

Mathematica in the teaching of mathematics for computer science students in H.E.

Experimenting and visualising mathematics applications in computing

Neil Gordon

Department of Computer Science,
University of Hull,
Cottingham Road,
Hull, HU6 7RX,
England.
n.a.gordon@hull.ac.uk

This paper considers the use of *Mathematica* in the teaching of mathematics to Computer Science students in Higher Education (H.E.). This is based on experiences in the English H.E. system, and includes examples of the kinds of illustrations and approaches that can be used in order to attempt the successful teaching of students with a diverse background and differentiated mathematical abilities. Whilst this paper considers the situation in the context of the English Higher Education environment many of the issues are relevant across the globe. The wide variety of student backgrounds now encountered in H.E. often gives rise to difficulties when attempting to teach diverse groups; the use of computer tools, examples, and interactive worksheets can all assist the student in gaining an appreciation and understanding of the applicability and underpinning nature of mathematics to the discipline of computer science.

Historically, many computing departments were founded in mathematics departments, and students entering computing degrees in Higher Education would require some form of advanced mathematical understanding. However, the changing nature of student groups and their earlier educational experience is leading to a diversification in the students' mathematical backgrounds. This paper considers how to utilise *Mathematica* in dealing with these issues. The strength of *Mathematica's* visualization tools, as well as the extensive discrete mathematics packages, provides a useful tool in attempting to engage computer science students with a topic many find "off task" or plain difficult.

This can be used in a variety of ways, which we investigate in the paper.

- 1) Interactivity in the lecture room
- 2) Interactivity in the workshop/laboratory
- 3) Complexity of examples
- 4) Visualization of problems, and solutions
- 5) Examples of the linking of the underlying mathematics with the application area

■ Introduction

Mathematics and computing are two disciplines with a close relationship, yet they remain distinct tribes [1]. The power of the two together – as demonstrated through solutions such as *Mathematica* – is well recognised, but may sometimes elude students in their typical studies, which are often focussed on one discipline in particular.

Furthermore, the growing distinction and competition between disciplines can lead to a greater separation. In computing in England, the requirements for advanced study in mathematics prior to entry (typically via A-levels) for computing students is now limited to a small number of departments [2]. Within this context, students often enter computing courses without any clear expectation of requiring mathematics skills, or of any real desire to develop them. One way to attempt to deal with this is to use subject based examples to motivate the content, along with tools to allow students to concentrate on the application rather than necessarily becoming proficient in carrying out the calculations themselves. Whilst *Mathematica* is useful as a tool for many topics, it is not necessarily an obvious choice for teaching the full range of discrete mathematics topics required for computing students. Furthermore, there is a danger of students using facilities as a black box solution, and of failing to understand or engage with the actual underlying concepts. However, the use of suitable examples can encourage such students to do the mathematics particularly since the use of the computer as a tool here reflects the natural interests of the students and helps them to develop semantic understanding. Developing material to assist students in understanding material which is an essential underpinning to their studies is a growing problem [3] which has been acknowledged since the early 1990's in the U.K. [4, 5]. *Mathematica* provides a useful environment in which to develop such material and we consider that later.

□ Module Context

The material in this paper covers usage within a module taught at level 4 (year 1 of the undergraduate programme) Computer Science degree. Students taking the course will have a variety of backgrounds in terms of their mathematical maturity, skill base and fluency, and may have strong existing barriers about engaging with mathematics. The module contributes 20 credits out of 120 credits taken in the year in question. The module is taught by traditional style lectures, supported by mediated workshops and laboratories, as well as informal support for the foundation mathematics material via a local study advice service. The workshops are an opportunity for students to work through examples and to have a less intimidating environment in which to ask questions about the material. The laboratories provide a similar space, but with the opportunity to use the computer as a tool in working through material. Many students prefer the computer as a medium and online forums appeal more than face-to-face dialogue in workshops.

