|  |  |  |  |
| --- | --- | --- | --- |
| **Lab 1.** | | | |
| **제출일** | **09/30** | **학번** |  |
| **전공** | **전자공학과** | **이름** |  |

|  |  |
| --- | --- |
| **①** **문제** | **1번:** X1(t), X2(t), X3(t), X4(t)의 3주기 그래프를 각각 그리기.  **3번:** X5(t)의 3주기 그래프 그리기. |
| **②** **주요 변수** | **- amplitude, frequency, phase**  차례대로 진폭, 주파수, 위상 값을 다루는 Signal 클래스의 private 변수로, 이들을 통해 코사인 함수 값을 구해낼 수 있으며, 코사인 함수 간의 합도 가능하다.  **- time\_values**  코사인 함수 값을 구해내는 기준이 되는 시간 데이터들을 담고 있는 벡터이다.  **- x1\_values, ~~**  각각의 함수의 코사인 값을 담고 있는 벡터이다. 해당 데이터를 .txt 파일에 저장하고, 이를 기반으로 엑셀을 통해 함수의 그래프를 그려낸다. |
| **③** **알고리즘** | **- Signal 클래스**  진폭, 주파수, 위상을 멤버 변수로 지니며, 시간 t에 대한 사인 함수 값과 코사인 함수 값을 반환해주는 멤버 함수도 지니고 있다. 여기에 두 Signal 클래스가 주파수가 같다면 덧셈 연산자로 계산이 가능하도록 연산자를 오버로딩하였다.  주파수가 같은 두 sinusoid 함수의 합을 통해 도출된 신호의 진폭과 위상은 다음과 같이 구할 수 있으며, 해당 공식을 통해 Signal 클래스 간 덧셈을 구현했다.  **- 메인 함수 알고리즘**  문제에서 주어진 x1 ~ x4 까지의 함수와 대응하는 Signal 클래스를 4개 생성한다. 이 4개의 Signal이 합쳐져 Signal x5를 생성한다. Signal의 멤버 함수를 통해 설정한 간격에 따라 3주기 동안의 코사인 함수 값을 계산해내어 x1 ~ x5의 각각의 값을 구분하여 기록한 .txt 파일을 생성한다. |
| **④** **결과** | **- 문제 1번**  그림 1.  X1(t) = 5cos(2π(15)t + 0.5π)를 나타낸 그래프. 5cos(2π(15)t)에 대해서 t에 대해 음의 방향으로 0.0166…만큼 Shifting된 모습을 확인할 수 있다.  그림 2.  X2(t) = 5cos(2π(15)t - 0.25π)를 나타낸 그래프. 5cos(2π(15)t)에 대해서 t에 대해 양의 방향으로 0.00833…만큼 Shifting된 모습을 확인할 수 있다.  그림 3.  X3(t) = 5cos(2π(15)t + 0.4π)를 나타낸 그래프. 5cos(2π(15)t)에 대해서 t에 대해 음의 방향으로 0.0133…만큼 Shifting된 모습을 확인할 수 있다.  그림 4.  X4(t) = 5cos(2π(15)t - 0.9π)를 나타낸 그래프이다. 5cos(2π(15)t)에 대해서 t에 대해 양의 방향으로 0.03만큼 Shifting된 모습을 확인할 수 있다.  **- 문제 3번**  그림 5.  X1(t) ~ X4(t)의 합의 결과인 X5(t)를 나타낸 그래프이다. 주파수는 같지만 위상이 다른 Sinusoid 함수들을 더했음에도 불구하고 동일한 주파수를 가지는 Sinusoid 함수가 도출됨을 알 수 있다. 