


Computational Methods in Physics (PHY4605) - Programming Fundamentals

Table of Contents

Overview.....	1
Computational Physics.....	1
Software Environment.....	3
Command Window.....	4
Editor.....	4
Workspace.....	5
Documentation.....	6
Variables.....	6
Variable Naming.....	7
Data Import/Export.....	7
Computer Precision.....	8
Operators, Expressions and Statements.....	9
Mathematical Functions.....	11
Physics Pinpoint  : Uniformly Distributed Load.....	13

Overview

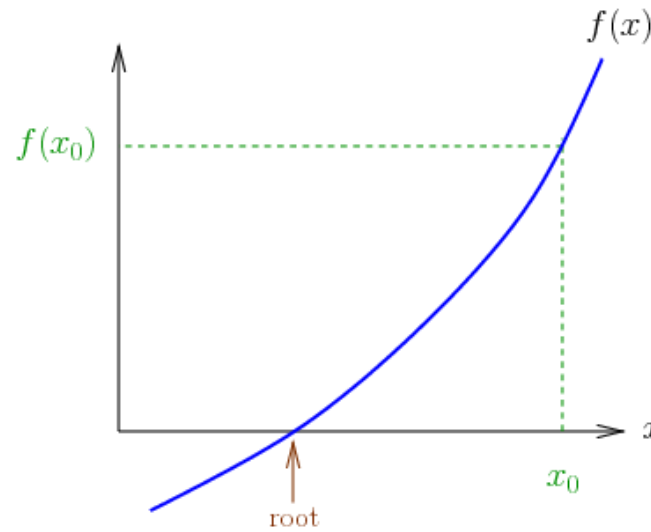
Computational physics is the study and implementation of numerical analysis to solve problems in physics for which a quantitative theory already exists. It is a subset of computational science and a vital tool for advancing physics research and applications. Mastering programming for computational physics is important because it enables physicists to create and modify computer simulations, models and algorithms that can handle complex and large-scale problems. Programming also helps physicists to visualize, analyze and communicate their results effectively.

Computational Physics

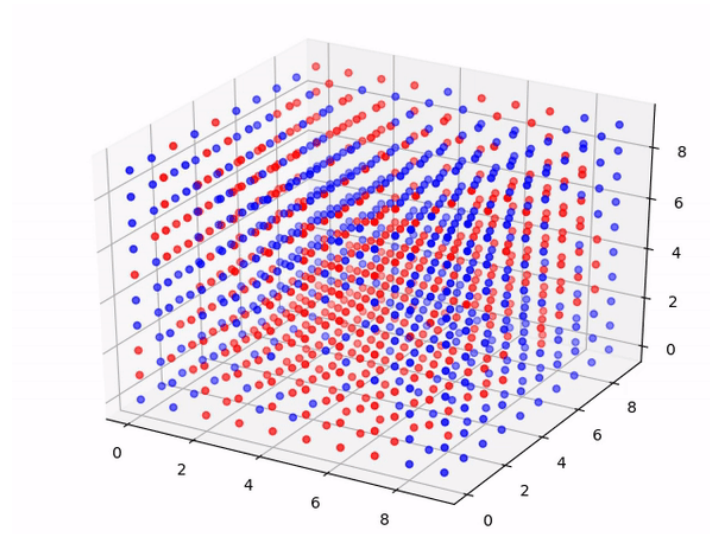
Computational physics is a subject that deals with numerical approximations of mathematical models in physics when solving them exactly is not feasible. It uses algorithms and computers to compute approximate solutions and errors. Some importance of computers in physics include:

- Solving problems in physics for which a quantitative theory already exists using **numerical analysis**.
- **Analyzing and visualizing data** from experiments and observations to communicate results and findings to other physicists and researchers.
- Running **experiments and monitoring equipment** using computer-controlled instrumentation.

