Hochschule Ruhr West

Bachelor Thesis

A Framework for Individualised Mathematical Assignments with Solutions in LaTeX

Thomas Flinkow

A thesis submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

Course of Study:	Applied Computer Science dual
Supervisors:	Prof. Dr. rer. nat. Jürgen Vorloeper Prof. Dr. rer. nat. Ioannis Iossifidis
Submitted on:	15 October 2021

Abstract

Individualised assignments benefit both students and teachers. For the generation of individualised assignments using LaTeX, one usually has to resort to external computer algebra systems such as Maple or MATLAB because LaTeX lacks advanced computation features. Using LuaLaTeX, it is possible to embed code written in the general purpose language Lua directly in LaTeX documents, which allows for integrating third-party software libraries into LaTeX code. This thesis proposes a new method of generating individualised assignments with solutions for undergraduate mathematics using LuaLaTeX in conjunction with existing libraries for symbolic computation.

Kurzfassung

Von individualisierten Aufgaben profitieren sowohl Studierende als auch Lehrende. Nutzt man für die Erstellung von individualisierten Aufgaben LaTeX, muss oft auf externe Computeralgebrasysteme wie Maple oder MATLAB zurückgegriffen werden, da LaTeX kaum Funktionen zur mathematischen Berechnung bietet. Mit LuaLaTeX ist es möglich, in der Programmiersprache Lua geschriebenen Code direkt in LaTeX-Dokumente zu integrieren und bestehende Softwarebibliotheken von Drittanbietern zu nutzen. In dieser Arbeit wird eine neue Methode präsentiert, um individualisierte Aufgaben mit Musterlösungen für die Mathematiklehre mittels LuaLaTeX in Kombination mit bestehenden Bibliotheken für symbolisches Rechnen zu erstellen.

Acknowledgement

I would like to express my gratitude to both of my supervisors who guided me throughout this thesis.

- I want to thank Prof. Dr. Jürgen Vorloeper for giving me the opportunity to work as a student assistant with him on an exciting project on investigating ways to incorporate numerical computation into LuaLaTeX. It allowed me to learn from his ideas and ultimately led to this thesis. Further, I want to thank him for his input, helpful feedback and frequent meetings during the writing of this thesis.
- I want to thank Prof. Dr. Ioannis Iossifidis for being supportive, for providing his valuable input and for always having the time for me when I had issues of any sort.

I wish to extend my special thank to Christian Mauve for enabling me to do the dual course of study at Mauve Mailorder Software.

Finally, I would like to thank my family and partner for their support during my studies and their help proofreading this text.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

Individualised assignment means providing each student with a different task of equal difficulty. The effectiveness of individualised assignments has been studied thoroughly: not only does it help combat plagiarism and cheating (e.g., by collusion among students), but it has also been shown to be more rewarding for students, improving their attitudes towards assignments in general, making their studies more successful by enforcing minimum participation and encouraging sound and solid learning with the potential to even arouse long-lasting interest [1]–[3].

Providing students with individual assignments has been shown to be beneficial for their learning success.

LaTeX is a software system for high-quality typesetting of scientific documents and is commonly used in academia for publications and the creation of documents for teaching purposes. The creation of individualised exercise sheets in LaTeX, however, is a repetitive and time-consuming process. Consider, for example, Figure 1.1 on the following page, which shows a simple exercise from undergraduate calculus. The task is to determine the area between a parabola $f(x) = ax^2$ and a line $g(x) = mx + b$ between the points where f and g intersect.

The creation of individualised assignments in LaTeX is both repetitive and time-consuming.

Creating different, individualised versions of this task means that for each version, one has to

1. choose different, suitable parameters a, b and m,
2. calculate the points of intersections between the functions,
3. evaluate the integral between these points, and, finally,
4. compile the file to create the resulting PDF file.

Task 1. Calculate the area between $f(x) = \frac{1}{2}x^2$ and $g(x) = x+4$.

Solution: The graphs of the functions f and g intersect at $P_1(-2,2)^T$ and $P_2(4,8)^T$. Thus, the area between them is

$$A = \int_{-2}^{4} (g(x) - f(x))\,\mathrm{d}x = \int_{-2}^{4} \left(x + 4 - \frac{1}{2}x^2\right)\,\mathrm{d}x$$
$$= \left[4x + \frac{x^2}{2} - \frac{x^3}{6}\right]_{-2}^{4} = 18.$$

Sketch:

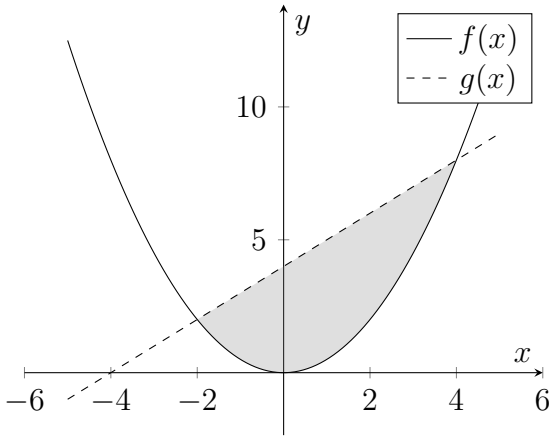


Figure 1.1: An example task from undergraduate calculus.

LaTeX offers no advanced calculation features.

Despite being useful for the typesetting of mathematical expressions, LaTeX provides only little functionality to compute mathematical expressions. Hence, there is a need to resort to external computer algebra systems (such as MATLAB) for the actual computations because they cannot be done directly in the LaTeX files. Being able to perform these computations directly in LaTeX and to automate the process of randomisation so that multiple individualised versions can be generated at once has the potential to save teaching staff a lot of time and to simplify the process of creating assignment sheets in general.

1.1 Related work

Much like individualised assignment not being new, automating the process of creating individualised assignments in LaTeX using various software systems has been a point of research and development for a long time. All of the present approaches have in common the use of external tools to perform the actual computations. Often, Python or MATLAB are used because they both provide a broad variety of numeric and symbolic computational features.

Creating and automating exams using LaTeX and Python.

One approach, using LaTeX in combination with Python for the creation and automation of exams is presented in [4]. Therein, mathematical tasks are individualised, and the expected result is encoded in a QR code printed next to the task using pythontex [5]. Because of the way pythontex works, the LaTeX file needs to be compiled once, and then, pythontex needs to be run before the LaTeX document is compiled one more time to obtain the resulting PDF file [6].

Randomising assignments using LaTeX with SageMath.

Another approach that also uses Python is shown in [7]. A LaTeX package called SageTeX is used to integrate the scientific computation library SageMath (examined in detail in Section 3.3.2 on page 18) into LaTeX to create individualised mathematical assignments. The compilation process is similar to the one explained above, which means each `.tex` file needs to be compiled twice.

Creating individualised assignments with solutions using MATLAB.

The MaTeX project [8], which is developed in a joint effort at Esslingen University and MINT-Kolleg Baden-Württemberg (a research institute of the University of Stuttgart and Karlsruher Institute of Technology (KIT)), uses MATLAB to do all of the computation and individualisation. Users interact with MaTeX via a web interface to generate individualised exercise sheets with both numeric and symbolical solutions, as well as different plots and figures. MaTeX heavily relies on MATLAB and its Symbolic Math Toolbox to provide this functionality. MATLAB does not provide an easy way to integrate with LaTeX, so the core concept of MaTeX is to work with MATLAB files for the individualisation and computation, and to create LaTeX files from within MATLAB that can then be processed using a LaTeX compiler [8, Chapter 4]. This we consider a slight drawback because of the rather cumbersome way of creating LaTeX files, as shown exemplary in Listing 1.1 on the following page.

```
clear all
fID = fopen('Aufgabe3.tex', 'wt');

%% Aufgabenstellung formulieren
Text = ['\\textbf{Aufgabe}: L\\"osen Sie die Gleichung \n
↪  \\[ \n '];
fprintf(fID, Text);
```

Listing 1.1: MATLAB code from a MaTEX task generator to write text into a LATEX file. Taken from [8, Chapter 4].

Creating individualised exercises with solutions for electrical engineering with LATEX and MATLAB.

Another project also using MATLAB is presented in [9] and focuses on creating individualised exercises with solutions for electrical engineering. In addition to text and mathematical formulas, it also puts a large amount of focus on the automated generation of figures using PGFPLOTS and CircuiTikZ. As seen in MaTEX before, the actual randomisation process takes part in MATLAB and again relies on MATLAB to generate the LATEX files. Interestingly, the author notes in [9] how it would be helpful if the algorithms for creating tasks and solutions could be moved to LATEX.

1.2 Objective

Goal: to move as much programming work as possible to LATEX without the need for external tools.

The goal of this project is to evaluate one possibility to shift away from approaches using external tools such as MATLAB and to move as much programming work as possible to LATEX, making the whole process of generating individualised mathematical exercises with solutions both more straight-forward and maintainable, as well as faster in terms of compilation time.

One main objective is that users be able to write valid LATEX files, as opposed to hybrid documents that contain LATEX commands (such as the MATLAB files used by MaTEX). This allows users to utilise the full power of LATEX in a way they are used to, e.g., with LATEX commands, packages and editors of their choice. A framework is to be developed that supports users in writing standard LATEX code, equipping them with features for randomisation and symbolic computation directly in the LATEX code.

```
\begin{luacode*}
  symcomp = require "symcomp"
  random = require "random"

  f = random.parabola()
  diff = symcomp.diff(f, "x")
\end{luacode*}

The derivative of \(f(x)=\scprint{f}\) is
↪  \(\scprint{diff}\).
```

Listing 1.2: An example of calling Lua functions provided by the framework from within LaTeX code. The example generates random parabolas and prints their derivative.

Using LuaLaTeX instead of pdfLaTeX opens up a whole new world of opportunities. Not only is LuaLaTeX the designated successor of pdfLaTeX and offers native support for Unicode, but it also includes the general purpose programming language Lua [10] as an embedded scripting language, allowing users to arbitrarily extend LaTeX's functionality.

Using LuaLaTeX instead of pdfLaTeX allows for using Lua, an embedded scripting language.

In [11], LuaLaTeX has been used to extend LaTeX's capabilities and carry out numerical computations directly inside the document. All calculations therein were implemented directly in Lua, but because Lua provides a robust and fast C application program interface (API), it is also possible to write code in C, and more importantly, to integrate existing code accessible from C. For example, in [12], the well-reputed numerical computation libraries BLAS and LAPACK have successfully been used from within LuaLaTeX using Lua's C API.

Taking this development a step further, it should be possible to forego the need to use external tools for calculations, as long as there are suitable libraries available to Lua that offer the required functionality for symbolic and numeric computation.

It is possible to use existing software libraries from LuaLaTeX.

Listing 1.2 shows an example of a LaTeX file that uses the framework's randomisation and symbolic computation features to create individualised versions of an exercise about calculating the derivative of a parabola.

1.3 Remarks

This document itself was compiled using LuaLaTeX. A less traditional layout including margin notes has been chosen to allow readers to skim through the text faster and to provide them with additional information and figures without disrupting the flow of the text. All figures were created by the author using TikZ.

When evaluating performance, some benchmarks are presented in the following chapters. They were obtained by the author using a virtual machine running Ubuntu. The hardware setup of the system running the benchmarks is shown below:

```
CPU : Intel Core i7 10700-K (8 x 3.8 GHz)
RAM : 32 GB DDR4-3000
VM  : Oracle VirtualBox 6.1.26
```

The virtual machine can use up to 8 CPU cores and up to 24 GB of RAM.

