AEP 4380 HW#10: The Fast Fourier Transform (FFT) and Spectral Methods

Gianfranco Grillo

November 30, 2016

1 Background

1.1 Fourier Transforms and the FFT

Fourier transforms show up in many areas of the physical sciences. At its core, the basis behind the idea of a Fourier transform is simple: it is just a way of representing a certain function or data set as a sum of sines and cosines. This is useful in itself, because trigonometric functions have been studied for millenia, so it is often the case that it is easier to analyze the Fourier transform of a function than the original function itself, and thus Fourier transforms are a useful tool for indirectly studying the properties of the original function/data set. This homework follows this line of thinking by using Fourier transforms to approximate the solution to a second order differential equation.

A more technical way of looking at Fourier transforms is by regarding them as a way of transforming a signal from its time domain to its frequency domain, or viceversa. In other words, it allows one to take a function or data set that is sampled in the time domain and represent it in terms of its frequency, or the other way around. The process also works with other related units. For instance, one can Fourier transform a function that depends on position into one that depends on inverse wavelength. Fourier transforms come in two types: continuous and discrete. This homework makes use of Discrete Fourier Transforms (DFT). Given a function of position $h(x_i)$ sampled at discrete positions x_i, the DFT is given by

$$H(k_n) = \sum_{i=0}^{N-1} h(x_i) \exp(ik_n x_i) \tag{1}$$

$$h(x_i) = \frac{1}{N} \sum_{n=0}^{N-1} H(k_n) \exp(-ik_n x_i) \tag{2}$$

Here, (1) corresponds to the forward transform, and (2) corresponds to the inverse transform. k_n is a spatial frequency number, and N is the number of times the data was sampled.

Due to the DFT's multiple applications, numerical algorithms used to perform the transforms have a long development history. One of them, the Fast Fourier Transform, is among the fastest numerical algorithms available. This homework uses a variant of the FFT developed for C/C++ and Fortran, the FFTW (Fastest Fourier Transform in the West)[1], to perform the required transforms needed in order to solve the wave equation, as described in the next subsection.

1.2 Solving the 2D Wave Equation Using Spectral Methods

Certain partial differential equations can be numerically solved using spectral methods, which involve using FFTs in one way or another. The purpose of this homework is to use spectral methods to solve the 2D wave equation,

[1] More details about this algorithm, as well as the code itself, are available at www.fftw.org

given a set of initial conditions. For a displacement $\psi(x,\, y)$ as a function of position $\vec{x} = (x,\, y)$, and time t, the wave equation is given by

$$v^2 \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right) \psi = \frac{\partial^2 \psi}{\partial t^2} \tag{3}$$

where v is the wave's velocity of propagation. The spectral method assumes that (3) has a solution in the form of a Fourier series, given by

$$\psi(\vec{x},\, t) = \sum_{ij} a_{ij}(t) \exp(i\vec{k_{ij}} \cdot \vec{x}) \tag{4}$$

Here, $\vec{k_{ij}}$ and $\vec{x}$ are the 2D versions of the same variables as defined in the previous subsection, and a_{ij} is a complex constant. By substuting (4) into (3) and equating coefficients, one arrives to the solution

$$\psi(\vec{x},\, t) = \sum_{ij} \left[b_{ij} \exp(iv|\vec{k_{ij}}|t) + c_{ij} \exp(-iv|\vec{k_{ij}}|t) \right] \exp(i\vec{k_{ij}} \cdot \vec{x}) \tag{5}$$

where b_{ij} and c_{ij} are also complex constants. Finally, if we take the displacement to be released from a certain position initially at rest, such that $\frac{\partial \psi(\vec{x},\, t=0)}{\partial t} = 0$, $b_{ij} = c_{ij}$ and the final solution can be written as

$$\psi(\vec{x},\, t) = \sum_{ij} d_{ij} \cos(v|\vec{k_{ij}}|t) \exp(i\vec{k_{ij}} \cdot \vec{x}) \tag{6}$$

where $d_{ij} = 2b_{ij}$. For this homework, the spatial lengths L_x and L_y are set to $500\, m$, v corresponds to the speed of sound in air, $343\, ms^{-1}$, $N_x = N_y = 512$, and the initial displacement is taken to be the sum of three Gausians near the center, as follows:

$$\psi(\vec{x},\, t = 0) = \sum_{i=1}^{3} A_i \exp\left(-\frac{|\vec{x} - \vec{x_i}|^2}{s_i^2} \right) \tag{7}$$

where $\vec{x_i} = \begin{bmatrix} (0.4,\, 0.5,\, 0.6)L_x \\ (0.4,\, 0.6,\, 0.4)L_y \end{bmatrix}$, $s_i = (10,\, 20,\, 10)\, m$, and $A_i = (1,\, 2,\, 1)$. This setup implies that

$$|\vec{k_{ij}}| = \sqrt{\left(\frac{2\pi i}{L_x} \right)^2 + \left(\frac{2\pi j}{L_y} \right)^2} \tag{8}$$

Equation (6) can be solved by performing a Fourier transform on the initial displacement given by (7) in order to calculate the d_{ij} coefficients, and then multiplying these coefficients by the cosine factor from equation (6) with t set to the time in which the system wants to be observed, which in this case corresponds to 0.1, 0.2, and 0.4 seconds. The inverse transform of this result then is a representation of the state of the system at the chosen t.

