

The Fast Fourier Transform (FFT) and Spectral Methods

HW 10: Thursday, Apr. 18, 2013

DUE: Thursday, Apr. 25, 2013

READ: Numerical Recipes in C++, Section 12.0-12.4 pages 600-627, on FFT's
FFTW documentation (pdf, etc. from www.fftw.org)

OPTIONAL: Gerald and Wheatley, 7th edit.: section 5.4, pages 285-297

1. **FINAL PROJECT OUTLINE DUE ON TUES. APR. 23, 2013:** Please write a one page outline of your final project. This should include a title and a few paragraphs and equations outlining your project goals and how you expect to accomplish them. You should also include at least one reference. Your outline must be approved before turning in your final project report. You may start to work on the project before your outline is approved but you may need to change your project if the outline is not accepted. Also, if this is a joint project (with another course) you should state this in your outline and on your final report.

2. **FINAL PROJECT DUE ON TUE. MAY 14, 2013** (during exams): Please read the previous handout. This will be graded out of 20 points total. Because of the tight deadlines for senior grades etc. there will be a 2 point/day penalty for late reports. If you have another exam on this day I may consider short extensions (please see me in advance). Please plan to work on your project well in advance. If you have registered S/U for this course, you do not have to turn in a final project or outline if you have turned in all homework and have an average of 8/10 or better.

3. For your final project, I prefer that you send me a **pdf file** of size less than about 5MB with filename=yourname.pdf using the cornell drop box (dropbox.cornell.edu). Please include the source code in a separate file that I can compile if necessary. Have a nice summer vacation.

4. For your information the homework mean and standard deviation are 9.04 and 0.72 (excluding late HW) for homework number 1 through 8 (which is very good).

PROBLEM (10 points):

A. The FFTW Subroutine Package

The FFT (Fast Fourier Transform) algorithm is one of the fastest numerical methods available. Many problems are rephrased to use Fourier transforms to take advantage of this method. The popular FFTW package (www.fftw.org, the "Fastest Fourier Transform in the West"[1]) is one of the fastest and most sophisticated subroutine packages available. It is also free (under the GNU public license, GPL) and fairly well documented. It is no more difficult to use than the FFT in Numerical Recipes, and it is worth the effort to learn how to use it. For this homework, download and use the FFTW code and documentation. There are pre-compiled subroutine libraries (in binary) for most computer platforms so you don't need to compile it (its long). The installation and/or linking procedure is a little different for every compiler and operating system, so please refer to the FFTW documentation for your specific situation. Most popular platforms are supported. (Most Linux distro's have an FFTW option which should be used, on Windows gcc will link directly with the .dll file and MSVC will link with the .lib file, with the .dll in your path. Mac's require a little more effort, see the FFTW web sites.)

The speed of execution of many FFT's (particularly multidimensional FFTs) is as much dependent on the memory bandwidth as the number of floating point operations. FFTW uses a specialized memory allocation scheme to align the memory properly. It has its own plain C like memory allocation routine and data types which should be used. FFTW has a unique approach to obtaining maximum speed for repeated execution of the same size FFT (common in many large simulation). FFTW first tries (at run time) many different forms of FFT's to find the style that is the fastest on the computer its running on and for the size being used. This step may take significant time, but if the same size FFT is performed many times, FFTW can

yield and overall improvement of a factor of 2X or 3X in many calculations. This HW problem is too simple to see this improvement, but larger calculations can see a significant improvement. You can also use the FFTW_ESTIMATE mode in place of the FFTW_MEASURE mode to avoid this planning step and just guess at a good FFT style (faster overall if only one FFT is performed).

