Learning Outcomes

You should be able to

- articulate and justify the role of mathematics in the theory of Computer Science and its use in application development, software engineering, and simulation;
- define fundamental mathematical concepts and notation needed to study the scientific basis of software engineering and computer systems;
- perform simple calculations and analysis using mathematical tools;
- implement software that employs mathematical concepts and techniques.

Learning Resources

Coursework Assignment A software implementation based on a mathematical algorithm we will study during the module. It should take about 15–20 hours to complete, assuming that you have mastered the material in the tutorials. This assignment counts for 20% of the total mark.

Lectures As well as straight presentation of material, there will be simulations of difficult points, worked examples, case studies etc.

Lecture notes These will be issued in batches. I suggest that you add your own comments too. Most of the worked examples have a blank space for the solution: we will work together on them during the lecture. You will sometimes be asked to do some reading between lectures. There will be two lectures each week. Thanks to Dr. Alan Barnes and Dr. Dan Cornford.

More Resources

Tutorials All tutorials should be attended: I shall keep a register.

Problem sheets will be handed out at lectures and these should be attempted before the tutorial. The aim of the problem sheets is for you to practice mathematical skills (such as calculation, specification and analysis). At the tutorial the answers will be given and more general issues discussed; you will also have an opportunity to ask questions in a smaller group (of about 20 students). Some of the problem sheets will be assessed, and will count for 5% of the total mark.

There will be five tutorials in the module, taking place every two weeks. Make sure that you bring the problem sheet, your solution, and all your lecture notes to each tutorial.

Yet More Resources

Syllabus This is available on the CS web site. It specifies in detail the content of the module, together with useful books (all of which should be in the main library).

Exam There will be a formal 1.5 hour exam for this module, counting for 75% of the mark. The first question (40% of the marks) is compulsory and will test your facility with basic mathematical techniques, and the other 60% of the marks are awarded for your choice of two out of four other questions, which test more advanced ideas.

Contacting Me

Consultation Office hour: 1–2pm on Mondays and Tuesdays, or by appointment.

`i.t.nabney@aston.ac.uk`

and my room is 315C. See me early rather than late!

Web site URL

`http://www.ncrg.aston.ac.uk/~nabneyit/courses/MCS/`

or from the links in the Teaching Material section of the CS web site

`http://www.cs.aston.ac.uk/`

Prerequisites

CS1110 Introduction to Systematic Programming

Expectations:

- minimum ‘B’ grade at mathematics GCSE;
- can understand and calculate arithmetic expressions;
- aware of what equations are.

You will find a calculator useful for some of the exercises.

Role of Mathematics in Computer Science

- Discrete mathematics and mathematical logic lie at the heart of the discipline of computer science.
- The rise of the digital computer over the second half of the 20th century has coincided with (and pushed forward) a growth of interest in these fields; discrete mathematics is now an area of mathematics in its own right.
- Mathematics plays a role in computer science in many different ways; in this course we will consider just a few of them. We shall focus on mathematics that is useful in building software systems, as this is most relevant to the degree programme.

Value of Mathematics in system Development

description and specification: it is important to define precisely what system components are meant to do. This is particularly true on large, team-based projects, where confusion leads very quickly to error.

computation and algorithms: some applications are inherently difficult and require complex algorithms for their solution. For example, it is not possible to write down a simple algorithm that can take a speech waveform and output the corresponding text.

analysis and proof: analysis gives us a greater insight into a problem, and proof makes the results of analysis sure. For example, if you want to sort a large list of numbers, complexity analysis gives general results about how long each algorithm takes, while a proof gives confidence that these results can be relied on.

But Why Mathematics?

In the direction of what reasoning can accomplish, mathematicians have exercised the greatest care that the human mind is capable of to secure the soundness of their results. It is not accidental that mathematical precision is a byword. Mathematics is still the paradigm of the best knowledge available.

Kleene

Precision. The formality of mathematics allows us to be extremely precise in defining meaning. This is of great value in describing software systems and components. The field of formal methods essentially takes undiluted set theory and logic and applies it directly to software specification. This means that data structures and other features of a program are given unambiguous definitions, so there is more chance the implementation will do what it should.

Accuracy. Mathematical procedures consist of a logical sequences of operations. This makes it straightforward to translate them into executable programs that perform the same function. Thus we can be sure that a program carries out the task we expect as there is a close correspondence between the original form of the mathematical algorithm and the implementation in software.

Generality. Mathematical proofs are 'guarantees' of the truth of statements. They enable us to be sure that certain properties are true. It is also possible to prove that some things cannot be known. G odel's famous incompleteness theorem states (roughly) that in any logical system which is powerful enough to include integer arithmetic there are results that are true but which cannot be proved. Thus in mathematics we can be sure of what we know, but can also say something about the limits of our knowledge.

Understanding. Mathematical analysis helps increase our understanding of problems in a way that simply repeating large number of experiments and observing the outcomes cannot. For example, the time to sort a list using insertion sort grows quadratically with the size of the list (is $O(n^2)$), while the time for merge sort grows log-linearly (is $O(n \log n)$), which gives significant time savings for large lists. Such questions cannot be answered without the use of mathematics.

Overview of Module Content

Arithmetic You already understand standard arithmetic on integers, but arithmetic can be defined for other number systems and has applications to generating random numbers, error-correcting codes and cryptography.

When it comes to real numbers (i.e. decimals), computer arithmetic can only approximate true mathematical arithmetic. The gap between the two arithmetics, which at first sight seems trivial, can actually have enormous consequences (including catastrophic system failures).

Elementary Functions Functions other than arithmetic operations, such as trigonometric functions, exponential and logs, are required for many applications. These are often calculated using sequences (infinite strings of numbers) and series (sums of sequences), and we shall study how these can be implemented in computer arithmetic.

Geometry The real world is a three-dimensional space, so for any system that models that world (e.g. graphics, virtual reality) it is important to be able to define the shape, position and motion of objects. This is achieved using coordinate geometry, and as well as defining position vectors, we shall also show how to define surfaces (such as lines and planes) through algebraic equations.

Formal Methods In the first half of the 20th century, computer science was as much about logic as it was about hardware (for example, in the work of Alan Turing). The aim was to work out what tasks could be computed, and this involved defining precisely what an algorithm is so that it became possible to prove results about the boundaries of computation. Logic is also used in hardware design and plays a role in formulating database queries.

Set theory, relations and functions are the core building blocks for a formal view of defining objects and systems; they form the basis of the Z language used for specifying systems in a formal way.

Languages and Machines Computer languages themselves must be defined formally (how can a compiler work without a rigorous definition of the emphmeaning of a statement?). There are two stages to this: the syntax of a language defines what are valid statements; the semantics of a language defines the meaning of valid statements. We shall study Backus-Naur form: a formal language that can be used to define the syntax of other languages. We shall also show how machines (finite state automata) can be used to define and analyse languages in a way that translates directly to computer algorithms for tasks like parsing (extracting information from structured 'text') and user interfaces (defining the paths that users may take through a system).

Summary

1. The three main areas of system development where mathematics is valuable are: description and specification; computation and algorithms; analysis and proof.
2. The features of a mathematical approach that make it so valuable are: precision; accuracy; generality; understanding.
3. This module focuses on: arithmetic; elementary functions; geometry; formal methods; languages and machines.
4.
5.