2 Results

Figures 1-8 correspond to gray scale countour plots and 3D plots of the real part of $\psi(\vec{x},\, t)$ at the four different times indicated previously.

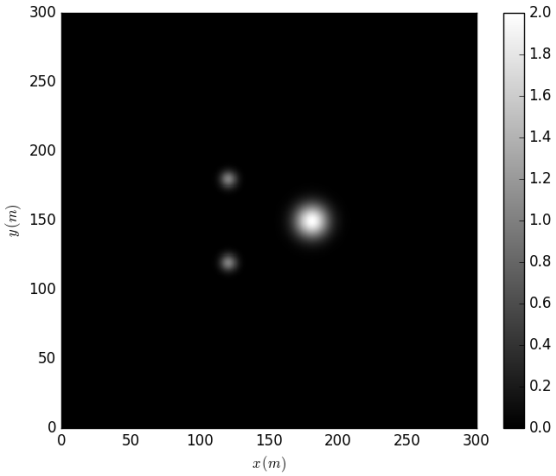

Figure 1: Grayscale plot of $Re[\psi(\vec{x},\, t=0)]$

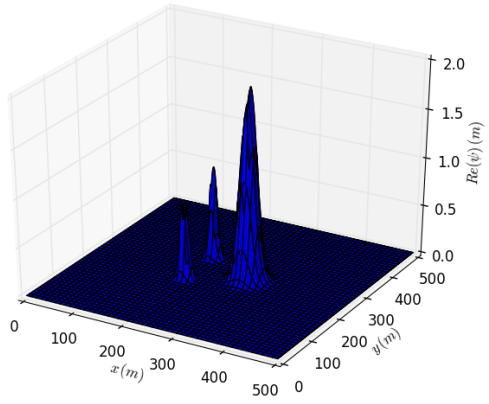

Figure 2: 3D plot of $Re[\psi(\vec{x},\, t=0)]$

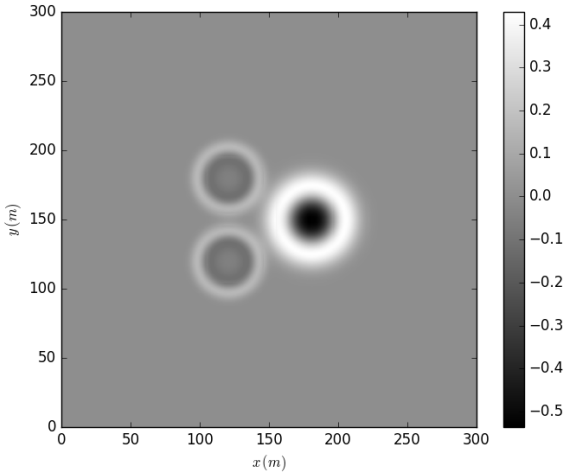

Figure 3: Grayscale plot of $Re[\psi(\vec{x},\, t=0.1\,s)]$

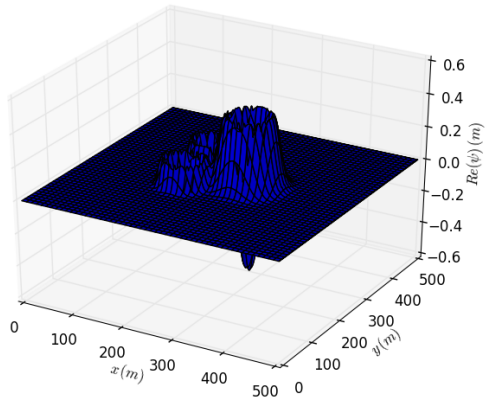

Figure 4: 3D plot of $Re[\psi(\vec{x},\, t=0.1\,s)]$

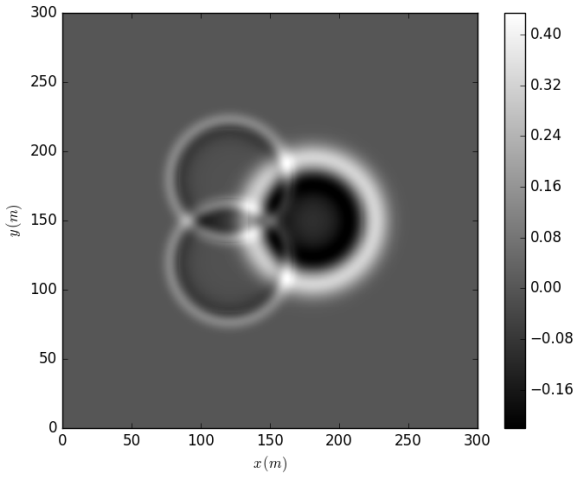

Figure 5: Grayscale plot of $Re[\psi(\vec{x},\, t = 0.2\, s)]$

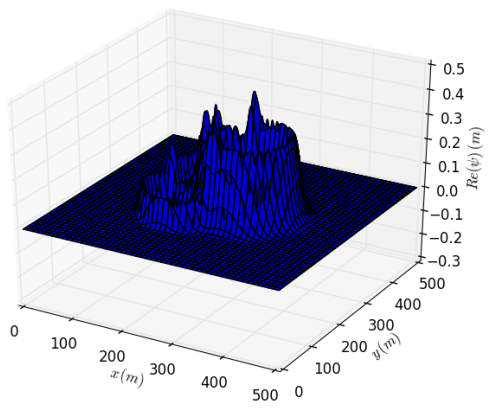

Figure 6: 3D plot of $Re[\psi(\vec{x},\, t = 0.2\, s)]$

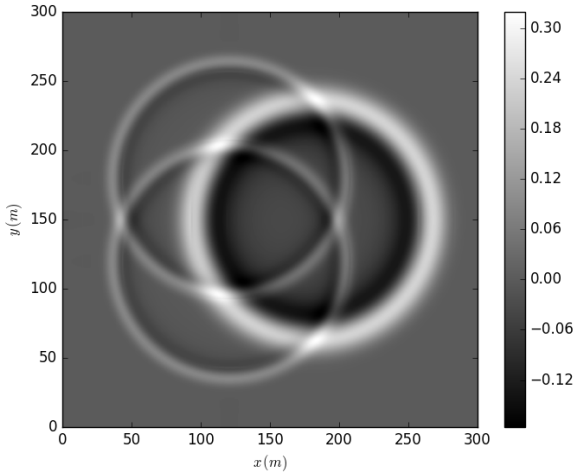

Figure 7: Grayscale plot of $Re[\psi(\vec{x},\, t = 0.4\, s)]$

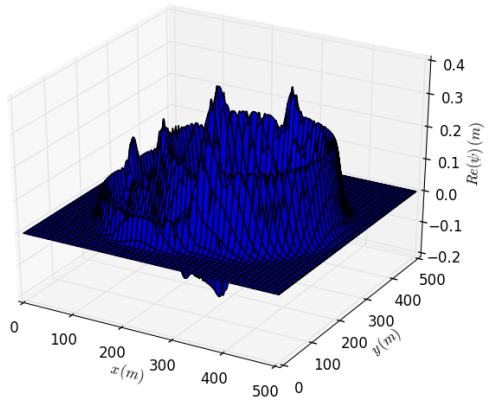

Figure 8: 3D plot of $Re[\psi(\vec{x},\, t = 0.4\, s)]$

3 Analysis

The results are in line with the expectations. The surface at $t = 0$ is flat except for three Gaussians near the center, with characteristics consistent to the ones we would expect to see given the values of the parameters that were used in equation (7), as can be clearly observed in Figures 1-2. Figures 3-8 show how the system evolves in time from this starting point, as the initial configuration gives rise to waves that expand outwards from the initial Gaussians and then proceed to interfere with each other. The peaks of the Gaussians proper appear to oscillate periodically between positive and negative values of ψ, much like a string would be expected to oscillate in an equivalent 1D system. The changes in color of the grayscale plots' backgrounds are simply the result of the surface

being observed from different altitudes, as necessary in order to account for the differing values of the Gaussian peaks at the different sampling times.