All background processes have been closed prior to benchmark execution to minimise disruptive influences. The results were obtained by averaging over a large number of iterations.

However, the benchmarks are not to be taken as the absolute truth. They are included to visualise the orders of magnitudes and to hint at approximate proportions instead.

Chapter 2

Architecture

Figure 2.1 illustrates the directory structure of the framework. All components that are listed in the figure will be explained in more detail with examples below. The `output` directory contains compiled, individualised versions of the same exercise sheet. Mathematical tasks are present in the `taskpool` directory. These tasks make use of the randomisation and symbolic computation features provided by the Lua modules `random.lua` and `symcomp.lua`, respectively.

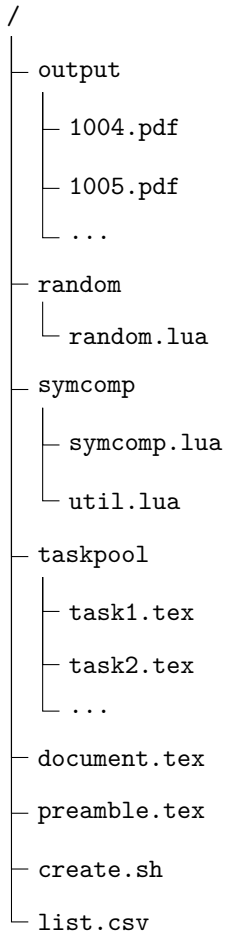


Figure 2.1: The directory structure of the framework.

2.1 Interacting with the framework

The idea is that users create LaTeX files that each represent one mathematical exercise in the so-called *task pool*. When creating these task files, users can make use of randomisation and symbolic computation features. For example, as opposed to calculating the derivative of a function by hand or using external tools, the Lua function `symcomp.diff(f, x)` can be called from within LaTeX code.

The randomisation features the framework provides are intended to be used instead of hard coding actual numerical values in the LaTeX files: for example, calling `random.parabola()` returns a string representing a parabola $ax^2 + b$ where the parameters a and b will be replaced with random numerical values at compile-time.

Exercise sheets usually consist of multiple exercises from the task pool. An example of an exercise sheet is shown in Listing 2.1 on the next page.

```
\documentclass[a4paper, 10pt]{exam}
\input{preamble}

\begin{document}
\begin{questions}

\input{taskpool/task1}
\input{taskpool/task42}

\end{questions}
\end{document}
```

Listing 2.1: The document structure of an exercise sheet consisting of multiple exercises from the task pool.

```
[[id]],"First name",Surname,"ID number"
```

Listing 2.2: The structure of a `.csv` file exported from the e-learning management system of the Hochschule Ruhr West. Only relevant columns are listed; there are some columns following after the `"ID number"` column which are not currently used by the framework.

Compiling an assignment creates two PDF files: one including solutions and one without.

E-learning systems such as Moodle allow for exporting information about the participants in a course (such as name and matriculation number) in the form of a `.csv` file. This file can be fed to the shell script `create.sh`, which then compiles different, individualised versions of an exercise sheet, two for each student in the course: one including solutions and one without. Generated files are going to be named `ID_FirstName_Surname_DocumentName.pdf` and `ID_FirstName_Surname_DocumentName_Solutions.pdf` where `ID`, `FirstName` and `Surname` are replaced with the respective values from the `csv` file. `DocumentName` can be chosen by the user.

```
./create.sh document Assignment1 list.csv
```

Listing 2.3: Compiling the file `document.tex` to create individualised versions for all entries in the `list.csv` file.

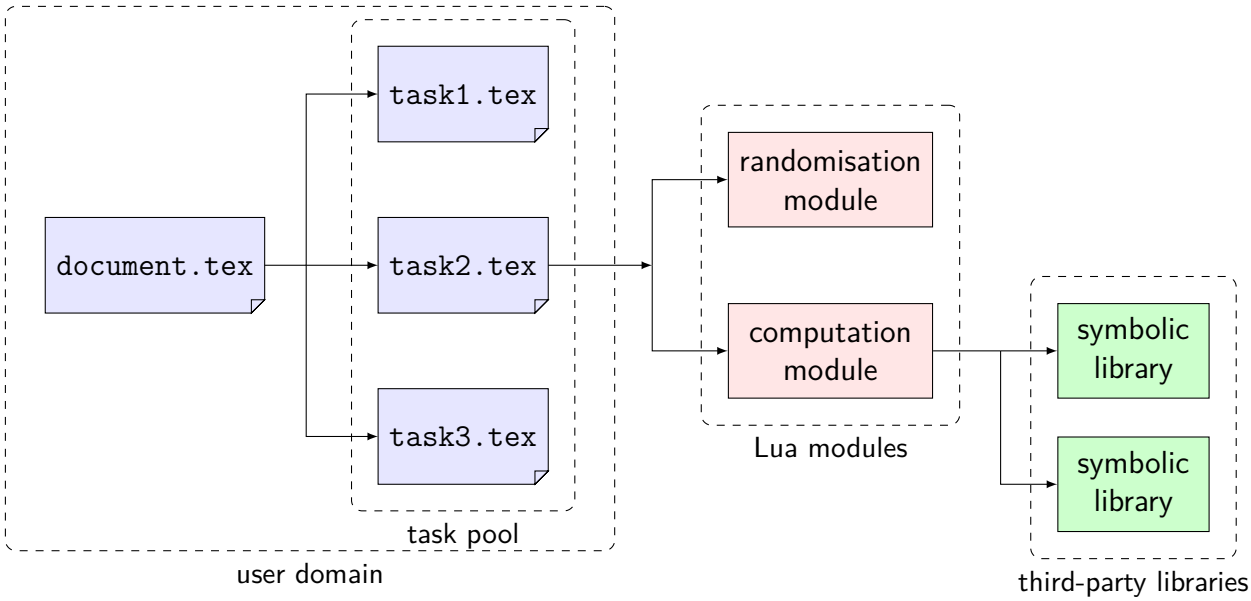


Figure 2.2: The architecture of the framework. Users (i.e., teaching staff) only ever have to deal with LaTeX files, mostly a main file (`document.tex`) that includes various exercises from the task pool that contains mathematical problems. Those problems are LaTeX files which call into the Lua modules to randomise parameters and to evaluate expressions. The computation module wraps third-party mathematical libraries.

2.2 Components

The main functionality of the framework is provided by a randomisation module and a symbolic computation module.

The randomisation and symbolic computation features are provided by two Lua modules: `random.lua` offers functionality to individualise mathematical expressions, and `symcomp.lua` provides all the required mathematical functions, such as symbolic differentiation and integration. Figure 2.2 gives a rough overview of the structure of the framework, including the two main modules and the LaTeX files that end users interact with.

The symbolic computation module uses third-party libraries for symbolic computation.

The randomisation module does not have any dependencies apart from the Lua standard library for the mathematical random function. The symbolic computation module, on the other hand, by design can have some dependencies on third-party computer algebra systems, as illustrated in Figure 2.2. Choosing suitable third-party libraries is vital and is discussed in detail in the next chapter, where multiple well-known software libraries for symbolic computation are assessed.

2.3 Exchanging data

The randomisation and the computation module are completely independent from each other.

It is important to note that the two modules are completely separated and are not dependent on each other. This produces the advantage that the randomisation module can generate mathematical expressions without requiring any knowledge of the actual mathematical objects used by the computation module, which makes writing code in the randomisation module more straightforward. At the same time, this separation allows for easier changes in the computation module because internal changes can be made there without the randomisation module having to adapt to those.

Although the two modules are separate from each other, they are meant to be used in conjunction. Expressions generated by the randomisation module are usually passed to the symbolic computation module, which further processes the expression; for example, the randomisation module may generate an expression which is then passed to the computation module to calculate the derivative. Therefore, a method is needed for the two modules to interact with each other and to exchange information.

The randomisation module returns strings and numbers.

The randomisation module exposes functions that accept various parameter types such as strings and numbers. It returns mathematical expressions as strings (such as `"3*sin(x)"`) and numbers, which can then be passed to methods provided by the computation module.

The computation module returns a Lua table with two values. The standard form and the LaTeX form of an expression.

The return value of the computation module is always a string tuple that is called *ExprRep* in the code. It stands for *Expression Representation* and is implemented in the C++ class `ExprRep`, as well as in Lua as a table with two elements. Essentially, it contains the 'basic' form of an expression (e.g., `"sin((x^2)/2)"`) and the LaTeX form of the same expression (e.g., `"\sin\frac{x^2}{2}"`).

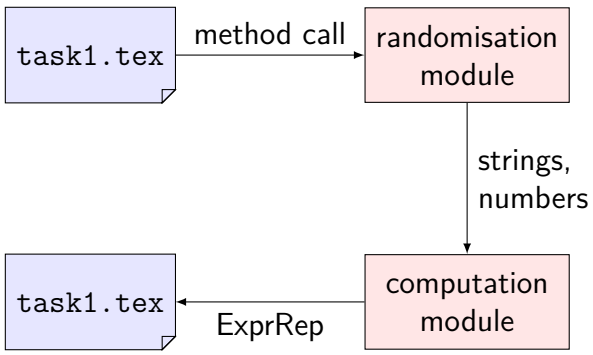


Figure 2.3: A visualisation of the interaction between LaTeX and the framework modules. Functions provided by the randomisation module are called from the LaTeX file and return strings and numbers, which are then passed to the computation module which returns an ExprRep object, that is, a table with two values: the *basic* form of an expression as the first entry, and the LaTeX form as the second.

Chapter 3

Symbolic computation

A computer algebra system (CAS) is a software system that manipulates mathematical expressions. Here, computation is exact and *symbolic* (as opposed to *numeric*). This means that expressions can contain variables that have no given value and that these variables are manipulated as symbols [13], [14].

Exact means that symbolic computation does not suffer from the loss of precision introduced by floating-point arithmetic.

CASs deal with mathematical expressions that express the relationships between operands and operators. They are usually represented internally using expression trees [15], [16]. For example, Figure 3.1 displays the expression $3x^2 + 7$. Expression trees place the main operation at the root of the tree and assist in evaluating the expression by making it possible for the sub-expressions, which are represented by the sub-trees, to be evaluated first. In Figure 3.1, it is apparent that $3x^2$ is a sub-expression of $3x^2 + 7$, as is x^2, but that $3 + 7$ is not.

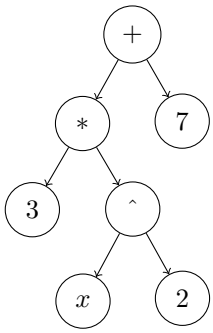


Figure 3.1: The expression tree for $3x^2 + 7$.

CASs implicitly apply simplifications to transform expressions into a standard, so-called *canonical* form. For example,

$$x + 3x + yy^2 + z^0 \to 4x + y^3 + 1\,. \tag{3.1}$$

For example, $8x$ is a simpler, equivalent expression of $3x + 7x - 2x$.

Structurally, $7x^2$ and $2x^2 + 5x^2$ are not equivalent; semantically, though, they are equivalent.

Two problems arise here: the problem of obtaining an equivalent but simpler form of an expression and the problem of finding unique representations for equivalent objects [14].

