PHYS 410 Project 1

Gavin Pringle, 56401938

October 14, 2024

Introduction

In this project, the problem of N identical point charges confined to free motion on the surface of a sphere is examined. Via a finite-difference approximation simulation, the dynamic behaviour of point charges each originating from random initial positions on the surface of the sphere is computed. Since like charges repel, if a velocity-dependent retarding force is present for each charge then it follows that all charges will eventually reach a stable equilibrium where the electrostatic potential energy is minimized.

Through examining the potential energy of these equilibrium configurations as well as conducting equivalence class analysis, the equilibrium configurations of the charges are cataloged and their symmetry is characterized. The equilibrium configurations in this problem are described in detail in the Thomson problem.

In order to test the validity of the numerical model constructed for this problem, convergence testing is also applied. This is done by simulating an identical scenario using multiple discretization levels, and analyzing the level-to-level differences. This convergence testing allows us to determine to which order our finite difference approximation is accurate.

Review of Theory

Equations of motion

In this simulation, natural units are used for all variables. This allows equations to be simplified by setting all masses and charges as well as the radius of the confining sphere centered at the origin equal to 1:

$$m_i = 1, \quad q_i = 1, \quad R = 1$$

Using Cartesian coordinates, the position of each charge is written as

$$\mathbf{r}_i(t) \equiv [x_i(t), y_i(t), z_i(t)] \quad , \quad i = 1, 2, \ldots, N,$$

where

$$r_i \equiv |\mathbf{r}_i| \equiv \sqrt{x_i^2 + y_i^2 + z_i^2} = 1 \quad , \quad i = 1, 2, \ldots, N.$$

The separation vectors between charges can be computed using the following formulas:

$$\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i$$

$$r_{ij} = |\mathbf{r}_j - \mathbf{r}_i|$$

$$\hat{r}_{ij} \equiv \frac{\mathbf{r}_j - \mathbf{r}_i}{r_{ij}} = \frac{\mathbf{r}_{ij}}{r_{ij}}\,.$$

The variable γ is used as the scaling parameter for the velocity-dependent retarding force. The equation for Newton's second law can then be written for each charge as:

$$m_i a_i = F_{i,\text{electrostatic}} + F_{i,\text{retarding}}$$

which can be expanded to

$$m_i \mathbf{a}_i = -k_e \sum_{j=1, j\neq i}^{N} \frac{q_i q_j}{r_{ij}} \hat{\mathbf{r}}_{ij} - \gamma \mathbf{v}_i, \quad i = 1, 2, \ldots N, \quad 0 \leq t \leq t_{\max}.$$

Using natural units ($k_e = 1$) and writing $\mathbf{a}_i$ and $\mathbf{v}_i$ as derivatives of $\mathbf{r}_i$, this expression can be simplified to

$$\frac{d^2\mathbf{r}_i}{dt^2} = -\sum_{j=1, j\neq i}^{N} \frac{\mathbf{r}_{ij}}{{r_{ij}}^3} - \gamma \frac{d\mathbf{r}_i}{dt}, \quad i = 1, 2, \ldots N, \quad 0 \leq t \leq t_{\max}. \tag{1}$$

The above equation is what is used to numerically solve the equations of motion using FDAs.

Electrostatic Potential Energy

The electrostatic potential energy of a point charge distribution is given by the following formula:

$$W = k_e \sum_{i=1}^{N} \sum_{j>i}^{N} \frac{q_i q_j}{r_{ij}}$$

Writing this potential energy in natural units and as a function of time, we can rewrite the above equation as:

$$V(t) = \sum_{i=2}^{N} \sum_{j=1}^{i-1} \frac{1}{r_{ij}} \tag{2}$$

Since in our scenario energy is not conserved (kinetic energy is dissipated via the velocity-dependent retarding force), we expect the potential to trend towards a minimum as $t \to \infty$.

Equivalence Classes

The concept of *equivalence classes* is introduced in order to characterize the different equilibrium configurations of the point charges on the unit sphere. As $t \to \infty$ and V is minimized, the equilibrium configuration for N charges becomes independent of the initial conditions. In words, the number of equivalence classes is the number of groups of charges that are indistinguishable in the equilibrium configuration.

