Chapter 1

1.1 - Radioactive Decay

Concerning radioactive decay, while trying to study the decay of a material it's best to consider having a large sample size of the material. If N is the number of a material's nuclei present in a sample at time, t, the behavior is governed by the differential equation

$$\frac{dN}{dt} = \frac{-N}{\tau}$$

Where T is the "time constant" for the decay. Using direct substitution, the solution to the equation is

$$N = N(0)e^{-t/\tau}$$

N(0) is the number of nuclei present at $t = 0$. At $t = \tau$ a fraction e^{-1} of the initial nuclei have not yet decayed.

1.2 - A Numerical Approach

The differential equation can be solved without using any numerical methods, although, there are various computational methods we can use here that will be useful later.

Our goal is to find N as a function of t. We already have N at one value ($t = 0$) we just have to find it at some later value. This is an **initial value problem**, and here we'll describe N using the Taylor expansion

$$N(\Delta t) = N(0) + \frac{dN}{dt}\Delta t + \frac{1}{2}\frac{d^2N}{dt^2}(\Delta t)^2 + \ldots,$$

$N(0)$ is value of our function of time $t = 0$, $N(\Delta t)$ is the value at $t = \Delta t$. If Δt is considered small, we can approximate that all the values above the second powers and higher can be ignored. Leaving us with

$$N(\Delta t) = N(0) + \frac{dN}{dt}\Delta t$$

The same result can be obtained from the definition of a derivative. The derivative N with respect to τ can be written as

$$\frac{dN}{dt} = lim_{\Delta t \to 0}\frac{(N(t + \Delta t) - N(t)}{\Delta t} \approx \frac{N(t + \Delta t) - N(t)}{\Delta t}$$

Assuming deltat is small and nonzero, we get

$$N(t+\Delta t) \approx N(t) + \frac{dN}{dt}\Delta t$$

A more functional form of the derivative is

$$N(t+\Delta t) \approx N(t) - \frac{dN}{dT}\Delta t$$

The last two equations are known as the **Euler Method**

1.3 - Design and Construction of a Working Program: Codes and Pseudocodes

Pseudocode is a description of essential parts of an algorithm expressed in common language, although, it isn't a precise coding language. This book will go into detail how each line of pseudocode translates to actual lines of code.

Throughout the book most code will only be in the form of pseudocode. Code works for Fortran, C, and Basic languages.

When writing code, it's best to keep everything from equations to variables organized so you, or other programmers can understand your code.

Beginning your program with comment strings is a good way to identify your programs and explain what each one does.

The names of each of your variables and arrays can also clarify what each one is used for and what they calculate.

See pages 4 - 10 for all of the code examples

The final job is to organize the results of your functions into plots and graphs to better understand them. The ability to display results graphically is essential for working in computational physics.

1.4 - Testing your Program

A program should never be considered to be working unless you check the program and make sure the output is correct. There are several things you can do to make sure the results of the program are correct.

Always having a rough idea of what the output should look like before using the program so you can check the program against your original results.

Checking your program's results against any outside sources that have already been proven to some capacity.

Always check that your program gives consistent results.

1.5 - Numerical Considerations

Numerical errors are central to computational solutions to any problem, errors are produced by the finite mathematical precision of any programming language, **round-off errors**. Generally, there is no guarantee that these errors will be negligible since numerical solutions for certain types of problems are inherently sensitive to round-off errors.

It is difficult to come up with general rules about what to watch out for. Instead we can follow guidelines for what to do in case errors appear.

- A calculation should always be repeated using several different values of the step size.
- The choice of the step sizes. Preferably, you'd pick one that's small compared to any characteristic time scales in the problem.
- And finally, there is no best method to solving an ordinary differential equation or any other class of problem. Choosing the right method is up to your understanding of the problem.

1.6 - Programming Guidelines and Philosophy

When constructing a program it is best to follow certain guidelines to ensure the program is easy to understand.

- **Program structure**: Using subroutines and functions to organize all of the tasks.
- **Descriptive names**: Giving your variables and functions names that are able to describe their function.
- **Comment statements**: Include comment statements to explain your functions and variables in depth.
- **Sacrifice for Clarity**: Rather than having your code clumped together in a compact form, give a few more lines or make more variables to make the code easier to read
- **Make your graphical outputs clear**: Think carefully about what quantities to plot and how. Axes labels, titles, legends, parameters, etc.