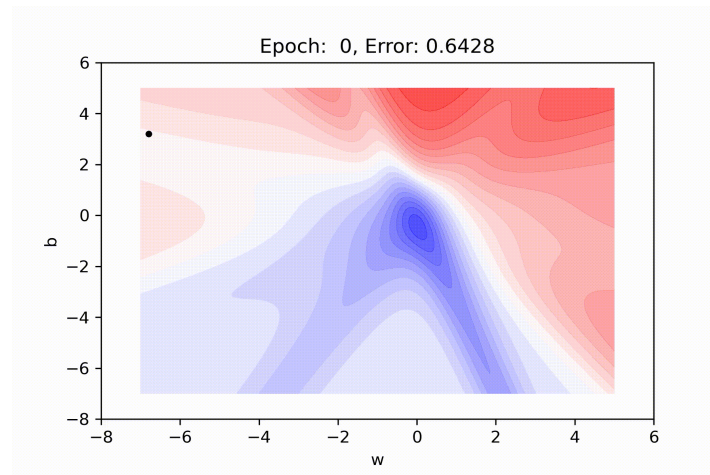
- **Simulating physical phenomena** that are difficult or impossible to observe directly.
- Processing large amounts of data, applying mathematical algorithms and learning from data patterns for **model development**.



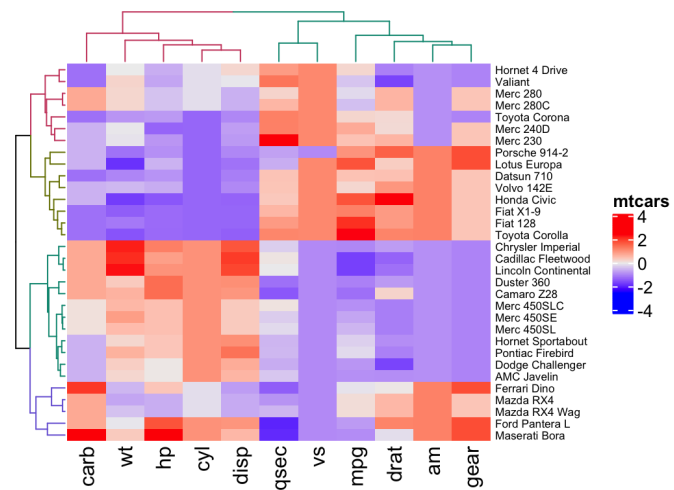
Newton-Raphson's method for finding roots



The Ising model simulation of magnetic materials



Machine learning application for optimization problems



Data visualization through heatmap

Software Environment



MATLAB is a powerful technical computing system for handling scientific and engineering calculations. The name MATLAB stands for **Matrix Laboratory**, because the system was designed to make matrix computations particularly easy. The MATLAB user interface consists of several components such as the Command Window, the Workspace browser, the Editor window, and the Help browser, collectively known as integrated development environment (IDE). The IDE allows you to interact with MATLAB by typing commands, editing and debugging code, managing files and variables, accessing documentation and online resources.

Command Window

To demonstrate running scripts as commands, enter the following in Command Window:

```
New balance:
1090
```

When you press Enter to run it, observe the output in the Command Window. Obviously you need to save the program if you want to use it again later. To write a script to be later saved, use Editor.

Editor

To get started, type the following script into the Editor :

```
Matrix = 3x3
    8     1     6
    3     5     7
    4     9     2

IdentityMatrix = 3x3
    1     0     0
    0     1     0
    0     0     1

PredictedIdentityMatrix = 3x3
```

```

1      0      0
0      1      0
0      0      1

```

Then click on the Run button just under the tab named Editor or Live Editor. The first time the script is executed you are asked to name the file. In this example use `MatrixMultiplication.m` as the name. If all lines are typed correctly, except those that are preceded by the % symbol (because those are comments to describe the code and are ignored when script is executed), outputs will be shown in Command Window.

