

Software Lab Computational Engineering Science

Group 12, Exception Handling

Aaron Floerke, Arseniy Kholod, Xinyang Song and Yanliang Zhu

Informatik 12: Software and Tools for Computational Engineering (STCE)
RWTH Aachen University

Analysis

- User Requirements
- System Requirements
- Class Model(s)

Implementation

- Development Infrastructure
- Source Code

Documentation

Software Tests

Project Management

Live Demo

Summary and Conclusion

- ▶ Extend cppNum v2.4 and v2.5 with appropriate C++ exception handling.
- ▶ Desing at least three scalable sufficiently distinct case studies.
- ▶ Compare general behavior and run times with the exception handling-free version.

- ▶ Exception handling is a programming concept used to manage errors and unusual conditions that arise during program execution. It allows for controlled responses to errors, ensuring the program can handle them gracefully without crashing. Key components include try, catch (or except), and finally blocks. (ChatGPT)

Functional:

▶ **Exception Handling:**

- ▶ An exception is thrown, if unintended input is put into the system
- ▶ An exception is thrown when the system behaves in an unintended way.
- ▶ An exception should be handled in such a way, as to prevent a potential crash of the system, if possible.
- ▶ The system must integrate C++ exception handling mechanisms in cppNum versions v2.4 and v2.5.
- ▶ The system must log all exceptions with appropriate error messages.
- ▶ A thrown exception should enhance the users ability to find bugs.

▶ **Case Studies:**

- ▶ The system must implement at least three scalable and distinct case studies to test the modified cppNum library.
- ▶ Each case study must include a specific scenario that can trigger exceptions.

▶ **Performance Comparison:**

- ▶ The system must compare the general behavior and run times of the modified cppNum versions with the original exception-free versions.
- ▶ The comparison results must be documented and include detailed performance metrics.

Nonfunctional:

▶ **Exception Structure:**

- ▶ An exception is a class object.
- ▶ All cppNum exception classes have a single parent class to provide a clear structure.
- ▶ All exception classes are inherited from `std::exception` to catch together with other exceptions, potentially generated by third-party libraries.

▶ **Exception Logic:**

▶ **Performance:**

- ▶ The system must ensure that the overhead introduced by exception handling is minimized.
- ▶ The system should not degrade the performance of cppNum versions v2.4 and v2.5.

▶ **Reliability:**

- ▶ The system must handle exceptions gracefully to prevent crashes and ensure smooth operation.
- ▶ The system must be able to recover from exceptions and continue processing if possible.

► Usability:

- The system must provide clear and informative error messages to users when exceptions occur.
- The system should document the scenarios under which exceptions are raised and how they are handled.

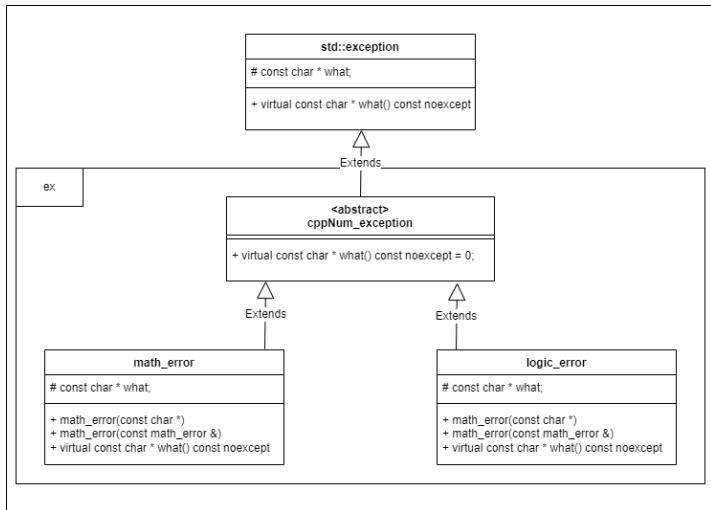
► Maintainability:

- The code implementing exception handling must be well-documented and follow coding standards.
- The system must use modular and clean code to facilitate future updates and maintenance.

