Solving Poisson's Equation using the FFT

The Fast Fourier Transform can be used to solve elliptic partial differential equations in multiple dimensions.

The basic idea is very simple. Consider the one-dimensional equation

$$\frac{\mathrm{d}^2\phi}{\mathrm{d}x^2} = \rho(x) \ .$$

Express f and ρ in terms of their Fourier transforms:

$$f(x) = \frac{1}{\sqrt{2\pi}} \int g(k) e^{ikx} \mathrm{d}k \ , \qquad \rho(x) = \frac{1}{\sqrt{2\pi}} \int \sigma(k) e^{ikx} \mathrm{d}k \ .$$

The equation is diagonalized in k-space

$$-k^2 g(k) = \sigma(k) \qquad \Rightarrow \qquad g(k) = -\frac{\sigma(k)}{k^2} \ .$$

The solution is given by the inverse transform

$$f(x) = -\frac{1}{\sqrt{2\pi}} \int \frac{\sigma(k)}{k^2} e^{ikx} \mathrm{d}k \ .$$

We need to impose appropriate boundary conditions, and also specify how to treat the singularity at $k = 0$ in the integral.

Boundary conditions and types of transforms

The boundary conditions determine the appropriate type of Fourier transform to solve the problem. Some problems, like conduction of heat through a rod which Fourier studied, require a sine transform. Others require a cosine or exponential transform.

Consider a domain $0 \leq x \leq L$ in one dimension. Choose a lattice of N equally spaced points $x_n = nL/N, n = 0, \ldots, N-1$.

The complex Fourier transform coefficients of $f(x)$ are

$$g_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} W^{kn} f_n \ , \qquad W = e^{2i\pi/N} \ .$$

The inverse transform

$$f_n = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} W^{-nk} g_k \ ,$$

will be *periodic* in x_n with period L. This complex Fourier transform is therefore appropriate for problems which satisfy *periodic boundary conditions*.

If the problem involves *Dirichlet boundary conditions*, for example, $f(0) = f(L) = 0$, then it would be appropriate to use a *sine Fourier transform*

$$f_n = \sqrt{\frac{2}{N}} \sum_{k=1}^{N-1} \sin\left(\frac{\pi n k}{N}\right) g_k \ .$$

If the problem involves *Neumann boundary conditions* it is appropriate to use is the *cosine Fourier transform*. One discrete version on $N+1$ points given in Numerical Recipes is

$$f_n = \frac{1}{\sqrt{2N}} [g_0 + (-1)^n g_N] + \sqrt{\frac{2}{N}} \sum_{k=1}^{N-1} \cos\left(\frac{\pi n k}{N}\right) g_k \ .$$

Note that the cosine and sine transforms are *not* just the real and imaginary parts of the complex exponential transform. This is because the sine, cosine, and exponential functions separately form complete sets, albeit with different boundary conditions. The basic phase factor for the sine and cosine transforms is π/N compared with $2\pi/N$ for the exponential:

the sine and cosine transforms require *twice as many lattice points* because they are real, compared with the exponential transform which is complex.

Fast Sine and Cosine transforms can be defined similarly to the FFT of Chapter 5. Details and programs are given in Section 12.3 of Numerical Recipes.

Poisson's equation in two dimensions

Let's consider the discrete form of Poisson's equation

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)\phi(x,y) \simeq \frac{1}{h^2}\left[\phi_{j+1,k} + \phi_{j-1,k} + \phi_{j,k+1} + \phi_{j,k-1} - 4\phi_{j,k}\right] = -\rho_{j,k}$$

on an $N \times N$ grid of points in the region $0 \le x, y \le 1$ with a given charge density, say a single point charge located at the center of the square.

For simplicity, we will impose periodic boundary conditions so that the exponential FFT can be used to obtain the solution.

The discrete Fourier transform is a *linear operation*, so we can apply it separately in the x and y directions, and it does not matter in which order the transforms are done.