You can also run the script by calling the file name without the ".m" extension in Command Window as follows:

```

Matrix = 3x3
      8      1      6
      3      5      7
      4      9      2
IdentityMatrix = 3x3
      1      0      0
      0      1      0
      0      0      1
PredictedIdentityMatrix = 3x3
      1      0      0
      0      1      0
      0      0      1

```

Workspace

Define some commonly used constant:

Enter the command `who` and `whos` to see the list of variables in the Workspace:

Your variables are:

AvogadroConst	Balance	Bar2KiloPascal	BaseLn	GravityAcceleration	IdentityMatrix	In
Name	Size	Bytes	Class	Attributes		
AvogadroConst	1x1	8	double			
Balance	1x1	8	double			
Bar2KiloPascal	1x1	8	double			
BaseLn	1x1	8	double			
GravityAcceleration	1x1	8	double			
IdentityMatrix	3x3	72	double			
Interest	1x1	8	double			

InverseMatrix	3x3	72	double
Matrix	3x3	72	double
PredictedIdentityMatrix	3x3	72	double
PredictionError	3x3	72	double
Rate	1x1	8	double
SmallestInterval	1x1	8	double
Value	1x1	8	double
m	1x1	8	double
n	1x1	8	double

The command `clearvars` removes all variables from the workspace:

Documentation

MATLAB provides self-contained documentation for all features and functions that it offers. The `help` and `doc` functions in MATLAB are both used to access the documentation of MATLAB functions and commands:

magic - Magic square
 This MATLAB function returns an n-by-n matrix constructed from the integers 1 through n2 with equal row and column sums.

Syntax

M = **magic**(n)

Input Arguments

n - Matrix order
 scalar integer

Examples

Third-Order Magic Square
 Magic Square Visualization

See also `ones`, `rand`

Introduced in MATLAB before R2006a
 Documentation for `magic`

The main difference between the functions is that `help` displays the information in the Command Window, while `doc` opens a separate window with more details and examples

Variables

A variable is created simply by assigning a value to it at the command line in a program – for example:

```
Number =  
98
```

All variables are considered arrays, whether they are single-valued (scalars) or multi-valued (vectors or matrices). In other words, a scalar is a 1-by-1 array – an array with a single row and a single column.

Variable Naming

A variable name must comply with the following rules:

- It may consist only of the letters a – z, the digits 0 – 9, and the underscore.
- It must start with a letter.
- It is case-sensitive

Some practices of variable naming are recommended:

1. Use **meaningful names** for variables. Variable name must define the exact explanation of its content. For physical quantities, embed the unit in the name, if possible. For examples, use BoxWidth, RoomTemperature and WattUsage instead of y, RT and Pwr.
2. **Single-letter lowercase** variables like i or j are usually used for **loop counters**; however, their usage are acceptable for well-recognized **mathematical unknowns** in an equation. Names that have one to four words are acceptable for other types of variables.
3. Obey programming language standards and **do not use lowercase/uppercase characters inconsistently**. Many MATLAB programmers use UpperCamelCase for variable names because it is arguably the easiest to read. To distinguish variables from functions, use camelCase for function names.
4. Mimic parts of speech of the English language. Use verbs for functions (e.g., extractData, plotData, showDataUnit), nouns for common variables (e.g., ExtractedData, PlottedData, DataUnit) and true/false questions for logical variables (e.g., IsDataExtracted, CanPlotData, HasDataUnit).

Data Import/Export

Data are saved in disk files in one of two formats: text or binary. In text format, data values are ASCII codes, and can be viewed in any text editor. In binary format, data values are not ASCII codes and cannot be viewed in a text editor. You may wish to import data to be plotted or operated on mathematically, and export data for future use. If you want to save data between MATLAB sessions, use save and load functions.

To export (save) the following array:

```
Array = 2x3
    1     2     3
    4     5     6
```

in delimited ASCII format in the file `MyData.txt` use the command:

The load command is the reverse of save, but has a curious twist. If the array `A` has been saved in `MyData.txt` as above the command creates a variable in the workspace with the same name as the file, minus the extension, i.e., `MyData`.

If you don't want the filename as the variable name use the functional form of the command, e.g.:

```
Numbers = 2x3
    1     2     3
    4     5     6
```

If the variables `x`, `y` and `z` exist in the workspace, the command:

```
x =
2
y = 1x3
    1     2     3
z = 3x1
    4
    5
    6
```

saves the variables `x`, `y` and `z` in the file `MyVariables.mat` in MATLAB proprietary binary format, i.e., such a MAT-file can only be used by MATLAB.

