руководитель практики _____ _____
подпись, дата *фамилия, и.о.*

Оценка _____

2019 г.

Оглавление

Введение

В ходе производственной практики, проходившей с 1 июля по 21 июля в АО «Научный исследовательский институт Точных Приборов» (далее НИИ ТП), я приобрел навыки командной работы в больших и сложных проектах, закрепил полученные в ходе обучения в университете навыки в области объектно-ориентированного программирования, вычислительных алгоритмов, программирования на языке C/C++, тестирования, а также работы в системе контроля версий GIT (в качестве среды использовался GitLab).

Основная часть

Индивидуальное задание

В рамках практики требовалось реализовать интерфейс класса для поиска минимума функции, интерфейс класса для вычитания изображений из заданного изображения, а также тестовые функции к каждой из задач.

Выполнение задания по поиску минимума функции

Прежде чем выполнять задачу по поиску минимума функции, в первую очередь нужно определится с методом, с помощью которого это производится. Вот некоторые из них:

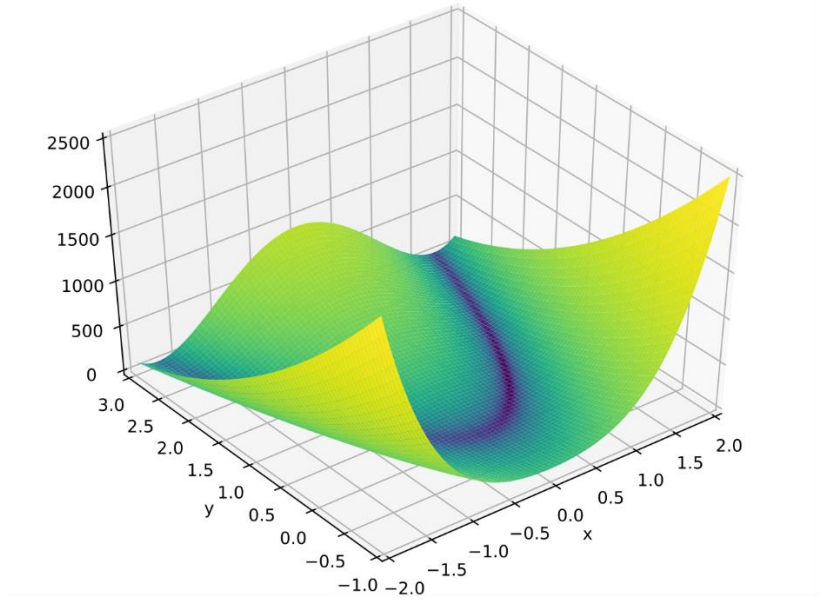
- Метод Ньютона
- Метод Наискорейшего Спуска
- Метод Левенберга-Марквардта

Выбор функции

В математической оптимизации есть функции, на которых тестируются новые методы. Одна из таких функция — функция Розенброка. В случае функции двух переменных она определяется как:

$$f(x_1, x_2) = (a - x_1)^2 + b * (x_2 - x_1^2)^2$$

где a и b – некоторые константы (при решении задачи в качестве значений для a и b были взяты числа 1 и 100 соответственно). Ее особенность заключается в поведении функции в области минимума (она имеет минимум 0 в точке (1, 1)).



Метод Левенберга-Марквардта

Поскольку предполагалось использовать только якобиан (матрица первых производных по всем переменным и всех функций системы) и система включала только одну функцию, я выбрал метод Левенберга-Марквардта.

Если коротко, суть алгоритма в том, чтобы найти такое значение вектора параметров $\boldsymbol{w}$, которое бы доставляло локальный минимум функции ошибки. Перед началом работы задается некоторое приближение, в последующих вычислениях определяется направление поиска. После каждой итерации вектор x заменяется на вектор x + Δx. $\Delta x = (J^T * J + \lambda * I)^{-1} * J^T * (y - f(x))$, где J – якобиан функции. x - вектор параметров (переменных). λ ≥ – параметр регуляризации. I – единичная матрица.

