C++ UI design for NAG Optimization Modelling Suite

Group 6

Konstantin Korkin, Maksim Feldman, Tran Man Khang, Yifei Huang, Ziya Valiyev[1]

[1]Informatik 12: Software and Tools for Computational Engineering, RWTH Aachen University, `info@stce.rwth-aachen.de`

Contents

Preface

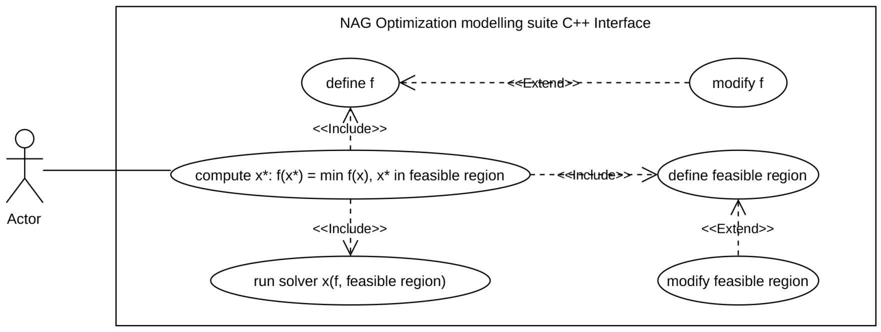
- administrative information about the project (e.g, topic issued by which institute)
- fit of topic into study program (e.g, sufficient prior knowledge)
- acknowledgement of supervision

Chapter 1

Analysis

1.1 User Requirements

This C++ interface ought to be made to to define and solve optimization problems with the help of the NAG Optimization Modelling Suite (NOMS).
The users would like to expect the software to firstly define the objective function and the constraints they give to the interface and then solve the optimization problem by itself without having the users to put in any code or derivatives of the function. Namely the software should be able to choose the appropriate solver for a specific optimization problem.


NAG Optimization modelling suite C++ Interface
define f
<<Extend>>
modify f
<<Include>>
Actor
compute x*: f(x*) = min f(x), x* in feasible region
<<Include>>
define feasible region
<<Include>>
<<Extend>>
run solver x(f, feasible region)
modify feasible region

The use case diagram above shows how the interface is supposed to process such a task of solving an optimization problem. To find an extreme of the function f, it defines the function and the feasible region which could be modified, then it chooses a certain solver to compute the extreme x^*. The user don't have to identify what kind the problem is (linear, non-linear) nor choose the right solver themselves to solve it.

1.2 Essential Technical Background

includes references into literature, e.g, [**?**]

1.3 System Requirements

Functional:

- defining optimization problems, including:
  - the ability to define and modify objective functions
- automatic computation of derivatives
- recognisation of linear contraints
  - automatic modification of problem to reflect linearity
- solve problems with either automatic solver choice or manual choice from the 3 solvers: e04mt, e04kf, e04st

Nonfunctional:

- backend: NAG Optimization Modelling Suite, C++ interface
- interface implementation in C++ using OOP
- testing with CTest and googletest
- documentation with doxygen
- automatic computation of derivatives and recognisation of linear contraints with dco/c++