Many different CASs are available, whether commercially (e.g., Reduce, Mathematica, Maple, MATLAB) or for free (e.g., Maxima, GiNaC, SageMath, SymPy). Some, such as Reduce, Maxima, Derive and Axiom, are based on Lisp [17]. Others, such as Maple and

Mathematica, are based on C. GiNaC, ViennaMath, and SymbolicC++ are written in C++, and SageMath and SymPy are based on Python.

CASs often offer a domain-specific language for interaction, making it difficult to use them as a library.

Often, CASs are meant to be used interactively, not programmatically. They often introduce a domain-specific language for the sake of interaction. For example, Mathematica offers the Wolfram language as a means of interacting with the software. This makes it hard to these CASs as a library in compiled code, such as software written in C or C++ [18].

The framework needs symbolic computation for the sake of analytically evaluating integrals, calculating the derivatives of symbolic expressions and dealing with mathematical expressions that contain both numbers and symbolic variables. Below, a selection of popular libraries offering symbolic computation are compared and evaluated with respect to their usability for the framework.

3.1 Criteria for suitable libraries

To evaluate whether a library is suited for the purposes of the framework, let us define a few key criteria.

Suitable libraries must be free and open-source.

First, because free/libre open source software (FLOSS) brings many benefits, no commercial or proprietary libraries (such as Maple or Mathematica) will be taken into consideration. Not only does using FLOSS not require paying license fees, but it is also often actively developed by large communities, in addition to being highly configurable and adjustable to different needs.

Suitable libraries must be usable for undergraduate mathematics topics such as linear algebra, calculus and analysis.

Concerning the scope of the features, libraries should be usable for all undergraduate mathematics topics. For example, in the case of basic linear algebra, working with vectors and matrices should be possible. A suitable library should also be able to handle polynomials and trigonometric functions, as well as calculate derivatives and basic definite and indefinite integrals.

Suitable libraries must be usable directly from LaTeX, Lua, or provide a C API.

Libraries need to be usable either directly from LaTeX or Lua. In addition to that, all CASs that can be called from C/C++ are feasible candidates since Lua provides a C API that allows them to be used from within LuaLaTeX. Besides providing a C API, it is

also important that interaction with this library be uncomplicated; an easily usable, well-documented API is essential.

Since the framework is designed to handle a large number of documents, each of which contains multiple separate exercises with each of those exercises utilising symbolic computation, it is clear that performance is important as well.

Suitable libraries must be usable in scenarios where performance matters.

In the following, a few popular free and open-source libraries for symbolic computation are assessed by taking into account the aforementioned key criteria.

To give a brief overview of API complexity, the code necessary to evaluate and render the integral in Equation (3.2) will be displayed for all assessed CASs where possible.

$$\int_{-2}^{4} \left(-\frac{1}{2}x^2 + x + 4 \right) \mathrm{d}x \tag{3.2}$$

3.2 Computer algebra systems in Lua

As LuaLaTeX allows to embed code written in the general purpose programming language Lua in LaTeX documents, it is natural to look for CASs written in Lua to easily integrate into the framework's symbolic computation module.

There do not seem to be many libraries for scientific computation written in Lua. One of them is SciLua, which is a high-performance framework for numerical scientific computing, and another is Symbolic Lua, a library that can also perform symbolic computation. As SciLua does not provide symbolic computation features apart from symbolic differentiation via its `sci.diff` module, only Symbolic Lua will be considered for the framework in the following.

Not many libraries for scientific computing are available in Lua.

3.2.1 Symbolic Lua

Symbolic Lua is a library written exclusively in Lua. It is a free, open-source library licensed under the MIT license.

Originally developed for computational physics, it supports a broad variety of features for equation solving, linear algebra and calculus.

Symbolic Lua is available at https://github.com/thenumbernine/symmath-lua.

It is able to evaluate integrals analytically although not all integrals can be evaluated yet [19].

```
symmath.Integral(-1/2*x^2+x+4, x, -2, 4)()
```

Listing 3.1: Lua code to evaluate $\int_{-2}^{4} \left(-\frac{1}{2}x^2 + x + 4\right) \mathrm{d}x$ using Symbolic Lua.

Despite not having a large community of developers, Symbolic Lua is actively developed, and issues opened on GitHub are quickly resolved.

Symbolic Lua has the slight drawback of being unable to always simplify expressions; in some cases, it runs into so-called simplification loops, which are noticeable in the drastically reduced performance.

Because of being implemented in Lua, Symbolic Lua is directly usable from LuaLaTeX and easily installable using the LuaRocks package manager. However, being implemented in Lua, there are some performance drawbacks. Table 3.1 shows the time per call for calculating the derivative of a polynomial of degree 5 in Symbolic Lua compared with SymEngine, which is a C++ library for symbolic manipulation. Compared with only 25 µs of time needed with the C++ library SymEngine, Symbolic Lua takes 360 µs, which is a significant difference.

Table 3.1: Comparison of the time it takes to differentiate a polynomial of degree 5 using SymEngine compared with Symbolic Lua.

Library	Time
SymEngine	25 µs
Symbolic Lua	360 µs

3.3 Computer algebra systems in Python

Python is a popular choice for programming in the sciences, especially in data science and machine learning. It is a general purpose programming language which has a particularly simple syntax, making it an easy-to-learn language and accessible to a broad audience.

Python has a big community and a wide range of frameworks and libraries such as SciPy and NumPy are available for scientific computing. A few of them are assessed in the following.

3.3.1 SymPy

SymPy [20] is a popular, free and open-source CAS written in Python and licensed under BSD. It can be used both interactively and as a programmatic library. The current release is version 1.8 (released in April 2021).

SymPy can be used for a broad variety of tasks.

SymPy offers a large number of features, including algorithms for calculus, group theory, differential geometry, logic, matrices and number theory. Note that both definite and indefinite integrals are supported, and improper integrals of non-polynomial functions such as the Gaussian integral given in Equation (3.3)

$$\int_{-\infty}^{\infty} \exp\left(-x^2\right) \mathrm{d}x = \sqrt{\pi} \tag{3.3}$$

can be computed. The sole code necessary is presented in Listing 3.2.

Note SymPy's concise, intuitive and easy-to-use API: two 'o' characters represent ∞.

```
integrate(exp(-x**2), (x, -oo, oo))
```

Listing 3.2: SymPy code to calculate the Gaussian integral $\int_{-\infty}^{\infty} \exp\left(-x^2\right) \mathrm{d}x$.

SymPy is actively developed and has a large community, allowing it to rapidly respond to the user's wishes. It provides a very concise, intuitive and easy-to-use API that is also very well-documented. One reason why SymPy provides such an intuitive API is the fact that it is written in Python, a programming language designed with readability and simplicity in mind. That simplicity, especially the fact that it is a dynamically-typed and interpreted language, naturally comes with some drawbacks regarding performance.

SymPy can be used directly in LaTeX using the sympytex [1] [21] package, making it a seemingly obvious choice as a library for the framework.

[1] available at https://ctan.org/pkg/sympytex

However, as described in detail in Section 3.3.3 on the following page, the way sympytex works internally has a remarkable negative impact on performance.

3.3.2 SageMath

Sage stands for *System for Algebra and Geometry Experimentation*.

SageMath [22] is another free open-source CAS written in Python and licensed under the GPL. The current release of SageMath is version 9.4 (released in August 2021).

SageMath incorporates a large number of other open-source mathematical software packages (both numeric and symbolic), such as NumPy, SciPy and even SymPy, too. Therefore, it is no surprise that SageMath offers the same variety of features that SymPy does. Moreover, it enhances SymPy's symbolic computation capabilities and offers sophisticated algorithms for number theory and abstract algebra. Note that SymPy and SageMath are not the same and should not be used interchangeably. Apart from syntax differences, they followed different design philosophies from the very beginning: SymPy aims to be a lightweight Python module, while SageMath aims to combine many useful open-source mathematics software packages.

[2] available at https://ctan.org/pkg/sagetex

SageMath can be used directly in LaTeX using the SageTeX [2] package, but this approach also suffers from the same performance drawbacks as sympytex.

3.3.3 Performance of sympytex and SageTeX

As mentioned before, both sympytex and SageTeX come with not negligible declines in performance, which are caused by the way these packages work.

Using either of these packages requires a LaTeX file to be compiled twice before the final PDF file is created. Figure 3.2 on the next page illustrates the compilation process using SageTeX in more detail. The process is similar for sympytex. Having to compile the document twice already seems slow without having measured and having seen any numbers. To compare actual numbers, sympytex and SageTeX are now being compared with each other and with the C++ library SymEngine, which was designed with performance in mind.

Table 3.2: Comparison of the build times between SageTeX, sympytex and the C++ library SymEngine.

Library	Duration
SageTeX	1.6 s
sympytex	1.1 s
SymEngine	0.4 s

Table 3.2 compares the average build duration for a `.tex` document whose only content is a call to the respective library to compute $\frac{\mathrm{d}}{\mathrm{d}x} 3x^2 + 7x$ and to render the result. Listing 3.3 on the next page

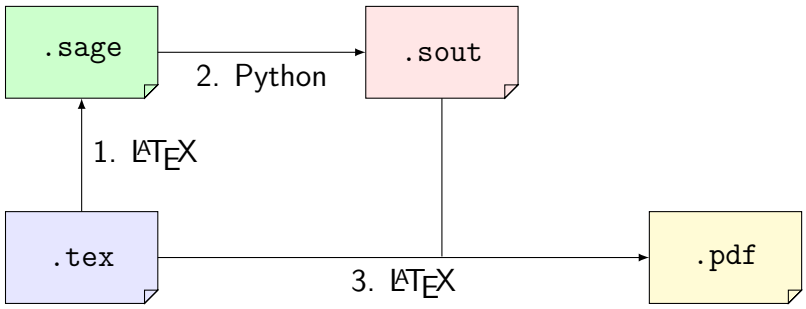


Figure 3.2: Overview of the compilation process using SageTeX. 1. Compiling the LaTeX file for the first time creates a `.sage` file. 2. This file then has to be processed by SageMath using Python to create a `.sout` file. 3. Finally, compiling the LaTeX for a second time using the .sout file creates the PDF file.

displays the whole content of the LaTeX file that uses SageTeX. The file that uses sympytex looks similar, and the file that uses SymEngine requires a C++ source file as well and therefore, is not shown here because it would sacrifice readability.

It can be observed that SageTeX is taking approximately four times as long and sympytex taking about three times as long as the LuaLaTeX version using a C++ library. Therefore, focusing on performance, neither sympytex nor SageTeX can be considered optimal choices due to their compilation process. Instead, the goal is to find one or more suitable libraries written in C/C++ (or at least, usable from C/C++) to use from Lua and to decrease the number of compilations required.

Extrapolating from the numbers in Table 3.2 on the facing page, compiling 50 documents would take about 1 minute and 20 seconds using SageTeX compared with only 20 seconds using the C++ library.

```
\documentclass{article}
\usepackage{sagetex}

\begin{document}
\(\sage{diff(3*x**2+7*x)}\)
\end{document}
```

Listing 3.3: LaTeX code to calculate $\frac{\mathrm{d}}{\mathrm{d}x}3x^2 + 7x$ using SageTeX.