The module is supported via our local Virtual Learning Environment, with forums, email groups and intranet facilities. The use of computer based diagnostic testing [6] is designed to encourage students in focussing their own studies, along with a regime of formative and summative assessment to encourage suitable engagement.

The module usually has a cohort of approximately 100 students, the majority of whom have not studied mathematics formally beyond secondary school (age 16, English G.C.S.E. level). Their pre-university experiences and preconceptions of mathematics as being a hard subject can cause potential barriers in their initial engagement with the module. One of the main aims of using *Mathematica* is to overcome such barriers.

■ *Mathematica* and the Syllabus

The syllabus for the module under consideration covers most of the typical discrete topics identified [8, 9, 10] as appropriate for a computing degree. These underpin a number of traditional computer science topics – and are a common part of many computer science degrees. However, the language and approach of mathematics, and the focus on abstract underlying ideas, can cause difficulties for students expecting direct examples, who lack the necessary mathematical language and fluency to engage with more abstract approaches. When using *Mathematica* within the context of a student's overall study account needs to be taken of the student's experiences of specific programming languages and paradigms. This can mean that less efficient (in terms of *Mathematica*'s own facilities) approaches and solutions are used, at least in the early stages. Examples of this are included below, such as the use of a For loop to explicitly calculate the sum of a series, rather than using the Sum function. Similarly in many examples Print statements are used to illustrate results rather than returning value, which is done to link in with the early programming experiences of the students. #

The detailed typical syllabus is listed in the next subsection.

□ A Computer Science Discrete Mathematics Syllabus

A typical "mathematics for computing" type syllabus would include a range of discrete mathematics topics, many of which of course have applications in continuous mathematics. However, the discrete mathematics has the benefit that students who are less fluent and familiar with mathematics can frequently understand the examples and the concepts more easily than many of the continuous concepts, such as calculus. Whilst the precise content of a syllabus varies, a typical content has been identified [7] and is listed here.

Set theory (language of sets, algebra and Venn diagrams)
 Relations and functions
 Propositional Logic (including truth tables)
 Predicate Calculus
 Boolean Algebra
 Induction
 Matrix algebra
 Graph Theory

□ Examples of *Mathematica* illustrations of the syllabus

As should be expected – given the intended application area – the syllabus topics identified above are amenable to approaches via computer tools, and in this context *Mathematica* offers a number of relevant functions, packages and a useful interactive environment for delivery – in both the lecture room and the workshop or laboratory. However, the support for these can implicitly assume existing understanding of the topics, whilst in teaching them we often wish to develop this understanding with the tools. The following subsections illustrate ways in which we can use *Mathematica* to let students develop examples, to develop code to implement the structures and ideas, and generally to illuminate the topics for students. The examples that follow attempt to avoid overly complex code, but to allow the students to link their own computing skills and knowledge with the underpinning mathematical concepts.

Set Theory

We can use the List structure and built in operations to allow students to easily develop set theoretic functionality and examples.

At the most trivial level, students can define a list of elements

```
L = {g, g, r, b, g, r, b}
```

but these ordered structures do not mirror true sets. However, emphasising the differences allows the students to convert the list to a true set

```
ListToSet[x_] := Union[x]
S = ListToSet[L]
```

Requiring students to implement a set version of equality ensures that they are aware of the different meanings of equality

```
We can use Mathematica's list
structures to mirror Set theory concepts for students.
SetEquals[A_, B_] := Union[A] == Union[B]
S2 = {r, b, g}
```

and then explain the difference between the statements

```
S2 == L
False
SetEquals[S2, L]
True
```