An example of some of the typical steps in using FFTW are:

```
#include "fftw3.h" // FFT routines from FFTW 3
:
int main()
{
    :
    fftw_complex *y, *y2; // complex waves
    fftw_plan planTf, planTi, planTf2, planTi2; // FFTW plans
    :
    /* ----- 1D FFT's -----*/
    y = (fftw_complex*) fftw_malloc( n * sizeof(fftw_complex) );
    //<check for NULL==y etc.>
    :
    planTf = fftw_plan_dft_1d( n, y, y, FFTW_FORWARD, FFTW_ESTIMATE ); //forward in place
    //planTf = fftw_plan_dft_1d( n, y, y, FFTW_FORWARD, FFTW_MEASURE ); //forward in place
    :
    // initialize y[i][0] = real part and y[i][1] = imag. part etc.
    :
    fftw_execute_dft( planTf, y, y ); // forward transform
    :
    /* ----- 2D FFT's -----*/
    y2 = (fftw_complex*) fftw_malloc( nx*ny * sizeof(fftw_complex) );
    //<check for NULL==y2 etc.>
    :
    // forward in place
    planTf2 = fftw_plan_dft_2d( nx, ny, y2, y2, FFTW_FORWARD, FFTW_ESTIMATE ); //slower execution
    //planTf2 = fftw_plan_dft_2d( nx, ny, y2, y2, FFTW_FORWARD, FFTW_MEASURE );
    // inverse in place
    planTi2 = fftw_plan_dft_2d( nx, ny, y2, y2, FFTW_BACKWARD, FFTW_ESTIMATE ); //slower execution
    :
    // initialize y2[iy + ix*ny][0] = real part and y2[iy + ix*ny][1] = imag. part etc.
    :
    fftw_execute_dft( planTf2, y2, y2 ); // forward transform
    :
}
```

The last `fftw_execute` statement is usually executed many times but the other statement are usually only executed once. FFTW does NOT normalize, so you have to divide by N (or $Nx * Ny$ for 2D) for a round trip (forward plus inverse).

B. Spectral Methods

Some partial differential equations with simple boundary condition can sometimes be solved using what are called *spectral methods* (see for example section 6.4 of [2]). This means that a solution is developed in Fourier Transform space and then transformed back into real space. The Fast Fourier Transform (or FFT) is an efficient numerical method to calculate Fourier Transforms and will be used in this homework assignment.

Consider the 2D wave equation for a displacement $\psi(x, y)$ as a function of position $\vec{x} = (x, y)$ and time t :

$$v^2 \left[\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right] \psi = \frac{\partial^2 \psi}{\partial t^2} \quad (1)$$

where v is the velocity of the wave propagation. This equation might model the behavior of a water wave, elastic membrane or electromagnetic wave. Express the solution as a Fourier series in the form:

$$\psi(\vec{x}, t) = \sum_{ij} a_{ij}(t) \exp(i\vec{k}_{ij} \cdot \vec{x}) \quad (2)$$

where $\vec{k}_{ij} = (k_{xi}, k_{yj})$ is a 2D spatial frequency vector ($\sum_{ij} = \sum_i \sum_j$). This uses the traditional physics placement of the 2π factor but remember that the FFT uses an explicit 2π as $\exp(2\pi i\vec{k}_{ij} \cdot \vec{x})$, and the frequencies comes out in a rearranged order. Insert this expression into the original equation (1):

$$-v^2 \sum_{ij} k_{ij}^2 a_{ij}(t) \exp(i\vec{k}_{ij} \cdot \vec{x}) = \sum_{ij} a''_{ij}(t) \exp(i\vec{k}_{ij} \cdot \vec{x}) \quad (3)$$

Equating coefficients of $\exp(i\vec{k}_n \cdot \vec{x})$ yields the solution:

$$-v^2 k_{ij}^2 a_{ij}(t) = a''_{ij}(t) \quad (4)$$

$$a_{ij}(t) = b_{ij} \exp(\pm iv|\vec{k}_{ij}|t) \quad (5)$$

$$\psi(\vec{x}, t) = \sum_{ij} \left[b_{ij} \exp(+iv|\vec{k}_{ij}|t) + c_{ij} \exp(-iv|\vec{k}_{ij}|t) \right] \exp(i\vec{k}_{ij} \cdot \vec{x}) \quad (6)$$

where b_{ij} and c_{ij} are complex constants independent of time. If the displacement is released from a specified position initially at rest:

$$\frac{\partial \psi(\vec{x}, t=0)}{\partial t} = 0 = \sum_{ij} [b_{ij} - c_{ij}] (iv|\vec{k}_{ij}|) \exp(i\vec{k}_{ij} \cdot \vec{x}) \quad (7)$$

