

Math 364- Introduction to Scientific Computing

Lecture #1

August 30, 2005

Almost any event in real world has a mathematical/physical explanation. Moreover, some events can be simulated in a computer allowing us to repeat it as many times as needed, analyzing behavior and making inferences about it. Therefore the use of computers is an important tool in mathematical modeling. The goal of this course is that students learn some techniques for solving math problems in a computer and learn one of the most commonly used programming languages for scientific research MATLAB. Also students will learn the basics of LaTeX for scientific publishing.

1 Introduction

For a computer to be of any significant use in scientific problem solving, this must become programmable. This in the sense that a sequence of calculations and decision making criteria to direct these calculations can be set out and made to control the steps taken by the calculator. The set of instructions which describe the process is called a **program**. Any programmable computer allows to execute the instruction sequence described in such a program.

At the lowest level, the program is written in machine code, which is in a format appropriate for direct interpretation by the computer but this language level is definitely not a human language. In fact it will be quite inappropriate for the description of the methods to solve scientific problems. To describe our scientific problems we use another type of language, which we will call a **higher level language** - like MATLAB. We can make an association between a language level and a computer that executes instructions of that language, so that the choice of a programming language implicitly imposes the machine on which that language will execute. Since one can construct translators from one language level to another, it is quite possible for a user to write programs at one language level which will actually be executed on a real machine of a much lower language level. In a well designed environment, the details of the actual machine on which the user's program executes should be of no direct interest to the user.

In the solution of many scientific problems, the natural language is that of conventional mathematics. In the sense that this involves expressions with addition, multiplication, and similar operations, as well as function evaluations. One of the first computer languages to provide significant facilities

in this manner was the language FORTRAN (FORmula TRANslator) developed in the mid 1950's. While still used extensively, FORTRAN has been followed by a number of more modern languages ranging from BASIC to PASCAL, and C. The choice of language should normally be dictated by the problem to be solved, and at least one measure by which a particular language should be judged is the ease with which the language can be used to solve the greatest breadth of problems - having to learn a new language to solve a slightly different problem is not an efficient way to use human time.

A recent development has been in the area of what are called scripting languages, such as **Perl** and **Python**. These languages have their origins in the older "shell languages" such as the Bourne Shell (sh) or the C-Shell (csh) that were used primarily to provide the interactive environment on UNIX and other computer systems. These scripting languages are derived from the need for a language which provided easy access to the many well written programs standardly available on such systems in a manner that would easily allow these programs to be connected together to perform composite tasks.

2 Algorithms

An algorithm can be described as a list of steps that are required to perform a particular task. The study of algorithms began as a subject in mathematics and in fact, the search for algorithms has been an important activity of mathematicians long before the development of today's computers. Discovering an algorithm for solving a problem is essentially discovering a solution for the problem. It is important to point out that without an algorithm the problem cannot be solved by any computer. That is, computers are able to perform solutions based on an algorithm.

Importance: There are problems that cannot be solved analytically BUT can be solved *numerically*. That is, we can find the numeric approximation of these problems by designing an algorithm and implementing it.

However, there are problems without algorithmic solutions. See **Kurt Godel's Incompleteness Theorem** which basically states that there are statements that can be neither proved nor disproved. The machine representation of an algorithm is called a **program**.

Algorithms + Programs = Software

Exercise: Read about the Incompleteness Theorem

Once an algorithm for solving a specific problem has been discovered, it has to be represented in such a way that it can be communicated to a computer and expressed in such a way that others can understand it. In other words it is transformed from its conceptual form into a set of instructions. The form, structure and "grammar" of those instructions is what we know as **programming languages**.

Perhaps the most difficult part of finding a numerical solution to a problem is to actually be sure that the program and the solution output of the program are correct. For the first one once the coding has been compiled then we can print out some preliminary results and analyze them, by analyzing I mean either plot them and use some knowledge or if it is not too hard follow the code lines with some hand computations. As for the second one - the task could take much more time than someone can

think of because in most cases numerical approximations are used because there are no analytical solution to the problem. So figuring out that the solution is correct will need more knowledge and time.