In order to calculate the number of equivalence classes, the magnitude of the displacement vector from charge i to charge j is defined as

$$d_{ij} = |\mathbf{r}_j - \mathbf{r}_i| \quad i, j = 1, 2, \ldots, N$$

For charges i and i' in the same equivalence class, the lists of magnitudes d_{ij} and $d_{i'j}$ to every other charge j are the same.

Numerical Approach

Finite Difference Equations

For this assignment, the following second-order accurate FDAs are used for the first and second vector time derivatives:

$$\left.\frac{d\mathbf{r}_i}{dt}\right|_{t=t^n} \to \frac{\mathbf{r}_i^{n+1} - \mathbf{r}_i^{n-1}}{2\Delta t} \tag{3}$$

$$\left.\frac{d^2\mathbf{r}_i}{dt^2}\right|_{t=t^n} \to \frac{\mathbf{r}_i^{n+1} - 2\mathbf{r}_i^{n} + \mathbf{r}_i^{n-1}}{\Delta t^2} \tag{4}$$

with t^n and Δt defined as

$$\text{Total number of time steps:} \quad n_t = 2^\ell + 1$$

$$\text{Time step length:} \quad \Delta t = \frac{t_{\max}}{n_t - 1} = 2^{-\ell} t_{\max}$$

$$\text{Time at time step } n: \quad t^n = (n-1)\Delta t, \quad n = 1, 2, \ldots n_t$$

Plugging (3) and (4) into (1), we can create a new FDA for the equations of motion (evaluated at time t^n):

$$\frac{\mathbf{r}_i^{n+1} - 2\mathbf{r}_i^{n} + \mathbf{r}_i^{n-1}}{\Delta t^2} = -\left.\sum_{j=1, j\neq i}^{N} \frac{\mathbf{r}_{ij}}{{r_{ij}}^3}\right|_{t=t^n} - \gamma \frac{\mathbf{r}_i^{n+1} - \mathbf{r}_i^{n-1}}{2\Delta t}$$

This equation can be reorganized to produce a formula for $\mathbf{r}_i^{n+1}$, the next position vector following $\mathbf{r}_i^n$:

$$\mathbf{r}_i^{n+1} = \left(\frac{1}{\Delta t^2} + \frac{\gamma}{2\Delta t}\right)^{-1} \left(\left(\frac{\gamma}{2\Delta t} - \frac{1}{\Delta t^2}\right)\mathbf{r}_i^{n-1} + \frac{2}{\Delta t^2}\mathbf{r}_i^n - \left.\sum_{j=1, j\neq i}^{N} \frac{\mathbf{r}_{ij}}{{r_{ij}}^3}\right|_{t=t^n}\right)$$

In this implementation, the current time step n is treated as the time step to be calculated, producing the following equation which is implemented in code:

$$\mathbf{r}_i^{n} = \left(\frac{1}{\Delta t^2} + \frac{\gamma}{2\Delta t}\right)^{-1} \left(\left(\frac{\gamma}{2\Delta t} - \frac{1}{\Delta t^2}\right)\mathbf{r}_i^{n-2} + \frac{2}{\Delta t^2}\mathbf{r}_i^{n-1} - \left.\sum_{j=1, j\neq i}^{N} \frac{\mathbf{r}_{ij}}{{r_{ij}}^3}\right|_{n-1}\right) \tag{5}$$

Since this FDA is a vector equation, it is evaluated three times for each charge at every time step for each of the three Cartesian coordinate directions. Luckily, MATLAB syntax allows a vector equation such as this one to be computed simultaneously for each coordinate.

Additionally, since this FDA does not constrain charges to be on the unit sphere, each charge is re-normalized after by dividing its position by its magnitude:

$$\tilde{\mathbf{r}}_i^n \to \frac{\mathbf{r}_i^n}{|\mathbf{r}_i^n|} = \frac{[x_i^n, y_i^n, z^n]}{\sqrt{(x_i^n)^2 + (y_i^n)^2 + (z_i^n)^2}} \tag{6}$$

This normalization assumes that the distance the FDA (5) moves each charge radially off the unit sphere is negligible.