The command:

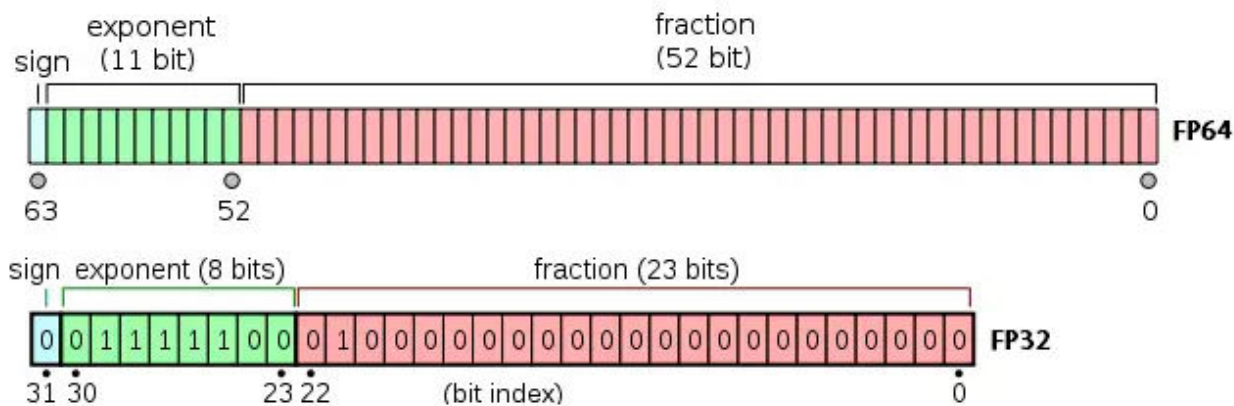
loads all the variables from `MyVariables.mat` into the workspace; see help for all the load options.

Computer Precision

Computer precision is a measure of how accurately a numerical value can be represented in a computer memory. It depends on the number of bits used to store the value and the format of the value. Floating-point format is one of the common formats for representing fractional or very large values by using a "floating point". It is called floating point because the decimal point can "float" to any position necessary to represent a number. For example, 0.00123 can be written as 1.23×10^{-3} where the decimal point has moved three places to the left.

Double and **single floating-point** are two types of floating-point formats that differ in their precision and range. Double floating-point uses 64 bits to store a value, while single floating-point uses 32 bits. This means that double floating-point can represent more digits and larger values than single floating-point, but it also takes more memory space and computation time.

A floating-point number consists of three parts: a sign bit, an exponent part, and a fraction part. For a double floating-point number, the sign bit (1 bit) indicates whether the number is positive or negative. The exponent part (7 bits for numbers and 1 bit for its sign) determines the magnitude or scale of the number. The fraction part (52 bits) specifies the digits after the decimal point.



The largest positive value that can be represented by double floating-point is about $(2 - 2^{-52}) \times 2^{1023}$, while for single floating-point it is about $(2 - 2^{-23}) \times 2^{127}$. Check using `realmax` function:

```
ans =
1.7977e+308

ans = single

3.4028e+38
```

Operators, Expressions and Statements

Numbers can be represented in the usual decimal form (fixed point) with an optional decimal point:

```
ans = 1x4
10^9 ×
    0.0000    -0.0000     0.0000     1.2345
```

The default numeric data type is double precision; all MATLAB computations are in `double`. MATLAB also supports signed and unsigned integer types and single-precision floating-point, by means of functions such as `int8`, `uint8`, `single` and so on.

The arithmetic operations on two scalar constants or variables are shown below:

Operation	Algebraic form	MATLAB
Addition	$a + b$	<code>a + b</code>
Subtraction	$a - b$	<code>a - b</code>
Multiplication	$a \times b$	<code>a * b</code>
Right division	a/b	<code>a / b</code>
Left division	b/a	<code>a \ b</code>
Power	a^b	<code>a ^ b</code>

The precedence rules for the operators in are shown below:

Precedence	Operator
1	Parentheses (round brackets)
2	Power, left to right
3	Multiplication and division, left to right
4	Addition and subtraction, left to right

MATLAB has four additional arithmetic operators, as shown below that work on corresponding elements of arrays with equal dimensions. They are sometimes called array or element-by-element operations because they are performed element by element:

Operator	Description
<code>.*</code>	Multiplication
<code>./</code>	Right division
<code>.\</code>	Left division
<code>.^</code>	Power

For example:

```
ans = 1×3  
    12     8    10
```

Compare the output with:

```
ans = 1×3  
    12     8    10
```

An expression is a formula consisting of variables, numbers, operators, and function names.

```
ans =  
6.2832
```

MATLAB statements are frequently of the form *variable = expression*.