4 Source code

```
/* AEP 4380 HW#10
  Fast Fourier Transform and Spectral Methods
  Run on core i7 with g++ 5.4.0 (Ubuntu)
  Gianfranco Grillo 11/26/2016
*/

#include <cstdlib>
#include <cmath>

#include <iostream>
#include <fstream>
#include <iomanip>
#include <fftw3.h>
#include "arrayt.hpp"
#include <string>

using namespace std;

int main() {
    fftw_complex *psi, *coeff;
    fftw_plan planTf, planTi;
    int Nx = 512, Ny = 512, Lx = 500, Ly = 500, i, j;
    arrayt<double> psimat(Nx,Ny), kmat(Nx,Ny);
    arrayt<int> auxindex(Nx);
    double x, y, arg1, arg2, arg3, x1n, y1n, x2n, y2n, x3n, v = 343.0, kx, ky, pi =
        4.0*atan(1.0);
    void invtrans(fftw_complex*, fftw_complex*, arrayt<double>&, arrayt<double>&,
       double, fftw_plan, double&, char*);
    psi = (fftw_complex*) fftw_malloc(Nx*Ny*sizeof(fftw_complex));
    coeff = (fftw_complex*) fftw_malloc(Nx*Ny*sizeof(fftw_complex));
    ofstream fp;

    planTf = fftw_plan_dft_2d(Nx, Ny, psi, psi, FFTW_FORWARD, FFTW_ESTIMATE); //
       forward
    planTi = fftw_plan_dft_2d(Nx, Ny, psi, psi, FFTW_BACKWARD, FFTW_ESTIMATE); //
       inverse

    for (i = 0; i < Nx; i++) for (j = 0; j < Ny; j++) { // initial psi
        x = i*Lx/Nx;
        y = j*Ly/Ny;
        x1n = x-0.4*Lx;
        y1n = y-0.4*Ly;
        arg1 = -(x1n*x1n+y1n*y1n)/100.0;
        x2n = x-0.5*Lx;
        y2n = y-0.6*Ly;
        arg2 = -(x2n*x2n+y2n*y2n)/400.0;
```

```
        x3n = x-0.6*Lx;
        arg3 = -(x3n*x3n+y1n*y1n)/100.0;
        psi[i*Nx + j][0] = exp(arg1)+2*exp(arg2)+exp(arg3); // initial real part
        psimat(i,j) = psi[i*Nx + j][0]; // save to regular matrix for easier output
        psi[i*Nx + j][1] = 0; // initial imaginary part
    }

    for (i = 0; i <= Nx/2-1; i++) { auxindex(i) = i; } // to unscramble data
    j = 0;
    for (i = Nx/2; i <= Nx; i++) {
        auxindex(i) = -Nx/2 + j;
        j++;
    }

    fp.open("t=0.dat");
    for (i = 0; i < Nx; i++) {
        for (j = 0; j < Ny; j++) { fp << psimat(i, j) << "␣␣␣␣"; }
        fp << endl;
    }
    fp.close();

    fftw_execute_dft(planTf, psi, psi); // execute initial forward transform

    for (i = 0; i < Nx; i++) for (j = 0; j < Ny; j++) {
        coeff[i*Nx + j][0] = psi[i*Nx + j][0]; // store coefficients
        coeff[i*Nx + j][1] = psi[i*Nx + j][1];
        kx = 2*pi*auxindex(i)/Lx;
        ky = 2*pi*auxindex(j)/Ly;
        kmat(i,j) = sqrt(kx*kx+ky*ky); // generate |k| matrix
    }

    invtrans(coeff, psi, kmat, psimat, 0.1, planTi, v, "t=0.1.dat"); // inverse
       transform for corresponding t
    invtrans(coeff, psi, kmat, psimat, 0.2, planTi, v, "t=0.2.dat");
    invtrans(coeff, psi, kmat, psimat, 0.4, planTi, v, "t=0.4.dat");
    return(EXIT_SUCCESS);
}

void invtrans(fftw_complex* coeff, fftw_complex* psi, arrayt<double>& kmat, arrayt<
   double>& psimat, double t, fftw_plan planTi, double& v, char* filename) {
    int i, j, Nx, Ny;
    ofstream sp;
    Nx = kmat.n1();
    Ny = kmat.n2();
    double Nsqr = Nx*Ny;

    for (i = 0; i < Nx; i++) for (j = 0; j < Ny; j++) {
        psi[i*Nx + j][0] = coeff[i*Nx + j][0]*cos(v*t*kmat(i,j)); // multiply
           coefficients by cosine
        psi[i*Nx + j][1] = coeff[i*Nx + j][1]*cos(v*t*kmat(i,j));
    }
```

```
    fftw_execute_dft(planTi, psi, psi); // execute inverse transform
    sp.open(filename); // output to file
    for (i = 0; i < Nx; i++) {
        for (j = 0; j < Ny; j++) {
            psimat(i,j) = psi[i*Nx + j][0]/Nsqr; // normalize first
            sp << psimat(i, j) << "␣␣␣␣";
        }
        sp << endl;
    }
    sp.close();
    return;
}
```

References

[1] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery *Numerical Recipes, The Art of Scientific Computing, 3rd edit.*, Camb. Univ. Press 2007.