Chapter 2

Design

2.1 Class Candidates

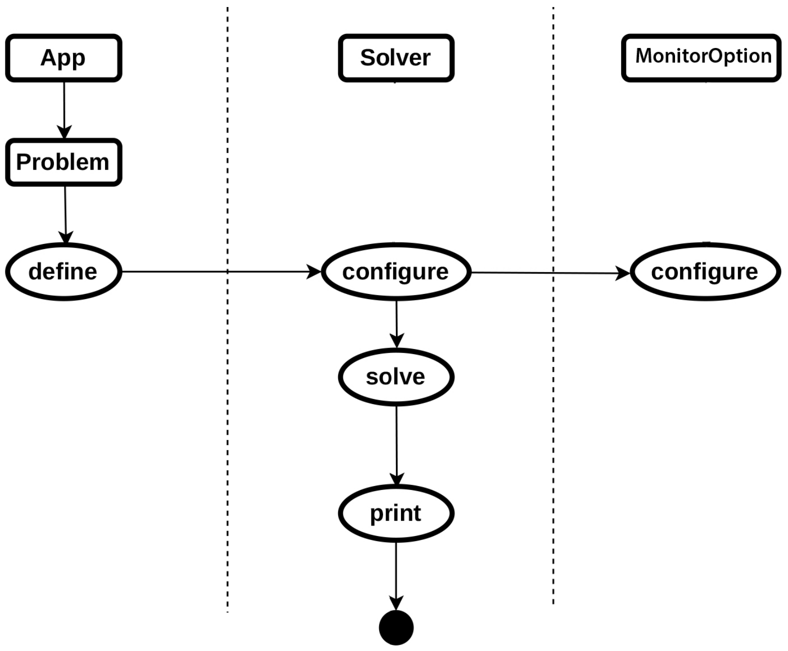
includes libraries that you built on explained briefly and references to further information
Some keywords/classes from the existing C++ interface in the NAG optimization modelling suite (NOMS)[1] which could be used as classes in ours:

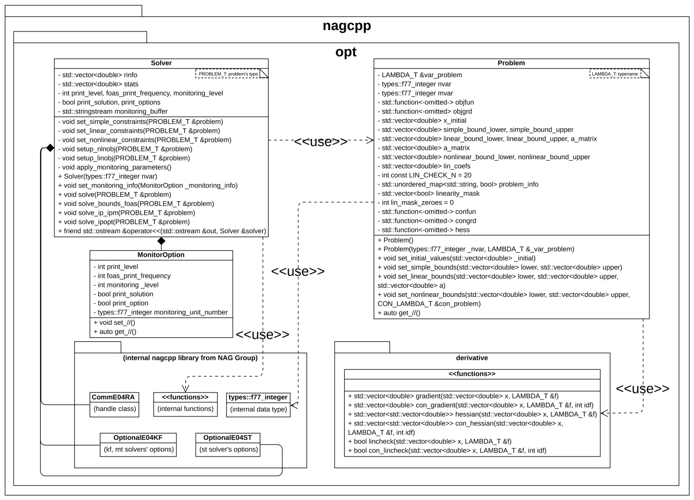
- problem
- solver
- dco/derivative
- bound
- function
  - objectiveFunction
  - constraintFunction
- linearInfo

2.2 Class Model

UML Class diagram(s) and description; should link into overall design through reference of application programming interfaces (API) of third-party software

[1]`https://www.nag.com/numeric/nl/nagdoc_latest/flhtml/e04/e04intro.html#optsuite`


