

Computational Physics

There are a very few physical systems which can be described by an explicit formula like

$$x(t) = x_0 \cos(\omega t) + \frac{v_0}{\omega} \sin(\omega t) ,$$

which represents the motion of a harmonic oscillator and is the solution of Newton's equation of motion

$$m\ddot{x} = -m\omega^2 x .$$

This is a second order ordinary differential equation.

In most cases, the equations of motion are too complicated to solve exactly, and approximate numerical solutions must be found. This is the subject of computational physics.

Computational science

Computational physics shares with computational science the following methodology:

1. *Choose a sufficiently simple model.* Most natural phenomena are too complicated to describe exactly. A simplified representation of the physical system must be chosen. The model must be described by mathematical equations which can be solved in practice. The model must represent a *computable function*.
2. *Find the most efficient algorithms to solve the model equations.* There are many remarkable algorithms, like the Fast Fourier Transform, which have enabled the computational solution of otherwise intractable problems.

3. *Implement the algorithms to take maximum advantage of available computer hardware and software.* For a long time, Fortran was the language of choice for numerical programming. Now, languages like C/C++ and Java are being used increasingly for scientific programming, as are higher level programming systems like Mathematica, Matlab, and Maple.
4. *Visualize and interpret the output of the computation.* This is especially important when computer programs generate huge amounts of numerical data rather than a single numerical answer. In *computer simulations* visualizing the intermediate steps in the computation can also be very useful. Computational scientists need to be able to use computer graphics and animation techniques.

Topics in Computational Physics

In the lectures this semester we will develop computer codes in C/C++ to solve typical problems in various interesting application areas:

- **Molecular Dynamics.** The 2-D Lennard Jones system was studied last semester. This semester we will start by developing a program to solve the 3-D problem using Verlet's neighbor lists. The physics involved is very simple: the system is governed by Newton's classical equations of motion.
- **The N-Body Problem.** This involves solving Newton's equations of motion with long range forces, for example in a charged plasma, or the dynamics of stars in a galaxy. We will implement *tree code* algorithms to calculate the instantaneous accelerations of the N bodies efficiently.
- **Monte Carlo Methods.** Last semester we studied the statistical mechanics of the 2-

D Ising Model. This semester we will implement *cluster algorithms* to study this type of system. We will also study Quantum Monte Carlo methods.

- **Multigrid Methods.** These methods are becoming very popular in a variety of applications. We will see how these methods can be used to greatly speed up the solution of Poisson's equation and similar problems.
- **Computational Fluid Dynamics.** This is a vast and important subject. We will study some simple nonlinear wave equations, some of which have very interesting *solitary wave* solutions. If time permits, we may study the *finite element* approach to solving fluid dynamic equations.
- **Cellular Automata.** These very simple models have been used to model a variety of very complex physical systems such as avalanches, forest fires, and ecological systems. We will develop programs to study some of these systems as time permits.

Homework will be assigned approximately every two weeks based on the topics discussed in class. In addition, you must choose a topic, either from the list above or any other problem that you find interesting, and complete a semester project on it. The time table for completing the semester project is as follows:

- Choice of topic and one page abstract due Friday, January 31. This project proposal should consist of a title, a brief description of the problem you would like to simulate, and a list of references you plan to use.
- Mid-semester project report due Friday, March 7. This should consist of an updated proposal, and a description of results obtained so far, or of problems that you have encountered. A printout of programs you are working on may be submitted if you have not yet obtained

any results.

- The semester project report is due at the end of the semester. A reasonable length for the report is 20–30 pages, which should include a discussion of theory, algorithms used, results presented graphically if possible, and listing of interesting parts of the computer programs you developed. Sources used for your project should be properly referenced.
- You will make an oral presentation of your project to the class at the end of the semester. Each presentation will last approximately 30 minutes.

C++ programming

A simple C++ program

The following program prints a greeting on the user's console:

```
#include <iostream>
using namespace std;

int main() {
    cout << "Hello, World!" << endl;
}
```

You should be familiar with compiling and running such a program. If the code is contained in a file called `hello.cpp`, then on a Unix system you can use the GNU compiler

```
> g++ hello.cpp  
> ./a.out
```

Here is an equivalent program written in C:

```
#include <stdio.h>  
  
int main(void) {  
    printf("hello, world\n");  
    return 0;  
}
```

The Simple Harmonic Oscillator

For those students who have not taken PHY 410-505, we discuss a program to solve the harmonic oscillator equation

$$\frac{d^2x}{dt^2} = -\omega^2 x ,$$

for which we wrote a C++ program last semester.