► Scalability:

- The system must be able to handle large datasets and complex computations in the case studies without significant performance degradation.
- The system must be designed to easily incorporate additional case studies in the future.

- ▶ Third-Party libraries: Eigen and AD.
- ▶ Wrap with try-catch every connection point to Eigen and AD to handle possible exceptions. All exceptions that could be thrown by Eigen and AD are inherited from `std::exception`.
- ▶ Rethrow exceptions recursively to the highest level function. This provides so-called stack-trace inside `cppNum`.
- ▶ Implement exception classes inherited from `std::exception` to handle exceptions generated by `cppNum` itself.
- ▶ Check applicability of LU and LLT decompositions utilizing functions from Eigen. Generate custom exceptions in error cases.



▶ 1. Operating System:

- ▶ Xubuntu

▶ 2. Programming Language and Compiler:

- ▶ Programming Language: C++.
- ▶ Compiler: GCC.

▶ 3. Libraries:

- ▶ Eigen: A C++ library for linear algebra, providing efficient matrix and vector operations.
- ▶ AD: Provide a complete C++ solution for implementing algorithmic differentiation in numerical computations.

▶ 4. Version Control System:

- ▶ GitHub: Remote code repositories for team collaboration, code reviews, and version control. https://github.com/ArseniyKholod/stce_ss24_ex12

▶ 5. Frameworks:

- ▶ Doxygen: Used for generating project documentation, helping the team understand and maintain the code better.
- ▶ Makefile: For build management.

cppNum v2.4

exceptions

cppNum_exception.hpp

math_error.hpp

logic_error.hpp

convexObjective

objective.hpp

newton.hpp

minimizer.hpp

algebraicSystem

system.hpp

solver.hpp

newton.hpp

linearAlgebra.hpp

iteration.hpp

derivative.hpp

approximation.hpp

cppNum v2.5

exceptions

cppNum_exception.hpp

math_error.hpp

logic_error.hpp

differentialSystem

system.hpp

integrator.hpp

implicitEuler.hpp

algebraicSystem

system.hpp

solver.hpp

newton.hpp

linearAlgebra.hpp

iteration.hpp

derivative.hpp

approximation.hpp

evolution.hpp

```
#pragma once
#include <exception>
namespace ex{
    /// Abstract basic class for all cppNum exceptions
    class cppNum_exception: public std::exception{
    public:
        virtual const char* what() const noexcept =0;
    };
}
```

```
#pragma once
#include "cppNum_exception.hpp"
#include <string>
namespace ex{
    /// An exception class to handle mathematical errors
    class math_error: public cppNum_exception {
    protected:
        /// Error message
        const char* what_arg;
    public:
        /// Constructor to initialize error message
        math_error(const char*);
        /// Copy constructor
        math_error(const math_error&);
        /// Returns explanatory string
        virtual const char* what() const noexcept{
            return what_arg;
        }
    };

    math_error::math_error(const char* what_arg) : what_arg(what_arg){}
    math_error::math_error(const math_error& err){
        what_arg = err.what_arg;
    }
}
```

```
#pragma once
#include "cppNum_exception.hpp"
#include <string>

namespace ex{
    /// An exception class to handle logical errors
    class logic_error: public cppNum_exception {
    protected:
        /// Error message
        const char* what_arg;
    public:
        /// Constructor to initialize error message
        logic_error(const char*);
        /// Copy constructor
        logic_error(const logic_error&);
        /// Returns explanatory string
        virtual const char* what() const noexcept{
            return what_arg;
        }
    };

    logic_error::logic_error(const char* what_arg) : what_arg(what_arg){}
    logic_error::logic_error(const logic_error& err){
        what_arg = err.what_arg;
    }
}
```

```

...
template<typename T>
struct lu_solver_t {
    static la::vector_t<T> run(const la::matrix_t<T>& A, const la::vector_t<T>& b) {
        try{
            //matrix have to be square
            if(A.cols() != A.rows())
                throw(ex::math_error("Matrix is not square, LU decomposition is not applicable"));
            //matrix and vector must have equal number of rows
            if(A.rows() != b.rows())
                throw(ex::math_error("Matrix and rhs—vector have different number of rows, linear system is
                not uniquely solvable"));
            //matrix have to be invertible
            if(A.determinant() == 0)
                throw(ex::math_error("Matrix is singular, applying LU algorithm for solving a linear system is
                not possible."));
            return A.lu().solve(b);
        }
        catch(...){
            std::cerr<<"Exception was caught in la::lu_solver_t::run, throw it further."<<std::endl;
            throw;
        }
    };
};
...

