

```
//-----Question 1-----  
-----
```

```
/* make_fourier_data.cc
```

```
generates a list of doubles to mimic the uniform  
sampling of a time-dependent signal.
```

```
output file: fourier_raw.dat
```

```
output file is eventually read in by  
fourier.cc to generate the digital  
fourier transform of the data created in the  
present code.
```

```
*/  
  
#include<iostream>  
#include <cmath>  
#include <fstream>  
  
using namespace std;  
  
const double two_pi = 2*acos(-1.0);  
double func(double x);  
  
int main()  
{  
  
    int i_max = 1000 ; // max number of points to sample  
    // int i,n;  
    // double data[n];  
    double y = 0.0;  
    double step = 0.001; // NOT the step size if the dft algorithm  
  
    const double f = 10;  
  
    ofstream outfile;  
    outfile.open("fourier_raw.dat"); // store output data  
  
    /*  
  
    for (i = 0; i < i_max; ++i)  
    {  
        data[i] = 0.0;  
    }  
  
    for (i = n / 3; i < 2 * n / 3; i++)  
    {  
        data[i] = 1.0;  
    }  
  
    for (i = 0; i < n; i++)  
    {  
        printf ("%d %e\n", i, data[i]);  
    }  
    printf ("\n");  
  
*/  
}
```

```
for (int k = 0 ; k < i_max; ++k){ // "sample" the waveform
    y = func(step *k*two_pi);
    outfile << k << "\t" << y << endl;
}

outfile.close();

cout << "data written to fourier_raw.dat" << endl;
return 0;
}

double func(double x){ // sampled waveform
    return sin(2*f*two_pi*x)+2*sin(3.0*f*two_pi*x) + 5*sin(4*f*two_pi*x); //function here
}

//-----dft -----
/*
dft.cc: Discrete Fourier Transformation

input file format: real y(t) values separated by whitespace
input file name:   fourier_raw.dat

output file name:   dft.dat
output file format: frequency index \t real part \t imaginary part

related programs: inv_dft.cc

*/

#include <iostream>
#include <cmath>
#include <fstream>

using namespace std;

const int i_max = 1000; // max number of input data
const double PI = acos(-1.0); // let the machine define pi

int main()
{
    double imag, real, input[i_max], x;
    int i=0,j,k;

    ifstream input_file;
    ofstream output_file;

    input_file.open("fourier_raw.dat"); // get input data
    output_file.open("dft.dat"); // save processed data

    for (i = 0; i < i_max; ++i) // zero out working array
    {
        input[i] = 0.0;
    }
}
```

```

    }

    i = 0; // reset i

    while (!input_file.eof() && (i < i_max)){ // fill sampled data array
        input_file >> j >> x;
        input[i] = x;
        ++i;
    }

    int N = i; // number of data points read

    for (j=0; j < N ; ++j) // loop over frequency index
    {
        real = 0.0;
        imag = 0.0; //clear variables
        for (k = 0; k < N; ++k) // loop for sums
        {
            real += input[k]*cos((2*PI*k*j)/N);
            imag += input[k]*sin((2*PI*k*j)/N);
        }

        output_file << j << "\t" << real << "\t" << imag << endl;
    }

    input_file.close();
    output_file.close();
    cout << "data stored in dft.dat" << endl;
    return 0;
}

/*
run these two programs then plot 'dft.dat',
the Hn is imaginary
*/

//-----Question 2-----

/* make_fourier_data.cc

generates a list of doubles to mimic the uniform
sampling of a time-dependent signal.

output file: fourier_raw.dat

output file is eventually read in by
fourier.cc to generate the digital
fourier transform of the data created in the
present code.

*/

#include<iostream>
#include <cmath>
#include <fstream>

using namespace std;

