Tia Snyder
PHYS250 - Reading Summary 1

**Chapter 1:**

**1.1 Radioactive Decay**

- Many nuclei are unstable (like $^{235}U$ which has a small probability for decaying into two nuclei of approximately half its size)
  - $(dN_U/dt) = -N_U/\tau$ is the behavior governed by differential equations where $N_U(t)$ is the number of uranium nuclei that are present in a sample at time t
    - T is the "time constant" for decay
  - $N_U = N_U(0)e^{-t/\tau}$ where $N_U(0)$ is the number of nuclei present at t = 0

**1.2 A Numerical Approach**

- $N_U$ should be obtained as a function of t
  - Taylor expansion:
    - Given that delta t is small, then we can determine:

$$N_U(\Delta t) = N_U(0) + \frac{dN_U}{dt}\Delta t + \frac{1}{2}\frac{d^2N_U}{dt^2}(\Delta t)^2 + \cdots,$$

$$\frac{dN_U}{dt} \equiv \lim_{\Delta t \to 0} \frac{N_U(t+\Delta t) - N_U(t)}{\Delta t} \approx \frac{N_U(t+\Delta t) - N_U(t)}{\Delta t},$$

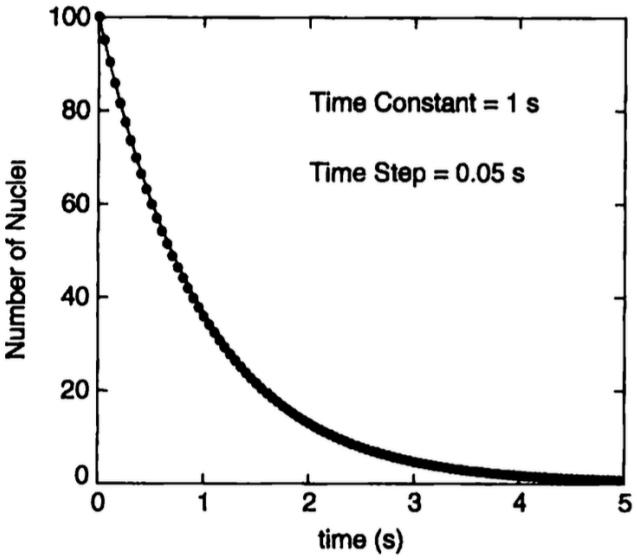
- The derivative of $N_U$ evaluated at time t can be written as:

$$\frac{dN_U}{dt} \equiv \lim_{\Delta t \to 0} \frac{N_U(t+\Delta t) - N_U(t)}{\Delta t} \approx \frac{N_U(t+\Delta t) - N_U(t)}{\Delta t}$$

- The equation above is an approximation
  - The error terms were dropped in deriving this new equation
  - We will use pseudocode along with actual code in order to see how the translation from pseudocode to an actual code can be done
    - It is important to understand code that we write as well as code written by others

**1.3 Structures of Coding**

- Code structure consists of (1) declaring necessary variables, (2) initializing all variables and parameters, (3) doing the calculations, and (4) storing the results
  - The code should start with a few comment statements that identify the program and tell what it is supposed to do
- Although the code will display results as numbers in a file for how $N_U$ varies with time, we still need to understand the result
  - Best done by examining the results in graphical form (VERY ESSENTIAL)
  - Graphs should look something like the one below:




**1.4 Testing Your Program**

- A working program considers not only no errors in syntax or spelling but also that the output is correct
- Guidelines to debugging
    - Does the output look reasonable?
        - Is it consistent with your intuition and instincts?
    - Does your program look reasonable?
        - Check each step of every calculation leading up to the final result
    - Check that the program gives the same answer for different values such as “step sizes”
        - The final answer should be independent of the values of parameters such as the time-step variable

**1.5 Numerical Considerations**

- Issues with numerical errors is central to the computational solutions of any problem
- You need to choose the best algorithm for a particular problem
    - How to estimate the numerical errors associated with the algorithm is very important
- With a radioactive decay program, errors were introduced by the approximation used to estimate the solution of the differential equation
    - Also produced by finite numerical precision in any programming language
        - Round-off errors
    - Time was treated as a discrete variable
        - Converted the differential equation into a difference equation
            - Allows us to compute estimates for $N_U$ at the discrete times n*delta t where n is an integer

- Always check that the calculated result converges to a fixed value as the step size is made smaller
    - Make sure that your step size is a reasonable value

**1.6 Programming Guidelines and Philosophy**

- Program structure: use subroutines to organize the major tasks and make the program more readable and understandable
    - The main program for the decay problem was basically an outline for the program
        - Subroutines or functions should be used when they are no more than a few lines of code or are required repeatedly
- Use descriptive names: choosing names of variables and subroutines according to the problem will make it easier to follow
    - Built in comment statements also make the code more understandable
        - Explain the program logic and describe the variables
- Sacrifice everything for clarity: compact code may occasionally run faster but it is a less reliable indication of computational efficiency
    - Compact code will always have a cost in terms of clarity and readability