```

```

...
template<typename T>
struct llt_solver_t {
    static la::vector_t<T> run(const la::matrix_t<T>& A, const la::vector_t<T>& b) {
        try{
            //matrix have to be squared
            if(A.cols() != A.rows())
                throw(ex::math_error("Matrix is not square, LLT decomposition is not applicable"));
            //matrix and vector must have equak number of rows
            if(A.rows() != b.rows())
                throw(ex::math_error("Matrix and rhs—vector have diffirent number of rows, linear system is
                not uniquely solvable"));
            //matrix have to be invertible
            if(A.determinant() == 0)
                throw(ex::math_error("Matrix is singular, applying LLT algorithm for solving a linear system
                is not possible."));
            //matrix have to be symmetric positive definite
            if(A.llt().info())
                throw(ex::math_error("Matrix is not symmetric positiv definite, LLT decomposition is not
                applicable."));
            return A.llt().solve(b);
        }
        catch(...){
            std::cerr<<"Exception was caught in la::llt_solver_t::run, throw it further."<<std::endl;
            throw;
        }
    }
};

```


Content of each function in cppNum/derivative.hpp was wrapped in try and followed by catch. E.g. for dFdx, for all other functions exactly in the same way.

```
...
static la::matrix_t<T> dFdx(const la::vector_t<T>& x_v, const la::vector_t<T>& p_v) {
    try{...
    }
    catch(...){
        std::cerr<<"Exception was caught in derivative_t::dFdx, throw it further."<<std::endl;
        throw;
    }
}
...
```

```
...
template<typename T, typename SYSTEM_T, typename LINEAR_SOLVER_T>
la::vector_t<T> newton_solver_t<T,SYSTEM_T,LINEAR_SOLVER_T>::run(la::vector_t<T> x,
    const la::vector_t<T> &p) {
    try{...
    }
    catch(...){
        std::cerr<<"Exception was caught in as::newton_solver_t::run, throw it further."<<std::endl;
        throw;
    }
}
...

```

```

...
la::vector_t<AS.T> newton_minimizer_t<T,LINEAR.SOLVER.T>::F(const la::vector_t<AS.T> &x, const la::vector_t<AS.T>& p) {
    try{
        return derivative_t::dfdx<objective_t,AS.T>(x,p);
    }
    catch(...){
        std::cerr<<"Exception was caught in co::newton_minimizer_t::F, throw it further." <<std::endl;
        throw;
    }
}

...
la::vector_t<T> newton_minimizer_t<T,LINEAR.SOLVER.T>::run(la::vector_t<T> x, const la::vector_t<T> &p) {
    la::vector_t<T> x_initial(x);
    try{
        ...
        return x;
    }
    catch(const std::exception & e){
        std::cerr<<"std:exception was caught in co::newton_minimizer_t::run with following message:" <<std::endl<<e.what()<<std::endl;
    }
    catch(...){
        std::cerr<<"Exception of unknown type was caught in co::newton_minimizer_t::run." <<std::endl;
    }
    std::cerr<<"co::newton_minimizer_t::run returns an initial value of x. Check the correctness of the input." <<std::endl;
    return x_initial;
}

...

