AEP4380 HW10
The FFT and Spectral Methods

Dan Girshovich

April 25, 2013

1 Overview

This document details the results of the included C++ program which solves a two-dimensional wave equation using a discrete fast Fourier transform (FFT). With this method, the propagation of the wave is reduced to a basic manipulation in the frequency domain. The implementation uses the `fftw` package, found at `fftw.org`.

2 System to Solve

2.1 Wave equation

The two dimensional wave equation is

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

where v is the velocity of the wave.

2.2 Discretization

The initial wave is contained in an square grid with length L and spacing Δ, so the number of points in each dimension is $N=\frac{L}{\Delta}$. The x and y coordinates for the wave equation are then defined as:

$$x=i\Delta,\ i=0,1,...N-1$$
$$y=j\Delta,\ j=0,1,...N-1$$

2.3 Initial Conditions

The wave is initially at rest, with values given by:

$$\psi(\vec{x},t=0)=e^{-\frac{|\vec{x}-\vec{x_A}|^2}{s_A^2}}+S_B(\vec{x}),$$

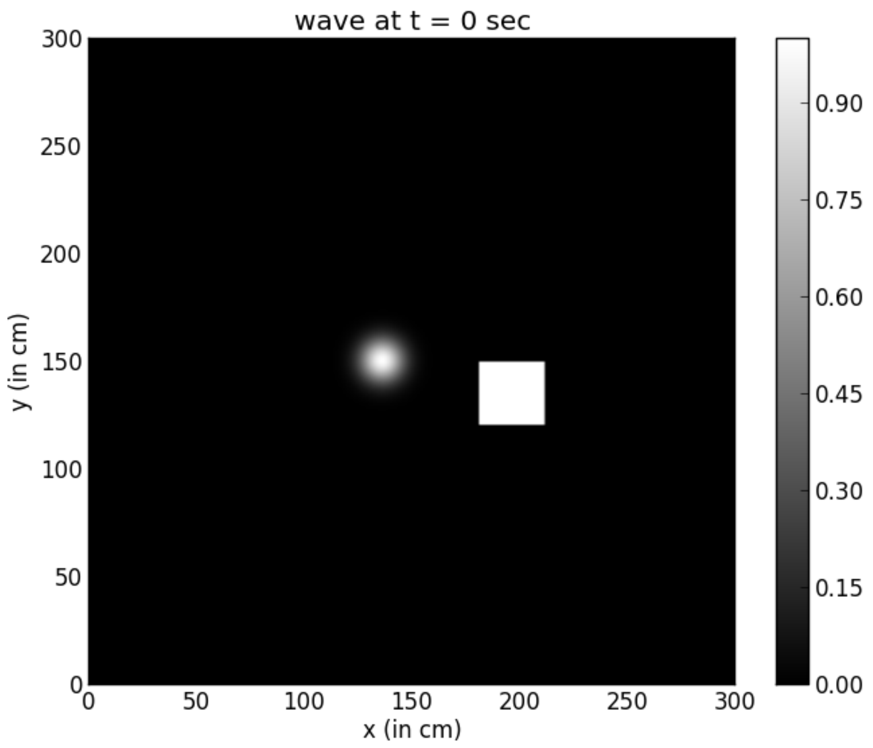

Figure 1: $\psi(\vec{x}, t = 0)$

which is implemented in `psi_init` and shown in Figure 1. The first term is a Gaussian and has height 1, width $s_A = 10$, and center $\vec{x_A} = (0.45L, 0.5L)$. The second term is a square pulse, which is zero everywhere except for $0.6L < x < 0.7L$ and $0.4L < y < 0.5L$, where it is 1. x, y, and ψ all have units of cm. This initial function and the wave equation determine $\psi(\vec{x}, t)$ for all $t > 0$.