Алгоритм останавливается, если:

- $\Delta x_i < \varepsilon$, где ε – задаваемое значение
- f(x) < E, где E – задаваемое значение

```
class sys_Rosenbrock: public nsolve_systemND
{
public:
        sys_Rosenbrock() {}
        virtual ~sys_Rosenbrock() {}

        virtual void compute(const t_coord* X, t_coord* Y) const override
        {
                t_coord a = 1;
                t_coord b = 100;

                Y[0] = __sqr(a - X[0]) + b * __sqr(X[1] - __sqr(X[0]));
        }

        virtual void partials(const t_coord*, const t_coord* X, t_coord* P) const override
        {
                P[0] = 2 * (X[0] - 1) + 400 * X[0] * (__sqr(X[0]) - X[1]);
                P[1] = 200 * (X[1] - __sqr(X[0]));
        }

        virtual void compute_partials(const t_coord*, const t_coord* X, t_coord* Y, t_coord* P) const
override
        {
                t_coord a = 1;
                t_coord b = 100;

                Y[0] = __sqr(a - X[0]) + b * __sqr(X[1] - __sqr(X[0]));

                P[0] = 2 * (X[0] - 1) + 400 * X[0] * (__sqr(X[0]) - X[1]);
                P[1] = 200 * (X[1] - __sqr(X[0]));
        }

        virtual lia::e_matrix_format partials_format() const override
        {
                return lia::emf_dense;
        }
```

```
        virtual lia::e_matrix_format partials_format() const override
        {
                return lia::emf_dense;
        }

        virtual int var_count() const override
        {
                return 2;
        }

        virtual int eq_count() const override
        {
                return 1;
        }
};

template<class N>
int tmpl_nsolve_Rosenbrock(N NSOLVE, const nsolve_systemND& sys)
{
        t_coord                                         eps = 10e-6;
        nsolve_conv                                     sys_conv(100000, 0, 10e-12);
        lia::t_dvector                          x;
        lia::t_dvector                          step;
        int                                                     res;

        x.resize(sys.var_count());
        step.resize(sys.var_count());

        if(x.is_null())
        {
                return -1;
        }

        if(step.is_null())
        {
                return -1;
        }
```

```
nsolve_conv                                    sys_conv(100000, 0, 10e-12);
lia::t_dvector                          x;
lia::t_dvector                          step;
int                                                  res;

x.resize(sys.var_count());
step.resize(sys.var_count());

if(x.is_null())
{
        return -1;
}

if(step.is_null())
{
        return -1;
}

x.fill(0);
step.fill(1e-10);
x[0] = 8;
x[1] = 0;

res = NSOLVE(sys, x.get_data(), step.get_data(), sys_conv);

if(res != 0)
{
        return -1;
}

if(__eq(x[0], 1.0, eps) && __eq(x[1], 1.0, eps))
{
        return 0;
}

return -1;
```