Convergence Testing

The validity of the FDA computed can be evaluated by conducting convergence analysis. Convergence analysis assumes that the error between the true solution and the computed solution is given by a *function* $e_2(t^n)$ scaled by the time-step duration:

$$\lim_{\Delta t \to 0} u_*(t^n) - u(t^n) \approx \Delta t e_2(t^n) \quad , \quad \text{If the approximation is first order}$$

$$\lim_{\Delta t \to 0} u_*(t^n) - u(t^n) \approx \Delta t^2 e_2(t^n) \quad , \quad \text{If the approximation is second order}$$

where $u_*(t^n)$ is the true solution at each time step and $u(t^n)$ is the computed solution at each time step.

Convergence analysis is done by computing the FDA for each time-step at different discretization levels l and comparing the results. For a first order approximation this looks like

$$du_l = u_l(t^n) - u_{l+1}(t^n) \approx -\frac{1}{2}\Delta t_l e_2(t^n)$$

$$du_{l+1} = u_{l+1}(t^n) - u_{l+2}(t^n) \approx -\frac{1}{4}\Delta t_l e_2(t^n)$$

For a second order approximation this looks like

$$du_l = u_l(t^n) - u_{l+1}(t^n) \approx -\frac{3}{4}\Delta t_l^2 e_2(t^n)$$

$$du_{l+1} = u_{l+1}(t^n) - u_{l+2}(t^n) \approx -\frac{3}{16}\Delta t_l^2 e_2(t^n)$$

Therefore, for a first order approximation we expect to have

$$du_{l+1} \approx \frac{du_l}{2^1}, \quad du_{l+2} \approx \frac{du_l}{2^2}, \quad du_{l+3} \approx \frac{du_l}{2^3}, \quad \dots \tag{7}$$

and for a second order approximation we expect to have

$$du_{l+1} \approx \frac{du_l}{4^1}, \quad du_{l+2} \approx \frac{du_l}{4^2}, \quad du_{l+3} \approx \frac{du_l}{4^3}, \quad \dots \tag{8}$$

In our implementation we are using second order FDA to compute both the first and second time derivatives of position so the per-step error is second order. However, since the number of steps to get to t_{max} varies like $\frac{1}{\Delta t}$, we expect the overall simulation to have first order accuracy and therefore we expect to convergence of curves as described by (7).

Implementation

Refer to Appendix A for the full source code implementation of the simulation, including the finite-difference approximation, electrostatic potential computation, and equivalence class computation.

In order to generate random initial positions of charges for code development as well as in `plotv.m` and `survey.m`, x, y and z coordinates for each charge are first chosen at random locations on $[-1, 1]$. For each charge, its initial position vector $\mathbf{r}_i^0$ is then re-normalized after by dividing it by its magnitude. This is shown in the code snippet below:

```
% Generate nc random inital locations for charges
r0 = 2*rand(nc,3) - 1;
for i = 1:nc
    r0(i,:) = r0(i,:)/(norm(r0(i,:)));
end
```

The equivalence classes are computed in the suggested way described in the project outline. First, a matrix `dij` is generated which for each index `(i, j)` holds the magnitude of the vector from $\mathbf{r}_i^{n_t}$ to $\mathbf{r}_j^{n_t}$. This is implemented in MATLAB via the line `dij(i, j) = norm(r(j,:,nt) - r(i,:,nt))` placed inside a nested for loop. Each row of this matrix is then sorted in ascending order, which is interpreted as the list of distances from charge i to each other charge sorted in ascending order. In order to find which charges have matching rows (meaning they are in the same equivalence class), a nested for loop is used that compares each row to each other row in `dij`, while keeping track of which rows have already been placed into an equivalence class and how many rows (or charges) are in each equivalence class. Refer to Appendix A for the full source code.

Please refer to Appendices B, C, and D for the source code for `convtest.m`, `plotv.m`, and `survey.m`, respectively.