3 Spectral Methods

3.1 Overview

Functions like $\psi(\vec{x}, t)$ above can be transformed into equivalent representations in the frequency domain. These representations sometimes allow straightforward solutions to differential equations. For the wave equation, the progression of the solution in time becomes a simple manipulation of the frequency components in the frequency domain. This allow the frequency representation of the wave function to be time evolved without any numerical propagation through time (as required when working in the time domain). After the frequency representation of the wave function is found for a given time, the inverse transform produces the desired time-domain wave function at that time.

3.2 Finite Fourier Series

The solution for $\psi(\vec{x}, t > 0)$ can be found by working with the wave function in the frequency domain. First, express the function as a Fourier series:

$$\psi(\vec{x}, t) = \sum_{ij} a_{ij}(t) e^{i\vec{k_{ij}}\cdot\vec{x}}.$$

This can be substituted into the wave equation to find:

$$a_{ij} = b_{ij} e^{iv|\vec{k_{ij}}|t} + c_{ij} e^{-iv|\vec{k_{ij}}|t}.$$

Now, the fact that the wave is initially at rest implies $b_{ij} = c_{ij}$, so:

$$a_{ij} = d_{ij} \cos(v|\vec{k_{ij}}|t).$$

3.3 Fast Fourier Transform

The coefficients d_{ij} are produced by taking the FFT of $\psi(\vec{x}, t = 0)$, since $a_{ij}(0) = d_{ij}$. Specifically, the FFT outputs amplitudes for N frequency components between $-f_c$ and f_c, where $f_c = \frac{1}{2\Delta}$ is the Nyquist critical frequency. A convention makes use of the N periodicity of the discrete transform by representing the values in the $-\frac{1}{2\Delta} \leq f < 0$ range by the (same) values in the $\frac{1}{2\Delta} < f \leq \frac{1}{\Delta}$ range. This shift can be seen in implementation of `mag_k`, which allows the amplitudes d_{ij} to be paired with the corresponding frequencies.

With the frequency amplitudes d_{ij}, the wave can be time evolved in the frequency domain to time t by shifting the amplitudes from $a_{ij}(0) = d_{ij}$ to $a_{ij}(t) = d_{ij} \cos(v|\vec{k_{ij}}|t)$. This is very straightforward, and is implemented in `calc_dcos`. Besides matching the amplitudes to frequencies, as discussed above, the only other complication is a conversion from frequency (used by the FFT library) to angular frequency (used in the equations above).

After the frequency representation for a given time is calculated, the values of $\psi(\vec{x}, t)$ are finally produced by the inverse FFT. This procedure is identical to the FFT, except it uses a sign change in the exponentials of the transform and adds an overall scaling factor (N^2 for our case). The inverse FFT is used to find $\psi(\vec{x}, t)$ for any $t > 0$ in `propagate_to`.

4 Results

The spectral method described above was used to calculate $\psi(\vec{x}, t)$ at $t = 0.1, 0.2,$ and 0.3 seconds. These functions are shown in Figures 2, 3, and 4, respectively. The results show the wave behaving as if it were a disturbance on a membrane or fluid, as expected. Specifically, the raised regions quickly fall and create ripples which propagate radially outwards. Note, the flat area surrounding the waves has changed from black to grey because the color scale now accounts for negative values of ψ.