App
Solver
MonitorOption
Problem
define
configure
configure
solve
print


nagcpp
opt
Solver
PROBLEM_T: problem's type
- std::vector<double> rinfo
- std::vector<double> stats
- int print_level, foas_print_frequency, monitoring_level
- bool print_solution, print_options
- std::stringstream monitoring_buffer
- void set_simple_constraints(PROBLEM_T &problem)
- void set_linear_constraints(PROBLEM_T &problem)
- void set_nonlinear_constraints(PROBLEM_T &problem)
- void setup_nlnobj(PROBLEM_T &problem)
- void setup_linobj(PROBLEM_T &problem)
- void apply_monitoring_parameters()
+ Solver(types::f77_integer nvar)
+ void set_monitoring_info(MonitorOption _monitoring_info)
+ void solve(PROBLEM_T &problem)
+ void solve_bounds_foas(PROBLEM_T &problem)
+ void solve_ip_ipm(PROBLEM_T &problem)
+ void solve_ipopt(PROBLEM_T &problem)
+ friend std::ostream &operator<<(std::ostream &out, Solver &solver)
<<use>>
Problem
LAMBDA_T: typename
- LAMBDA_T &var_problem
- types::f77_integer nvar
- types::f77_integer mvar
- std::function<-omitted-> objfun
- std::function<-omitted> objgrd
- std::vector<double> x_initial
- std::vector<double> simple_bound_lower, simple_bound_upper
- std::vector<double> linear_bound_lower, linear_bound_upper, a_matrix
- std::vector<double> a_matrix
- std::vector<double> nonlinear_bound_lower, nonlinear_bound_upper
- std::vector<double> lin_coefs
- int const LIN_CHECK_N = 20
- std::unordered_map<std::string, bool> problem_info
- std::vector<bool> linearity_mask
- int lin_mask_zeroes = 0
- std::function<-omitted-> confun
- std::function<-omitted-> congrd
- std::function<-omitted-> hess
+ Problem()
+ Problem(types::f77_integer _nvar, LAMBDA_T &_var_problem)
+ void set_initial_values(std::vector<double> _initial)
+ void set_simple_bounds(std::vector<double> lower, std::vector<double> upper)
+ void set_linear_bounds(std::vector<double> lower, std::vector<double> upper, std::vector<double> a)
+ void set_nonlinear_bounds(std::vector<double> lower, std::vector<double> upper, CON_LAMBDA_T &con_problem)
+ auto get_//()
MonitorOption
- int print_level
- int foas_print_frequency
- int monitoring _level
- bool print_solution
- bool print_option
- types::f77_integer monitoring_unit_number
+ void set_//()
+ auto get_//()
<<use>>
<<use>>
(internal nagcpp library from NAG Group)
CommE04RA
(handle class)
<<functions>>
(internal functions)
types::f77_integer
(internal data type)
OptionalE04KF
(kf, mt solvers' options)
OptionalE04ST
(st solver's options)
derivative
<<functions>>
+ std::vector<double> gradient(std::vector<double> x, LAMBDA_T &f)
+ std::vector<double> con_gradient(std::vector<double> x, LAMBDA_T &f, int idf)
+ std::vector<std::vector<double>> hessian(std::vector<double> x, LAMBDA_T &f)
+ std::vector<std::vector<double>> con_hessian(std::vector<double> x, LAMBDA_T &f, int idf)
+ bool lincheck(std::vector<double> x, LAMBDA_T &f)
+ bool con_lincheck(std::vector<double> x, LAMBDA_T &f, int idf)
<<use>>

Chapter 3

Implementation

3.1 Development Infrastructure

- Target platform: Linux devices, specifically the Virtual Machine provided by Prof. Naumann
- Programming language: C++ 20
- Compiler: gcc
- Build system: CMake
- Runtime library: NAG Library with C++ interface, dco/c++, the standard ones (std, math.h, etc.)

3.2 Source Code

overview of source code structure (file names, directories); build instructions; references into source code documentation e.g, doxygen[1]; short (!) code listings

```
#include<iostream>
int main() {
  std::cout << "Leave me alone world!" << std::endl;
  return 42;
}
```

only if helpful (must come with detailed explanation)

3.3 Software Tests

e.g, googletest[2]

[1]https://github.com/doxygen/doxygen

[2]https://github.com/google/googletest

Chapter 4

Project Management

Our project workflow was devided into five phases in a time span of 6 months: analysis, design and implementation, testing, documentation and final report writing.

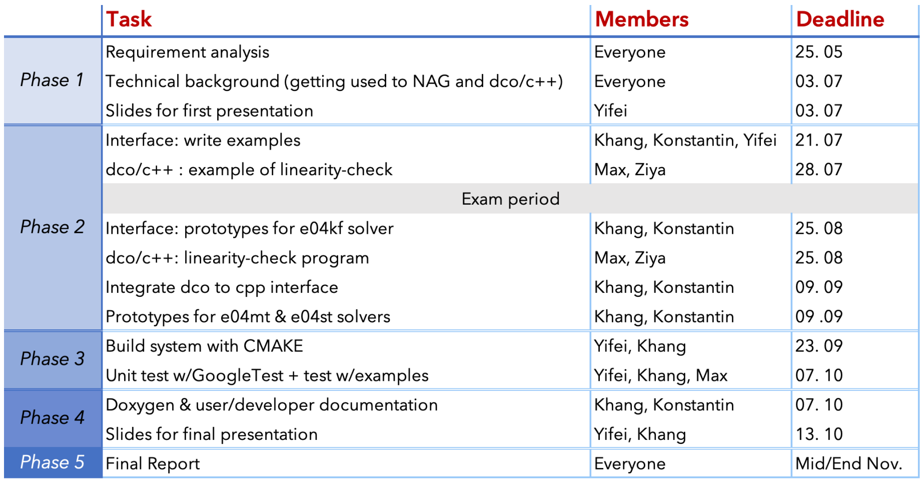
At the start of this project, we discussed and worked together for the requirement analysis and technical background so that everyone could have the basic understanding of the project assignment and also get to know the NAG library for later development. At the beginning of July, we then presented the requirement analysis of our project and came up with preliminary division of work of the rest phases.