```

The highest level function in v2.4 is `co::newton_minimizer_t::run`, it does not rethrow an error, it prints the message and returns an initial value to user.

```

...
template<typename T>
la::vector_t<T> implicitEuler_integrator_t<T>::run(la::vector_t<T> x, const la::vector_t<T> &p) {
    la::vector_t<T> x_initial(x);
    try{...
        return x;
    }
    catch(const std::exception & e){
        std::cerr<<"std:exception was caught in ds::implicitEuler_integrator_t::run with following message
            : "<<std::endl<<e.what()<<std::endl;
    }
    catch(...){
        std::cerr<<"Exception of unknown type was caught in ds::implicitEuler_integrator_t::run."<<std
            ::endl;
    }
    std::cerr<<"ds::implicitEuler_integrator_t::run returns an initial value of x. Check the correctness of
        the input."<<std::endl;
    return x_initial;
}
...

```

The highest level function in v2.5 is `ds::implicitEuler_integrator_t::run`, it does not rethrow an error, it prints the message and returns an initial value to user.

In all plot functions was checked, whether requested state exists. These functions were wrapped in try and followed by catch. E.g. for `evolution.t::plot` from v2.5, all other plots in the same way.

```

















template<typename T>
void evolution.t<T>::plot(const std::string& filename, int i) const {
    try{
        std::ofstream ofs(filename);
        assert(_states.size()==_times.size());
        //check if requested state exists
        if(i < 0 || i >= _states[0].rows())
            throw(ex::logic_error("State outside range is requested."));
        for (size_t k=0; k<_times.size(); ++k)
            ofs << _times[k] << ' ' << _states[k](i) << std::endl;
    }
    catch(const std::exception & e){
        std::cerr<<"std:exception was caught in evolution.t::plot with following message:"<<std::endl<<
            e.what()<<std::endl;
    }
    catch(...){
        std::cerr<<"Exception of unknown type was caught in evolution.t::plot."<<std::endl;
    }
}
    
```

Example is for v2.4, analogically for v2.5.

Class Hierarchy

[Go to the graphical class hierarchy](#)

This inheritance list is sorted roughly, but not completely, alphabetically:

 derivative_t	
▼  std::exception	
▼  ex::cppNum_exception	Abstract basic class for all cppNum exceptions
 ex::logic_error	An exception class to handle logical errors
 ex::math_error	An exception class to handle mathematical errors
▼  iteration_t< T >	
▼  approximation_t< T >	
▼  as::solver_t< T, system_t >	
 as::newton_solver_t< T, SYSTEM_T, LINEAR_SOLVER_T >	
 as::solver_t< T, SYSTEM_T >	
▼  co::minimizer_t< T >	
 co::newton_minimizer_t< T, LINEAR_SOLVER_T >	
 la::lt_solver_t< T >	
 la::lu_solver_t< T >	
 co::objective_t	
 as::system_t	

ex::cppNum_exception Class Reference

[abstract](#)

Abstract basic class for all cppNum exceptions. [More...](#)

```
#include <cppNum_exception.hpp>
```

Inheritance diagram for ex::cppNum_exception:

Collaboration diagram for ex::cppNum_exception:

Public Member Functions

```
virtual const char * what () const noexcept=0
```

Detailed Description

Abstract basic class for all cppNum exceptions.

ex::math_error Class Reference

An exception class to handle mathematical errors. [More...](#)

```
#include <math_error.hpp>
```

Inheritance diagram for ex::math_error:

Collaboration diagram for ex::math_error:

Public Member Functions

```
math_error (const char *)  
Constructor to initialize error message.
```

```
math_error (const math_error &)  
Copy constructor.
```

```
virtual const char * what () const noexcept  
Returns explanatory string.
```

Protected Attributes

```
const char * what_arg  
Error message.
```

Detailed Description

An exception class to handle mathematical errors.

ex::logic_error Class Reference

An exception class to handle logical errors. [More...](#)

```
#include <logic_error.hpp>
```

Inheritance diagram for ex::logic_error:

Collaboration diagram for ex::logic_error:

Public Member Functions

```
logic_error (const char *)  
Constructor to initialize error mess
```

```
logic_error (const logic_error &)  
Copy constructor.
```

```
virtual const char * what () const noexcept  
Returns explanatory string.
```

Protected Attributes

```
const char * what_arg  
Error message.
```

Detailed Description

An exception class to handle logical errors.