Support for notions such as set complements, union and intersection is of course built in

```
U = {a, b, c, d, e, f, g, r, s, t, u}
{a, b, c, d, e, f, g, r, s, t, u}
```

and so allow students to interact with these structures directly

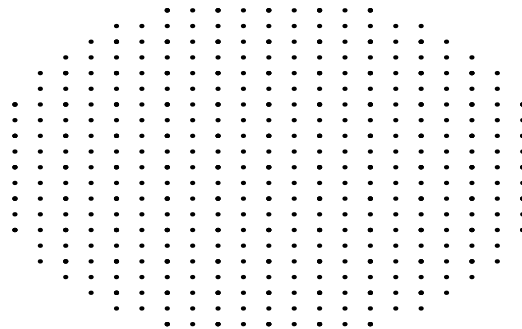
```
SC = Complement[U, S]
{a, c, d, e, f, s, t, u}
```

and this allows students to explore standard results such as

```
SetEquals[Complement[U, SC], S]
True
```

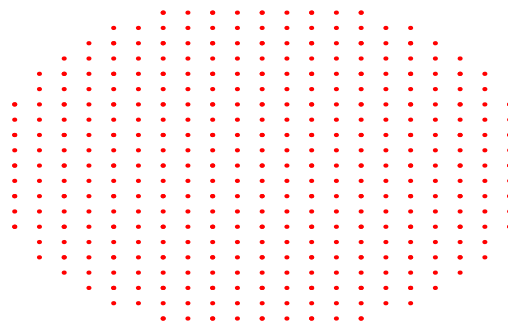
Venn diagrams are a natural representation of finite sets, and generating some examples of these – where students can think of the elements as individual points – allows students to interact with them and produce their own figures illustrating set theoretic results.

```
A = {};
For[x = 0, x < 50, x = x + .5,
  For[y = 0, y < 50, y = y + .5,
    If[((x - 25) ^ 2 + (y - 25) ^ 2) ≤ 30,
      {A = Append[A, {N[x, 1], N[y, 1]}]}]]]]
B = {}
For[x = 5, x < 55, x = x + .5,
  For[y = 0, y < 50, y = y + .5,
    If[((x - 30) ^ 2 + (y - 25) ^ 2) ≤ 30,
      {B = Append[B, {N[x, 1], N[y, 1]}]}]]]]
AV = Show[Graphics[Map[Point, A]]]
```



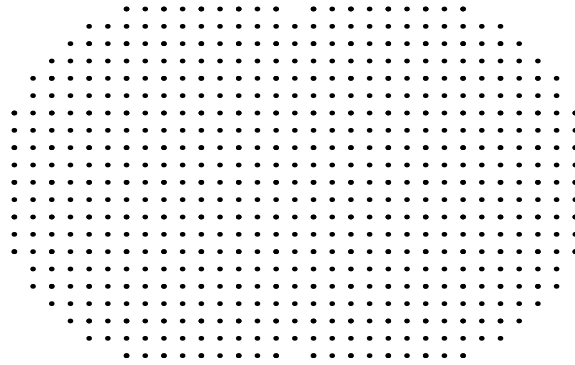
- Graphics -

```
BV = Show[Graphics[{RGBColor[1, 0, 0], Map[Point, B]}]]
```



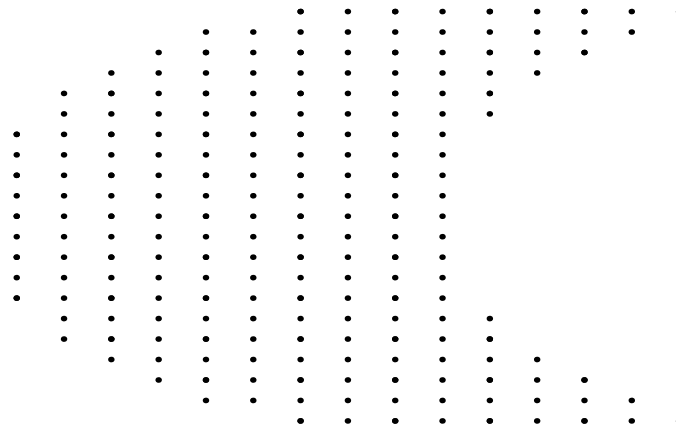
- Graphics -

```
Show[Graphics[Map[Point, Union[A, B]]]]
```



```
- Graphics -
```

```
Show[Graphics[Map[Point, Complement[A, B]]]]
```



```
- Graphics -
```

Relations and functions

Mathematica's approach to expressions and its functional paradigm clearly make it a good platform for developing these skills in computing undergraduates.