The Euler-Cromer algorithm

To solve the equation of motion it is simplest to write it as a set of two coupled first order differential equations:

$$\begin{aligned}\frac{dx}{dt} &= v , \\ \frac{dv}{dt} &= -\omega^2 x .\end{aligned}$$

These equations can be solved using by choosing a small time step dt , choosing an initial position $x(0)$ and velocity $v(0)$ at time $t = 0$, and using the recurrence relations

$$\begin{aligned}v(t + dt) &= v(t) + a(t)dt , \\ x(t + dt) &= x(t) + v(t + dt)dt .\end{aligned}$$

Note that the new velocity $v(t + dt)$ at the next time step is used to obtain the new position $x(t + dt)$. Using the old value $v(t)$ yields the *Euler algorithm*, which turns out to be unstable for this problem and does not conserve energy

$$E = \frac{1}{2}mv^2 + \frac{1}{2}m\omega^2x^2 .$$

A C++ program to implement this algorithm

We first include various standard library headers:

```
#include <cmath>
```

```
#include <fstream>
#include <iostream>
#include <string>
using namespace std;
```

`cmath` defines the arctangent function `atan` which will be used to compute π , `fstream` allows us to open a file linked to an `ofstream` object, and `string` allows us to process the name of the output file.

Next, we define a set of *global* variables:

```
double omega;           // the natural frequency
double x, v;            // position and velocity at time t
int periods;            // number of periods to integrate
int stepsPerPeriod;     // number of time steps dt per period
string fileName;        // name of output file
```

and the *prototypes* of some functions we will use in the program

```
void getInput();         // for user to input parameters
void EulerCromer(double dt); // takes an Euler-Cromer step
double energy();         // computes the energy
```

The `getInput` function uses the iostream objects `cout` to write to the user's console and `cin` to read input from the keyboard:

```
void getInput ( ) {  
    cout << "Enter omega:  ";  
    cin >> omega;  
    cout << "Enter x(0) and v(0):  ";  
    cin >> x >> v;  
    cout << "Enter number of periods:  ";  
    cin >> periods;  
    cout << "Enter steps per period:  ";  
    cin >> stepsPerPeriod;  
    cout << "Enter output file name:  ";  
    cin >> fileName;  
}
```

The `EulerCromer` function advances the global variables `x,v` by one time step `dt`:

```
void EulerCromer (double dt) {  
    double a = - omega * omega * x;  
    v += a * dt;  
    x += v * dt;
```

```
}
```

The **energy** function computes the current value of the energy from the global position and velocity:

```
double energy ( ) {  
    return 0.5 * (v * v + omega * omega * x * x);  
}
```

Finally, we define the **main** function which gets the input parameters from the user

```
int main ( ) {  
  
    getInput();
```

opens the output data file

```
    ofstream file(fileName.c_str());  
    if (!file) {  
        cerr << "Cannot open " << fileName << "\nExiting ...\n";  
        return 1;
```

```
}
```

defines π , computes the time step and writes the initial data in the file

```
const double pi = 4 * atan(1);  
double T = 2 * pi / omega;  
double dt = T / stepsPerPeriod;  
double t = 0;  
  
file << t << '\t' << x << '\t' << v << '\n';
```

steers the calculation for the desired number of periods and time steps per period

```
for (int p = 1; p <= periods; p++) {  
  
    for (int s = 0; s < stepsPerPeriod; s++) {  
        EulerCromer(dt);  
        t += dt;  
        file << t << '\t' << x << '\t' << v << '\n';  
    }  
  
    cout << "Period = " << p << "\ttt = " << t
```

```
<< "\tx = " << x << "\tv = " << v  
<< "\tenergy = " << energy() << endl;  
}
```

and finally closes the output file

```
file.close();  
}
```

Zeroth assignment

This assignment need not be submitted!

- If you don't have a computer account, get one as soon as possible.
- Learn how to compile and run the two programs discussed today. If you are using a PC you will need to install a C/C++ compiler and a programmers editor.
- Take a look at the recommended textbooks. Buying a textbook is not required, but having one will be helpful to supplement the lecture notes. Thijssen's book is a good choice for physics graduate students: it discusses many topics suitable for projects. Gould and Tobochnik's book is a good choice for undergraduate students.
- Start thinking about a topic for your semester project. You will not gain much from this course just by listening to the lectures. Working on the homework assignments, and espe-

cially the semester project, will really help you learn computational physics.