The two-dimensional Fourier coefficients are given by

$$\tilde{\phi}_{m,n} = \frac{1}{N}\sum_{j=0}^{N-1}\sum_{k=0}^{N-1} W^{mj+nk}\phi_{j,k} \ ,$$

$$\tilde{\rho}_{m,n} = \frac{1}{N}\sum_{j=0}^{N-1}\sum_{k=0}^{N-1} W^{mj+nk}\rho_{j,k} \ .$$

The inverse transforms are

$$\phi_{j,k} = \frac{1}{N}\sum_{m=0}^{N-1}\sum_{n=0}^{N-1} W^{-jm-kn}\tilde{\phi}_{m,n} \ ,$$

$$\rho_{j,k} = \frac{1}{N}\sum_{m=0}^{N-1}\sum_{n=0}^{N-1} W^{-jm-kn}\tilde{\rho}_{m,n} \ .$$

Plugging these expressions into the discretized equation and equating coefficients of W^{-mj-nk} gives

$$\frac{1}{h^2}\left[W^m + W^{-m} + W^n + W^{-n} - 4\right]\tilde{\phi}_{m,n} = -\tilde{\rho}_{m,n} \ ,$$

which is easily solved for

$$\tilde{\phi}_{m,n} = \frac{h^2\tilde{\rho}_{m,n}}{4 - W^m - W^{-m} - W^n - W^{-n}} \ .$$

The inverse Fourier transform then gives the potential.

PoissonFFT.cpp

```
#include <cmath>
#include <complex>
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <fstream>

#include "Vector.hpp"

int main() {

    std::cout << " FFT solution of Poisson’s equation\n"
              << " ----------------------------------\n";
    std::cout << " Enter number points in x or y:  ";
    int N;
```

```
std::cin >> N;
int n = 1;
while (n < N)
    n *= 2;
if (n != N)
    std::cout << " must be a power of 2, using " << (N=n) << std::endl;

double h = 1 / double(N - 1);        // assume physical size in x and y = 1

std::clock_t t0 = std::clock();

double q = 10;                       // point charge
std::complex<double> rho[N][N];
for (int j = 0; j < N; j++) {
    for (int k = 0; k < N; k++) {
        if (j == N/2 && k == N/2)    // at center of lattice
            rho[j][k] = q / (h * h);
        else
            rho[j][k] = 0.0;
    }
}

cpl::ComplexVector f(N);             // to store rows and columns
cpl::FFT fft;                        // FFT object for 1-D transforms

// FFT rows of rho
for (int j = 0; j < N; j++) {
    for (int k = 0; k < N; k++)
        f[k] = rho[j][k];
    fft.transform(f);
    for (int k = 0; k < N; k++)
        rho[j][k] = f[k];
}
// FFT columns of rho
for (int k = 0; k < N; k++) {
    for (int j = 0; j < N; j++)
        f[j] = rho[j][k];
    fft.transform(f);
    for (int j = 0; j < N; j++)
        rho[j][k] = f[j];
}

// solve equation in Fourier space
std::complex<double> i(0.0, 1.0);
double pi = 4 * std::atan(1.0);
std::complex<double> W = std::exp(2.0 * pi * i / double(N));
std::complex<double> Wm = 1.0, Wn = 1.0;
for (int m = 0; m < N; m++) {
    for (int n = 0; n < N; n++) {
        std::complex<double> denom = 4.0;
        denom -= Wm + 1.0 / Wm + Wn + 1.0 / Wn;
        if (denom != 0.0)
            rho[m][n] *= h * h / denom;
        Wn *= W;
    }
    Wm *= W;
}
```

```
// inverse FFT rows of rho
for (int j = 0; j < N; j++) {
    for (int k = 0; k < N; k++)
        f[k] = rho[j][k];
    fft.inverseTransform(f);
    for (int k = 0; k < N; k++)
        rho[j][k] = f[k];
}
// inverse FFT columns of rho
for (int k = 0; k < N; k++) {
    for (int j = 0; j < N; j++)
        f[j] = rho[j][k];
    fft.inverseTransform(f);
    for (int j = 0; j < N; j++)
        rho[j][k] = f[j];
}

std::clock_t t1 = std::clock();
std::cout << " CPU time = " << double(t1 - t0) / CLOCKS_PER_SEC
     << " sec" << std::endl;

// write potential to file
std::cout << " Potential in file PoissonFFT.data" << std::endl;
std::ofstream dataFile("PoissonFFT.data");
for (int j = 0; j < N; j++) {
    double x = j * h;
    for (int k = 0; k < N; k++) {
        double y = k * h;
        dataFile << x << '\t' << y << '\t' << std::real(rho[j][k]) << '\n';
```