즉 신호의 시간 축에 대한 Shifting을 나타내는 위상 차이는 신호의 주기적 진동을 나타내는 주파수에 영향을 주지 않음을 짐작할 수 있다. 추가적으로, 해당 문제로는 확인할 수 없지만, 신호의 진동 폭을 나타내는 진폭 또한 주파수에 영향을 주지 않는다.    그림 6.  Signal에 대한 덧셈 연산자가 정상적으로 작동하였는지 검증해보기 위해 프로그램을 통해 얻은 X5(t)의 데이터로 나타낸 그래프와 엑셀의 덧셈 함수로 X1(t) ~ X4(t)까지의 데이터를 직접 더한 결과값으로 나타낸 그래프를 비교한 결과이다. 두 그래프가 거의 일치함을 확인할 수 있으며, 나타나는 약간의 오차는 프로그래밍의 부동소수점 연산 오류에 의한 것임을 짐작할 수 있다.  **- 느낀점**  처음으로 프로그래밍 수업이 아닌 수업에서 프로그래밍을 적용해보았는데, class, operator overloading, reference, const method 등 지난 학기에 배웠던 중요한 개념들과 개인적으로 알고 있었던 Latex 문법 등도 다시 상기시킬 수 있어서 좋은 경험이었다. 특히 주파수의 불변성을 그래프를 통해 직관적으로 확인할 수 있었기에 추상적이던 개념을 확실하게 각인할 수 있었다. 또한 각각의 신호를 class로 나타낸 점을 보고, X5(t)에 대한 데이터를 X1(t) ~ X4(t) 까지의 데이터의 합을 요소로 하는 벡터와 같이 단순하게 구현하는 방법 대신, 연산자 오버로딩을 통해 X5(t) 신호에 대해서도 Signal class로 구현할 수 있도록 코드를 모색해보았다. 해당 코드를 구현해내면서 주파수가 같은 신호를 합한 결과의 진폭과 위상을 구하는 공식에 대해서도 이해하는 과정이 필요하였는데, 이때 주파수의 불변성은 위상에 대해서만이 아니라 진폭에 대해서도, 코사인 함수에 국한된 것이 아닌 sinusoid 함수 전체에 대해서도 유효하다는 것을 파악할 수 있었다. 따라서 주어진 함수들의 진폭을 다르게 하거나, 코사인 함수로 통일하지 않고 사인 함수를 섞어보는 등 문제의 조건을 변형하여 그래프를 구현해보는 것도 해당 과제의 연장선으로써 학습 효과를 높이는 좋은 방법이 될 수 있을 것 같다.  **- 클래스의 개념에 대해서**  클래스란 C++에서 기본으로 제공하는 데이터 타입 외의 새로운 타입을 직접 정의할 수 있는 템플릿으로, 클래스에 따라 생성된 인스턴스를 객체라고 한다. 대부분의 클래스는 멤버 변수와 메소드, 생성자 등으로 구성되어 있다. 멤버 변수는 객체가 가질 수 있는 데이터를 의미하며, 메소드는 객체가 호출할 수 있는 함수를 의미하고, 생성자는 객체를 생성할 때 초기화하는 특수 메소드다. 클래스를 통해 생성된 객체는 기존 데이터 타입처럼 동작하지 않을 수 있으므로 함수와 연산자에 대한 오버로딩을 통해 정상적으로 동작할 수 있도록 코딩해주는 작업이 필요하다. 이번 과제에서는 Signal이라는 클래스를 정의하여 진폭, 주파수, 위상을 데이터로 가지고 있고, (멤버 변수) 이를 통해 사인 또는 코사인 함수의 값을 계산할 수 있는 (메소드) 객체를 생성해낼 수 있도록 하였다. 또한 Signal 객체에 대하여 덧셈 연산자를 오버로딩하여 Signal 객체 간 덧셈이 용이하도록 하였다. 이처럼 클래스는 사용자가 특정한 데이터와 기능을 묶어 객체로 정의하고, 이를 통해 코드의 재사용성을 높여 체계적이고 효율적인 코드 설계를 가능하게 해준다는 점에서 프로그래밍에서 매우 중요한 개념이다. |