Software and Tools for Computational Engineering

RWTH AACHEN UNIVERSITY

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

Contents

- Software always has a working domain.
- User of the software is not aware of all limitations.
- Software developer helps user by introducing appropriate exception handling.
- Our task is to introduce an exception handling to cppNum v2.4 and v2.5.

- 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 the system is not able to produce the correct result
  - 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:

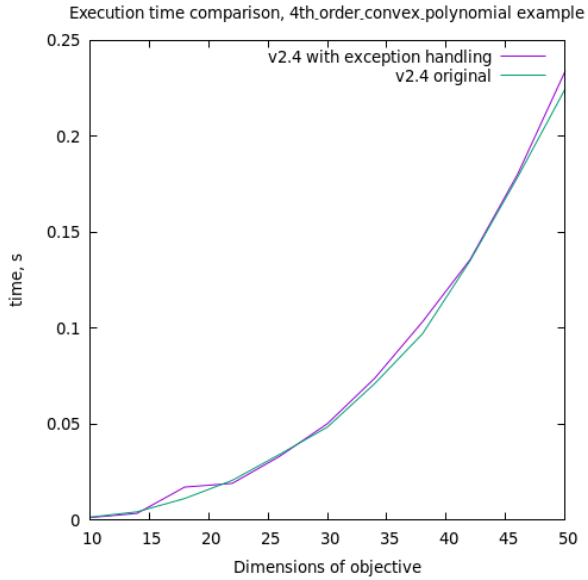
- **Exception Structure:**
  - An exception is a class object.
  - All cppNum exception classes have a single parent class to provide a clear structure.
  - All exception classes are inherited from std::exception to catch together with other exceptions, potentially generated by third-party libraries.
- **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 by more than 10
- **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.

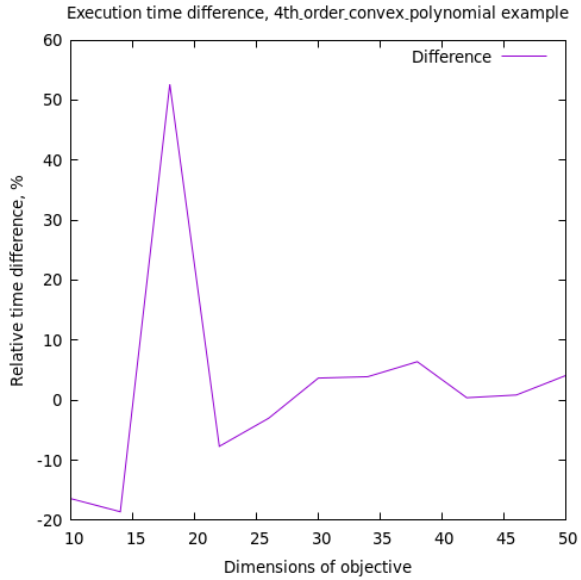
- **Usability:**
  - The system must provide clear and informative error messages to users when exceptions occur.
  - The system should document the scenarios under which exceptions are raised and how they are handled.
- **Maintainability:**
  - The code implementing exception handling must be well-documented 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.

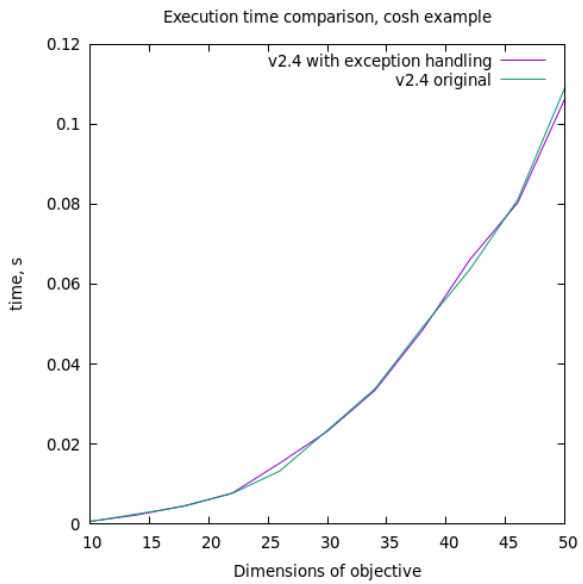
- Third-Party libraries: Eigen and AD.
- Wrap with try-catch every connection point to Eigen and AD to handle possible exceptions. All exceptions that could be thrown by Eigen and AD are inherited from std::exception.
- Rethrow exceptions recursively to the highest level function. This provide so-called 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.