3.4 Computer algebra systems in C++

Being a very high-performance language and providing features for object-orientated programming, it is no wonder C++ is frequently used for scientific computation. In addition to that, C code can almost seamlessly integrate with C++ code, making libraries written in C as well as libraries written in any other language providing a C API also usable from C++. Therefore, a broad variety of libraries for scientific and symbolic computation in C++ is available, and a few popular ones are assessed in the following.

3.4.1 SymEngine

SymEngine is a promising standalone symbolic computation library written in C++. It provides a convenient and intuitive API, allowing it to be used from C, C++ and Python.

SymEngine is a C++ library intended to replace SymPy's back end for more performance.

The resemblance of the names *SymEngine* and *SymPy* is not coincidental: SymEngine is intended to replace performance-critical parts of SymPy's back end over time [23]. This design goal makes SymEngine a promising candidate if we expect the same variety of features that SymPy provides, here paired with the high-performance characteristics of a C++ library. SymEngine's API is concise and intuitive. Listing 3.4 on the next page illustrates the code required to calculate

$$\frac{\mathrm{d}}{\mathrm{d}x} 3x^4 + \sin x = 12x^3 + \cos x \tag{3.4}$$

as SymEngine cannot handle integration yet[3]. SymEngine offers LaTeX output and a basic parser for reading mathematical expressions from strings at run-time.

[3] A GitHub issue requesting an integration method to be implemented was opened in 2019 and is still not fixed. The issue can be found at https://github.com/symengine/symengine/issues/1597.

At the time of writing, SymEngine is still under development. Some features (such as simplification of expressions) are not implemented yet, and documentation is either hard to find or virtually nonexistent. With missing features and a lack of documentation being the only drawbacks, it is reasonable to expect this to change in the future, taking into account SymEngine's large community. Therefore, for the framework, it makes sense to use SymEngine hand in hand with other, more mature libraries for the features SymEngine is not yet capable of handling.

```cpp
#include <symengine/expression.h>
#include <iostream>

int main()
{
  SymEngine::Expression x("x");

  auto ex = 3 * SymEngine::pow(x, 4) + SymEngine::sin(x);
  auto result = ex.diff(x);

  std::cout << SymEngine::latex(result) << std::endl;

  return 0;
}
```

Listing 3.4: C++ code to evaluate $\int_{-2}^{4} \left(-\frac{1}{2}x^2 + x + 4\right) \mathrm{d}x$ using SymEngine.

3.4.2 GiNaC

GiNaC [24] is a framework for symbolic computation written in C++. Originally designed for loop calculations in quantum field theory, GiNaC supports a multitude of features such as arbitrarily sized numbers, multivariate polynomials, linear algebra, symbolic differentiation and symbolic integration, which render it useful for a general symbolic computation library as the back-end of the framework.

GiNaC is a recursive acronym standing for *GiNaC is not a CAS*.

The code shown in Listing 3.5 on the following page returns LaTeX code that renders the integral as follows:

$$\int_{-2}^{4} dx\,(4 - (0.5)x^2 + x) = 18.0$$

Since GiNaC has its origins in physics, the differential operator is placed closer to the integral itself, which might be an unusual sight for undergraduate students of other disciplines, but can be changed using custom print functions.

GiNaC was clearly written with performance in mind, and it trades speed for predictable output [25]. Canonical forms of expressions are not consistent across multiple runs because terms are ordered based on an internal hash function. In high-energy physics applications (where GiNaC is generally used), this is no drawback: when dealing with 10^3–10^6 symbolic terms, the order of the terms does not exactly

GiNaC might happen to represent $x - y$ as $-y + x$ or $-(y - x)$ as well.

```cpp
#include <iostream>
#include <ginac/ginac.h>

int main()
{
  GiNaC::symbol x("x");
  GiNaC::ex integrand = -1/2.0 * GiNaC::pow(x, 2) + x + 4;
  GiNaC::ex e = GiNaC::integral(x, -2, 4, integrand);

  std::cout << GiNaC::latex
            << e << " = " << e.eval_integ()
            << std::endl;

  return 0;
}
```

Listing 3.5: C++ code to evaluate $\int_{-2}^{4} \left(-\frac{1}{2}x^2 + x + 4\right) \mathrm{d}x$ using GiNaC.

matter. For teaching purposes, this presents a slight inconvenience because readability should not be sacrificed for performance.

3.4.3 ViennaMath

ViennaMath [26, Section 5] is another free, open-source symbolic computation library written in C++. It was developed at the Vienna University of Technology alongside other mathematical libraries such as ViennaCL, ViennaData and ViennaFEM. The current release is version 1.0.0, which was released in 2012.

ViennaMath allows for expressions to be evaluated at compile-time.

It is a header-only library and allows mathematical expressions to be evaluated not only at run-time, but also at compile-time thanks to extensive template meta-programming. Compile-time evaluation of expressions certainly yields great performance advantages over run-time evaluation, and no other library assessed herein is capable of performing compile-time evaluation. For the purposes of the framework, this is, however, not required—even more so, it is not usable at all because the actual mathematical expressions will be generated by the randomisation module at run-time and, thus, will have to be evaluated at run-time as well.

ViennaMath is unable to evaluate integrals symbolically at runtime [27] and approximates them using numerical quadrature rules instead. The n-point Gaussian quadrature rule is displayed in Equation (3.5).

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{b-a}{2}\sum_{i=1}^{n} w_i f\left(\tfrac{b-a}{2}\xi_i + \tfrac{b+a}{2}\right) \tag{3.5}$$

For a polynomial f, the quadrature rule is exact if $\deg(f) \leq 2n-1$. Because the only inbuilt quadrature rule of ViennaMath is a 1-point Gaussian quadrature rule, it is clear that no *interesting* function can be integrated exactly using ViennaMath. In the case of $n = 1$, Equation (3.5) is simplified using $\xi_i = 0, w_i = 2$ to

$$\frac{b-a}{2}\sum_{i=1}^{n} w_i f\left(\tfrac{b-a}{2}\xi_i + \tfrac{b+a}{2}\right) = (b-a)\, f\left(\tfrac{b+a}{2}\right). \tag{3.6}$$

Therefore, calculating Equation (3.2) on page 15 using ViennaMath yields

$$\int_{-2}^{4} \left(-\frac{1}{2}x^2 + x + 4\right) \mathrm{d}x \approx 27 \tag{3.7}$$

instead of the correct, exact solution of 18.

Listing 3.6 on the following page illustrates the code required to evaluate the integral Equation (3.2) on page 15. Comparing the code that utilises ViennaMath with the code using GiNaC in Listing 3.5 on the facing page, one can easily see that ViennaMath's API is rather verbose. Many things are not hidden from the user for convenience: for example, no other library assessed requires to specify the integration limits using a call to a library function (`viennamath::interval`).

The LaTeX code produced by ViennaMath is displayed in Equation (3.8).

$$\int_{[-2;4]} -0.5 \cdot (x_0 \cdot x_0) + x_0 + 4\,\mathrm{d}x_0 = 27 \tag{3.8}$$

We are unsure whether symbolic variables necessarily have must have an index, but we have found no way to not add indices to variables.

As a side note, ViennaMath by default produces code that typesets the differential operator as an upright d, complying with ISO standard 80000-2:2019 [28]. No other library presented herein does.

```cpp
#include <iostream>

#include "viennamath/expression.hpp"
#include "viennamath/manipulation/latex.hpp"
#include "viennamath/runtime/numerical_quadrature.hpp"

int main()
{
  viennamath::variable x(0);
  viennamath::latex_translator to_latex;

  viennamath::numerical_quadrature integrator(new viennamath::gauss_quad_1());

  viennamath::expr integrand = -1/2.0*(x*x) + x + 4;
  viennamath::interval interval(-2.0, 4.0)
  viennamath::expr e = viennamath::integral(interval, integrand, x);

  std::cout << to_latex(e) << " = " << integrator(e)
            << std::endl;

  return 0;
}
```

Listing 3.6: C++ code to evaluate $\int_{-2}^{4} \left(-\frac{1}{2}x^2 + x + 4\right) \, \mathrm{d}x$ using ViennaMath.

Based on the fact that ViennaMath was last released almost 10 years ago at the time of writing combined with the inability to symbolically evaluate most integrals and its quite shallow documentation, ViennaMath is deemed not suitable as a library for the framework.

3.4.4 SymbolicC++

SymbolicC++ [17] is a CAS written in C++ and published under the GPL. The last release is version 3.35, which was published in September 2010.

SymbolicC++ is a general purpose CAS that can even integrate improper and indefinite integrals, and integration is not limited to polynomial integrands. It is a header-only library, making compilation both easier (because no libraries have to be linked into the

```cpp
#include <iostream>

#include "symbolicc++.h"

int main(void)
{
  Symbolic x("x");
  Symbolic integrand = -1/2.0 * Power(x,2) + x + 4;

  std::cout << integrate(integrand, x, -2, 4);

  return 0;
}
```

Listing 3.7: C++ code to evaluate $\int_{-2}^{4} \left(-\frac{1}{2}x^2 + x + 4\right) \mathrm{d}x$ using SymbolicC++.

main library) and slower (because each header file of SymbolicC++ must be parsed each time the main library is compiled).

Listing 3.7 shows the intuitive and concise API of SymbolicC++ using Equation (3.2) on page 15 as an example.

Despite being a fully featured CAS that has not seen any update for more than 10 years now, there seems to be no LaTeX output functionality. This, in addition to the lack of a parser or other ways to specify expressions at run-time, makes SymbolicC++ not usable as the sole symbolic computation library for the framework, but in conjunction with a library that can parse SymbolicC++'s output and display the result as LaTeX (such as SymEngine or GiNaC), SymbolicC++ could be a good substitute solution for providing the features other libraries do not, such as calculating antiderivatives or improper integrals.

SymbolicC++ does not provide LaTeX output nor a parser or other ways to specify expressions at run-time.

3.5 Choice of third-party libraries

The framework uses SymEngine as the main and Symbolic Lua as a temporary auxiliary library.

For the framework, it was decided to use SymEngine as the main library, alongside Symbolic Lua as a temporary, auxiliary library to take on the tasks that SymEngine is not yet capable of. For example, symbolic and numerical integration will be handled by Symbolic Lua until SymEngine is mature enough to take on that functionality as well.

To unify their APIs and use the framework's infrastructure (e.g., the *ExprRep* object as the return value), a thin wrapper was created for each third-party library. There is `wsymmath.lua`, which wraps Symbolic Lua and is written in Lua, and there is `wsymengine.so`, which is a thin wrapper over SymEngine and is written in C++, hence providing a C API so as to be usable from Lua. The actual symbolic computation module `symcomp.lua` provides methods such as `diff(f, x)` and `integrate(f, x [, l, u]]` and redirects to the specific wrapper for implementation. For example, `diff` is implemented by `wsymengine`, and `integrate` is implemented by `wsymmath`.

Because the randomisation and computation module are separate from each other, changes (such as replacing third-party libraries) can be done easily.

As explained in Chapter 2 on page 7, the fact that users of the symbolic computation module have no knowledge about the internals of the module allows for easier changes in the future. Once SymEngine is mature enough to replace Symbolic Lua, this change under the hood can be done without users of the symbolic computation module noticing or having to modify existing code.

The features implemented in the symbolic computation module are mainly concerned with symbolic derivation and integration, as well as manipulation of polynomials and the evaluation of functions at a given input value. Some matrix operations are also available although functionality to solve systems of equations is yet to be implemented. A complete overview of the implemented functions is given in Appendix A.1 on page 43.