Example:

Find an algorithm for solving the following program: Given a positive integer n , find the list of positive integers whose product is the largest among all the list of positive integers whose sum is n .
sol.

$n = 1, n = 2$ are simple list,

$n = 3, \{3\}, \{2,1\}, \{1,1,1\};$

$n = 4, \{4\}, \{3,1\}, \{2,1,1\}, \{1,1,1,1\};$

$n = 5, \{5\}, \{4,1\}, \{3,2\}, \{3,1,1\}, \{2,2,1\}, \{2,1,1,1\}, \{1,1,1,1,1\};$

Algorithm:

Step 1: Divide n by 3, you get a quotient q and a remainder r ,

Step 2: If $r = 0$ then list contains q 3s

Step 3: If $r = 1$ then list contains $q - 1$ 3s and two 2s

Step 4: If $r = 2$ then list contains q 3s and one 2

MATLAB implementation:

We will use *rem* and *floor* built-in functions.

Exercise: Find the lists for $n=765, 987, 2001$

3 Basics of Matlab

MATLAB includes tools for: Data acquisition, Data analysis and exploration, Visualization and image processing, Algorithm prototyping and development, Modeling and simulation, Programming and application development.

- Matlab is an interactive program which means that you can give commands at the command prompt \gg but you can also write script files or programs, called “*m-files*”. A “*m-file*” contains a sequence of standard Matlab commands. The file must always have a “*m*” suffix, e.g., myfile.m.
- To run a “.m-file, it is necessary to be running Matlab in the same directory in which the “*m-files*” sits, which is the path Matlab will look for by default. For a different directory, you should change your path setting inside Matlab (File -> Set path). To see where you are just type *pwd* which gives you the path for the working directory. To change directories within Matlab use the “*cd*” command.
- There are two types of m-files: Script and Function
- For comments within the code use the symbol % preceding a comment.
- A semicolon ; at the end of the instruction is used to suppress output, quite useful when dealing with 1,000,000 x 1,000,000 matrices!

- Variables: The name of the variable can be as long or as short as one wish, it can be a combination of letters (case sensitive), numbers and only one symbol is allowed - the underscore '_'.
- Arithmetic operators are carried out from left to right so $a * b + c = (a * b) + c \neq a * (b + c)$. Note that this does not apply to the power operator “^”. The use of parenthesis is always useful.

Examples:

1. Matrix-vector manipulation:

- » [1,2,3] - creates a row vector (also [1 2 3]);
- » [1;2;3] - creates a column vector;
- » [1 2 3; 4 5 6; 7 8 9] - creates a 3x3 matrix

$$\gg A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

2. The colon operator

- » A[:,1] - extracts the first column
- » A[2,:] - extracts the second row
- » A[2:3,1:2] - extracts

$$A[2 : 3, 1 : 2] = \begin{bmatrix} 4 & 5 \\ 7 & 8 \end{bmatrix}$$

- » y=m:s:n - creates y=(m, m+s, m+2s, ..., n)
- » y = eye(3) - Identity
- » y = zeros(2,3) - Zeros 2x3 matrix
- » y = ones(5,2) - Ones 5x2 matrix
- » y = rand(3) - 3x3 matrix of random numbers from a Uniform(0,1) distribution
- » y = randn(2) - 2x2 matrix of random numbers from a Normal(0,1) distribution

3. Simple loop constructs

```
for index = j:k
-   statements
end
```

- » x= [1 2 3];
- » for i=0:3; M(:, i + 1) = (x').ⁱ; end

$$\gg M = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \end{bmatrix}$$

or to compute successive members of a sequence e.g $x_n = \frac{ax_{n-1}}{n}$

4. 5. The “dot” before an arithmetic operator

To perform array operations on an element-by-element basis, that is the vector with elements $(2^4 \ 6^2 \ 7^{-2})$ will be $\gg [2 \ 6 \ 7].\wedge[4 \ 2 \ -2]$