The function is: $f(x, p) = \sum_{i=0}^{n-1} (0.5 \cdot (p(i) + x(i))^2 + 0.2 \cdot (x(i) - p(i))^4)$

```
#pragma once
#include "cppNum/convexObjective/objective.hpp"
#include <cassert>
#include <cmath>

template<typename T>
T co::objective_t::f(const la::vector_t<T> &x, const la::vector_t<T> &p) {
    using namespace std;
    int n=x.size(); assert(n>=1); assert(p.size()==n);
    T y=0;
    for (int i=0;i<n;++i) y+=0.5*pow(p(i)+x(i),2) + 0.2*pow(x(i)-p(i),4);
    return y;
}
```

For time measurements:

- ▶ Initial value of x: $[-4, -4, \dots, -4]$
- ▶ Parameters p: $[2, 2, \dots, 2]$

The function is: $f(x, p) = \sum_{i=0}^{n-1} (\cosh(x(i) + p(i)))$

```
#pragma once
#include "cppNum/convexObjective/objective.hpp"
#include <cassert>
#include <cmath>

template<typename T>
T co::objective_t::f(const la::vector_t<T> &x, const la::vector_t<T> &p) {
    using namespace std;
    int n=x.size(); assert(n>=1); assert(p.size()==n);
    T y=0;
    for (int i=0;i<n;++i) y+=cosh(x(i)+p(i));
    return y;
}
```

For time measurements:

- ▶ Initial value of x: $[-4, -4, \dots, -4]$
- ▶ Parameters p: $[2, 2, \dots, 2]$

Investigate level of water in consecutive connected fountains.

Modellierung: Dreistufige Brunnenkaskade

$$\begin{aligned}\frac{dh_i}{dt}\bigg|_t &= \frac{1}{A_i} (q_{\text{ein},i}(t) - q_{\text{aus},i}(t)), h_i(t_0) = h_{i,0}, i = 1, \dots, 3 \\ q_{\text{aus},i}(t) &= s_i \sqrt{2gh_i(t)} \\ q_{\text{ein},3}(t) &= q_{\text{aus},2}(t) \\ q_{\text{ein},2}(t) &= q_{\text{aus},1}(t) \\ q_{\text{ein},1}(t) &= u(t), u(t) = \text{bekannt} \\ y(t) &= q_{\text{aus},3}(t)\end{aligned}$$

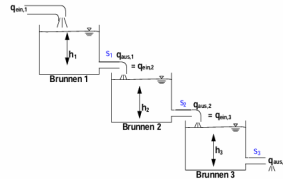


Figure: Fountain Chain

*Figure was taken from Simulationstechnik II, Prof. Alexander Mitsos, Ph.D.

ODE: fountain chain

$$r(0) = \frac{1}{p_2} (p_0 - p_3 \sqrt{2p_1 x_0})$$

$$r(i) = \frac{1}{p_{2i+2}} (p_{2i+1} \sqrt{2p_1 x_{i-1}} - p_{2i+3} \sqrt{2p_1 x_i}), \quad i = 1, 2, \dots, n-1$$

```
#pragma once
#include "cppNum/differentialSystem/system.hpp"
#include <cassert>

template<typename T>
la::vector_t<T> ds::system_t::G(const la::vector_t<T> &x, const la::vector_t<T> &p) {
    int n = x.size();
    assert(p.size() == 2*n + 2);
    assert(n >= 2);
    la::vector_t<T> r(n);
    r(0) = (1/p(2))*(p(0) - p(3)*sqrt(2*p(1)*x(0)));
    for(int i = 1; i < n; i++) {
        r(i) = (1/p(2*i + 2))*(p(2*i + 1)*sqrt(2*p(1)*x(i - 1)) - p(2*i + 3)*sqrt(2*p(1)*x(i)));
    }
    return r;
}
```