The first time computing students meet relations is usually as the concept of an ordered pair, with the first and second entry in each pair coming from two distinct sets. The List structure allows these to be defined intuitively

```
R = {{a, red}, {b, green}, {c, blue}, {a, black}}
```

and by defining an appropriate function students can extract the specific relation for a specific value

```
Rel[R_, x_] := Select[R, #[[1]] == x &]
```

```
Rel[R, a]
```

```
{{a, red}, {a, black}}
```

The concept of the Cartesian Product of two sets (as the set of all ordered pairs of elements) is again something that students can be encouraged to explore and explain using built in functions. With some guidance students should be able to find an efficient way of generating this, such as

```
CartProd[A_, B_] := Tuples[{A, B}]
CartProd[{0, 1}, {hi, lo}]
{{0, hi}, {0, lo}, {1, hi}, {1, lo}}
```

Propositional Logic

In the case of propositional logic, *Mathematica* offers an efficient way of allowing student interaction with basic propositional logic expressions. For example, we can illustrate the logical OR operator and produce the relevant truth table using code such as the following:

```
or[A_, B_] := Print[Infix[{A, B}, "or"], "=", A || B];
```

This can then be used to produce a truth table with the following approach

```
Boolean = {True, False}
Table[or[Boolean[[i]], Boolean[[j]]], {i, 2}, {j, 2}]
```

True or True=True

True or False=True

False or True=True

False or False=False

Similar definitions for And, Implies and IFF allow the students to interact with these standard propositional logic statements in what is to them an intuitive environment.

Example code for implies is

```
implies[A_, B_] :=
  Print[Infix[{A, B}, "=>"], "=", Not[A] || B];
```

Using this approach, students can easily investigate logical equivalence as well as building up more complex statements which can be then be evaluated for all possible logical states.

Predicate Calculus

Whilst *Mathematica* has built in support for predicate logic— through the `ForAll` and `Exists` predicates – these are designed to allow *Mathematica* to reason. In attempting to imbue students with this understanding, we need a different approach which allows them to explore the distinctions between meanings.

For students familiar with relational databases and set theoretic concepts, we can do this through the set theory approach to properties, and ask the students to produce code to test for the validity of statements such as `ForAll x something is true etc.`

As an example of one approach to this, we have

```
R =
  {{red, colour}, {green, colour}, {blue, colour}, {sour, taste}}
```

Then a predicate statement such to test something such as `ForAll x in R, x is a colour`, can be tested using a `For` loop – chosen as this links the predicate concept to the programming experiences of the students.

```
For[x = 1, x ≤ Length[R], x++,
  {Print[R[[x]][[2]]];
  If[R[[x]][[2]] != "colour", Print["False"]]}]
False
```

Boolean Algebra

The logic syllabus considered above naturally leads on to more complex notions such as Karnough maps and disjunctive normal forms. The built in `LogicalExpand` function allows students to develop further understanding and try their own examples and check their own answers

```
LogicalExpand[(a || b) && (a || c)]
a || (b && c)
```

Induction

As computing students will meet concepts such as iteration and recursion, they need some understanding of induction; at the initial introduction students often find the overall concept difficult. One way of assisting is in providing evidence to support the equation being proved.

A typical example would be for the sum of the first n integers. Here we can help the students in believing the result – prior to an inductive proof, using examples that they can interact with.