Выполнение задания по вычитанию изображений

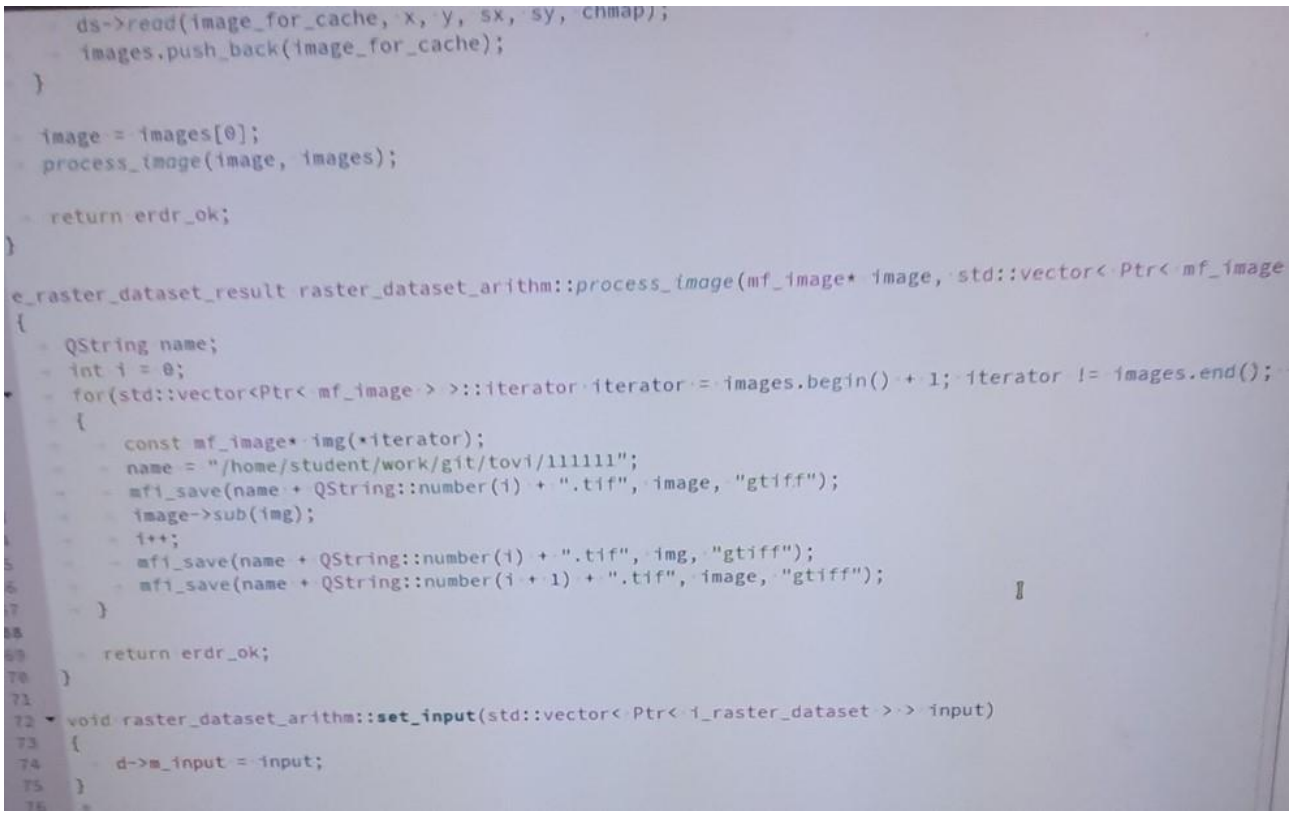
Вычитание изображения из изображения предполагает собой вычитание из каждого значения пикселя изображения 1 соответствующего значения пикселя изображения 2. На вход подается вектор объектов, содержащих считанные параметры изображений, на выход мы получаем один объект, являющийся разностью изображения 1 и всех остальных изображений.

Существует два вида разности изображений: абсолютная (если получаемое значение отрицательное, берется его модуль) и относительная (если значение меньше нуля, то оно приравнивается нулю). В рамках задачи использовалась относительная разность.

В ходе разработки получился класс, предоставляющий интерфейс для выполнения данной задачи. А именно, класс хранит вектор объектов вида *std::vector < класс_изображения*> images*, а также предоставляет метод для определения области вычитания – получает на вход область памяти, которая должна хранить результат, и параметры определяемой области (длина, ширина и смещение относительно края оригинального изображения) - а также метод для непосредственного вычитания – получает на вход вычитаемое-изображение, массив вычитателей-изображений, на выходе выдает новый объект в виде изображения (частное операции вычитания).

```
#include "image_io.h"
#include "qdebug.h"
#include "raster_dataset_arithm.h"

struct raster_dataset_arithm::data
{
    std::vector< Ptr <i_raster_dataset> > m_input;        //!< List of inputed images
};

raster_dataset_arithm::raster_dataset_arithm() : d(new data())
{

}

raster_dataset_arithm::~raster_dataset_arithm()
{
    delete d;
}

e_raster_dataset_result raster_dataset_arithm::read(mf_image* image, size_t x, size_t y, size_t sx, size_t
                                                    const size_t* chmap)
{
    if(!image)
    {
        qDebug() << UNIQUE_SOURCE_TEXT_MARKER << "image == NULL";
        return erdr_fail;
    }

    if(d->m_input.size() < 2)
    {
        qDebug() << UNIQUE_SOURCE_TEXT_MARKER << "Needs two or more images";
        return erdr_fail;
    }
```

```
    qDebug() << UNIQUE_SOURCE_TEXT_MARKER << "Needs two or more images";
    return erdr_fail;
}

image->fill(0);
std::vector< Ptr< mf_image > > images;

for(std::vector< Ptr< i_raster_dataset > >::iterator iterator = d->m_input.begin(); iterator != d->m_
    ++iterator)
{
    const i_raster_dataset* ds(*iterator);
    Ptr< mf_image >    image_for_cache(mf_image::create(sx, sy, ds->get_channel_count(),
                                                        ds->get_data_type()));
    ds->read(image_for_cache, x, y, sx, sy, chmap);
    images.push_back(image_for_cache);
}

image = images[0];
process_image(image, images);

return erdr_ok;
}

e_raster_dataset_result raster_dataset_arithm::process_image(mf_image* image, std::vector< Ptr< mf_image >
{
    QString name;
    int i = 0;
    for(std::vector<Ptr< mf_image > >::iterator iterator = images.begin() + 1; iterator != images.end(); +
    {
        const mf_image* img(*iterator);
        name = "/home/student/work/git/tovi/111111";
        mfi_save(name + QString::number(i) + ".tif", image, "gtiff");
        image->sub(img);
        i++;
```



```
    ds->read(image_for_cache, x, y, sx, sy, chmap);
    images.push_back(image_for_cache);
  }

  image = images[0];
  process_image(image, images);

  return erdr_ok;
}

e_raster_dataset_result raster_dataset_arithm::process_image(mf_image* image, std::vector< Ptr< mf_image
{
  QString name;
  int i = 0;
  for(std::vector<Ptr< mf_image > >::iterator iterator = images.begin() + 1; iterator != images.end();
  {
    const mf_image* img(*iterator);
    name = "/home/student/work/git/tovi/111111";
    mfi_save(name + QString::number(i) + ".tif", image, "gtiff");
    image->sub(img);
    i++;
    mfi_save(name + QString::number(i) + ".tif", img, "gtiff");
    mfi_save(name + QString::number(i + 1) + ".tif", image, "gtiff");
  }

  return erdr_ok;
}

void raster_dataset_arithm::set_input(std::vector< Ptr< i_raster_dataset > > input)
{
  d->m_input = input;
}
```

Заключение

Производственная практика показала мне внутреннее устройство IT-проекта, специфики работы в команде. В ходе обучения я узнал новые методы вычисления минимума функций, автоматическое тестирование проекта в GitLab, развил навыки тестирования, участвовал в действительном рабочем проекте и многое другое. Большую трудность составляло изучение внутренних библиотек проекта, что показало важность работы с документацией.

Приложение

График прибытия и отбытия.

2.07.2019: 10:00 - 15:40

4.07.2019: 11:20 - 17:20

5.07.2019: 11:25 - 18:10

8.07.2019: 11:30 - 18:10

9.07.2019: 11:30 - 17:40

10.07.2019: 11:45 - 17:00

11.07.2019: 11:25 - 18:30

12.07.2019: 11:30 - 18:20

15.07.2019: 11:40 - 19:35

16.07.2019: 11:42 – 17:30

Список использованных источников

Литература:

1. Rosenbrock, H. H. (1960): An automatic method for finding the greatest or least value of a function
2. K. Madsen, H.B. Nielsen, O. Tingleff (2004): Methods for non-linear least square
3. Florent Brunet(2011): Basics on Continuous Optimization

Веб-источники:

1. Алгоритм Левенберга — Марквардта для нелинейного метода наименьших квадратов и его реализация на Python - habr.com/ru/post/308626/
2. Функция Розенброка - en.wikipedia.org/wiki/Rosenbrock_function

Подись научного руководителя