```
#include "pdesolver.h"

PDESolver::PDESolver(int nt, int nx, double Dt, double Dx)
{
    Nt = nt; dt = Dt; Nx = nx; dx = Dx;
}

void PDESolver::RandomWalk(vec *x)
{
    double l0, r; vec X; long idum; int N,n; ofstream myfile;
    X = *x; l0 = sqrt(2*dt); idum = -1; myfile.open("MC_uniform_movie.txt");

    for (int t=0;t<=Nt;t++){
        N = X.n_rows; n = 0;            // n will determine how many new particles to be
added in x = 0

        for (int i=0;i<N;i++){
            r = ran0(&idum);            // Getting a random number

            if (X(i) < 1e-13) {n += 1;} // This particle will move away from x0. Need to
add one more.

            if (r>0.5) {X(i) += l0;}    // Move to the right
            else       {X(i) -= l0;}    // Move to the left

            if (abs(X(i)) < 1e-13) {n -= 1;} // This particle has arrived at x0. Need to
add one less
        }
        int i = 0;
        while (i<N){
            if      (X(i) > 1){X.shed_row(i); N-=1;} // Remove particle and reduce total
particle #
            else if (X(i) < 0){X.shed_row(i); N-=1;} // Remove particle and reduce total
particle #
            else              {i++;}
        }
        for (int i=0;i<n;i++){
            X.insert_rows(0,1); X(0) = 0; // Maintaining # of particles at x=0
        }
        for (int elem=0; elem<X.n_rows; elem++){myfile << X(elem) << " ";}; myfile <<
endl;
    }
    *x = sort(X);
}

void PDESolver::GaussRandomWalk(vec *x){
    double l0, r, theta, bin, L; vec X; long idum; int N,n; ofstream myfile;
    X = *x; l0 = sqrt(2*dt); idum = -1; bin = 1./20; myfile.open("MC_normal_movie.txt");

    for (int t=0;t<=Nt;t++){
        N = X.n_rows; n = 0;            // n will determine how many new particles to be
added in x = 0

        for (int i=0;i<N;i++){
            r = sqrt(-log(1. - ran0(&idum)));           // Getting a random number
            theta = 2*pi*ran0(&idum);

            L = l0 * r*cos(theta); // From normal distribution

            if (X(i) < bin) {n += 1;} // This particle will move away from bin 0. Need
to add one more.

            X(i) += L; // Particle changes position following the normal distribution

            if (X(i) < bin && X(i) >= 0) {n -= 1;} // This particle has arrived at bin
0. Need to add one less
        }
        int i = 0;
        while (i<N){
```

```
            if      (X(i) > 1){X.shed_row(i); N-=1;} // Remove particle and reduce total
particle #
            else if (X(i) < 0){X.shed_row(i); N-=1;} // Remove particle and reduce total
particle #
            else              {i++;}
        }
        for (int i=0;i<n;i++){
            X.insert_rows(0,1); X(0) = 0; // Maintaining # of particles at x=0
        }
        for (int elem=0; elem<X.n_rows; elem++){myfile << X(elem) << " ";}; myfile <<
endl;
    }
    *x = sort(X);
}




void PDESolver::Analytical(vec *U){
    vec u; ofstream myfile;
    myfile.open("Analytical_movie.txt");

    for (int j=0; j <= Nt; j++){
        u = zeros<vec>(Nx);
        for (int i=0; i<Nx; i++){
            for (int n=1; n < 15; n++){
                u(i) += -2/(n*pi)*sin(n*pi*i*dx) * exp(-(n*n*pi*pi*j*dt));
            }
            u(i) += 1-i*dx;
            myfile << u(i) << " ";}; myfile << endl; }; myfile.close();
    *U = u;
}

void PDESolver::CrankNicolson(vec *v){
        double a,a2,a3; vec A1 = zeros<vec>(Nx), A2 = zeros<vec>(Nx), A3 =
zeros<vec>(Nx), vtilde, vnew;
        a = dt / dx / dx; a2 = 2 - 2*a; a3 = 2 + 2*a; ofstream myfile;
        myfile.open("CrankNicolson_movie.txt");

        // Setting the diagonals on of the LHS-matrix.
        for (int i=0; i<Nx; i++){
            A1(i) = -a; A2(i) = a3; A3(i) = -a;
        }

        vnew = *v; vtilde = vnew;
        for (int j=1; j<Nt; j++){
            // Chaning the RHS vector v_old into vtilde.
            for (int i=1; i<Nx-1; i++){
                vtilde(i) = a*vnew(i-1) + a2*vnew(i) + a*vnew(i+1);
            }
            vnew = PDESolver::tridiagonal(A1,A2,A3,vtilde);
            for (int i=0; i<Nx; i++){myfile << vnew(i) + 1 - i*dx << " ";}; myfile <<
endl;
        }
        myfile.close();
        for (int i=0; i<Nx; i++){vnew(i) += 1 - i*dx;}
        *v = vnew;
}

vec PDESolver::tridiagonal(vec a1, vec a2, vec a3, vec b){
        // Forward Substitution. Row reducing the matrix equation
        vec v = zeros<vec>(Nx);
        for (int i = 1; i < Nx; i++){
            float factor = a1(i)/a2(i-1);
            a2(i) = a2(i) - a3(i-1)*factor;
            b(i) = b(i) - b(i-1)*factor;
        }

        // Backward Substitution. Solving the equation for vector v.
```

```
        v(Nx-1) = b(Nx-1)/a2(Nx-1);
        for (int k = Nx-2; k >= 0; k--){
            v(k) = (b(k) - a3(k) * v(k+1))/a2(k);
        }
        return v;
}
```