This approach utilises the students' natural understanding in terms of programming concepts. As a non-recursive approach we can write a simple iterative function to literally calculate the result

```
s[n_] := Module[{t, i}, t = 0;
  For[i = 1, i ≤ n, i++, t = t + i];
  t]
```

Stating the result to be proved, namely

```
f[n_] := n * (n + 1) / 2
```

the students can verify its validity in a number of test cases (and so helping them differentiate the distinction between testing and proving)

```
Table[{n, f[n] == s[n]}, {n, 0, 10}]
```

and even more efficiently using a recursive function

```
r[n_] := If[n > 0, n + r[n - 1], n]
```

which can also be tested for validity

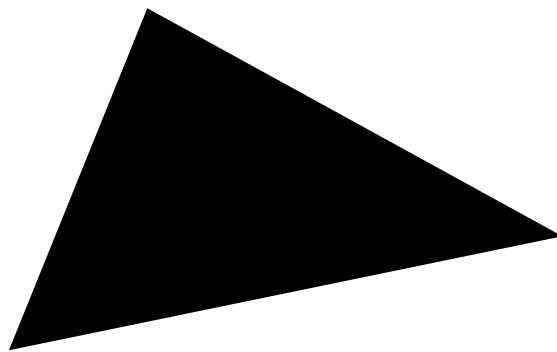
```
Table[{n, f[n] == r[n]}, {n, 0, 100}]
```

Note: in the above example a For loop was used in preference to the built in Sum command in order to link in with students' knowledge of other programming languages.

Matrix Algebra

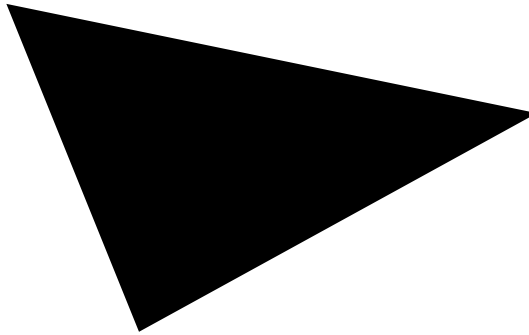
The array of built in support for matrices means that there is little extra work needed in providing students with an interactive experience, apart from covering the list of list data structure and the syntax required. However, since many students will be meeting matrices as a prelude to graphics this can be a good opportunity to demonstrate the applications of matrices – both 2d, 3d and 4d representations (to include translations along with scaling and rotations).

```
Shape = {{0, 0}, {1, 3}, {4, 1}, {0, 0}}
{{0, 0}, {1, 3}, {4, 1}, {0, 0}}
Show[Graphics[Polygon[Shape]]]
```



```
- Graphics -
M = {{1, 0}, {0, -1}}
{{1, 0}, {0, -1}}
```

```
Show[Graphics[Polygon[Map[M.# &, Shape]]]]
```



```
- Graphics -
```

In lecture or classroom, the use of animation can bring the topic to life. For example, to demonstrate the role of a rotation matrix in 2D

```
rm[t_] := {{Cos[t], Sin[t]}, {-Sin[t], Cos[t]}}
ani = Table[
  Graphics[Polygon[Map[rm[t].# &, Shape]], {t, 0, 2*Pi, .4}]
Map[
  Show[#, AspectRatio -> 1, PlotRange -> {{-5, 5}, {-5, 5}}] &, ani]
```

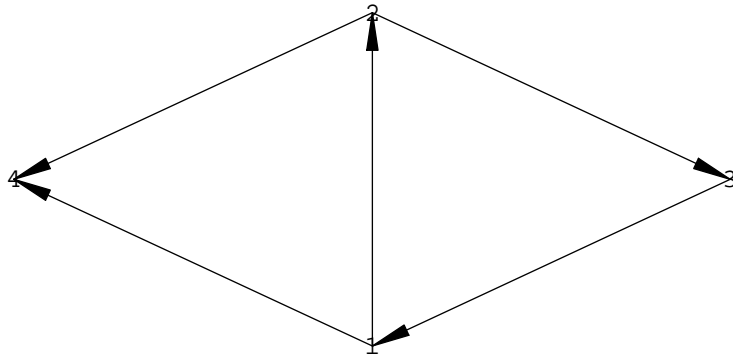
The use of 3D graphics and associated transformation matrices, and the use of 3D and 4D matrices to represent the range of computer graphics related manipulations, can be very effective in showing students the power of these mathematical objects, and in keeping the – sometimes quite abstract – mathematics closer to the application area.