Results

Through gradual development of

convtest.m output

plotv.m output

survey.m output

Video of sample evolution

The attached file `charges.mp4`

Conclusions

Appendix A - charges.m Code

```
%% 5.2 - Top-level function for simulation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% charges: Top-level function for solution of charges-on-a-sphere
% problem.
%
% Input arguments
%
% r0: Initial positions (nc x 3 array, where nc is the number of
% charges)
% tmax: Maximum simulation time
% level: Discretization level
% gamma: Dissipation coefficient
% epsec: Tolerance for equivalence class analysis
%
% Output arguments
%
% t: Vector of simulation times (length nt row vector)
% r: Positions of charges (nc x 3 x nt array)
% v: Potential vector (length nt row vector)
% v_ec: Equivalence class counts (row vector with length determined
% by equivalence class analysis)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [t, r, v, v_ec] = charges(r0, tmax, level, gamma, epsec)

    nt = 2^level + 1;           % Number of timesteps
    dt = tmax / (nt - 1);       % Time period between steps
    t = linspace(0, tmax, nt);  % Vector of simulation times

    % Number of charges
    nc = height(r0);

    % Intialize array for locations of charges throughout all timesteps
    r = zeros(nc, 3, nt);
    r(:,:,1) = r0; r(:,:,2) = r0; % No initial velocity

    % Array for storing the potential at each time step
    v = zeros(1, nt);

    % Compute for all time steps
    for n = 1:nt
        % Compute the position of each charge at the current time step
        for i = 1:nc
            % We already know all positions at the first two time steps
            if n == 1 || n == 2
                continue
            end

            % Compute electrostatic forces on the current charge
            es_disp = [0 0 0];
            for j = 1:nc
                if j == i
                    continue
                end
                es_disp = es_disp + (r(j,:,n-1) - r(i,:,n-1)) / ...
                          norm(r(j,:,n-1) - r(i,:,n-1))^3;
```

```
            end

            % Displacement term depending on location of the charge from 1
                ts ago
            one_ts_disp = (2 / dt^2) * r(i,:,n-1);
            % Displacement term depending on location of the charge from 2
                ts ago
            two_ts_disp = (gamma/(2 * dt) - 1/(dt^2)) * r(i,:,n-2);

            % Combining all computed displacements prior to normalization
            pos_before_norm = (two_ts_disp + one_ts_disp - es_disp) / ...
                              (1 / dt^2 - gamma/(2 * dt));

            % Normalize the position
            pos_norm = pos_before_norm / (norm(pos_before_norm));
            % Add to r matrix
            r(i,:,n) = pos_norm;
        end

        % Compute potential at the current time step
        for i = 2:nc
            for j = 1:i-1
                v(n) = v(n) + 1/(norm(r(j,:,n) - r(i,:,n)));
            end
        end
    end

    % Compute equivalence classes

    % Compute dij matrix
    dij = zeros(nc);
    for i = 1:nc
        for j = 1:nc
            dij(i, j) = norm(r(j,:,nt) - r(i,:,nt));
        end
    end
    % Sort each row of matrix to be in ascending order
    dij_sorted = sort(dij, 2);

    % Vector for storing the number of charges in each equivalence class.
    % The number of equivalence classes is the number of nonzero entries
    v_ec = zeros(1, nc);

    % Array of flags for if each row has been sorted into an equivalence
        class
    rows_in_ec = zeros(1, nc);

    for i = 1:nc
        if rows_in_ec(i) == 1
            continue
        end

        % If not already matched, create a new equivalence class
        v_ec(i) = v_ec(i) + 1;
        rows_in_ec(i) = 1;

        for j = 1:nc
            if j == i || rows_in_ec(j) == 1
```

```
                    continue
                end
                % Check if rows are the same
                if all(abs(dij_sorted(j,:) - dij_sorted(i,:)) < epsec)
                    v_ec(i) = v_ec(i) + 1;
                    rows_in_ec(j) = 1;
                end
            end
        end

        % Remove zero entries and sort in descending order
        v_ec(v_ec == 0) = [];
        v_ec = sort(v_ec, 'descend');

end
```