Because all of the required infrastructure is already present, it should be rather easy to implement more functions in the computation module, that is to make more of SymEngine's functions (and Symbolic Lua's, where necessary) available.

Chapter 4

Randomisation

In the following, the two sets

$$\mathbb{Z}_{\mathrm{R}} := \{x \in \mathbb{Z} \mid \texttt{random.MIN\_INT} \leq x \leq \texttt{random.MAX\_INT}\}$$
$$\mathbb{R}_{\mathrm{R}} := \{x \in \mathbb{R} \mid \texttt{random.MIN\_REAL} \leq x \leq \texttt{random.MAX\_REAL}\}$$

denote the set of all possible integer and the set of all possible real numbers that can be generated by the randomisation module. The lower and upper limits are used to limit randomly generated numbers to a useful range and their default values are shown in Table 4.1.

Table 4.1: The default lower and upper limits for random numbers generated by the randomisation module.

Variable	Value
`MIN_INT`	-10
`MAX_INT`	10
`MIN_REAL`	-1
`MIN_REAl`	1

The randomisation module is responsible for creating individualised variants of user-specified mathematical expressions. It allows users to specify 'templates', that is, a general family of expressions with parameters. When compiling the same LaTeX document multiple times, the randomisation module will replace the parameters with actual, randomised values to create different mathematical expressions, taking into account any user-imposed constraints on parameters.

For example, Listing 4.1 on the next page shows how to specify the family of curves displayed in Equation (4.1).

$$f_{a,b}(x) = ax^2 + b \quad \text{with } a \in \mathbb{Z}_{\mathrm{R}} \text{ and } b \in \mathbb{Z}_{\mathrm{R}}, b \leq 10. \tag{4.1}$$

At compile-time, individualised 'instances' from the 'template' are generated by replacing the specified parameters with actual values; for example, the expression `"a*x^2+b"` from Listing 4.1 on the next page might be replaced by $3x^2 + 4$ or $-5x^2 + 9$.

The randomisation module replaces parameters of expressions when compiling the LaTeX file.

```lua
local constraints =
{
    { "a", random.integer() },
    { "b", random.integer(10) }
}

local expr = random.create("a*x^2+b", constraints)
```

Listing 4.1: Exemplary code showing how to create randomised versions of the expression ax^2+b with the constraints $a \in \mathbb{Z}_{\mathrm{R}}$ and $b \in \mathbb{Z}_{\mathrm{R}}, b \leq 10$ using the randomisation module.

The randomisation module does not have knowledge of the mathematical objects used by the computation module.

As mentioned in Chapter 2 on page 7, the randomisation module does not have any knowledge of the symbolic computation module. By clearly separating the randomisation module from the symbolic computation module, it is easier to make changes in either of the two. For example, changing the internals of the symbolic computation module (such as replacing one third-party library with another) does not entail any changes in the randomisation module.

The randomisation module generates numbers and strings which the computation module parses into objects.

Therefore, the randomisation module cannot generate actual mathematical objects such as matrices, but rather generates strings that represent those objects. These strings are then parsed by the symbolic computation module, which turns them into actual objects. In addition to that, the randomisation module also provides functions returning numbers (e.g., `random.integer` and `random.real`). A complete overview of all functions of the randomisation module is given in Appendix A.2 on page 49.

4.1 Persisting values between multiple compilation runs

As described in Chapter 2 on page 7, creating an assignment sheet for each student means compiling two versions, one including solutions and one without. Now, as described before, the randomisation module creates new randomised variants of mathematical expressions each time a document is compiled. What this means is that random values are generated once when compiling the version

without solutions, and *completely different* random values are generated when compiling the version with solutions. Hence, a method is needed to persist generated randomised values between different compilation runs so that compiling the same document for a single student multiple times does not produce different values each time. One way this could be done is by writing the randomised values into a file and reading those values from that file when compiling the version with solutions. However, this would be both fragile and lead to more cumbersome Lua code.

Random values must not change between compilations of the same document for a single student.

Lua's random number generator will generate the same sequence of pseudo-random numbers when the same seed is used [10]. The randomisation module does not need to be cryptographically secure, so we can use the deterministic behaviour of the random number generator to our advantage: by using a student's matriculation number as the seed for the random number generator, we can ensure that the same random values are generated for each student, regardless of the number of times a document is compiled. The matriculation number is passed as a command line argument to the LuaLaTeX compiler by the shell script, which reads a student's first and last name as well as the matriculation number from the list of participants of a course.

Seeding the random number generator with a student's matriculation number to get predictable, consistent random numbers.

4.2 Constructing good problems

The creation of a good mathematical problem usually includes carefully chosen numerical values so that the calculations are not unnecessarily complicated, allowing students to focus more on the techniques and strategies for solving problems of that kind.

By randomising the numerical values of a problem, it is likely that calculations may be more complicated compared with carefully selected values, which can create confusion and might distract students from focusing on problem solving strategies and techniques. Consider the simple example from an introductory linear algebra course shown in Figure 4.1 on the next page. The task is to find the eigenvalues of a matrix with integer entries. The entries are chosen so that all eigenvalues are also integer and so that calculations can be done without a calculator, allowing students to focus on practising the techniques and strategies for solving problems of that kind. By

Using randomised values instead of carefully chosen ones may lead to more complicated calculations and may distract students.

Task 1. Find the eigenvalues of the matrix

$$A = \begin{bmatrix} -2 & 1 \\ 12 & -3 \end{bmatrix}.$$

Solution: First, calculate $A - \lambda I_2$:

$$A - \lambda I_2 = \begin{bmatrix} -2 & 1 \\ 12 & -3 \end{bmatrix} - \begin{bmatrix} \lambda & \\ & \lambda \end{bmatrix} = \begin{bmatrix} -2-\lambda & 1 \\ 12 & -3-\lambda \end{bmatrix}.$$

Now, calculate the characteristic polynomial $\det(A - \lambda I_2)$:

$$\det(A - \lambda I_2) = (-2-\lambda)(-3-\lambda) - 1 \cdot 12 = \lambda^2 + 5\lambda - 6$$

Solving this yields the eigenvalues $\lambda_1 = -6$ and $\lambda_2 = 1$.

Figure 4.1: An example task from an introductory linear algebra course.

simply allowing the matrix entries to be randomly chosen integer values, the probability of the resulting matrix having even only one integer eigenvalue is almost zero, as shown in [29]. Therefore, it is probably not a good idea to simply generate random integer matrix entries.

Randomly choosing values from a set can be realised using `random.oneof`.

An easy way to bypass this problem is to use `random.oneof` to randomly choose from a set of matrices that are known to have integer eigenvalues, as shown in Listing 4.2 on the facing page. Obviously, this has the disadvantage that n different matrices with integer eigenvalues have to be known and specified in the code to be able to create n different, individualised versions of a document. Instead of saving time, users would then have to spend time looking for n matrices meeting the criteria of having integer entries and integer eigenvalues.

Using the features provided by the symbolic computation module, one idea is to multiply a matrix that is known to have integer eigenvalues by an integer. The code necessary for this is shown in Listing 4.3 on the next page.

```
local possible =
{
  "[-2, 1] [12, -3]", -- matrix has eigenvalues -6 and 1
  "[-5, -3] [-3, 3]", -- matrix has eigenvalues -6 and -4
  "[4, -5] [-5, 4]",  -- matrix has eigenvalues -1 and 9
}

local matrix = symcomp.matrix(random.oneof(possible))
```

Listing 4.2: Randomly choosing from matrices that are known to have integer eigenvalues.

```
local base =  symcomp.matrix("[-2, 1] [12, -3]")
local matrix = symcomp.scalarMul(random.integer(), base)
```

Listing 4.3: Creating randomised matrices with integer eigenvalues by scalar multiplication with a matrix that is known to have integer eigenvalues.

The remainder of this section is left to show that if a 2×2 matrix A has integer eigenvalues, so does kA for an integer k. The characteristic polynomial of kA is given by

$$\det(kA - \lambda I_2) = \lambda^2 - \operatorname{tr}(kA)\lambda + \det(k^2 A) \tag{4.2}$$
$$\det(kA - \lambda I_2) = \lambda^2 - k\operatorname{tr}(A)\lambda + k^2 \det(A) \tag{4.3}$$

since $\det(rA) = r\det(A)$ and $\operatorname{tr}(rA) = r\operatorname{tr}(A)$ for a matrix A and some scalar r.

The solutions of $\det(kA - \lambda I_2) = 0$, that is, the eigenvalues of kA are then given by

$$\lambda_{1,2} = \frac{k\operatorname{tr}(A)}{2} \pm \sqrt{\frac{(-k\operatorname{tr}(A))^2}{4} - k^2\det(A)} \tag{4.4}$$
$$= k\left(\frac{\operatorname{tr}(A)}{2} \pm \sqrt{\frac{\operatorname{tr}(A)^2}{4} - \det(A)}\right) \tag{4.5}$$

which shows that the eigenvalues of kA are still integer eigenvalues and, more concisely, are integer multiples of the eigenvalues of A.

4.3 Performance considerations

The randomisation module is implemented in Lua.

In contrast to the symbolic computation module, the randomisation module is implemented exclusively in Lua. This is mainly because there are no dependencies on any third-party libraries requiring to be integrated via C/C++, but also because we considered code written in Lua to be more maintainable. This choice does introduce a slight performance penalty because although Lua is a high-performance language, it still is an interpreted language and cannot be as fast as C/C++ code translated directly into machine code.

To evaluate the overhead of performance between Lua and C++, this section compares the performance between a Lua and a C++ implementation of the algorithm shown in 1, which generates strings in $\mathcal{O}(n)$ time that represent the polynomial

$$p(x) = \sum_{k=0}^{n} a_k x^k \quad \text{with } n \in \mathbb{N}, a_k \in \mathbb{Z}_{\mathtt{R}}. \tag{4.6}$$

Algorithm 1: The algorithm for creating string representations of polynomials.

input : the degree of the polynomial $n \geq 0$
output : a string representation of $\sum_{k=0}^{n} a_k x^k$

$s \leftarrow$ ""
$k \leftarrow 0$
while $k \leq n$ **do**
    $a_k \leftarrow$ randomInteger();
    $s \leftarrow s +$ toString(a_k) + "x^" + toString(k);
    $k \leftarrow k + 1$;
end
return s;

As can be seen in Figure 4.2 on the facing page, there is no big difference in performance for polynomials with degrees lower than or equal to 10: for example, it takes about 2.5 µs to create a polynomial of degree 6 in Lua as opposed to 1.3 µs in C++. The first noticeable difference occurs when generating polynomials of degree 25, where the Lua implementation takes about 10 µs compared with roughly 3 µs in C++.

