A guide to the lecture notes for MAT-INF11001(L)

Knut Mørken

November 16, 2014

This document is a navigation tool for reading *Numerical Algorithms and Digital Representation*, tailored to students in MAT-INF1100. Hopefully, this will simplify the task of reading and studying the notes.

Chapter 1: Introduction

This chapter provides background information and context for the rest of the lecture notes. For those who are not familiar with programming, sections 1.3–1.5 contain a simple introduction to the pseudo language that is used in later chapters for describing algorithms.

About numbers

From *Kalkulus* we know that there are both rational and irrational numbers. For normal purposes it is usually best to view the numbers as expansions in the decimal numeral system. Some rational numbers can be represented with a finite number of digits, but most real numbers require an infinite number of digits. This is the major challenge when we represent numbers in a computer: Because of finite resources we can only represent numbers with a finite number of digits. And we can only represent a finite collection of numbers.

Chapter 2: 0 and 1

The core of this chapter is the simple fact that if we want robust communication in the presence of noise it is wise to code information in terms of as few symbols as possible. Since we need at least two symbols to get something interesting, it's a good idea to use exactly two symbols, for example 0 and 1. This is one of the main reasons for representing information in a computer in terms of 0s and 1s.

Chapter 3: Numbers and Numeral Systems

We are used to thinking of real numbers as decimal numbers, that is, numbers represented as a sequence of digits with a point somewhere in the expansion, as in

$$0.1,\ 4.6,\ 3.14,\ 354.65878723,\ 0.33333333\cdots,\ 1.414213562373095\cdots$$

The last two numbers we recognise as $1/3$ and $\sqrt{2}$—the $\cdots$-notation indicates that there are infinitely many digits to the right which means that we can never write these numbers accurately as decimal numbers.

The decimal numeral system uses 10 as the base and makes use of the 10 digits 0–9. In chapter 3 we show that there is nothing magical about 10—numbers may be represented in any integer base $\beta > 1$. Regardless of the base, irrational numbers will always require infinitely many digits to the right of the point. For rational numbers however, it is not completely obvious which numbers require infinitely many digits to the right of the point; this actually depends on the base β, see Lemma 3.22. In particular, the only rational numbers that can be represented with a finite number of digits in base $\beta = 2$ are numbers in the form $a/2^n$ for integers a and n. This means that numbers like 0.1 and 3.4 can *not* be represented exactly in standard computer hardware.

Chapter 4: Computers, Numbers, and Text

This chapter describes how integers, decimal numbers and text are represented in a computer. This is quite factual, but is based on the following simple ideas:

1. The hardware of a computer is built to handle groups of 32 or 64 bits (4 or 8 bytes) as a unit.
2. Integers represented with 32 bits use 31 bits for the digits and 1 bit for the sign, and similarly for 64-bit integers. The detailed representation is based on so-called two's complement.
3. Real numbers are represented in binary normalised form, see section 4.2.2, these are called floating-point numbers. For 32 bit floating-point numbers 23 bits are used for the significand and 11 bits for the exponent. For 64 bit numbers 53 bits are used for the significand (the digits) and 11 bits for the exponent (the size of the number).
4. Text is represented as a sequence of characters. A character is represented as an integer that points into a table; the table lists all the characters available for this particular character encoding, see section 4.3.
5. The Unicode table contains virtually all the characters that are in use today, see section 4.3.3.

Chapter 5: Computer Arithmetic and Round-Off Errors

Section 5.1 gives a simple description of what can go wrong in integer arithmetic. The arithmetic of floating-point numbers is more challenging, and this is described in section 5.2. The key observation is that subtraction of two almost equal numbers may lead to a result with very few, if any, correct digits, see section 5.2, and in particular example 5.12.

There are different ways to represent errors and this is discussed in section 5.3. The most important observation here is that the relative error provides a natural way to measure the number of correct digits in an approximation, see observation 5.20.

Two expressions that are mathematically equivalent may behave very differently on a computer. In section 5.4 we discuss how some expressions may be rewritten to be less sensitive to round-off errors.

Chapter 6: Numerical Simulation of Difference Equations

This is the first chapter that discusses numerical computations. The chapter overlaps with the chapter in *Kalkulus* about difference equations so you do not necessarily need to read everything. The core is the discussion of round-off errors in section 6.5.

1. Section 6.1 is just a short introduction to equations in general, may be skipped.
2. Section 6.2 define difference equations, more generally than in *Kalkulus*, and also discusses different types of difference equations. In particular, linear difference equations are defined. Observation 6.8 makes an important distinction between simulating a difference equation and finding an explicit formula for the solution.
3. Section 6.3 discusses simple algorithms for simulation of difference equations. This may be useful if you are not so familiar with programming.
4. Section 6.4 reviews the theory of linear difference equations. Most of this material can also be found in *Kalkulus*, except for the discussion of equations of general order in section 6.4.3.
5. Difference equations provide a simple setting where round-off errors may cause serious problems. This is discussed in section 6.5, particularly example 6.27 and the explanation in section 6.27.

Chapter 7: Lossless Compression

The theme of this chapter is how information may be stored compactly. Two methods are discussed in detail, namely Huffman coding (section 7.2) and Arithmetic coding (section 7.4). Section 7.3 provides some additional theory, while the rest are useful facts. In 2014 only sections 7.1-7.2 are part of the syllabus (for MAT-INF1100; Chapter 7 is not part of the syllabus for MAT-INF1100L).

Chapter 8: Digital Sound

In this chapter we discuss how sound can be stored and manipulated in a computer. This is peripheral in the course, and only simple facts are relevant for the exam.

