Chapter 1: Introduction to Computational Physics

1. History and Evolution of Computational Physics

- Digital computers emerged in the 1940s and were almost immediately used for solving physics problems.
- ENIAC (Electronic Numerical Integrator and Computer) was the first digital computer in the U.S., completed in 1946.
    - Contained 17,000 vacuum tubes and over 1,000 relays.
    - Performed complex physics calculations in nuclear physics, statistical physics, and condensed matter.
    - Later replaced by faster computers after the invention of transistors and microprocessors.
- Computers have since become essential tools in physics, transforming nearly every major physics discovery.

2. Importance of Computational Physics

- Traditional physics education focuses on fundamental theories with problems that can be solved using:
    - Pen, paper, and basic algebra or calculus.
    - Perseverance and mathematical manipulation.
- However, in real-world physics:
    - Many problems cannot be solved purely analytically.
    - Some questions require theoretical calculations because:
        - Experiments are impossible (e.g., modeling early universe conditions).
        - Theory helps interpret experimental data.
        - Simulations provide deeper insights that experiments alone cannot.
- "Numerical" methods (i.e., using computers) dominate modern physics research.
-

3. Categories of Computational Problems in Physics

Most numerical computations fall into key mathematical operations, including:

- Integrals & derivatives: Needed for solving equations involving change.
- Linear algebra operations:
    - Matrix inversion (important for solving systems of equations).
    - Eigenvalue calculations (used in quantum mechanics and material science).
- Differential equations:

    - Ordinary Differential Equations (ODEs) and Partial Differential Equations (PDEs).
    - Used for physics simulations like heat diffusion and wave equations.

4. Approach in This Book

- The book focuses on computational techniques step by step:
    1. Learn the mathematical operations behind physics problems.
    2. Implement these operations in code (mainly Python).
    3. Apply them to a wide range of physics applications.
- Goal: Develop an intuition for computational physics beyond just programming.

5. Examples of Computational Physics Applications

- Dark Matter Simulations:
    - Example from Volker Springel's astrophysical calculations.
    - Used billions of particles to simulate the universe's structure over a 2 billion light-year scale.
    - Computation required a supercomputer running for over a month.
- Fluid Dynamics Simulation:
    - Example from Hans Johnston and Charles Doering.
    - Modeled Rayleigh-Bénard convection—how heated fluids move under turbulence.
    - Showed temperature variations and convection rolls in fluid systems.
- Condensed Matter & Statistical Physics:
    - Example from David Adams simulating diffusion-limited aggregation.
    - Studied how atoms cluster together randomly to form structures.

6. Introduction to Python for Computational Physics

- Why Python?
    - Widely used in physics due to its simple syntax.
    - Can handle:
        - Complex numbers, vectors, matrices, tensors.
        - Scientific calculations like Fourier transforms and calculus.
    - Built-in visualization tools for graphs and animations.
    - Free & open-source, widely supported in the scientific community.
- First Python Example
    - Calculates hydrogen emission spectra using the Rydberg formula
        - $1/\lambda = R((1/m^2) - (1/n^2))$
    - A simple Python program prints wavelengths of hydrogen's spectral lines for different values of m.

7. Key Takeaways on Python and Computational Physics

- Even without prior coding experience, Python is straightforward to learn.
- Enables solving physics problems with just a few lines of code.
- Python is constantly evolving:
    - Version 3 is currently the standard.
    - Older Version 2 is largely obsolete but still in some legacy systems.
- Other computational physics languages include C, C++, Fortran, Java, but Python is a great starting point.