PHYS 410 Project 1

Gavin Pringle, 56401938

October 13, 2024

Introduction

In this project, the problem of

Review of Theory

Equations of motion

Electrostatic Potential Energy

Equivalence Classes

Numerical Approach

Finite Difference Equations

Convergence Testing

Implementation

Results

convtest.m output

plotv.m output

survey.m output

Video of sample evolution

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 lenght 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);
```