Chapter 9: Polynomial Interpolation

The topic of this chapter is a generalisation of Taylor approximation. Instead of determining a polynomial of degree n by matching the function value and first n derivatives of a function at a point, the idea is to determine an approximating polynomial by matching $n + 1$ function values of a given function.

1. Section 9.1 reviews Taylor polynomials. Most of the material here can also be found in *Kalkulus*, but there are some examples of deriving error estimates that may be helpful.

2. Section 9.2 introduces the interpolation problem and defines the Newton form of the interpolating polynomial, which simplifies the solution of the interpolation problem.

3. Section 9.3 discusses divided differences and the divided difference table which simplifies the determination of the interpolating polynomial. This is not part of the syllabus for MAT-INF1100 and MAT-INF1100L in 2014.

4. Sections 9.4 and 9.5 and not part of the syllabus for MAT-INF1100 and MAT-INF1100L in 2014.

Chapter 10: Zeros of Functions

Determining zeros of functions is important in many applications. The tradition in mathematics is that solving an equation means finding a formula for the solution. However, this approach is only possible in some special cases. The alternative is to compute numerical approximations to the solution, which can be done for a large class of equations, and this chapter discusses three such methods. All three methods are based on very simple ideas and are best remembered by these ideas.

1. Section 10.1 discusses some examples where finding zeros is necessary, this is not essential reading.

2. The bisection method in section 10.2 is a very simple method for finding zeros, based on the intermediate value theorem. For this method it is very easy to derive an error estimate.

3. The secant method in section 10.3 is based on approximating the given function f by a secant through two points and using the zero of the secant as an approximation to the zero of the function. By repeating this, a sequence of numbers are generated that hopefully converges to a zero of f.

4. Newton's method in section 10.4 is similar to the secant method, but is based on approximating the given function by its tangent at a point, using the zero of the tangent as an approximation of the zero. By repeating this, a sequence is obtained that also hopefully converges to a zero of f.

Both Newton's method and the secant method work very well in most cases and usually only require a few iterations to arrive at a good approximation to a zero. However, it is quite easy to find cases where both methods fail.

For Newton's method it is relatively easy to show that if it converges and the function f is 'nice', then it converges quadratically, which means that the number of correct digits roughly doubles for each iteration, see Lemma 10.19. A similar result holds for the secant method, but this is harder to prove, and for this method the number of correct digits only increases by about 60 % per iteration.

Chapter 11: Numerical Differentiation

In many situations we need to differentiate a function whose values are only known at isolated points. This is the case when the function values are given by measurements or the function is given as a computer program—in either case symbolic differentiation is not possible.

The recommended way to approach this chapter is to study the simplest method in section 11.1 in detail. The method is simple, but all the features of the more complicated methods are present. This includes the effects of both the truncation error (replacing the derivative with a divided difference, section 11.1.2) and the round-off error (subtraction of almost equal numbers, section 11.1.3).

Section 11.2 gives a general procedure for deriving numerical differentiation methods, and some such methods are discussed in sections 11.3–11.5. The essence here is to understand the principles: How to derive the methods, and how to analyse them. All the details are not so important, but it is useful to remember how the errors depend on h.

Chapter 12: Numerical Integration

This chapter is similar to the previous, but discusses numerical integration instead of numerical differentiation. The general approach to the construction of the methods is similar: Replace f locally by a polynomial p and integrate p instead of f. For numerical integration it turns out that round-off errors are not critical. On the other hand, the analysis of the truncation error now consists of two parts. First the derivation of an error estimate on one subinterval, and then considering what happens when we assemble the results from all the subintervals to obtain the approximation to the complete integral.

Section 12.1 gives some background information on integration, and the simplest method is analysed and formulated as an algorithm in section 12.2. In particular it is possible to estimate a subinterval width that will ensure that the total error is smaller than a given tolerance.

For the methods in sections 12.3 and 12.4 it is sufficient to understand the ideas behind the methods and be familiar with the final error estimates.

Chapter 13: Numerical Solution of Differential Equations

This is a long chapter that may seem overwhelming. The core of the chapter is Euler's method in section 13.3 and its simple refinement, Euler's midpoint method in section 13.7.1, and the extension of these methods to systems of equations in section 13.8. Sections 13.5 and 13.6 are not part of the syllabus, and the details of the error analysis is not part of the syllabus either, see the podcast from the lecture that covers this material.

Section 13.1 provides some background information and motivation for studying differential equations, including a classification of different types of equations in section 13.1.3. Section 13.2 provides some basic information about first order differential equations, with a geometric interpretation in section 13.2.2 and discussion of what a numerical solution is in section 13.2.4. Euler's method is an intuitive method for solving differential equations based on a simple geometric idea and is described in section 13.3. Section 13.4 gives an error analysis for Euler's method. The mathematics should be understandable, but the total analysis is more advanced than what is expected for the exam. In general it is only expected that you are familiar with the convergence order of the numerical methods in this chapter.

Section 13.6 discusses Taylor methods which are based on differentiation of the differential equation, see section 13.5. Note that it is sufficient to understand differentiation for a specific equation—differentiation may appear quite abstract for a general equation. Section 13.7 discusses Runge-Kutta methods. Finally, in section 13.8 we show that a general system of differential equations of arbitrary orders may be expressed as a system of first-order differential equations.

Chapter 14: Functions of two variables

Not part of the syllabus

Chapter 15: Digital images and image formats

In this chapter we discuss how images can be stored and manipulated in a computer. This is peripheral in the course, and only simple facts are relevant for the exam.