Common problems

Common Problems

Installing the development environment

The key point here is to do this before Easter, so you can ask for expert help in Cambridge if you run into difficulties. Be careful to follow the step-by-step "getting started" instructions to the letter. Do not think everything is working until you have successfully compiled and run the C++ source code.

Assignment 1: Verlet integrator

The instructions do not dictate what to do on the first iteration. This is up to you: do something sensible. Python has some nice syntax for indexing into lists from the end instead of the start. For example, **x_list[-1]** is the last entry in **x_list**, and **x_list[-2]** is the penultimate entry.

Assignment 2

Compared with Assignment 1, the most significant change is that your state variables will now be 3-element vectors, not scalars. Consequently, the trajectory records will be *n*x3 matrices, not vectors. You might as well model Mars in this assignment: values for its mass and radius can be found online or in the C++ header files. For straight down descent, make sure you initialize the body above the surface. The expression for the gravitational force is (of course) only valid above the surface, so you should probably add a check for impact and terminate the simulation at that point (use Python's **break** command). Investigate whichever orbits and hyperbolic escapes you like. A back-of-the-envelope calculation should give you the initial tangential velocity for a circular orbit at radius *r*.

Assignment 3

Before measuring execution times, don't forget to comment out any file-writing or graph-plotting parts of the programs, leaving just the Euler iterations. **trajectories.txt** is created in the same folder as the compiled (machine code) executable file. This can be rather difficult to find: for example, Xcode buries it somewhere deep inside your **Library** folder. For more control over the file's location, specify a full file path when opening the file, e.g. **fout.open("/Users/XXX/trajectories.txt")** for macOS, where XXX is your login identifier. For the C++ Verlet implementation, you will need to access previous position values from **x_list**. **x_list[x_list.size()-1]** is the last entry in the list, while **x_list[x_list.size()-2]** is the penultimate entry.

Assignment 4

Before posting a query to the forum, make sure you have read and absorbed Appendix II of the instructions. If your integrators do not work as expected, add some **cout** commands inside **numerical_dynamics** to display the contents of the variables as you calculate them: this should help you spot any bugs. The Verlet integrator requires the use of one static variable, and should be structured something like this:

```
static vector3d previous_position;
vector3d new_position;

if (simulation_time == 0.0) {

   do an Euler update for the first iteration
   i.e. new_position = .... (Euler update, using position and velocity)
   velocity = .... (Euler update, using acceleration)

} else {

  do a Verlet update on all subsequent iterations
  i.e. new_position = .... (Verlet update, using position and previous_position)
  velocity = ... (Verlet update, using new_position and position)

}

previous_position = position;
position = new_position;
```

Finally, when calculating the lander's mass, do not forget to account for the amount of fuel that has been burned.

Assignment 5: exporting data to Python

The **ofstream** method you saw in **spring.cpp** in Assignment 3 will get you most of the way there, though you may need to find a way to make **open()** append to a file instead of overwrite it. A quick web search should suffice. Once you have the values recorded to a file (call it "results.txt" or similar), use **numpy.loadtxt** to read the data into Python, as detailed at the bottom of **spring.cpp** in Assignment 3.

Last modified: Tuesday, 28 September 2021, 8:57 PM

Linux Jump to... Forum > C++ programming