

Long range forces in a two-dimensional space

It is easier to understand and program tree code and fast multipole algorithms in a two-dimensional space rather than in 3 dimensions. There are also some physical systems that are effectively two dimensional: examples include vortices in a type II superconductor, and the 2-dimensional electron gas in a semiconductor heterostructure.

The gravitational or Coulomb forces in a two-dimensional space fall off like $1/r$. In a system of particles with approximately uniform density, the number of particles at distance r from a particular particle is proportional to the “surface” of a circle of radius r , i.e., its circumference $2\pi r$. Thus, the contributions of *all* particles, are important no matter how far away they are, just like for the $1/r^2$ force in 3 dimensions.

The Coulomb potential and field due to a charge q at the origin in two dimensional space $\mathbf{r} = (x, y)$ are given by:

$$U(r) = -q \ln r, \quad \mathbf{E}(\mathbf{r}) = -\frac{\partial}{\partial \mathbf{r}} U(r) = \frac{q}{r^2} \mathbf{r}.$$

If there are N charges q_i located at positions $\mathbf{r}_i$, the potential at point $\mathbf{r}$ is given by

$$U(\mathbf{r}) = -\sum_{i=1}^N q_i \ln |\mathbf{r} - \mathbf{r}_i|.$$

Viewing 2-d space as the complex plane

In many problems, it turns out to be mathematically convenient to view 2-dimensional space as the complex plane with points $z = x + iy$. The physical potential is the negative of the real part of a *complex potential*

$$U(z) = \sum_{i=1}^N q_i \ln(z - z_i).$$

The magnitude of the field is given by the magnitude of a *complex field*

$$E(z) = \frac{d}{dz} U(z) = \sum_{i=1}^N \frac{q_i}{z - z_i}.$$

The Cartesian components of the field are given by its real and (negative) imaginary parts:

$$E_x = \Re E(z) , \quad E_y = -\Im E(z) .$$

If we are interested in the potential and field at a point far away from the charges, i.e., $|z| \gg |z_i|$, then a *multipole expansion* can be used:

$$U(z) = a_0 \ln z - \sum_{k=1}^{\infty} \frac{a_k}{z^k} , \quad E(z) = \sum_{k=1}^{\infty} \frac{k a_k}{z^{k+1}} ,$$

where the *multipole moments* are given by

$$a_0 = \sum_{i=1}^N q_i \quad (\text{the net charge})$$

$$a_1 = \sum_{i=1}^N q_i z_i \quad (\text{the complex dipole moment})$$

$$a_k = \sum_{i=1}^N \frac{q_i z_i^k}{k} , \quad k = 2, 3, \dots$$

Computing the potential energy of N charges

The complex potential energy of N charges is given by

$$E_{\text{pot}} = \frac{1}{2} \sum_{\substack{i,j=1 \\ i \neq j}}^N q_i q_j \ln(z_i - z_j) = \sum_{\text{pairs } ij} q_i q_j \ln(z_i - z_j) .$$

This is a typical N -body computation because the number of pairs $N(N-1)/2$ is of $\mathcal{O}(N^2)$.

A particle-particle code to compute E_{pot}

A *particle-particle* algorithm is one which examines every pair of particles.

```
#include <iostream>
#include <cstdlib>
#include <cmath>
```

Here we include two additional standard headers:

- **complex** defines a template class for arithmetic with complex numbers. The class can be instantiated with various numeric types, **double**, **float**, **long int**, etc., and the arithmetic is done using those types.
- **ctime** defines some quantities that allow us to measure the CPU time used by the program.

```
#include <complex>
#include <ctime>
```

```
using namespace std;
```

```
int N = 1000;           // number of charges
double L = 10;          // side of square region containing charges
```

The positions in the complex plane and the complex potential energy are defined as **complex** quantities whose real and imaginary parts are **doubles**.

```
complex<double> *z;      // positions of particles
complex<double> Epot;    // total potential energy of system
```

```
double *q;                      // charges of particles

void initialize() {

    // allocate arrays
    z = new complex<double> [N];
    q = new double [N];

    // place particles with random charges at random locations
    for (int i = 0; i < N; i++) {
        q[i] = 2 * rand() / double(RAND_MAX) - 1;
        double x = L * rand() / double(RAND_MAX);
        double y = L * rand() / double(RAND_MAX);
        z[i] = complex<double>(x, y);
    }
}

void computeEpot() {
    Epot = 0;
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            if (i != j)
                Epot += q[i] * q[j] * log(z[i] - z[j]);
    Epot /= 2;
}
```

The `main` function will be defined to use command line arguments to read optional input for the program:

```
int main(int argc, char *argv[]) {  
  
    if (argc > 1)  
        N = atoi(argv[1]);  
    if (argc > 2)  
        L = atof(argv[2]);  
    cout << "Number of charges = " << N << endl;  
    cout << "Side of square box = " << L << endl;  
  
    initialize();  
}
```

The `clock` function, which approximates the CPU time in units of “clocks” that the current program has been running, will be used to time the N -body particle-particle computation of the potential:

```
clock_t t0 = clock();        // CPU time before computing potential  
computeEpot();  
clock_t t1 = clock();        // CPU time after computing potential  
cout.precision(16);  
cout << "  Potential energy = " << Epot << endl;  
cout << "          CPU time = " << double(t1 - t0) / CLOCKS_PER_SEC  
    << " sec" << endl;  
  
}
```

The constant `CLOCKS_PER_SEC` is used to convert from “clocks” to seconds.