

Assignment 2, Week 2

Gordon Erlebacher

2024-09-05

In this assignment, we will build on the particle simulation example discussed in class. Work is submitted via your github repository (we download the repository on once the assignment is due, so that changes are no longer accepted).

Sometimes I have not provided all necessary details, and you'll have to make your own decisions. You can ask us what to do, or alternatively, make a decision, and describe in your `README.md` the situation encountered and the decision made, along with a justification.

Enhanced Task

1. Extend the Vector Class:

- **Overload Additional Operators:**

- Implement additional operator overloads for the `-` (subtraction) and `*` (dot product) operators for 2D vectors. The `-` operator should allow for subtracting one vector from another, and the `*` operator should perform the dot product between two vectors.

2. Implement Vector Norms:

- **Support Different Norm Types:**

- Extend the `norm` function to support the following norms: (L1 norm, L2 norm, and infinity norm) using an additional argument specifying the norm type. See the skeleton file for the function signature.

3. Test with Various Scenarios:

- **Utilize GPT-4-o:**

- Use GPT-4-o to generate a wide range of test cases that include vectors of different dimensions, different types of norm calculations, and scenarios with multiple particles interacting through forces. Validate your operator overloading and norm implementations under various conditions, including edge cases. The tests should be in a separate file that includes the main `cpp` file.

Example Scenario to Implement

2D and 3D Particle Simulation:

- Create simulations where particles move in both 2D and 3D spaces. Apply forces to particles and observe how they move and interact, ensuring that your operators and norm calculations behave correctly.
- This requires the solution to the differential equation:

$$\frac{d^2 P}{dt^2} = F(t)$$

where F is an external force, which we'll take to be

$$F_{2D}(t) = (\sin(2t), \cos(2t))$$

and

$$F_{3D}(t) = (\sin(2t), \cos(2t), \cos(1.5 * t)),$$

in the 2D and 3D cases respectively. You'll use the Euler method to compute the velocity of the particle $P(t)$ in space:

$$v_{n+1} = v_n + \Delta t F_n$$

where $P(t^n)$ represents the particle position at time $t^n = n\Delta t$, which you should run with $\Delta t = 0.02$ and t ranging from 0 to 4. Plot the motion of a 2D and 3D particle in the x - y plane (x and y are the two components of P).

- Integrating the equation for velocity with respect to time, one derives the equation for particle position:

$$P_{n+1} = P_n + \Delta t v_n$$

In summary, the system of equations that provides the position and velocity of the particle subjected to a force $F(t)$ is:

$$P_{n+1} = P_n + \Delta t v_n \tag{1}$$

$$v_{n+1} = v_n + \Delta t F_n \tag{2}$$

Recall that a subscript n on any variable represents evaluation at time $t^n = n\Delta t$. We have assumed that the particle mass is unity (whatever the units used).

To calculate position P and velocity v requires initial values at time $t = 0$. Choose $P_0 = (0, 0)$ or $P_0 = (0, 0, 0)$ for the 2D or 3D case. For the velocity, choose $v_0 = (0., 0.)$ or $v_0 = (0., 0., 0.)$ for the 2D or 3D cases.

Expected Deliverables

Start from the file `homework2_skeleton.cpp`, which lists all the functions you must create and their signatures (arguments and argument types).

- **The following files:**

5 pts Makefile: Execute `make` to generate the executables `homework2.x` and `tests2.x`.

- `homework2.cpp`: contains all the enhancements to the `Vector` and `Particle` classes.
- `tests2.cpp`: contains a collection of unit tests to check that the code functions properly. You used AI to generate these tests. But they must run. The code generate might be hard to understand. Converse with the AI to better understand the code generated and your general knowledge.

25 pts `README.md`: contains explanations of what you've done. Include a description of how you used AI. Also include a section that describes some of the errors you encountered and how you resolved them. Include the two image generated in this file as well. The debugging section will count for 10 out of the 25 points.

- Any other files you feel are necessary. The required plots are generated with Python and are contained in the `images/` folder.

- **Extended Vector Class:**

20 pts A class that supports 2D and 3D dimensional vectors and implements overloaded operators for addition (5 pts), subtraction (5 pts), scalar product (5 pts), and the multiplication of a vector by a scalar (5 pts). The overloaded operations must work on 2D and 3D vectors. The operators that involve two vectors should `throw` or print an error if the number of elements of the two vectors don't match. 1 point is removed per operation if the error check is not provided. (Attention: points add up!)

- **Enhanced Particle Class:**

20 pts A class that simulates particle motion in 2D and 3D dimensional space, including the effects of forces.