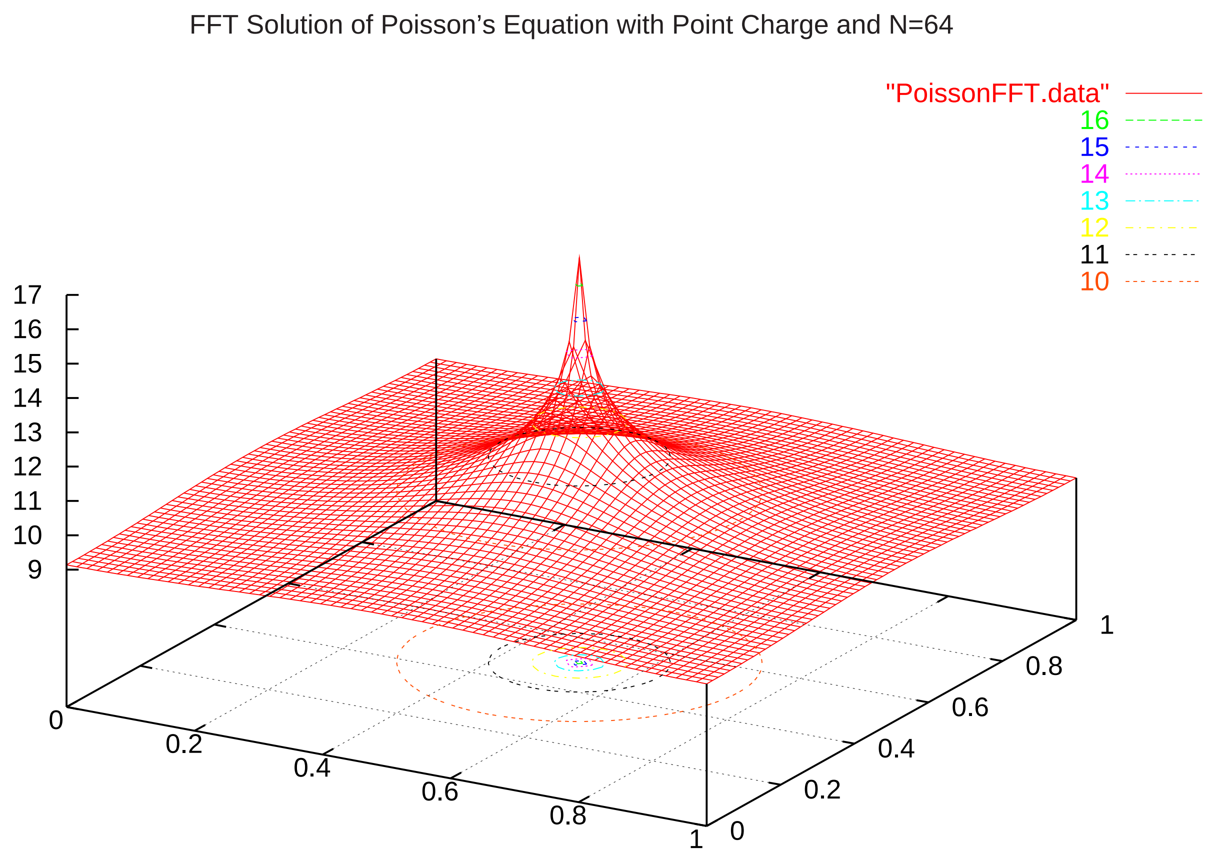
```
            }
            dataFile << '\n';
        }
        dataFile.close();
    }
```

Running the program

```
    > a.out
     FFT solution of Poisson's equation
     ----------------------------------
     Enter number points in x or y:  64
     CPU time = 0.06 sec
     Potential in file PoissonFFT.data
```

gives the potential in the figure on the next page, which was obtained using the following Gnuplot commands:

```
    gnuplot> set data style lines
    gnuplot> set surface
    gnuplot> set contour both
    gnuplot> set grid
    gnuplot> splot "PoissonFFT.data"
```


FFT Solution of Poisson’s Equation with Point Charge and N=64
"PoissonFFT.data"
16
15
14
13
12
11
10
17
16
15
14
13
12
11
10
9
0
0.2
0.4
0.6
0.8
1
0
0.2
0.4
0.6
0.8
1