Graph Theory

The `DiscreteMath`Combinatorica`` package provides a useful array of functions to assist in demonstrating key concepts to students, and provides students with useful tools with which to explore this topic.

The automatic positioning of vertices and the handling of digraphs are examples of how efficient this can be in the classroom environment. For example

```
GraphPlot[{1 -> 2, 2 -> 3, 3 -> 1, 1 -> 4, 2 -> 4},
  EdgeStyleFunction -> (Arrow[{#1, #2}, HeadWidth -> .3] &),
  VertexStyleFunction -> Automatic];
```



to generate a graph.

□ Other syllabus topics

Other topics that are covered in the module under discussion include

Algebra

Trigonometry

Vectors

Strings and Finite State Machines

Of these, the first three topics can all be supported via notebooks utilising the built in features and so provide a useful environment for student learning. Regarding the last topic, there is some support for strings but support for Finite State Machines is more limited. We consider this topic now

Strings, Languages and Finite State Machines

The wide range of functions and operations on strings themselves means that this topic is natural to cover

```
t = "a test string"
a test string
Characters[t]
{a, , t, e, s, t, , s, t, r, i, n, g}
```

And making use of some of the set theory issues met earlier, we can easily find the alphabet for this string

```
Alphabet = Union[Characters[t]]
{ , a, e, g, i, n, r, s, t}
```

Concatenation and other string operations and properties, such as the length of a string, or identifying substrings, can all be carried out immediately.

For example, concatenation via

```
A = "red"; B = " blue";
s = A <> B
red blue
```

and finding the length, often denoted by # in computing texts, by

```
StringLength[s]
```

```
8
```

The standard regular expression support – via the `RegularExpression` object is clearly very useful for teaching this topic.

```
ts = "the quick brown fox jumped over the lazy dog";
```

```
StringReplace[ts, RegularExpression["fox|dog"] -> "canine"]
```

```
the quick brown canine jumped over the lazy canine
```

However, this does not appear to give much insight into regular languages, and does not explicitly link in with Finite State Machines in a way that students are likely to appreciate. We can develop some tools to investigate Finite State Machine (FSM) concepts, and the following example for a Finite State Machine Acceptor demonstrates one approach to this. Finite State Machines are of particular interest here, as they provide a formal model of computation as well as being a practical technique used in software development. A Finite State Machine acceptor can be thought of as a collection of states, with a transition function describing how the machine changes states for given inputs. A subset of states are identified as the *accepting* states and if the machine ends in such a state it is said to have accepted – or recognised – the string.

We can define the component parts of our Finite State Machine Acceptor in a natural way, in particular the transition function

```
states = {s0, s1};
```

```
sigma = {0, 1};
```

```
d[s0, 0] = s1; d[s0, 1] = s0; d[s1, 0] = s0; d[s1, 1] = s1;
```

```
s0 = s0;
```

```
Accepting = {s1};
```

where the machine function can be simply implemented via something like

```
fsm[s_] := Module[{i},
  CS = states[[1]];
  For[i = 1, i ≤ Length[s], i++,
    CS = d[CS, s[[i]]]];
  CS]
```

```
fsm[{1}]
```

```
s0
```

```
fsm[{1, 0, 0}]
```

```
s0
```

and the issue of acceptance or otherwise can be easily tested too

```
Accepted[s_] := If[MemberQ[Accepting, s],
  Print["Accepted"], Print["Rejected"]]
Accepted[fsm[{1, 0, 0}]]
```

```
Rejected
```

```
Accepted[fsm[{0, 1, 0, 1, 1, 0}]]
```

```
Accepted
```

■ Issues in computer mediated learning

The use of technology in learning is well established and is no longer seen as being particularly innovative in itself. However, it can still be useful when making use of a tool to consider the different roles it could play, and this can aid in making use of it a success [11]. For the application considered in this paper, we now summarize the inclusion of more complex – and often more realistic – examples which can help students to see the benefits of the approach. Furthermore, it is often relatively quick to do one example, then repeat it in several more cases getting progressively more complex.