- **Comprehensive Test Suite:**

10 pts A set of test cases, stored in the file `tests2.cpp` generated using GPT-4-o, that thoroughly validate the behavior of the vector operations, norm calculations, and particle simulations.

- **Documentation:**

- 10 pts All functions in your code must be documented, including an explanation on how your code handles different vector dimensions, the mathematical operations implemented, and the scenarios tested.
- 10 pts Inclusion of plots that display the particle trajectory (position as a function of time) in the $x - y$ plane. For the 3D case, simply ignore the z component.

Certainly! I'll generate 20 questions based on the homework assignment and the skeleton code provided. Here's the LaTeX output with an enumerated list of questions:

1 C++ Homework Understanding Assessment, no grade

AI Assistant

This assessment is not graded, but will help you prepare for the quizzes given at a later time. Use the AI if you do not know the answers, and make sure you understand the responses. If you do not agree with the responses, ask the same question with different wording.

1. Explain the purpose of the `Vector` class in the provided skeleton code. How does it support both 2D and 3D vectors?
2. What is the significance of using the `auto` keyword and range-based loops as mentioned in the skeleton code comments?
3. Describe the implementation of the dot product operation for vectors. How does it differ from scalar multiplication?
4. How would you modify the `Vector` class to support vector subtraction, as required by the assignment?
5. Explain the concept of operator overloading in C++. How is it used in the `Vector` class?
6. What is the purpose of the `friend` keyword in the context of operator overloading? Provide an example from the skeleton code.
7. Describe the different types of vector norms mentioned in the assignment. How would you implement them in the `norm` function?
8. Explain the purpose of the `Particle` class. How does it relate to the `Vector` class?
9. What is the significance of the destructor in the `Particle` class? Why is it important to implement it?

10. How does the `updatePosition` function in the `Particle` class work? What physical concept does it represent?
11. Describe the Euler method for solving differential equations. How is it applied in the particle simulation?
12. What are the initial conditions for the particle simulation as specified in the assignment? Why are they important?
13. Explain the purpose of the `force` function in both the `Vector` and `Particle` classes. How do they differ?
14. How would you implement error checking for vector operations when the dimensions don't match?
15. Describe the process of generating test cases using GPT-4-o. What types of scenarios should be included?
16. What is the purpose of the `test_operators` function in the skeleton code? How would you expand it to test new functionality?
17. Explain the structure of the `main` function in the skeleton code. How does it demonstrate the use of the `Particle` class?
18. How would you modify the code to output the particle trajectories to files as specified in the assignment?
19. Describe the process of visualizing the particle trajectories using Python. What libraries might be useful for this task?
20. Explain the importance of proper documentation in the context of this assignment. What key elements should be included in function documentation?

More questions but more related to C++:

1. What is the purpose of the `const` keyword in function declarations? Provide an example from the assignment.
2. Explain the concept of function overloading in C++. How is it used in the `Vector` class?
3. What is the difference between pass-by-value and pass-by-reference in C++? When would you use each?
4. Describe the purpose of the `std::vector` container. How is it used in the `Vector` class implementation?
5. What is the significance of the `explicit` keyword in constructor declarations? When should it be used?

6. Explain the concept of move semantics in C++. How could it be applied to improve the efficiency of the **Vector** class?
7. What is the purpose of the **noexcept** specifier? Where might it be appropriate to use in the given code?
8. Describe the difference between stack and heap memory allocation in C++. How does this relate to the **Vector** and **Particle** classes?
9. What is a copy constructor? When might you need to implement one for the **Vector** class?
10. Explain the concept of RAII (Resource Acquisition Is Initialization) in C++. How does it apply to the **Particle** class?
11. What is the purpose of the **static** keyword in C++? How might it be useful in the context of this assignment?
12. Describe the difference between **std::cout** and **std::cerr**. When would you use each in error handling?
13. What is function inlining in C++? Which functions in the **Vector** class might benefit from being inlined?
14. Explain the concept of namespaces in C++. How are they used in the provided code?
15. What is the purpose of the **constexpr** keyword? How could it be used to optimize the **Vector** class?
16. Describe the difference between compile-time and runtime polymorphism in C++. Which type is used in the assignment?
17. What is the purpose of the **std::runtime_error** exception? How is it used in the code?
18. Explain the concept of template metaprogramming. How could it be applied to make the **Vector** class more flexible?
19. What is the difference between **std::endl** and **'\n'** for line breaks? Which is preferable in most cases and why?
20. Describe the concept of SFINAE (Substitution Failure Is Not An Error) in C++. How could it be used to create more robust vector operations?