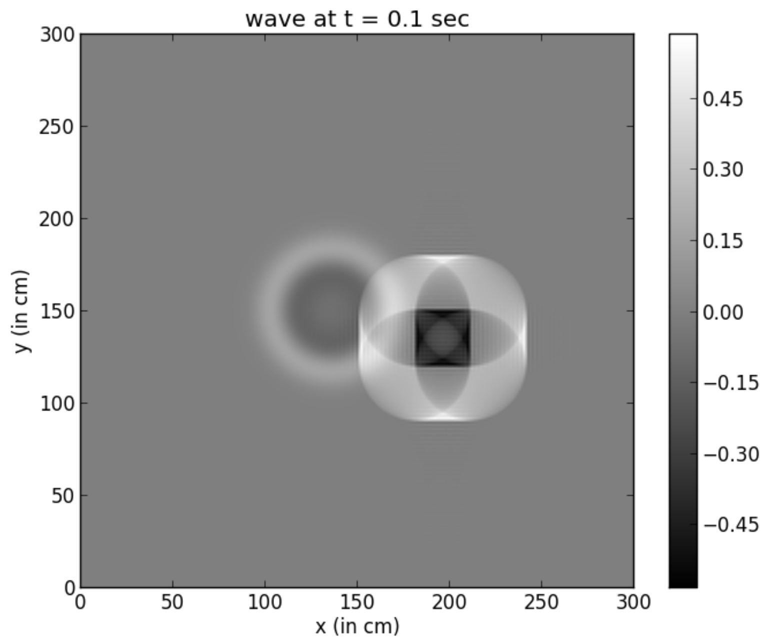

Figure 2: $\psi(\vec{x}, t = 0.1)$

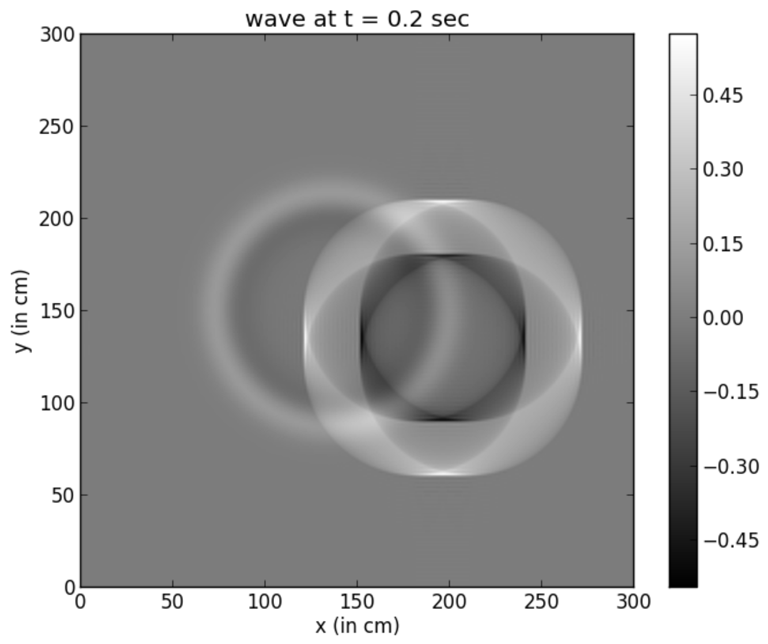

Figure 3: $\psi(\vec{x}, t = 0.2)$

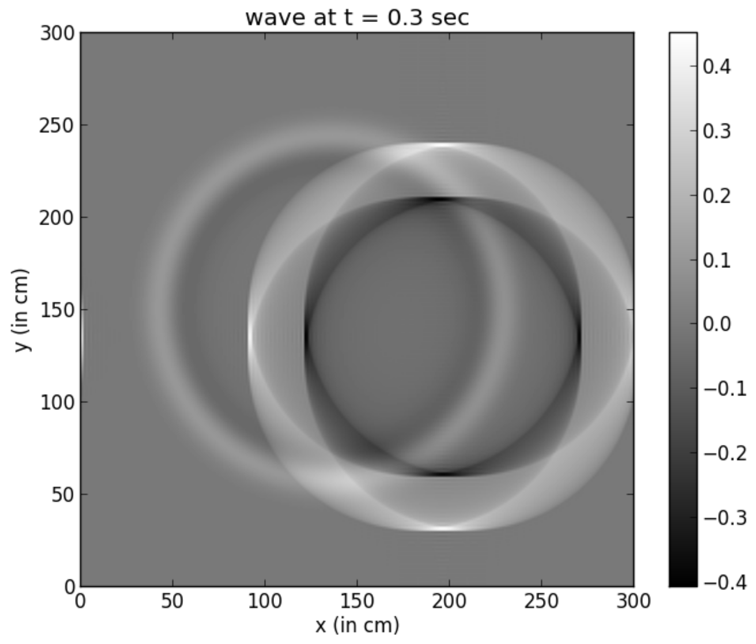

Figure 4: $\psi(\vec{x}, t = 0.3)$

5 Source Code

```
/*
AEP 4380 HW #10
Dan Girshovich
4/25/13
The FFT and Spectral Methods
Compile with: g++ -std=c++0x -O2 -o gen_data hw10.cpp
Tested on Mac OSX 10.8.2 with a Intel Core 2 Duo
*/

#include <cstdio>
#include <cstdlib>
#include <cmath>
#include <fstream>
#include <iostream>
#include <sstream>
#include <vector>
#include "_array.hpp"
#include <fftw3.h>

using namespace std;

const int N = 256;
const double L = 300; // cm
const double delta = L / N; // cm
const double v = 300; // cm/s
const double x_A = 0.45 * L; // cm
const double y_A = 0.5 * L; // cm
const double s_A = 10; // cm
const double pi = 4 * atan(1.0);

fftw_complex *psi;
fftw_complex *d;
fftw_complex *dcos;
fftw_plan f_plan;
fftw_plan b_plan;

// returns the value of psi(x, y, t=0) given x, y
double psi_init(double x, double y) {
    double norm = (x - x_A) * (x - x_A) + (y - y_A) * (y - y_A);
    double gaussian = exp((-1) * norm / (s_A * s_A));
    // add square pulse
    if (x > (0.6 * L) && x < (0.7 * L) && y > (0.4 * L) && y < (0.5 * L)) {
        return gaussian + 1;
    } else {
        return gaussian;
    }
}

// sets psi to the inital value of gaussian + square pulse
void init_psi() {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            psi[i * N + j][0] = psi_init(i * delta, j * delta);
        }
    }
}

// initalizes fftw data for computing ffts
void init_fft() {