This restricts the solution to $b_{ij} = c_{ij}$ and:

$$\psi(\vec{x}, t) = \sum_{ij} d_{ij} \cos(v|\vec{k}_{ij}|t) \exp(i\vec{k}_{ij} \cdot \vec{x}) \quad (8)$$

where $d_{ij} = 2b_{ij}$. To solve this equation using a spectral method, first calculate the d_{ij} coefficients from a given initial shape $\psi(\vec{x}, t=0)$ using a discrete fast Fourier transform (FFT). Then multiple these d_{ij} values by the above factor and calculate $\psi(\vec{x}, t > 0)$ at some later time using an inverse FFT.

In this homework problem assume $L_x = L_y = 300$ cm., $v = 300$ cm./sec $N_x = N_y = 256$, and set the initial displacement to be one small Gaussian and one small square pulse near the center (to watch propagation and interference):

$$\psi(\vec{x}, t=0) = A_a \exp \left[-\frac{|\vec{x} - \vec{x}_A|^2}{s_A^2} \right] + A_b S_B(\vec{x}) \quad (9)$$

where $\vec{x}_A = (0.45N_x, N_y/2)$ and $S_B(\vec{x})$ is zero except for the region between $0.60N_x < x < 0.70N_x$ and $0.40N_y < y < 0.50N_y$ (in pixels). The amplitudes should be $A_a = 1$ and $A_b = 1$.

Display the real part of $\psi(\vec{x}, t)$ for times 0.0, 0.1, 0.2 and 0.3 sec. You should use the FFTW subroutines. You may display the results as a contour plot (as in HW 6) or as a gray scale image which is easier to interpret. In Matlab:

```
pixa = load( 'hw10a.dat' ); % 2D data in row/col format
x = load( 'hw10x.dat' ); % x,y coord. for axis
y = load( 'hw10y.dat' );
colormap(gray);
imagesc( x, y, pixa );
axis image;
colorbar;
```

Or in scilab:

```
pixa = read( 'hw10a.dat', -1, 256 ); // 2D data in row/col format
x = read( 'hw10x.dat', -1, 1 ); // x,y coord. for axis
y = read( 'hw10y.dat', -1, 1 );
xset("colormap",jetcolormap(64));
grayplot( x, y, pixa );
xs2eps(0,'fig10sci.eps'); // output graph as postscript file
```

Or in python:

```
# uses numpy, scipy, matplotlib packages
from pylab import *
#
pixa = loadtxt( 'hw10a.dat', 'float' )
#
img = imshow( pixa, aspect='equal', extent=(0,300,0,300) )
img.set_cmap( 'gray' ) # for greyscale
colorbar()
xlabel( 'x (in cm.)' )
ylabel( 'y (in cm.)' )
title( 'wave at t=0.1 sec' )
savefig('fig10py.eps') # change suffix for other file formats
show()
```

OPTIONAL-1: Try using the fast sin transform (in FFTW) which will impose a constraint on the end points and use only about half of the floating point operations.

OPTIONAL-2: Try this problem using matlab, scilab or python and compare results.

References

- [1] Matteo Frigo and Steven G. Johnson, "Design and Implementation of FFTW3", Proc. of the IEEE, 93 (2005), p. 216-231.
- [2] N. J. Giodano and H. Nakanishi, *Computational Physics, 2nd edit.*, (Prentice-Hall 1997,2006).
- [3] 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.
- [4] James S. Walker, *Fast Fourier Transforms, 2nd edit.*, CRC Press, 1996.
- [5] E. O. Brigham, *The Fast Fourier Transform and Its Applications*, Prentice Hall, 1988.