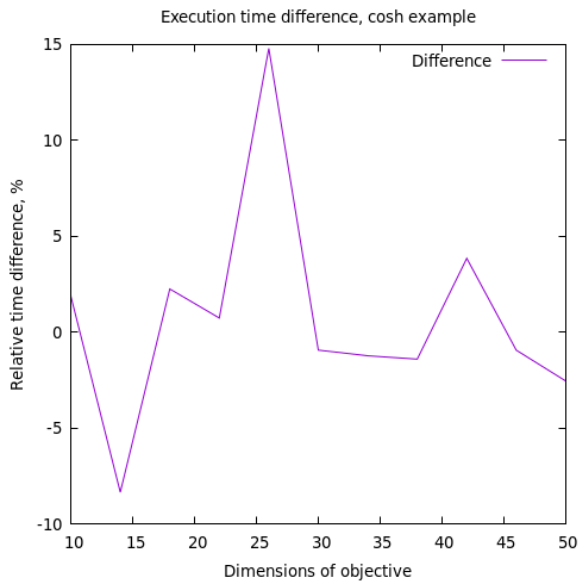
- **1. Operating System:**
  - Xubuntu
- **2. Programming Language and Compiler:**
  - Programming Language: C++.
  - Compiler: GCC.
- **3. Libraries:**
  - Eigen: A C++ library for linear algebra, providing efficient matrix and vector operations.
  - AD: Provide a complete C++ solution for implementing 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.