    psi = (fftw_complex *) fftw_malloc(sizeof(fftw_complex) * N * N);
    d = (fftw_complex *) fftw_malloc(sizeof(fftw_complex) * N * N);
    dcos = (fftw_complex *) fftw_malloc(sizeof(fftw_complex) * N * N);
    f_plan = fftw_plan_dft_2d(N, N, psi, d, FFTW_FORWARD, FFTW_ESTIMATE);
    b_plan = fftw_plan_dft_2d(N, N, dcos, psi, FFTW_BACKWARD, FFTW_ESTIMATE);
}

// returns |k_ij|
double mag_k(int i, int j) {
    if(i > (N / 2)) i = i - N;
    if(j > (N / 2)) j = j - N;
    double k_x = i / (N * delta), k_y = j / (N * delta);
    return 2 * pi * sqrt(k_x * k_x + k_y * k_y);
}

// returns d_ij * cos(v * t * |k_ij|)
void calc_dcos(double t) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            dcos[i * N + j][0] = d[i * N + j][0] * cos(v * t * mag_k(i, j));
            dcos[i * N + j][1] = d[i * N + j][1] * cos(v * t * mag_k(i, j));
        }
    }
}

// propagates the values in psi up to time t
void propagate_to(double t) {
    calc_dcos(t);
    fftw_execute(b_plan);
    // renormalize
    for(int i = 0; i < N * N; i++) psi[i][0] = psi[i][0] / (N * N);
}

// outputs the data file for psi at t
void write_psi(double t) {
    stringstream fname;
    fname << "dat/psi_" << t << ".dat";
    ofstream of;
    of.open(fname.str().c_str());
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            of << psi[j * N + i][0] << " ";
        }
        of << endl;
    }
    of.close();
```

```
}

int main() {
    init_fft();
    init_psi();
    // calculates and sets the d_ij coefficients for psi(x, t=0)
    fftw_execute(f_plan);
    double times[] = {0, 0.1, 0.2, 0.3};
    for (double t : times) {
        propagate_to(t);
        write_psi(t);
    }
}
```