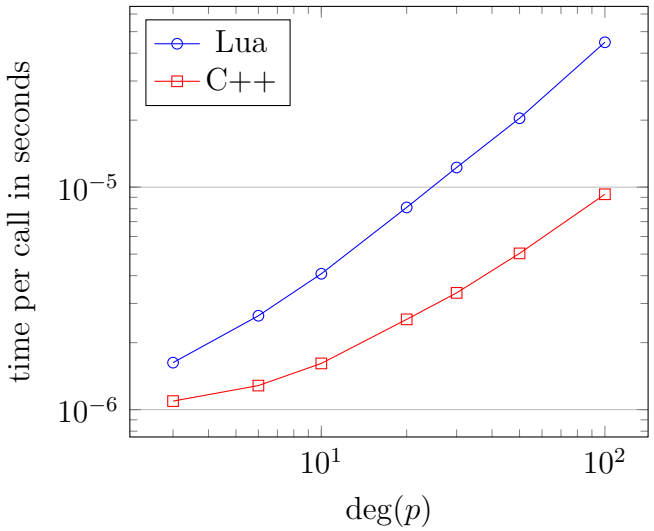


Figure 4.2: Visualisation of the time it takes to create polynomials of various degrees in Lua compared with in C++. Logarithmic scaling has been chosen for the y-axis to visualise the orders of magnitude because exact, actual numbers are not as relevant.

It is safe to say that because mathematical exercises will seldom have to deal with polynomials of degrees that high, it makes virtually no difference in performance whether one chooses Lua or C++. Hence, it was decided to implement the randomisation module exclusively in Lua to increase maintainability and to make the code more readable.

For the use case of the randomisation module, there is no significant difference in performance between a Lua and a C++ implementation.

Chapter 5

Evaluation

In this chapter, the framework will be evaluated in terms of aspects such as performance, code quality, scope of features and usability.

5.1 Quality assurance

As a substantial part of the project was the creation of software, various measures were put in place to ensure robust and maintainable code.

The wrapper for SymEngine has been written in C++ by adhering to the C++20 standard, and the C++ Core Guidelines proposed in [30] have been followed. They are a collection of rules, suggestions and proven programming best practices that aid programmers in creating safe, maintainable and modern C++ code. For example, no naked `new` and `delete` have been used, but rather, all heap allocations were managed by using smart pointers.

C++20 and the C++ Core Guidelines have been used to create maintainable and modern code.

As the supporting implementation of the C++ Core Guidelines, the GSL Guidelines Support Library provided by Microsoft has been used extensively, as has the static analyser clang-tidy to enforce the use of the C++ Core Guidelines and to diagnose various other programming errors, style violations and bugs.

Microsoft's implementation of the GSL can be found at https://github.com/Microsoft/GSL

To ensure correct behaviour of the code, unit tests have been written. The term *unit testing* means testing small components (units) of a software project in isolation to check whether each unit is working as intended. Unit tests are meant to improve the quality of code and to find defects and bugs early. In addition to this, they also serve

Unit tests are used to ensure the correct behavior of code.

```
TEST_CASE("Solve correctly solves an equation")
{
    auto f = "x^2-4";
    auto x = "x";

    auto solutions = symcomp::Solve(f, x);

    CHECK(solutions.size() == 2);       // count
    CHECK(solutions[0].Basic == "-2"); // value and order
    CHECK(solutions[1].Basic == "2");  // value and order
}
```

Listing 5.1: A unit test checking whether the equation $x^2 - 4 = 0$ is solved correctly, which means that the number of real solutions is 2, the solution set is $\{-2, 2\}$ and that the solutions are returned in ascending order, i.e., $x_1 < x_2$.

as documentation and allow for easier modifications to the code because they prove that the individual parts work correctly, and if the tests were successful before and after some changes, it is safe to say that these changes did not accidentally introduce bugs. Finally, unit tests also lead to better code because they force functions to be written so that they can be tested easily, which, amongst others, means that they should be adhering to the single-responsibility principle, thus only doing one thing instead of everything at once.

Most of the functions implemented in C++ have been written along with accompanying unit tests. Listing 5.1 shows an example unit test for a function from the SymEngine wrapper. The doctest[4] framework has been used for this as it is header only and easy to integrate into projects. It also claims to be the C++ unit test framework with the fastest compile-time [31].

[4] The header file is available at https://github.com/onqtam/doctest

There are also tests for all Lua functions implemented in the symbolic computation module. No test framework is used here, but instead, simple assertions are used for checking whether the functions return the expected output for a given input.

In general, assertions are often used both in Lua and in C++ code to check whether internal assumptions (mostly preconditions) hold. For example, from Lua's point of view, it is perfectly acceptable

to call any function with any number of parameters. A function `add(a, b)` can be called as `add(1, 2, 3, 4)` without Lua throwing an error. To fail fast and provide helpful error messages to developers, functions requiring exactly k parameters check for that exact parameter count in the C++ part of the symbolic computation module.

Code comments are used in Lua and C++ to explain ideas and to state a function's input and output.

The documentation of code is another very important thing to keep code maintainable in the future, especially once other contributors join. In addition to code comments explaining the intent and giving insights into complex places, all Lua functions contain a comment describing what this function does and what inputs it expects. All C++ functions contain a Doxygen-style comment in the header.

Further, the APIs of the randomisation and computation module are documented in Markdown with descriptions, usage examples and possible errors.

5.2 Usability of the framework

Users interact with the framework on different 'layers' as shown in Figure 2.2 on page 9. Individualised assignments are generated by including mathematical problems from the task pool into one LaTeX document and compiling this document for a list of students. The problems in the task pool are isolated LaTeX files which each represent one mathematical task and use the framework's randomisation and symbolic computation features. They are written just like any other LaTeX files but with the added opportunity to perform computations directly in the file. Example task files are shown in Appendix B on page 53.

The computation module needs to provide more features so that task files for more mathematical topics can be created.

A limiting factor when creating task files is the scope of features provided by the symbolic computation module. Currently, functions are implemented to evaluate, differentiate and integrate expressions and to do basic matrix operations such as calculating the determinant and finding its eigenvalues.

The creation of exercise sheets happens without users having to interact with the randomisation or symbolic computation modules directly. Instead, exercise sheets simply `\input` exercises from the

task pool, as shown in Listing 2.1 on page 8. Then, after downloading a list of the participants of a course from an e-learning platform such as Moodle, users can generate individualised versions of the assignments sheets: one version for each participant consisting of a PDF file without solutions and one PDF file of the assignment with solutions included. The generation of the individualised versions occurs by calling a shell script.

The `.csv` files is parsed in Bash, which is rather fragile and could be improved by using tools designed for handling `.csv` files.

This is where improvements could be made: as of now, the `.csv` file containing the participants is parsed directly in Bash. Bash is not the most suitable tool for `.csv` file parsing, and the script relies on the fields in the file to be in a specific order. A more sophisticated approach, one probably using other tools for the `.csv` file parsing, would provide improved robustness.

Each LaTeX file is compiled twice in an attempt to correctly generate references, which might not always be correct. Using tools such as `latexmk` that can automatically figure out the number of compilations required would be an improvement.

Another improvement can be made in the shell script, too. For the actual creation of the exercise sheets, the script calls `lualatex` twice for each document to create and use `.aux` files, for example, to correctly display a table of contents or to make referencing labels work. Compiling twice may not always be the right number of compilation runs. In some cases, compiling once can suffice, while in other cases, compiling only twice does not produce a correct, finished PDF file. Using `latexmk` to figure out the required number of compilation runs would improve performance in the first case and ensure the creation of correct files in the latter case.

5.3 Performance

Figure 5.1 on the next page shows the overhead of using framework functions a certain number of times in a LaTeX file to generate a random polynomial of degree 5 and to calculate its derivative as shown in Listing 5.2 on the facing page. The overhead was evaluated by measuring the time it takes to compile this document and subtracting from that the time it takes to compile a document with as many hard-coded values that does not use the framework. Two things can be observed: first, the overhead is virtually constant over the range of 10 to 1000 function calls. And second, the overhead in general is tolerable, being approximately 0.4 s for documents with 10 to 200 function calls and even below 1 s for documents with 10 000 function calls.

The overhead is tolerable, being around 0.5 s to create a document where the framework is used between 10 and 1000 times.

```
for i=1,limit do
  p = random.polynomial(5)
  diff = symcomp.diff(p)
  tex.print("$" .. diff .. "$")
end
```

Listing 5.2: Lua code to generate random polynomials of degree 5 and to calculate their derivatives a certain number of times.

Consider the following realistic scenario: compiling an assignment sheet consisting of 3 task files that each use the randomisation and symbolic computation module a multitude of times for 50 students took 3 minutes and 30 seconds, which means around 4 seconds per student to create one version with solutions and one without while compiling each twice to account for references. This shows that compiling a LaTeX file once takes 1 s with the overhead of about 0.4 s already included. We therefore consider the overhead of using the framework justifiable, especially when taking into account the amount of time saved when creating individualised assignments simply by choosing from the task pool and executing the shell script.

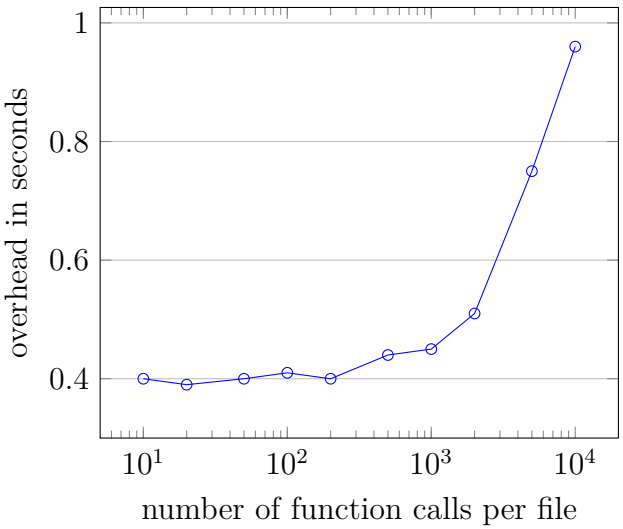


Figure 5.1: Visualisation of the compilation overhead when using the framework. The x-axis displays the number of times the derivative of a random polynomial of degree 5 is calculated in a LaTeX file.

Chapter 6

Conclusion and outlook

The goal of this thesis was to develop a framework for the creation of individualised mathematical assignments in LaTeX. As opposed to approaches using external tools such as MATLAB for the computation and individualisation of tasks, the framework is intended to support users in writing standard LaTeX code and to provide them with features for the symbolic evaluation and randomisation of mathematical expressions directly in the LaTeX files.

The framework presented herein serves as a prototype showing the possibility of integrating symbolic computation and randomisation features into LaTeX. It consists of one module for randomising mathematical expressions and one module for symbolic computation. Those modules are implemented in Lua and C++ and provide functions that are intended to be used from within LaTeX by using the LuaLaTeX compiler. The symbolic computation module makes available the functionality provided by existing third-party software libraries for symbolic computation to Lua code.

In its current state, the framework can be used to create individualised assignments for a few topics in undergraduate mathematics. Given a LaTeX file and a list of students that can be exported from e-learning management systems such as Moodle, the process of creating individualised versions happens automated.

The framework is provided as free and open-source software licensed under the MIT license, thus imposing minimal restrictions on user and allowing for easy modifying and extending of the framework. All code can be found at https://github.com/flinkow/individualised-assignments-latex.

6.1 Future work

Future work mainly focuses on extending the provided functions for symbolic computation and to add more files to the task pool to be able to generate assignments covering more mathematical topics. A task type that can profit from individualisation is solving systems of linear equations. Currently, no function is implemented to solve systems of linear equations.

Implementing more functions for symbolic computation should be a straightforward task because the framework was designed to be easily extendable. The basic infrastructure for passing values from LaTeX to Lua and C++ and the way back is present and can be used to make more features a third-party symbolic computation library offers accessible to LaTeX.