Appendix B - convtest.m Code

```
%% 6.1 - 4-level convergence test

close all; clear; clc;

% Simulation parameters as described in project description
nc = 4;
tmax = 10;
gamma = 1;
epsec = 1.0e-5;

% Initial conditions of charges
r0 = [[1, 0, 0]; [0, 1, 0]; [0, 0, 1]; (sqrt(3)/3) * [1, 1, 1]];

% Run computation at each discretization level
[t10, r10, v10, v_ec10] = charges(r0, tmax, 10, gamma, epsec);
[t11, r11, v11, v_ec11] = charges(r0, tmax, 11, gamma, epsec);
[t12, r12, v12, v_ec12] = charges(r0, tmax, 12, gamma, epsec);
[t13, r13, v13, v_ec13] = charges(r0, tmax, 13, gamma, epsec);

% x coordinate values of charge one at level 10
x10 = reshape(r10(1,1,:), size(t10));
x11 = reshape(r11(1,1,:), size(t11));
x12 = reshape(r12(1,1,:), size(t12));
x13 = reshape(r13(1,1,:), size(t13));

% Calculating the level-to-level differences, taking every second
% value of the larger length array
dx10 = downsample(x11, 2) - x10;
dx11 = downsample(x12, 2) - x11;
dx12 = downsample(x13, 2) - x12;

% First plot all curves for rho = 2
fig1 = figure(1);
rho = 2;
hold on
plot(t10, dx10, 'LineWidth', 2);
plot(t11, rho*dx11, 'LineWidth', 2);
plot(t12, rho^2*dx12, 'LineWidth', 2);
xlabel("Time");
ylabel("Difference between level");
legend('dx10', 'rho * dx11', 'rho^2 * dx12');
title("Convergence Test: rho = 2");
ax = gca;
ax.FontSize = 12;

% First plot all curves for rho = 4
fig2 = figure(2);
rho = 4;
hold on
plot(t10, dx10, 'LineWidth', 2);
plot(t11, rho*dx11, 'LineWidth', 2);
plot(t12, rho^2*dx12, 'LineWidth', 2);
xlabel("Time");
ylabel("Difference between level");
legend('dx10', 'rho * dx11', 'rho^2 * dx12');
title("Convergence Test: rho = 4");
```

```
57 ax = gca;
58 ax.FontSize = 12;
```

Appendix C - plotv.m Code

```
%% 6.2 - Time evolution of potential for 12-charge calculation

close all; clear; clc;

% Simulation parameters
nc = 12;
tmax = 10;
level = 12;
gamma = 1;
epsec = 1.0e-5;

% Generate nc random inital locations for charges
r0 = 2*rand(nc,3) - 1;
for i = 1:nc
    r0(i,:) = r0(i,:)/(norm(r0(i,:)));
end

% Run simulation
[t, r, v, v_ec] = charges(r0, tmax, level, gamma, epsec);

% Plot V(t) vs. t
plot(t,v)
xlabel("Time t")
ylabel("Total potential energy V(t)")
title({"Potential Energy V(t) vs. time t", ...
       "nc = 12, tmax = 10, level = 12, gamma = 1, epsec = 1.0e-5"})
ax = gca;
ax.FontSize = 12;
```

Appendix D - survey.m Code

```
%% 6.3 Survey of V(t_max; N) and v_ec(N) for various values of N

close all; clear; clc;

tmax = 500;
level = 12;
gamma = 2;
epsec = 1.0e-3;

% Creating file to be written to
fid_v = fopen('vsurvey.dat','w');
fid_ec = fopen('ecsurvey.dat', 'w');

for nc = 2:60
    r0 = 2*rand(nc,3) - 1;
    for i = 1:nc
        r0(i,:) = r0(i,:)/(norm(r0(i,:)));
    end
    [t, r, v, v_ec] = charges(r0, tmax, level, gamma, epsec);

    fprintf(fid_v, '%3d %16.10f\n', nc, v(end));
    fprintf(fid_ec, '%3d ', nc);
    fprintf(fid_ec, '%d ', v_ec);
    fprintf(fid_ec, '\n');
end

fclose(fid_v);
fclose(fid_ec);
```