Software Tests 2.4: Convex Polynomial and Cosh Function

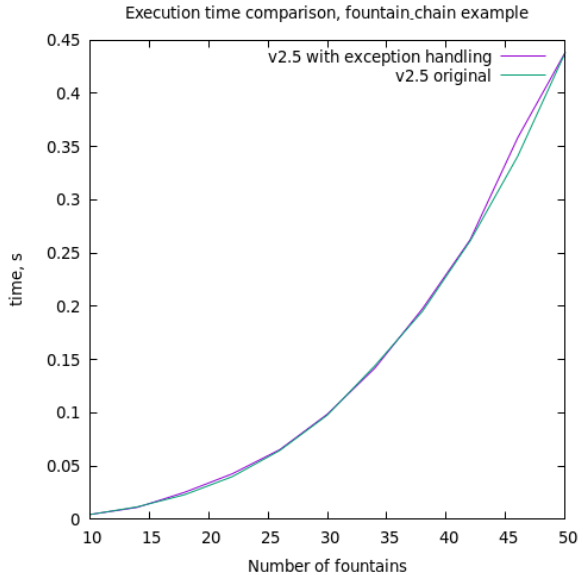


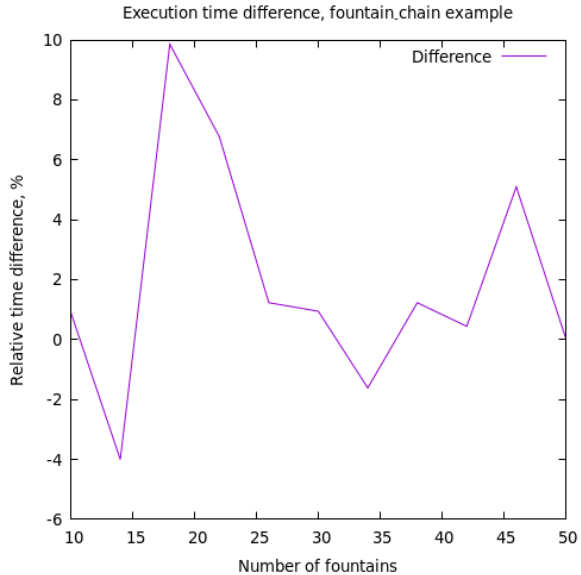


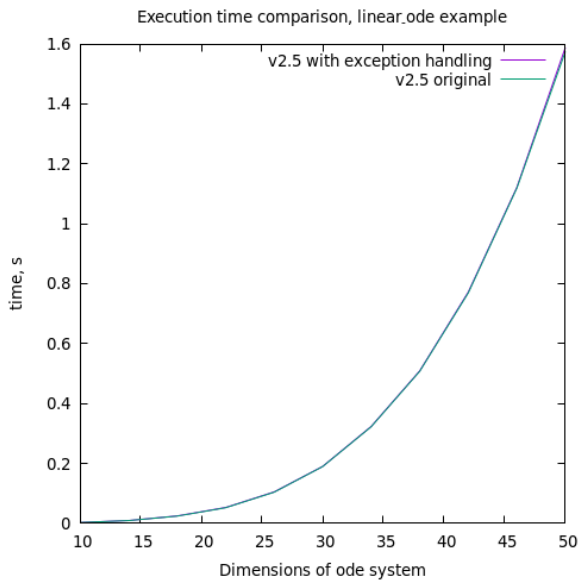


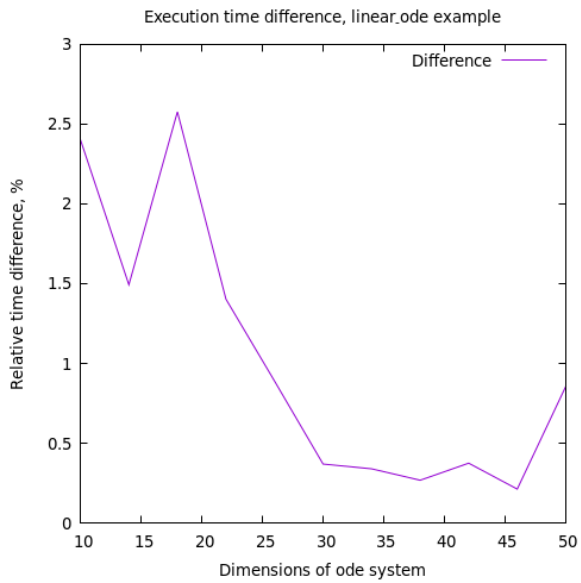


Software Tests 2.5: Fountain Chain and Linear ODE









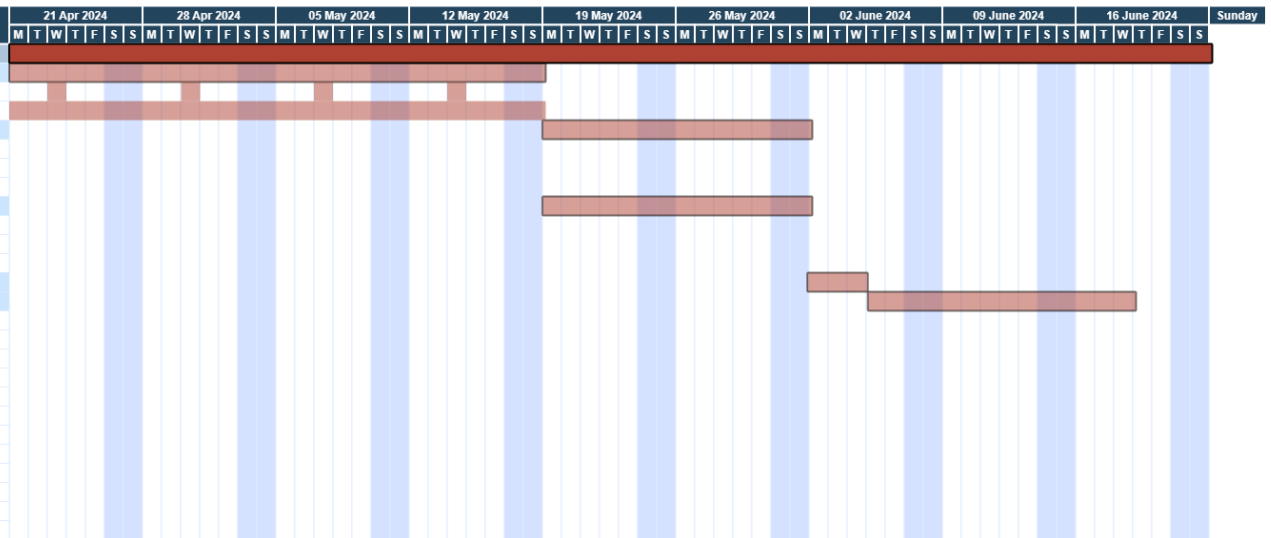
- Differences between the versions with and without exception handling:
  - The process of checking the matrix occurs before the program calculations begin, so the difference between the exception handling-free versions and the original program is most noticeable at the beginning.
  - Subsequently, most operations involve differentiation, integration and matrix calculations, etc., where there is essentially no difference between the two versions.

- **1.Self-study course:**
  - Read source code.
  - Discuss problems in group in Discord.
- **Extend cppNum with exception handling:**
  - Classification of Exceptions
  - Add exception classes as specified in the class diagram and in alignment with the standard exception library
  - Couple Exception Classes with Original Program
- **3.Design scalable case studies:**
  - Implent new cases
  - Add timing and plotting method
  - Test and Debug
- **4.Run time difference Analysis**
- **5.Design scalable case studies:**
  - 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.