For time measurements:

- ▶ Initial value of x : $[0, 0, \dots, 0]$
- ▶ Parameters p : $[1, 9.81, 10, 1, \dots, 10, 1]$, where $p(0) = 1$ is input stream, $p(1) = 9.81$ is acceleration of free fall, $p(2i) = 10$ is area of fountain, $p(2i + 1) = 1$ is area of exit hole.

$$\frac{dx}{dt} = Ax$$

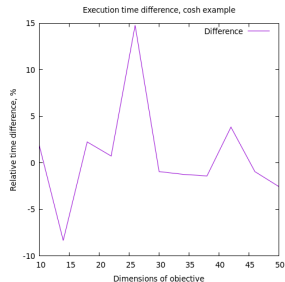
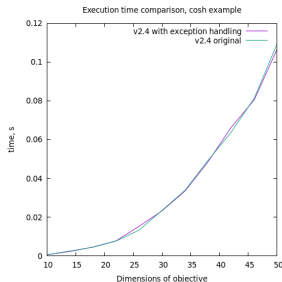
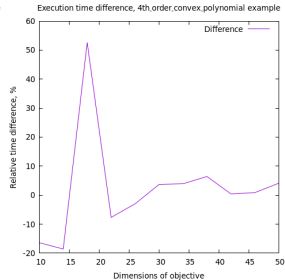
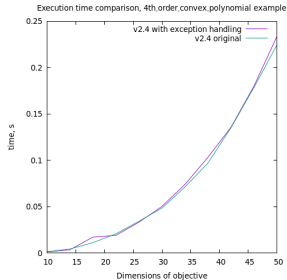
```
#pragma once
#include "cppNum/differentialSystem/system.hpp"
#include <cassert>

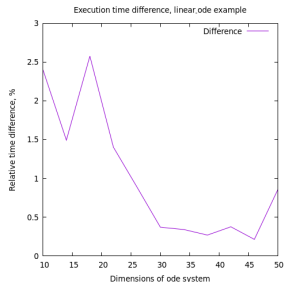
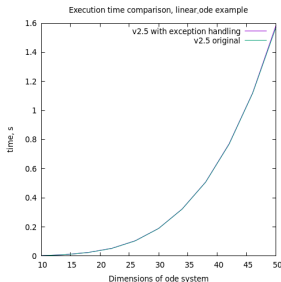
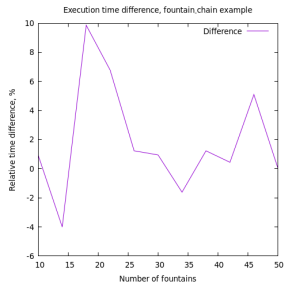
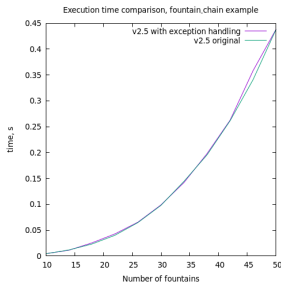
template<typename T>
la::vector_t<T> ds::system_t::G(const la::vector_t<T> &x, const la::vector_t<T> &p) {
    int n=x.size();
    assert(p.size()==n*n);
    la::vector_t<T> r=la::vector_t<T>::Zero(n);
    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
            r(i)+=p(i*n+j)*x(j);
    return r;
}
```

For time measurements:

- ▶ initial value of x : $[1, 1, \dots, 1]$

- ▶ Parameters p contain matrix $A = \begin{bmatrix} -1 & -1 & 0 & 0 & \dots & 0 \\ 1 & -1 & 0 & 0 & \dots & 0 \\ 0 & 0 & -1 - 2/n & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 0 & -1 - (n-1)/n \end{bmatrix}$