In the design and implementation phase, Khang and Konstantin took charge of the main C++ interface (prototypes for the three solvers, etc.) and implemented the code in C++20 whilst Ziya was responsible for applying dco/c++ for automatic differentiation and linearity check. Despite the one-month-long exam period for all members, we still managed to stick to our plan.

After the source code was implemented and runned successfully with all the expecting features, Yifei constructed a build system based on CMake[1] and Khang optimized the system to make it aesthetically well-structured. Meanwhile, Yifei and Khang looked into googletest[2] and CTest to create unit-tests for the interface, together with Valgrind to ensure it was free of memory leaks and invalid memory accesses. Max implemented three examples for each solver to assure us of the functionality of the interface.

Before the final presentation of the project, Khang and Konstantin worked on user- and developer-documentation with doxygen[3] as they were the most familiar with the source code. With Khang summarizing and arranging the content of the presentation, Yifei made the slides in LaTeX for both presentation and also the report of the project.

[1]`https://cmake.org/`
[2]`http://google.github.io/googletest/`
[3]`https://github.com/doxygen/doxygen`



	Task	Members	Deadline
Phase 1	Requirement analysis	Everyone	25. 05
	Technical background (getting used to NAG and dco/c++)	Everyone	03. 07
	Slides for first presentation	Yifei	03. 07
Phase 2	Interface: write examples	Khang, Konstantin, Yifei	21. 07
	dco/c++ : example of linearity-check	Max, Ziya	28. 07
	Exam period		
	Interface: prototypes for e04kf solver	Khang, Konstantin	25. 08
	dco/c++: linearity-check program	Max, Ziya	25. 08
	Integrate dco to cpp interface	Khang, Konstantin	09. 09
	Prototypes for e04mt & e04st solvers	Khang, Konstantin	09 .09
Phase 3	Build system with CMAKE	Yifei, Khang	23. 09
	Unit test w/GoogleTest + test w/examples	Yifei, Khang, Max	07. 10
Phase 4	Doxygen & user/developer documentation	Khang, Konstantin	07. 10
	Slides for final presentation	Yifei, Khang	13. 10
Phase 5	Final Report	Everyone	Mid/End Nov.

Figure 4.1: Project Management table

As a group, we kept our communication both internal and with our supervisor Johannes, managed the time and overcame exceptional situations such as sickness, which was precious experience in software development for all of us.

We sticked to our roles as originally planned and made sure to have both online group meetings every week to discuss and online meetings with our supervisor every 2 week to have them check on our progress so that everything was kept on track very well. To keep everything up-to-date on everyone's end, we made good use of git[4] for software version control. As a prevention of unexpected circumstances, tasks were always assigned to a pair of our group members. If one person is unavailable, there is still another to keep up, which proved much of use.

[4]https://git.rwth-aachen.de/tranmankhang1705/nag-optimization-modelling-suite-ui

Appendix A

User Documentation

A.1 Building

e.g, using cmake[1] and make[2]

A.2 Testing

e.g, `make test`

A.3 Running

documented sample session(s); e.g, `make run`

[1]`https://cmake.org/`

[2]`https://www.gnu.org/software/make/`