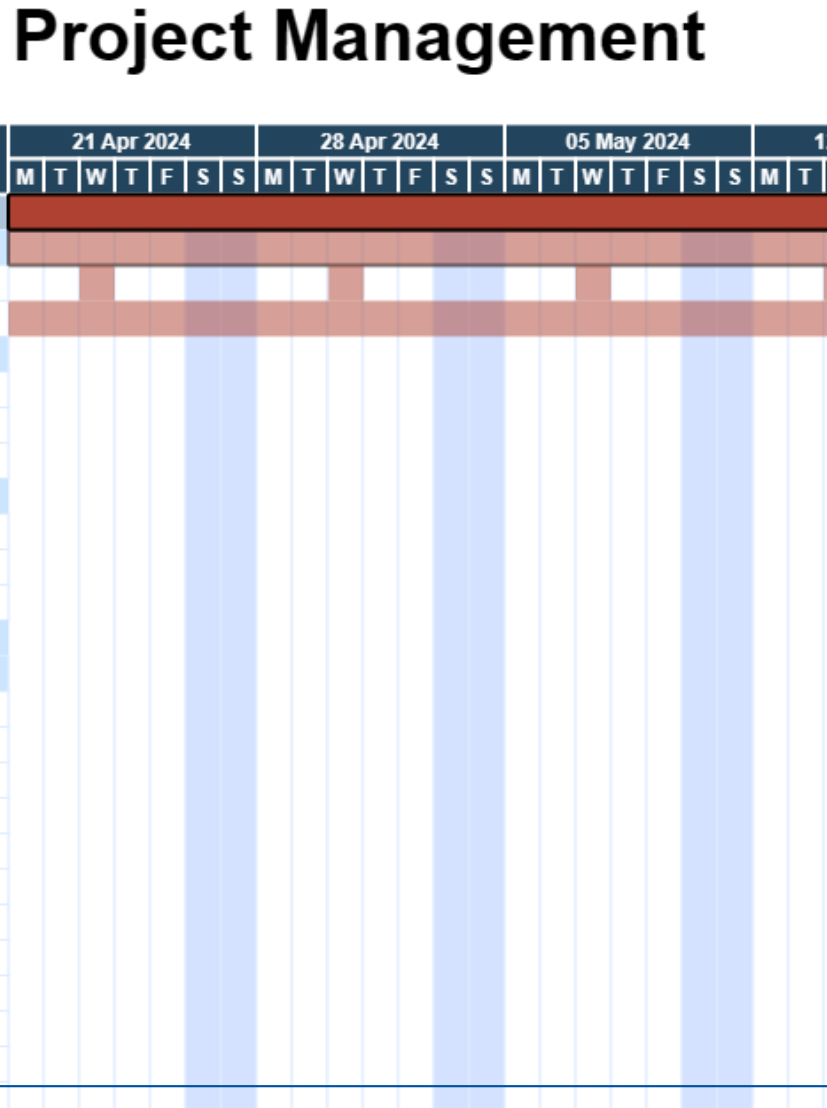
A = ALL members in group
K = Kholod Arseniy
F = Floerke Aaron Albert
S= Song Xinyang
Z = Zhu Yanliang

Project Management

	Task Name	Duration	Start	ETA
1	Exercise 12	56 days	21.04	16.06
A	Self-lerning-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	Adding Exception Classes	/	/	/
A	Coupling Exception Classes with Original Program	/	/	/
3	Design scalable case studies	14 days	12.05	26.05
K	Implent functions	/	/	/
K	Add timing and plotting code	/	/	/
A	Test and Debug	/	/	/
4	Run time difference Analysis	3 days	26.05	29.05
5	Documentation	14 days	29.05	12.06
A	Task Assignment ()	1 days		
K	Analysis (user requirements , use case)			
F	Source code and Design			
S	Project management and live-demo(test)			
Z	Case study(example) and run time analysis			



A = ALL members in group
K = Kholod Arseniy
F = Floerke Aaron Albert
S= Song Xinyang
Z = Zhu Yanliang



	Task Name	Duration	Start	ETA
1	Exercise 12	56 days	21.04	16.06
A	Self-lerning-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	Adding Exception Classes	/	/	/
A	Coupling Exception Classes with Original Program	/	/	/
3	Design scalable case studies	14 days	12.05	26.05
K	Implent functions	/	/	/
K	Add timing and plotting code	/	/	/
A	Test and Debug	/	/	/
4	Run time difference Analysis	3 days	26.05	29.05
5	Documentation	14 days	29.05	12.06
A	Task Assignment ()	1 days		
K	Analysis (user requirements , use case)			
F	Source code and Design			
S	Project management and live-demo(test)			
Z	Case study(example) and run time analysis			

- **1.Make:**
  - make depend
  - make
  - make test
  - make clean
- **2.Execute executable files:**
  - ./main.exe + Command-line arguments, (such as testing exception cases).
- **3.Plot:**
  - gnuplot gnuplot.plt

Summary and Conclusion