There also needs to be more integration with e-learning management systems. As of now, the generated individualised assignments are not automatically sent to the students or uploaded to Moodle. Because the list of participants that can be exported from Moodle also includes the students' email addresses, a first step could be to automatically send the assignments to the students by email.

Another useful step could be trying to port the framework to Windows to make its functionality available to more users.

6.2 Outlook

A necessary action is to gather feedback both from teaching academics using the framework when preparing assignments and from students working on assignments generated using the framework. It must be verified that the framework is indeed easy to use and allows teaching staff to save time when creating assignments. Equally as important is evaluating whether students benefit from individualised assignments of the form the framework generates and where improvements could be made.

Appendix A

Function reference

A.1 Symbolic computation

The symbolic computation module `symcomp.lua` currently provides the following functions.

To use these functions, one has to load the module first, for example by using `symcomp = require "symcomp"`.

A.1.1 Evaluating a function at a point

The function `symcomp.evalAt(f, x, x0)` evaluates a function at the specified point x_0.

```
f = symcomp.expr("-1/2*x^3")
res = symcomp.evalAt(f, "x", 1) -- result: -1/2
```

Listing A.1: Lua code to evaluate a function $f(x) = -\frac{1}{2}x^3$ at $x = 1$.

A.1.2 Subtracting objects from another

The function `symcomp.sub(a, b)` calculates $a - b$ where a and b can be polynomials or other mathematical objects.

```
f = symcomp.expr("2*x^4+7*x-3")
g = symcomp.expr("3*x^3-2*x+7")

res = symcomp.sub(f, g) -- result: 2*x^4-3*x^3+9*x-10
```

Listing A.2: Lua code to calculate $f - g$ with $f(x) = 2x^4 + 7x - 3$ and $g(x) = 3x^3 - 2x + 7$.

A.1.3 Finding roots of a function

The function `symcomp.solve(f, x)` calculates $f(x) = 0$.

```
f = symcomp.expr("x^2-4")
res = symcomp.solve(f, "x") -- result: -2 and 2
```

Listing A.3: Lua code to calculate $f(x) = 0$ with $f(x) = x^2 - 4$.

A.1.4 Calculating a derivative

The function `symcomp.diff(f, x)` calculates $\frac{\mathrm{d}}{\mathrm{d}x} f(x)$.

```
f = symcomp.expr("-1/2*x^3")
res = symcomp.diff(f, "x") -- -*x^2
```

Listing A.4: Lua code to calculate $\frac{\mathrm{d}}{\mathrm{d}x} f(x)$ with $f(x) = -\frac{1}{2}x^3$.

A.1.5 Evaluating an integral

The function `symcomp.integrate(f, x, [l, u])` calculates either the indefinite integral $\int f(x)\,\mathrm{d}x$ or the definite integral $\int_l^u f(x)\,\mathrm{d}x$ if l and u are given.

```
f = symcomp.expr("1/3*x^2")

indefinite = symcomp.integrate(f, "x") -- result: x^3/9
definite = symcomp.integrate(f, "x", 0, 3) -- result: 3
```

Listing A.5: Lua code to calculate the indefinite integral of $f(x) = \frac{1}{3}x^2$ and the definite integral of f from 0 to 3.

A.1.6 Creating an identity matrix

The function `symcomp.identityMatrix(s)` returns the identity matrix of the specified size.

```
i3 = symcomp.identityMatrix(3)
```

Listing A.6: Lua code to create the identity matrix $I_3 = \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix}$.

A.1.7 Creating a matrix

The function `symcomp.matrix(str)` returns a matrix with the specified entries. Note that a $m \times n$ matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

needs to be entered as

```
"[a_11, a_12, ..., a_1n] [a_21, a_22, ..., a_2n] ...
↪  [a_m1, a_m2, ..., a_mn]"
```

```
m = symcomp.matrix("[1, 2] [3, 4]")
```

Listing A.7: Lua code to create the matrix $m = \left[\begin{smallmatrix} 1 & 2 \\ 3 & 4 \end{smallmatrix}\right]$.

A.1.8 Subtracting one matrix from another

The function `symcomp.matrixSub(A, B)` calculates $A - B$ where A and B are matrices.

```
A = "[1, 2] [3, 4]"
B = "[0, 1] [0, 2]"

res = symcomp.matrixSub(A, B) -- result: [1, 1] [3, 2]
```

Listing A.8: Lua code to calculate $A - B$ with $A = \left[\begin{smallmatrix} 1 & 2 \\ 3 & 4 \end{smallmatrix}\right]$ and $B = \left[\begin{smallmatrix} 0 & 1 \\ 0 & 2 \end{smallmatrix}\right]$.

A.1.9 Multiplying a matrix by a scalar

The function `symcomp.scalarMul(k, A)` multiplies the matrix A by the scalar k.

```
A = "[1, 2] [3, 4]"
k = 4

res = symcomp.scalarMul(k, A) -- result: [4, 8] [12, 16]
```

Listing A.9: Lua code to calculate kA with $A = \left[\begin{smallmatrix} 1 & 2 \\ 3 & 4 \end{smallmatrix}\right]$ and $k = 4$.

A.1.10 Calculating the determinant of a matrix

The function `symcomp.det(A)` returns the determinant of the matrix A.

```
A = "[1, 2] [3, 4]"

d = symcomp.det(A) -- result: -2
```

Listing A.10: Lua code to calculate $\det(A)$ with $A = \left[\begin{smallmatrix} 1 & 2 \\ 3 & 4 \end{smallmatrix}\right]$.

A.1.11 Finding eigenvalues of a matrix

The function `symcomp.eigenvalues(A)` returns the eigenvalues of the matrix A.

```
a = "[-5, -3] [-3, 3]"

evs = symcomp.eigenvalues(a) -- result: -6 and 4
```

Listing A.11: Lua code to find the eigenvalues of $A = \left[\begin{smallmatrix} -5 & -3 \\ -3 & 3 \end{smallmatrix}\right]$.

A.1.12 Printing the intersections of two functions

`symcomp.printintersections(f, g, x)` returns a string listing the intersections of the two functions f and g, handling cases where the functions do not intersect, intersect at one, two or more points.

```
s = symcomp.printintersections("x^2", "x+6", "x")
print(s)
```

Listing A.12: Lua code printing the intersections of two functions that intersect at two points. The resulting string will be The functions intersect at $P_1(-2, 4)^T$ and at $P_2(3, 9)^T$.

```
s = symcomp.printintersections("x+4", "x+2", "x")
print(s)
```

Listing A.13: Lua code printing the intersections of two functions that do not intersect. The resulting string will be The functions do not intersect.

A.1.13 Printing the zeros of a function

The function `symcomp.printzeros(f, x)` returns a string listing the zeros of a function f, handling the cases where the function does not have zeros, has one, two, or more zeros.

```
s = symcomp.printzeros("x^2-4", "x")
print(s)
```

Listing A.14: Lua code printing the zeros of a function that has two zeros. The resulting string will be The function has zeros at $x_1 = -2$ and at $x_2 = 2$.

```
s = symcomp.printzeros("4", "x")
print(s)
```

Listing A.15: Lua code printing the zeros of a function that does not have zeros. The resulting string will be The function does not have zeros.

A.1.14 Printing the extremal points of a function

The function `symcomp.printminmax(f, x)` returns a string listing the minima and maxima of a function f, handling the cases where the function does not have extremal points, has one, two, or more.

```
s = symcomp.printminmax("x^2+4x", "x")
print(s)
```

Listing A.16: Lua code printing the extrema of a function. The resulting string will be The function has a minimum at $(-2, 4)$ because $f(x_1) > 0$ where x_1 is a zero of f'.

A.1.15 Printing the eigenvalues of a matrix

The function `symcomp.printeigenvalues(A)` returns a string listing the eigenvalues of the matrix A.

```
A = symcomp.matrix("[0, 1] [-2, -3]")
s = symcomp.printeigenvalues(A, "lambda")
print(s)
```

Listing A.17: Lua code printing the eigenvalues of a matrix. The resulting string will be The matrix has the eigenvalues $\lambda_1 = -2$ and $\lambda_2 = -1$.

A.2 Randomisation

The randomisation module `random.lua` currently provides the following functions.

To use these functions, one has to load the module first, for example by using `random = require "random"`.

A.2.1 Creating an expression

The function `random.expr(ex, rules)` creates a randomised version of the specified expression while respecting the given constraints.

```
local rules =
{
    { "a", random.rational() },
    { "n", random.integer(2, 5) },
    { "b", random.oneof(4, 6, 10) }
}

e = random.create("a*x**n+b", rules)
```

Listing A.18: Lua code to create the expression $ax^n + b$ with random values $a \in \mathbb{Q}_R, n \in \mathbb{Z}_R, 2 \leq n \leq 5$ and $b \in \{4, 6, 10\}$.

A.2.2 Choosing from a list of supplied values

The function `random.oneof(t)` randomly chooses from the values in the specified table.

```
v = random.oneof({ "1/2*x", "2/3*x", "1/4*x" })
```

Listing A.19: Lua code to randomly choose from $\frac{1}{2}x^2$, $\frac{2}{3}x$ and $\frac{1}{4}x$.

A.2.3 Random integers

The function `random.integer([a [, b]])` generates a random integer. If a is specified, the maximum generated value is a, and if a and b are specified then the generated value is between a and b. If b is not specified, it will be `random.MIN_INT=-10`, and if a is not specified, it will be `random.MAX_INT=10`.

```
x = random.integer() -- [random.MIN_INT, random.MAX_INT]
x = random.integer(10)      -- [random.MIN_INT, 10]
x = random.integer(-5, 20) -- [-5, 10]
```

Listing A.20: Lua code to generate random integers.

A.2.4 Random fractions

The function `random.rational()` generates a rational number $r \in [0, 1]$ as a string containing a fraction.

```
x = random.rational() -- for example, "4/5" or "-1/2"
```

Listing A.21: Lua code to generate random integers.

A.2.5 Random real numbers

The function `random.real([a [, b]])` generates a random floating point number. If a is specified, the maximum generated number is a, and if a and b are specified, the generated value is between a and b. If b is not specified, it will be `random.MIN_REAL=-1`, and if a is not specified, it will be `random.MAX_REAL=1`.

```
x = random.real() -- [random.MIN_REAL, random.MAX_REAL]
x = random.real(3.5)       -- [random.MIN_REAL, 3.5]
x = random.real(-2.9, 1.2) -- [-2.9, 1.2]
```

Listing A.22: Lua code to generate random real numbers.

A.2.6 Random lines

The function `random.line([m [, b]])` generates a random line equation $mx + b$ with slope m and y-axis intercept b. If m is specified but b is not, then b will be `random.integer()`. If m is also not specified, m will be `random.rational()`.

```
l = random.line()
l = random.line(4.5)
l = random.line(4.5, -10)
```

Listing A.23: Lua code to generate random lines.

A.2.7 Random parabolas

The function `random.parabola([a])` generates a random parabola ax^2. If a is not specified, it will be `random.rational()`.

```
l = random.parabola()
l = random.parabola(0.5)
```

Listing A.24: Lua code to generate random parabolas.

A.2.8 Random polynomials

The function `random.polynomials(deg)` generates random polynomials of the specified degree.

```
p = random.polynomial(4)
```

Listing A.25: Lua code to generate random polynomials.