```

```
const double two_pi = 2*acos(-1.0);
double func(double x);
const double f = 10.5;

int main()
{

    int i_max = 1000 ; // max number of points to sample
    // int i,n;
    // double data[n];
    double y = 0.0;
    double step = 0.01; // in case of f1, change this to 0.1

    ofstream outfile;
    outfile.open("fourier_raw.dat"); // store output data

    /*

    for (i = 0; i < i_max; ++i)
    {
        data[i] = 0.0;
    }

    for (i = n / 3; i < 2 * n / 3; i++)
    {
        data[i] = 1.0;
    }

    for (i = 0; i < n; i++)
    {
        printf ("%d %e\n", i, data[i]);
    }
    printf ("\n");

    */

    for (int k = 0 ; k < i_max; ++k){ // "sample" the waveform

        y = func(step *k*two_pi);
        outfile << k << "\t" << y << endl;
    }

    outfile.close();

    cout << "data written to fourier_raw.dat" << endl;
    return 0;
}

double func(double x){ // sampled waveform
    return sin(f*two_pi*x); //function here
}

//-----dft -----

/*
```

dft.cc: Discrete Fourier Transformation

input file format: real y(t) values separated by whitespace

input file name: fourier_raw.dat

output file name: dft.dat

output file format: frequency index \t real part \t imaginary part

related programs: inv_dft.cc

*/

```
#include <iostream>
```

```
#include <cmath>
```

```
#include <fstream>
```

```
using namespace std;
```

```
const int i_max = 1000;
```

```
// max number of input data
```

```
const double PI = acos(-1.0);
```

```
// let the machine define pi
```

```
int main()
```

```
{
```

```
double imag, real, input[i_max], x;
```

```
int i=0,j,k;
```

```
ifstream input_file;
```

```
ofstream output_file;
```

```
input_file.open("fourier_raw.dat"); // get input data
```

```
output_file.open("dft.dat"); // save processed data
```

```
for (i = 0; i < i_max; ++i) // zero out working array
```

```
{
```

```
input[i] = 0.0;
```

```
}
```

```
i = 0; // reset i
```

```
while (!input_file.eof() && (i < i_max)){ // fill sampled data array
```

```
input_file >> j >> x;
```

```
input[i] = x;
```

```
++i;
```

```
}
```

```
int N = i; // number of data points read
```

```
for (j=0; j < N ; ++j)
```

```
// loop over frequency index
```

```
{
```

```
real = 0.0;
```

```
imag = 0.0;
```

```
//clear variables
```

```
for (k = 0; k < N; ++k)
```

```
// loop for sums
```

```
{
```

```
real += input[k]*cos((2*PI*k*j)/N);
```

```
imag += input[k]*sin((2*PI*k*j)/N);
```

```
}
```

```

    output_file << j << "\t" << real << "\t" << imag << endl;
}

input_file.close();
output_file.close();
cout << "data stored in dft.dat" << endl;
return 0;
}

//-----inv-dft-----

/*

inv_dft.cc: Inverse Discrete Fourier Transformation

(re)generates y(t) given its dft.

input file:    dft.dat (created by dft.cc)
input file format:  frequency index \t real part \t imaginary part

output file format: same as input

*/

#include <iostream>
#include <cmath>
#include <fstream>

using namespace std;

const int i_max = 1000;
const double PI = acos(-1.0);

int main()
{
    double imag, real, input [2][i_max]
, x, y;
    int i = 0, j, k;

    ifstream input_file;
    ofstream output_file;

    input_file.open("dft.dat");           //get data
    output_file.open("invers_dft.dat");   //save data

    while (!input_file.eof() && (i < i_max)){
        input_file >> j >> x >> y ;
        input[0][i] = x;
        input[1][i] = y;
        ++i;
    }

    int N = i;    // number of sampled values

    for (j=0; j < N; ++j)                // loop over frequency index
    {
        real = 0.0;                       // clear variables
        imag = 0.0;
        for (k = 0; k < N; ++k)          // loop for sum
        {
            real += input[0][k]*cos(2*PI*k*j/N)+input[1][k]*sin(2*PI*k*j/N);
            imag += input[1][k]*cos(2*PI*k*j/N)-input[0][k]*sin(2*PI*k*j/N);
        }
    }
}

```

```
    }
    output_file << j << "\t" << real/(double)N << "\t" << imag/(double)N << endl;
}

input_file.close();
output_file.close();
cout << "data saved in invers_dft.dat" << endl;
return 0;
}

/*
plot 'invers_dft.dat' in gnuplot,
*/
```