|  |  |  |  |
| --- | --- | --- | --- |
| **Lab 1.** | | | |
| **제출일** | **2023.10.09** | **학번** | **2022104263** |
| **전공** | **전자공학과** | **이름** | **석지원** |

|  |  |
| --- | --- |
| **①** **문제** | * **문제:**  1. 4개의 신호를 합쳐서 하나의 신호를 구하고 크기와 위상을 확인 하시오. (샘플링 주파수 를 결정하여 3주기의 를 출력) 2. Sampling Frequency를 변경하면서 Nyquist theory를 확인 하시오.  * **목적:**   C++를 이용하여 위의 4개의 sinusoid 함수들의 data를 sampling하고 합성 신호를 구한다. 이렇게 구한 합성 신호를 디지털에서 아날로그 신호로 복원하며 sample frequency를 변화시킬 때, 어떻게 복원되는지를 살펴본다. 주어진 신호의 최고 주파수 의 2배 이상의 주파수로 sampling을 하면 디지털에서 아날로그 신호로 복원하는 과정에서 원래의 신호를 보다 정밀하게 복원할 수 있다는 Nyquist theorem을 확인한다. |
| **②** **주요 변수** | * **Double:** double은 float의 두배의 크기로 더욱 높은 정밀도의 실수들을 다룰 수 있다. * **t:** 시간 * **freq:** 주파수 * **sample\_freq:** 사용자가 원하는 주파수 * **dt:** 1.0/sample\_freq로 dt와 sample\_freq는 역수 관계이다. 샘플링 주기. * **tt:** 각 cosine 함수의 시간을 double로 받는다. * **double aa, bb, cc, dd:** 각 함수의 값들을 double 형식으로 aa, bb, cc, dd에 저장한다. |
| **③** **알고리즘** | * **코드:**   #include <iostream> // c++의 입출력을 위함  #include <fstream> // file을 읽고 쓰기 위함  #include <cmath> // cosine 등의 수학을 사용하기 위함  using namespace std;  const double PI = 3.141592653589793238462643383279502884197; // pi의 값 정의  // x1(t) = 2 그래프에 대한 함수 값  double x1() {  double aa = 2;  return aa;  }  // x2(t) = 2 \* cos((30 \* PI \* tt) - (0.2 \* PI))에 대한 함수 값  double x2(double tt) {  double bb = 2 \* cos((30 \* PI \* tt) - (0.2 \* PI));  return bb;  }  // x3(t) = 4 \* cos((30 \* PI \* tt) - (0.25 \* PI))에 대한 함수 값  double x3(double tt) {  double cc = 4 \* cos((30 \* PI \* tt) - (0.25 \* PI));  return cc;  }  // x4(t) = 3 \* cos((30 \* PI \* tt) - (0.4 \* PI))에 대한 함수 값  double x4(double tt) {  double dd = 3 \* cos((30 \* PI \* tt) - (0.4 \* PI));  return dd;  }  int main() {  // x축 값인 time과 y축 값인 cosine 그래프에 대한 value를 저장하는 file 생성  ofstream out\_time;  ofstream out\_value;  out\_time.open("cosine\_signal\_time.txt");  out\_value.open("cosine\_signal\_value.txt");  double freq, sample\_freq, t, dt; //double로 주파수, 샘플 주파수, 시간, 시간 변화율 선언  t = 0;  freq = 440;  cout << "sample frequency : "; // sample frequency의 값을 입력 해야함을 알려주기 위해 출력  cin >> sample\_freq; // 사용자가 원하는 sample frequency를 입력 받음  dt = 1.0 / sample\_freq;  for (int i = 0; i < 30; i++, t += dt) {  out\_time << t << endl; // x축 value인 time  out\_value << x1() + x2(t) + x3(t) + x4(t) << endl; // y축 value인 4개의 신호를 합친 하나의 신호  }  // 열었던 file들을 닫는다.  out\_time.close();  out\_time.close();  }   * **알고리즘:**  1. <iostream>, <cmath>, <ofstream> 해더파일 선언. 첫번째 해더파일부터 c++의 입출력을 위함. Cosine 등의 수학을 사용하기 위함. File을 읽고 쓰기 위함. 2. 값을 설정. Const를 이용하여 double 형식으로 PI를 3.141592653589793238462643383279502884197로 정의. 3. 아래의 세 sinusoid 함수들에 대한 각각의 함수를 만든다. Main 함수를 줄이기 위함. 4. Ofstream을 이용하여 x축 값인 time과 y축 값인 cosine 그래프에 대한 value를 저장하는 file 생성. 총 두개의 파일을 만들어 x축 값과 y축 값 따로 저장. 5. For loop를 이용하여 dt만큼의 간격으로 30번 sampling한다. Time은 t 값, Value는 네 함수의 값들, 즉 4개의 신호를 합친 하나의 신호의 value들이다. 6. 열었던 file들을 다시 닫아주며 main함수를 끝낸다. |
| **④** **결과** | * **Sample\_freq = 15**     4개의 신호를 합친 신호의 주기와 같은 샘플링 주기를 가진다. 즉, 한 주기에 한 번만 데이터를 sampling한다. 따라서 일직선 형태의 그래프를 볼 수 있다.   * **Sample\_freq = 30**     4개의 신호를 더한 신호의 의 2배인 30을 sample frequency로 가진다. 즉, 한 주기에 두 번 sampling한다. Sample\_freq = 30부터 디지털에서 아날로그 신호로 복원하여도 대략적인 형태는 복원되지만 완벽하지는 않다.   * **Sample\_freq = 90**     Sample frequency가 의 2배인 30을 넘어간 후부터는 대략적인 그래프의 모양은 비슷하고 디테일한 신호 복원의 정밀도가 달라진다.   * **Sample\_freq = 120**     Sample frequency가 최대 주파수 의 2배인 30을 넘어간 이후부터는 대략적인 모양은 비슷하다. 하지만 sample frequency가 120이기 때문에 sample frequency가 90인 그래프보다 더 정밀하게 디지털 신호를 아날로그 신호로 복원한 것이다.  최대 주파수 가 30인 합성 신호의 sample frequency를 15, 30, 90, 120으로 변경시키며 그래프를 그렸다. 이 과정에서 최대 주파수 의 2배인 30을 sample\_freq로 가질 때부터 대략적인 그래프의 개형을 얻을 수 있었다. 이는 Nyquist theorem을 증명한다. 또한 sample frequency가 올라갈수록, 더욱 정밀하게 디지털 신호를 아날로그 신호로 복원할 수 있음을 보았다. |