Appendix B

Example tasks

```
\begin{luacode*}
  symcomp = require "symcomp"
  random = require "random"

  local baseMatrices = -- integer matrices with integer
  ↪  eigenvalues
  {
    "[-2, 1] [12, -3]",
    "[-5, -3] [-3, 3]",
    "[4, -5] [-5, 4]"
  }

  local base = symcomp.matrix(random.oneof(baseMatrices))
  A = symcomp.scalarMul(random.integer(-5, 5), base)

  I = symcomp.identityMatrix(2)
  lambdaI = symcomp.scalarMul("lambda", I)
  AminusLambdaI = symcomp.matrixSub(A, lambdaI)

  det = symcomp.det(AminusLambdaI)
\end{luacode*}

\question
Find all eigenvalues and eigenvectors of the matrix \[A =
↪  \scprint{A}\].

\begin{solution}

Calculate $A-\lambda I_2$:
\[A-\lambda I_2 = \scprint{A} - \lambda\scprint{I} =
↪  \scprint{AminusLambdaI}.\]
Then, calculate $\det(A-\lambda I_2)$. \[\det(A-\lambda
↪  I_2) = \scprint{det}\]
Now, we solve $\det(A-\lambda I_2)=0$.
↪  \scprint{symcomp.printeigenvalues(A, "A")}

\end{solution}
```

Listing B.1: The source file of the example task shown in Figure B.1 on the facing page. Note the Lua function calls directly in the LaTeX file, making use of the symbolic computation and randomisation modules the framework provides.

1. Find all eigenvalues and eigenvectors of the matrix

$$A = \begin{bmatrix} 2 & -1 \\ -12 & 3 \end{bmatrix}.$$

Solution:
Calculate $A - \lambda I_2$:

$$A - \lambda I_2 = \begin{bmatrix} 2 & -1 \\ -12 & 3 \end{bmatrix} - \lambda \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix} = \begin{bmatrix} 2 - 1.0\lambda & -1.0 \\ -12.0 & 3 - 1.0\lambda \end{bmatrix}.$$

Then, calculate $\det(A - \lambda I_2)$.

$$\det(A - \lambda I_2) = -12.0 + (3 - 1.0\lambda)(2 - 1.0\lambda)$$

Now, we solve $\det(A - \lambda I_2) = 0$.
The matrix A has the eigenvalues $\lambda_1 = -1$ and $\lambda_2 = 6$.

Figure B.1: An example task dealing with eigenvalues.

1. How big is the parabolic segment between the parabola $f(x) = \frac{x^2}{2}$ and the line $g(x) = 4 + x$? Sketch a graph to visualize the desired area.

Solution: The functions intersect at $P_1(-2, 2)^T$ and at $P_2(4, 8)^T$. Thus, the area is

$$A = \int_{-2}^{4} g(x) - f(x)\,\mathrm{d}x = \int_{-2}^{4} 4 + x + \left(\frac{-1}{2}\right) x^2\,\mathrm{d}x = \left[\frac{1}{6}x\left(24 + 3x - x^2\right)\right]_{-2}^{4} = -16 + 3x^2 - 0.5x^3$$

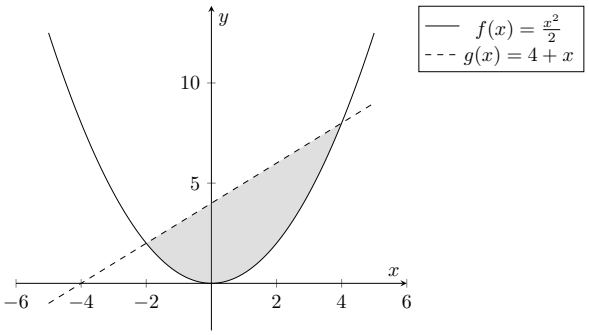


Figure B.2: An example task dealing with integration.

1. Given the function

$$f(x) = -6x^2 - x^3$$

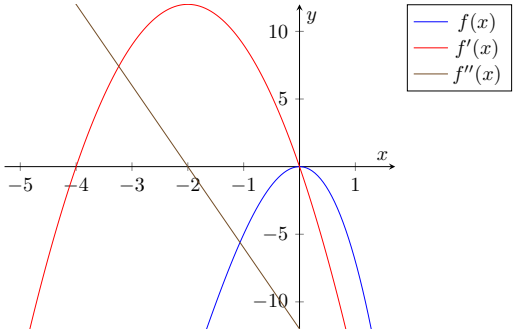
(a) Sketch f, f' and f'' in one coordinate system.

(b) Identify all of the minimum and maximum points and find its inflection points.

Solution:

(a) First, calculate the derivatives

$$f(x) = -6x^2 - x^3$$
$$f'(x) = -12x - 3x^2$$
$$f''(x) = -12 - 6x$$
$$f'''(x) = -6$$



(b) The function f has zeros at $x_1 = -6$ and at $x_2 = 0$. The function f' has zeros at $x_3 = -4$ and at $x_4 = 0$. The function f has a minimum at $(-4, -32)$ because $f''(x_3) > 0$ and a maximum at $(0, 0)$ because $f''(x_4) < 0$.

Figure B.3: An example task dealing with derivation.

Bibliography

[1] M. Rosser *et al.*, 'Using Excel to individualise basic mathematics assignments', *Computers in Higher Education Economics Review*, vol. 20, no. 1, pp. 13–19, 2008.

[2] P. Voumard, 'Computer based individualised assignment system', in *Proceedings IEEE 1st International Conference on Multi Media Engineering Education*, 1994, pp. 249–254. DOI: 10.1109/MMEE.1994.383208.

[3] N. Hunt, 'Individualised assessment in statistics', *Assessment methods in statistical education: An international perspective*, p. 203, 2010.

[4] U. Ziegenhagen, 'Creating and automating exams with LaTeX & friends', *TUGboat*, vol. 40, no. 2, pp. 163–166, 2019.

[5] G. M. Poore, 'PythonTeX: Reproducible documents with LaTeX, python, and more', *Computational Science & Discovery*, vol. 8, no. 1, p. 014 010, Jul. 2015. DOI: 10.1088/1749-4699/8/1/014010.

[6] U. Ziegenhagen, 'Combining LaTeX with Python', *TUGboat*, vol. 40, no. 2, pp. 126–128, 2019.

[7] S. W. Al-Safi, 'Randomising assignments with SageTeX', *TUGboat*, vol. 37, no. 1, pp. 25–27, 2016.

[8] A. Helfrich-Schkarbanenko, K. Rapedius, V. Rutka and A. Sommer, *Mathematische Aufgaben und Lösungen automatisch generieren*. Springer Berlin Heidelberg, 2018. DOI: 10.1007/978-3-662-57778-3.

[9] M. Magdowski, 'Automatically generate personalized tasks and sample solutions for the fundamentals of electrical engineering with PGFPLOTS and CircuiTikZ ', *TUGboat*, vol. 42, no. 2, pp. 159–163, 2021.

[10] R. Ierusalimschy, *Programming in Lua*, 4th ed. lua.org, 2016, ISBN: 978-8590379867.

[11] J. I. Montijano, M. Pérez, L. Rández and J. L. Varona, 'Numerical methods with LuaLaTeX', *TUGboat*, vol. 35, no. 1, pp. 51–56, 2014.

[12] T. Flinkow and J. Vorloeper, 'Numerisches Rechnen mit LuaLaTeX', 2021, Manuscript in preparation for *Die TeXnische Komödie (DTK)*.

[13] P. Zimmermann, A. Casamayou, N. Cohen, G. Connan, T. Dumont, L. Fousse, F. Maltey, M. Meulien, M. Mezzarobba, C. Pernet *et al.*, *Computational mathematics with SageMath*. SIAM, 2018.

[14] R. Albrecht, B. Buchberger, G. E. Collins and R. Loos, *Computer Algebra: Symbolic and Algebraic Computation*. Springer Science & Business Media, 2012, vol. 4.

[15] J. S. Cohen, *Computer Algebra and Symbolic Computation: Mathematical Methods*. CRC Press, 2003.

[16] M. Bronstein, 'Σ^{IT} — A strongly-typed embeddable computer algebra library', in *Design and Implementation of Symbolic Computation Systems*, J. Calmet and C. Limongelli, Eds., Springer Berlin Heidelberg, 1996, pp. 22–33, ISBN: 978-3-540-70635-9.

[17] Y. Hardy, K. S. Tan and W.-H. Steeb, *Computer Algebra with SymbolicC++*. Singapur: World Scientific, 2008, ISBN: 978-9-812-83360-0.

[18] L. Liberti, 'Ev3: A Library for Symbolic Computation in C++ using n-ary Trees', 2003. [Online]. Available: http://www.lix.polytechnique.fr/Labo/Leo.Liberti/Ev3.pdf.

[19] C. E. Moore, *Symmath-lua: Computer Algebra System written in Lua*. [Online]. Available: https://github.com/thenumbernine/symmath-lua#todo (visited on 3rd Oct. 2021).

[20] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, Š. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman and A. Scopatz, 'SymPy: Symbolic computing in

Python', *PeerJ Computer Science*, vol. 3, e103, Jan. 2017. DOI: 10.7717/peerj-cs.103.

[21] T. C. Molteno, 'SympyTEX: Embedding Symbolic Computation Into LATEX Documents', University of Otago, Tech. Rep. 2014-1.

[22] W. Stein and D. Joyner, 'Sage: System for algebra and geometry experimentation', *Acm Sigsam Bulletin*, vol. 39, no. 2, pp. 61–64, 2005.

[23] SymPy Development Team. (2021). 'SymPy Roadmap', [Online]. Available: https://www.sympy.org/en/roadmap.html (visited on 27th Aug. 2021).

[24] C. Bauer, A. Frink and R. Kreckel, 'Introduction to the GiNaC Framework for Symbolic Computation within the C++ Programming Language', *Journal of Symbolic Computation*, vol. 33, no. 1, pp. 1–12, 2002, ISSN: 0747-7171. DOI: 10.1006/jsco.2001.0494.

[25] J. G. U. Mainz, *GiNaC 1.8.1: An open framework for symbolic computation within the C++ programming language*, 8th Aug. 2021. [Online]. Available: https://www.ginac.de/tutorial.pdf.

[26] K. Rupp, F. Rudolf and J. Weinbub, 'A discussion of selected Vienna-libraries for computational science', *Proceedings of C++ Now (2013)*, 2013.

[27] K. Rupp, *ViennaMath 1.0.0*, 2012. [Online]. Available: http://viennamath.sourceforge.net/viennamath-manual-current.pdf.

[28] International Organization for Standardization, 'ISO 80000-2:2019: Quantities and units – Part 2: Mathematics', Standard ISO 80000-2:2019, Aug. 2019. [Online]. Available: https://www.iso.org/standard/64973.html.

[29] G. Martin and E. B. Wong, 'Almost all integer matrices have no integer eigenvalues', *The American Mathematical Monthly*, vol. 116, no. 7, pp. 588–597, 2009.

[30] B. Stroustrup and H. Sutter, 'C++ Core Guidelines', 2019. [Online]. Available: https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines (visited on 30th Sep. 2021).

[31] V. Kirilov, *Doctest Benchmarks*, Jul. 2019. [Online]. Available: https://github.com/onqtam/doctest/blob/master/doc/markdown/benchmarks.md (visited on 5th Oct. 2021).