With array operations, you can easily evaluate a formula repeatedly for a large set of data; this is called formula vectorization:

```
1.0e+04 *  
  
    0.0750    0.1776  
    0.1000    0.2367  
    0.3000    0.7102  
    0.5000    1.1837  
    1.1999    2.8406
```

MATLAB supports two types of syntax for calling functions: command syntax and function syntax. Command syntax passes inputs as character vectors and cannot pass variable values. Function syntax passes inputs as arguments enclosed in parentheses and can pass variable values. For example:

Command syntax is useful for interactive work when you want to avoid typing quotes and parentheses, but function syntax is more flexible and robust.

Mathematical Functions

Interesting problems in science and engineering are likely to involve special mathematical functions like sines, cosines, logarithms, etc. MATLAB comes with a large collection of such functions.

```
x =  
-3.1416  
  
y =  
2.0944  
  
ans =  
3.1416  
  
ans =  
3.1416 - 1.8115i  
  
ans =  
1.8115 + 3.1416i  
  
ans =  
-1.5708 + 1.8115i  
  
ans =  
-1.8623  
  
ans =  
-1.2626  
  
ans =  
2.5536  
  
ans =  
-0.3298 - 1.5708i  
  
ans =  
-3  
  
ans =  
-1  
  
ans =  
11.5920  
  
ans =  
8.1656e+15  
  
ans =  
-8.1656e+15  
  
ans =  
-3.1416
```

```
ans = datetime
      12-Oct-2024 08:14:51
```

```
ans =
      0.0432
```

```
ans =
      -3
```

```
ans =
      -4
```

```
ans =
      1
```

```
ans =
      1.1447 + 3.1416i
```

```
ans =
      0.4971 + 1.3644i
```

```
ans =
      -3.1416
```

```
ans =
      -3.1416
```

```
ans =
      -3.1416
```

```
ans =
      0.1133
```

```
ans =
      -3.1416
```

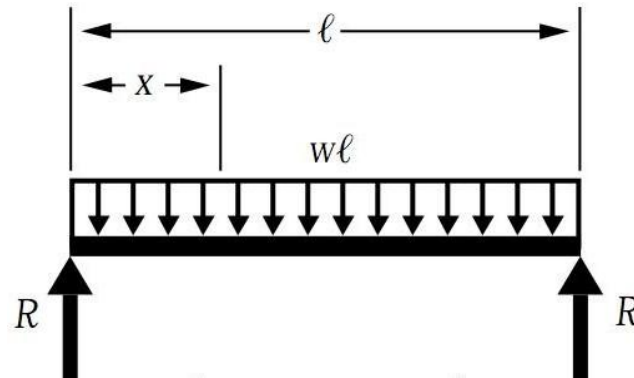
```
ans =
      0.2994
```

```
ans =
      1.7977e+308
```

```
ans =
      2.2251e-308
```

```
ans =
      -1.0472
```

Physics Pinpoint : Uniformly Distributed Load



A uniform beam is freely hinged at its ends $x = 0$ and $x = L$, so that the ends are at the same level. It carries a uniformly distributed load of W per unit length, and there is a tension T along the x -axis. The deflection y of the beam a distance x from one end is given by:

$$y = \frac{WEI}{T^2} \left[\frac{\cosh(a(L/2 - x))}{\cosh(aL/2)} - 1 \right] + \frac{Wx(L - x)}{2T}$$

where $a^2 = \frac{T}{EI}$, E being the Young's modulus of the beam, and I is the moment of inertia of a cross-section of the beam. The beam is 10 m long, the tension is 1 000 N, the load 100 N/m, and $EI = 10^4$. The script to compute and plot a graph of the deflection y against x is:

