USING AN OPEN SOURCE PLATFORM LIKE SCILAB OR OCTAVE AS A FRAMEWORK TO INTRODUCE STUDENTS OF BASIC MATHEMATICS TO COMPUTER PROGRAMMING

Mohamed I Jamaloodeen, Keith Erickson
Georgia Gwinnet College
School of Science and Technology
1000 University Center Lane
Lawrenceville, GA 30043
mjamaloo@ggc.edu
kerickso@ggc.edu

Abstract:

We discuss integrating computer-programming in a discrete mathematics course using open source platforms. We present assignments to help students design and analyze code to implement algorithms or pseudocode.

Introduction

Algorithms, figure prominently in mathematics. Even basic mathematics concepts like long division or the bisection method for solving equations entail an understanding of algorithms. While this is the case, students are often challenged when tasked with analyzing or implementing an algorithm.

One possible reason is that ideas of basic computer programming are not integrated into the mathematics curriculum as is done in schools in France, Singapore, and China.

Students enrolled in a course such as discrete mathematics, for instance, are expected to be familiar with basic programming; but most have not even had a rudimentary exposure to it. Typically these students first see programming using advanced object-oriented program languages like Java or C++. The challenge for them is to master the basic concepts such as algorithms, program flow, and pseudocode, while also learning the more involved syntax associated with these object oriented languages. Historically students—even computer science majors—first learned computer programming using introductory high level programming languages like BASIC, FORTRAN and Pascal.

While these appear to have gone out of fashion, there is now a vast array of open source computational programming environments that can be used to expose introductory mathematics students to more basic computer programming concepts. These environments include Python, Scilab, Octave, and the computational statistical programming environment R.

In this paper we present a framework for integrating computer programming concepts in an undergraduate discrete mathematics course using an open source platform like Scilab or Octave. We discuss sample programming assignments and projects consistent with this framework. We intend to teach students to design and analyze algorithms and corresponding pseudocode, and to implement these designs in Scilab or Octave.

We conclude by proposing methods to use the elements in a basic college algebra course.

In previous work, [4], we discussed several trends in STEM education including the trend to programming in high school, the trend to scientific computing in high schools and the trend to mobile computing. This paper, in which we focus on another trend that of integrating computing to basic college math courses, can be considered an extension of that previous work.

Introducing Scilab

Scilab is a freely distributed open source scientific computing software package which can be a standalone installation on a Linux, Windows or Mac OS X system. The Scilab computational features can also be harnessed using a cloud service as offered by http://hotcalcul.com/on-line-calculator/compute and https://cloud.scilab.in/ Scilab is similar to the commercial Matlab platform or the open source Octave platform but in many ways is just as powerful. Scilab consists of three main components:

- an interpreter
- libraries of functions (Scilab procedures)
- libraries of Fortran and C routines

The trend in STEM education to introduce scientific computing in basic college math courses

The use of technology such as the scientific computing platforms MATLAB, Scilab and Octave in linear algebra courses or basic numerical analysis courses is quite common. The ease of defining matrices and vectors and the ease with which students can perform computations and manipulations with vectors and matrices makes these scientific computing environments ideal for adoption in linear algebra courses. The libraries of functions available for these platforms, and, their access to libraries of FORTRAN, C, and Math Kernel Library (MKL) routines make them well-suited to basic numerical analysis courses. Increasingly, however, we see that programming is being introduced into courses such as discrete math as well as calculus. In, [7], the authors introduce a program oriented approach to discrete math using the ALLOY language to model and solve problems. In, [3], the Python programming language and environment is integrated in a discrete math course intended primarily for computer science majors. The authors, here, argue that running Python in interactive mode, gives students an efficient environment with which to explore and experiment with the more abstract concepts they

encounter in this course, including the major topics of logic, mathematical proof techniques, graph theory, and design and analysis of algorithms. There are various other implementations integrating programming into discrete math courses, such as in [2], and [8]. More recently in, [1], the authors present work in which Scilab was integrated into a college calculus course. They argue that using mathematical programming languages (MPL's), like Scilab promotes logical thinking, and deeper mathematical understanding by forcing students to translate between verbal and programming descriptions of mathematical concepts. They also argue that MPL's can help students visualize concepts relating math to other areas such as engineering and science.

Introducing Georgia Gwinnett College (GGC) Discrete Math (MATH 2300)

Students enrolled in discrete math at GGC are, primarily, information technology majors and not computer science or computer engineering majors. Each section of MATH 2300 has about 28 students. The major topics covered in the course include, propositional logic, sets and functions, complexity of algorithms, basic number theory including divisibility, modular arithmetic and number representation systems, mathematical induction and recursive algorithms, basic combinatorics, basic discrete probability, and basic graph theory and applications of graphs and trees. Key course outcome goals include having students,

- Perform combinatorial analysis to solve counting problems and analyze algorithms.
- Demonstrate algorithmic thinking using mathematical creativity and critical thinking by specifying algorithms, verifying that algorithms work, and analyzing the time required to perform specific algorithms.
- Use appropriate technology in the evaluation, analysis, and synthesis of information in problem-solving situations.
-

The text used in the course is Kenneth H. Rosen's, Discrete Math and Its Applications (7^{th} ed), [5]. One of the great benefits of using this standard textbook, is the Scilab textbook companion, [6](" Scilab Textbook Companion for Discrete Mathematics And Its Applications by K. H. Rosen"), in which Scilab programs are available, for every example from chapters 1 through 8, in either pdf form, or, as standalone, downloadable and editable scripts. Another advantage of Scilab is that Matlab, or Octave code, that may not be explicitly available in Scilab, can often be used verbatim. As an example, Appendix 1 includes compatible Octave code, from rosettacode.org, for the widely used quicksort algorithm, a widely used sorting technique. Scilab's intuitive high level syntax is also consistent with the pseudocode syntax used in the Rosen text. The text introduces pseudocode syntax for,

1. Procedures
2. Conditionals
3. For loops
4. While loops
5. Comments

The corresponding Scilab syntax is intuitive and easy to use:

1. Functions
   - ```
     function [output arguments]=  function_name [input arguments]
      statements
     endfunction
     ```

2. Conditionals
   - ```
     if expr1 then
     statements
     elseif expri then
     statements
     ....
     else
     statements
     end
     ```

3. For loops
   - `for variable=expression do instruction,…,instruction, end`

4. While loops
   - `while expr do instruction,…,instruction, end`

5. Comments
   - Any line which begins with two slashes "//" is considered by Scilab as a commend and is ignored

Illustrating with the linear search algorithm

Consider the problem of searching for a target integer, *x*, from a list of *n* integers. The basic algorithm can be found in example 2 of section 3.1 from the Rosen text, [5], a variation of which is given below.

Given a list *A* of *n* integers with values a_1,...., a_n, and target value *x*, the following procedure (in pseudocode) uses linear search to find the index of the target *x* in *A*.

1. Set *i* to 1.
2. If $a_i = x$, the search terminates successfully; return *i*.
3. Increase *i* by 1.
4. If $i \leq n$, go to step 2. Otherwise, the search terminates unsuccessfully.

This example uses the linear search algorithm, and its Scilab code is available in the Scilab textbook companion (see Appendix 2). Below is a screenshot of a Scilab session in which the integer, 19, is searched for in the list of numbers 1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22.

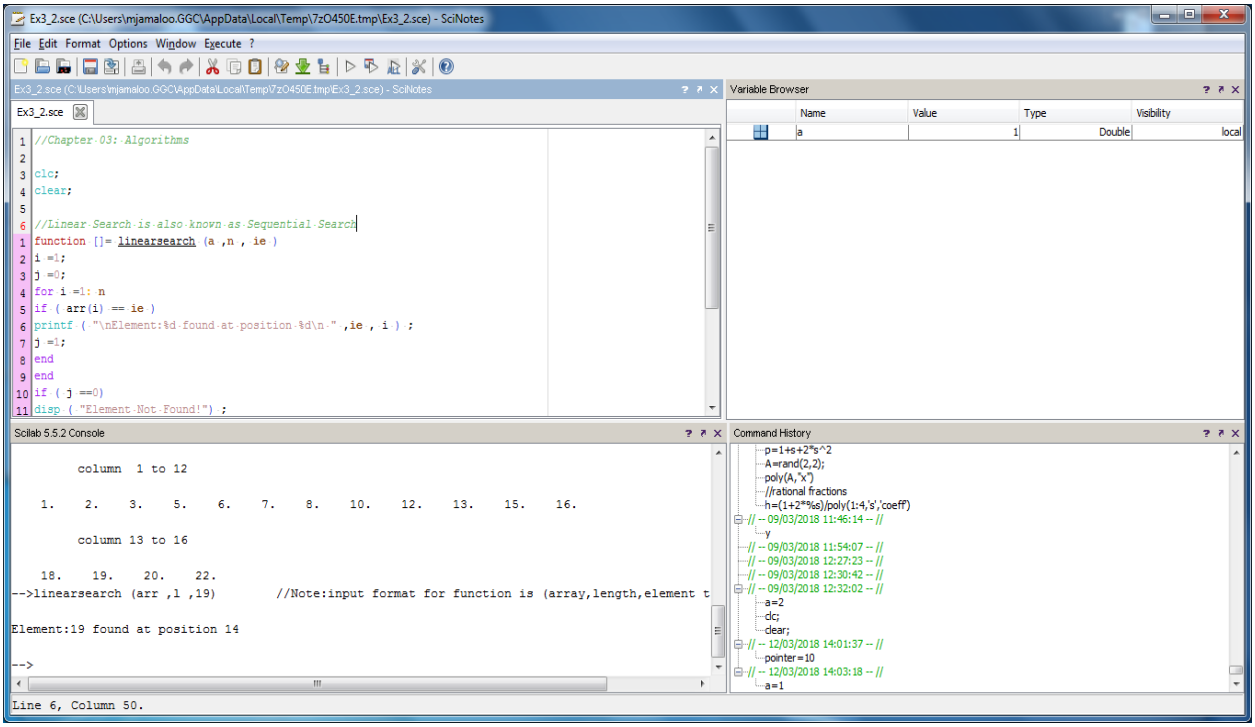


Figure 1. Screenshot of Scilab code and Scilab console running the linear search algorithm with Scilab code from Appendix 2.

Assigning programming homework assignments in GGC Discrete Math

Several programming assignments were designed beginning with basic familiarization with the Scilab programming and computing environment. Most of these were based on section exercises, chapter "Computer Projects," or chapter problems on "Computations and Explorations." The computer projects instruct students to write programs with specified inputs and outputs, and the problems on "Computations and Explorations" instruct students to apply computational programs they have written toward solving certain exercises. We provide a sample Scilab programming assignment in Appendix 3. The first part of this assignment comes from modification of exercises found in the text. The second part involves modifying the bubble sort procedure found in the Scilab textbook companion to sort a list of integers in decreasing order rather than increasing order. We provide the Scilab bubble sort algorithm from the companion in Appendix 4, with comments showing the modification required to reverse sort. All assignments involve specific input and output formats, and Scilab programs/scripts are turned in as text files on the GGC learning management system for testing and grading.

Preliminary findings

During this pilot implementation of the Scilab programming assignments students were offered the opportunity to use their cumulative programming assignment scores to replace their lowest in class test score worth 15 percent of their final grade. Although the class was encouraged to make use of this opportunity, and a majority of students did, several students chose not to work on the assigned programming questions. In addition to improving their lowest test grade, those students who completed the programming assignments improved their understanding of algorithms and programming, as assessed by comparing relevant quiz scores before and after the implementation of the optional programming homework. We plan to continue making these programming assignments optional over the next few iterations of this effort to refine some of the startup challenges faced during this pilot phase.

Introducing basic programming in a college algebra course

We are also exploring ways to introduce basic computer programming in GGC's college algebra course. A few of the course outcome goals here have students:

- Express and analyze relationships using functions in multiple ways (graphical, numerical, symbolic and verbal);
- Manipulate mathematical information and concepts to solve problems using multiple representations of polynomial, exponential and logarithmic functions;
- Use mathematical language appropriately;
- Use appropriate technology in the evaluation, analysis and synthesis of information in problem-solving situations.

While there is no Scilab textbook companion for the text we currently use, there are textbook companions freely available for other textbooks that can be adapted to our course. For instance, one algebra Scilab textbook companion has code for over 200 examples covering basic concepts such as order of operations, fractions, signed numbers, expressions, and factoring and equations, along with the more standard material of functions and lines, literal equations, simultaneous equations, linear inequalities, functions and their graphs, linear functions, quadratic functions, and exponential and logarithmic functions. Many concepts in college algebra, by their iterative or sequential nature lend themselves ideally for computational analysis. For example, understanding the intermediate value theorem and its use in locating zeros, especially of polynomial functions, is one possible simple programming assignment.

Conclusions

We have found that integrating the Scilab programming language and platform into our teaching of the GGC discrete mathematics course has been beneficial to the students in the class, most of whom are information technology majors. Our pilot implementation shows that Scilab programming allows students to actively engage with many of the algorithms encountered in this course including algorithms related to the major topics such as logic, combinatorics, number theory, and graph theory. The advantages of using an environment like Scilab are its intuitive

syntax, its compatibility with Matlab and Octave, the availability of code from repositories like Rosettacode.org, and that it is freely available as an open source platform. The major advantage of this platform, though, is the availability of the Scilab textbook companion with code for every example from Chapters 1 through 8 of the Rosen text ([5] and [6]). In future work, we will present results of integrating basic programming assignments in a college algebra course.

References

[1] Affane Aji, Chadia & Garrett, Lauretta & Biaz, Saad. (2013). The integration of Scilab into college calculus. Alabama Journal of Mathematics. Alabama Journal of Mathematics.

[2] Bolick, B., Gopalakrishnan, G., Jacobsen, C., & Tuttle, T. (2013).Toward Revamping Discrete Structures: Concurrency through Functional/Constraint Programming Integration. University of Utah, Formal Methods Research.

[3] Farahani, A. (2009). Use of Python in Teaching Discrete Mathematics. In American Society for Engineering Education. American Society for Engineering Education

[4] Jamaloodeen, M. I. (2015). Scientific computing and programming in the cloud using open source platforms: An illustration using weighted voting systems on the Scilab computing environment (Electronic Proceedings of the Twenty-sixth Annual International Conference on Technology in Collegiate Mathematics VOL26/S064 Proceedings

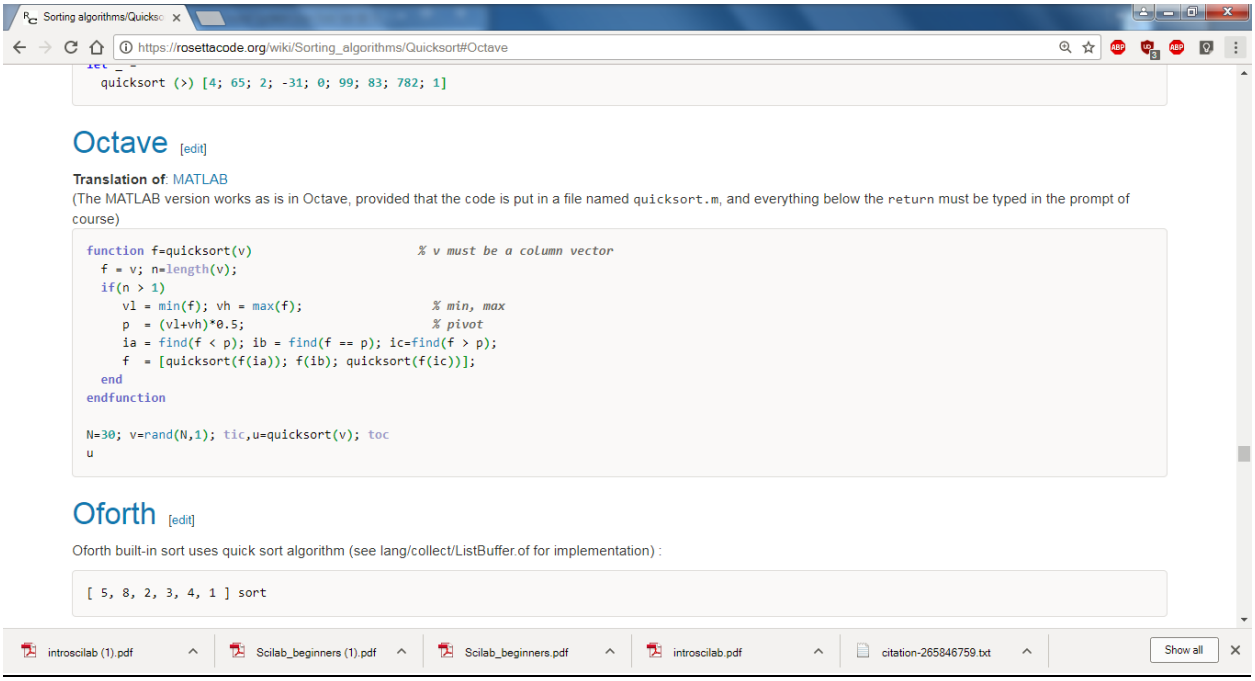
[5] Rosen, Kenneth H. (2012). Discrete Mathematics and Its Applications, 7th Edition (2012); McGraw Hill, New York, NY 10020

[6] Sai, L. (contributor). (2017). Scilab Textbook Companion for Discrete Mathematics And Its Applications by K. H. Rosen, Scilab India,(https://scilab.in/textbook_run/3808/82/7)

[7] Ureel, L. C., & C. Wallace, C. (2016). Discrete mathematics for computing students: A programming oriented approach with Alloy, IEEE Frontiers in Education Conference (FIE), Erie, PA, USA, 2016, pp. 1-5., doi: 10.1109/FIE.2016.7757641

[8] VanDrunen, T. (2011). The case for teaching functional programming in discrete math. In Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion (OOPSLA '11). ACM, New York, NY, USA, 81-86. DOI: https://doi.org/10.1145/2048147.2048180

Appendix 1: Rosetta Code Octave example for the quicksort algorithm that can be used verbatim in Scilab through a copy and paste (with Octave program comments (percentage signs) deleted)



Sorting algorithms/Quicksc

https://rosettacode.org/wiki/Sorting_algorithms/Quicksort#Octave

```
quicksort (>) [4; 65; 2; -31; 0; 99; 83; 782; 1]
```

Octave [edit]

Translation of: MATLAB

(The MATLAB version works as is in Octave, provided that the code is put in a file named `quicksort.m`, and everything below the `return` must be typed in the prompt of course)

```
function f=quicksort(v)                    % v must be a column vector
  f = v; n=length(v);
  if(n > 1)
     vl = min(f); vh = max(f);               % min, max
     p  = (vl+vh)*0.5;                       % pivot
     ia = find(f < p); ib = find(f == p); ic=find(f > p);
     f  = [quicksort(f(ia)); f(ib); quicksort(f(ic))];
  end
endfunction

N=30; v=rand(N,1); tic,u=quicksort(v); toc
u
```

Oforth [edit]

Oforth built-in sort uses quick sort algorithm (see lang/collect/ListBuffer.of for implementation) :

```
[ 5, 8, 2, 3, 4, 1 ] sort
```

introscilab (1).pdf | Scilab_beginners (1).pdf | Scilab_beginners.pdf | introscilab.pdf | citation-265846759.txt | Show all

Appendix 2: Scilab textbook companion code implementation of the linear search algorithm, [6]

```
//Chapter 03: Algorithms

clc;
clear;

//Linear Search is also known as Sequential Search
function []=linearsearch(a, n, ie)
i =1;
j =0;
for i =1: n
if ( arr(i) == ie )
printf ( "\nElement:%d found at position %d\n " ,ie , i ) ;
j =1;
end
end
if ( j ==0)
disp ( "Element Not Found!") ;
end
endfunction

arr =[1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22]
l=length(arr)
disp (arr , " Given array:" ) ;
linearsearch (arr ,l6 ,19)       //Note:input format for function is (array,length, target element to be searched)
```

Appendix 3: Sample Assignment

Discrete Math Math 2300 Programming Assignment #1

1)

a) Devise an algorithm that finds all terms of a finite sequence of integers that are greater than the sum of all previous terms of the sequence.

b) Implement the algorithm in Scilab and test it on the following input examples

testlist =[2 4 11 3 1 25]

testlist =[1 2 3 4 11 23 45 7 100]

Your output should give the corresponding integers and their positions in the list.

c) Submit your Scilab script file on the Brightspace Assignment 1 drop box for input of type “testlist“ call your script Part1.sci.

2)

a) Change the Bubble Sort Algorithm (4) from the text so that the list of integers is sorted in decreasing order rather than increasing order

b) Implement the reversed Bubble sort Algorithm in Scilab and test it on the following input examples

testlist =[2 4 11 3 1 25]

testlist =[1 2 3 4 11 23 45 7 100]

Your output should give a list sorted in decreasing order

c) Submit your Scilab script file on the Brightspace Assignment 1 drop box for input of type “testlist“ call your script Part2sci.

Appendix 4: Scilab textbook companion code implementation of the bubble sort algorithm, [6]

```
//Chapter 03: Algorithms

clc;
clear;

function [res]=bubblesort(a, n)
i =1;
j =1;
temp =0;
for i =1: n -1
for j =1: n - i
if ( a ( j ) >a ( j +1) )  //to reverse sort this line is changed by reversing the inequality
temp = a ( j ) ;
a ( j ) = a ( j +1) ;
a ( j +1) = temp ;
end
j = j +1;
end
i = i +1;
end
res = a ;
disp ( res ,"Sorted Array :") ;
endfunction

 a =[3 2 4 1 5]
 disp (a , " Given Array " )
a1 = bubblesort (a ,5)
```