4) Visualization of problems, and solutions

Whilst some students prefer an abstract approach, others are more suited to learning via visual stimuli. *Mathematica's* wide range of graphical support and the ability to animate, and to manipulate plots, allows students to have the opportunity to explore objects and ideas in a very visual manner.

5) Examples of the linking of the underlying mathematics with the application area
This follows on from (3), since the ability to handle complex problems in the classroom means that we can consider problems that are directly related to the application area. Furthermore, the underlying model in *Mathematica* means that for computing students the tool being used is a direct example of the linkage between the application area and mathematics.

Informal and formal feedback from students indicates that the use of animation, concrete examples and a computer based tool are all appreciated and have been identified in module review as something that is valuable and should be further developed. At this stage it is too early to determine the impact on student performance, but attendance figures at workshops and laboratories is improved on previous years.

■ Conclusions

This paper has considered the embedding of *Mathematica* in the delivery of a typical computer science discrete mathematics curriculum. The examples illustrate one approach to this, and how we can use more relevant examples to engage students with material. Whilst there are other learning environments and delivery methods for this type of content, we have seen that the ease of creating examples, along with the interactivity that can be provided in both the lecture room and the laboratory opens up useful opportunities for students to learn and develop their understanding of the mathematics that underpins computer science.

■ References

- [1] Becher, T., 1989, *Academic Tribes and Territories: Intellectual Enquiry and the cultures of the disciplines*, Edmondsbury Press
- [2] Mark McCartney, 2004, *Computer Scientists and Mathematics*, [Online], Available: http://www.ics.heacademy.ac.uk/Events/Past_events/mathsbirmingham04/birmingham%20minus%20cartoons.ppt [Accessed 1 March 2006]
- [3] N. Gordon, 2003, *Wither Maths, Whither Science, Teaching Mathematics and its Applications: An International Journal of the IMA*, 23, pp15–32
- [4] Engineering Council, 2000, *Measuring the Mathematics Problem*, Report published in June 2000.
- [5] Sutherland R and Pazzi S, 1995, *The Changing Mathematical Background of Undergraduate Engineers*, report published by The Engineering Council.
- [6] N. Gordon, 2005, *The use of diagnostic software in teaching a mathematics module for computer science students*, 6th Annual Conference of the Subject Centre for Information and Computer Sciences, Editors H. Steed and U. O'Reilly, pp 34 – 38, Higher Education Academy Subject Centre for ICS, Ulster
- [7] *Issues in Teaching Discrete Mathematics*, I. Anderson, M. Dammerell, N. Gordon, R. Haggarty, A. Hooper, D. Penman, J. Stone, Proceedings of the 2001 Undergraduate Mathematics Teaching Conference, Ed P. Egerton, Birmingham University Press, Birmingham, ISSN 1477–7134, 2001.
- [8] Gersting J.L., 1993, *Mathematical Structures for Computer Science*, Freeman
- [9] Frank Giannasi and Robert Low, 1995, *Maths for computing and Information Technology*, Prentice Hall.
- [10] Rod Haggarty, 2001, *Discrete Mathematics for Computing*, Addison Wesley
- [11] P. Ramsden, 1997, *Mathematica in Education: Old Wine in New Bottles or a Whole New Vineyard?*, "Innovation in Mathematics: Proceedings of the Second International Mathematica Symposium", (Rovaniemi, Finland, June 1997). Southampton UK, Boston MA: Computational Mechanics Publications.
- [12] Kadjevich D and Haapasalo L, 2001, *Linking procedural and conceptual mathematical knowledge through CAL*, *Journal of Computer Assisted Learning*, 17, pp156–165