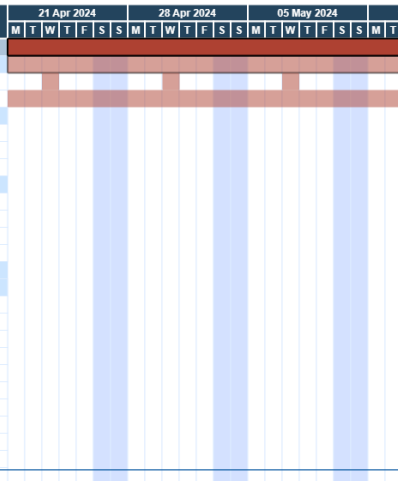


- ▶ Differences between the versions with and without exception handling under different problem sizes:
 - ▶ For small problems, the difference between versions varies between runs because small problems are particularly affected by fluctuations in processor performance (if the processor is busy with operating system tasks).
 - ▶ The overall difference is small because the resources used for exception handling in the program are very minimal and almost negligible.
 - ▶ Most of the resources are utilized for performing algorithmic differentiation and solving the linear algebraic system at each step.

- ▶ **1. Self-study course:**
 - ▶ Discuss problems in group in Discord.
 - ▶ Read source code.
- ▶ **2. Extend cppNum with exception handling:**
 - ▶ Classification of Exceptions
 - ▶ Create exception classes
 - ▶ Add exception classes with try-catch to source code
- ▶ **3. Design scalable case studies:**
 - ▶ Implement new cases
 - ▶ Add timing and plotting method
 - ▶ Visualization
 - ▶ Test and Debug
- ▶ **4. Run time difference Analysis:**
- ▶ **5. Presentation:**
 - ▶ Analysis (user requirements , use case)
 - ▶ Source code and design
 - ▶ Project management and live-demo(test)
 - ▶ Case study(example) and run time analysis
- ▶ *The following page of the PDF outlines the responsibilities of each person.

Person	Task Name	Duration	Start	ETA
0	Exercise 12	56 days	21.04	16.06
1	Self-study course	35 days	16.04	12.05
A	Discuss problems in group in Discord	Wednesday	/	/
A	Read source code	/	/	/
2	Extend cppNum with exception handling	14 days	12.05	26.05
A	Classification of exceptions	/	/	/
K	Create exception classes	/	/	/
K	Add exception classes with try-catch to source code	/	/	/
3	Design scalable case studies	14 days	12.05	26.05
Z	Implement new cases	/	/	/
K	Add timing and plotting method	/	/	/
F	Visualization	/	/	/
S	Test and debug	/	/	/
4	Run time difference Analysis	3 days	26.05	29.05
5	Presentation	14 days	29.05	12.06
A	Task assignment ()	1 days		
F	Analysis (user requirements , use case)			
K	Source code and design			
S	Project management and live-demo(test)			
Z	Case study(example) and run time analysis			

Project Management



- ▶ 1. Build executable: `make`
- ▶ 2. Execute: `./main.exe`

Use function: $y = \frac{1}{2}x_1^2 + \frac{1}{2}x_2^2 + x_1x_2$ for minimization with Newton method (cppNum v2.4).

Hessian matrix: $\partial^2 y = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ is singular, so minimization with Newton method is not applicable.

Show live demo

Output of the live demo:

```
Exception was caught in la::lu_solver_t::run, throw it further.  
Exception was caught in as::newton_solver_t::run, throw it further.  
std::exception was caught in co::newton_minimizer_t::run with following message:  
Matrix is singular, applying LU algorithm for solving a linear system is not possible.  
co::newton_minimizer_t::run returns an initial value of x. Check the correctness of the input.
```

In the output, one can observe a stack trace and a message describing the exception.

Summary and Conclusion