

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

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

where

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

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

$$\begin{aligned} \mathbf{r}_{ij} &= \mathbf{r}_j - \mathbf{r}_i \\ r_{ij} &= |\mathbf{r}_j - \mathbf{r}_i| \\ \hat{\mathbf{r}}_{ij} &\equiv \frac{\mathbf{r}_j - \mathbf{r}_i}{r_{ij}} = \frac{\mathbf{r}_{ij}}{r_{ij}}. \end{aligned}$$

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:

$$m_i \mathbf{a}_i = F_{i,\text{electrostatic}} + F_{i,\text{retarding}}$$

which can be expanded to

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

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

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

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:

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

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 $t \rightarrow \infty$.

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 $t \rightarrow \infty$ 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

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

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:

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

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

with t^n and Δt defined as

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

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

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

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

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

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

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

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:

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

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:

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

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:

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

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

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

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

For a second order approximation this looks like

$$\begin{aligned} du_l &= u_l(t^n) - u_{l+1}(t^n) \approx -\frac{3}{4} \Delta t_l^2 e_2(t^n) \\ du_{l+1} &= u_{l+1}(t^n) - u_{l+2}(t^n) \approx -\frac{3}{16} \Delta t_l^2 e_2(t^n) \end{aligned}$$

Therefore, for a first order approximation we expect to have

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

and for a second order approximation we expect to have

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

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 initial 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` element-by-element to a tolerance of `epsec`, 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 frequent use of the provided plotting scripts `charges_plot.m` and `charges_video.m`, the top-level function `charges.m` was developed until it was clear it was working correctly. Furthermore, the required test scripts described below produced output data that agreed with the expected behaviour of the simulation.

`convtest.m` output

Figures 1 and 2 below show the output of the script `convtest.m`, which conducts convergence analysis of the x-coordinate of the first charge throughout the simulation time across different discretization levels: $l = 10, 11, 12, 13$. The convergence analysis was conducted with the input parameters as described in the project outline document and in Appendix B.

From the plots, we can see near-coincidence of the curves for $\rho = 2$ and discrepancy between the curves for $\rho = 4$

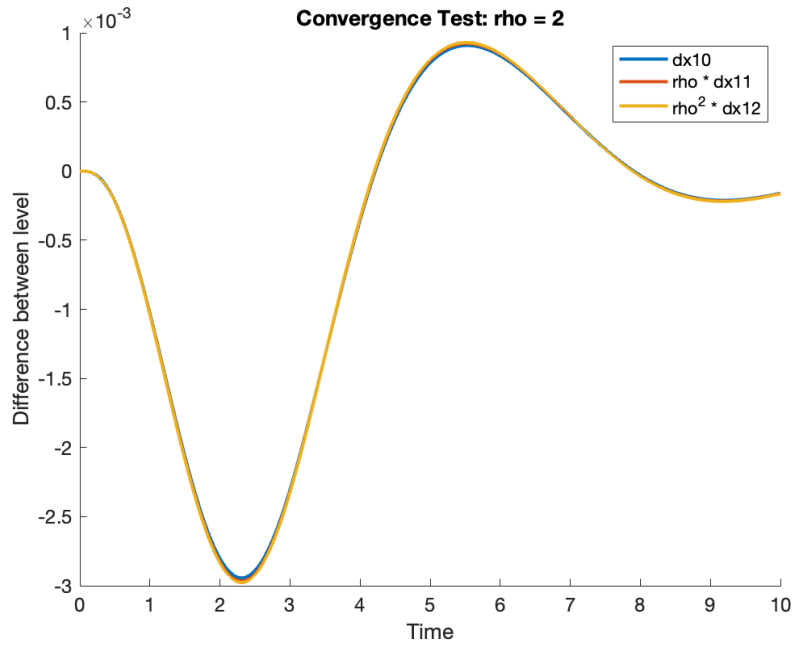


Figure 1: First order convergence test

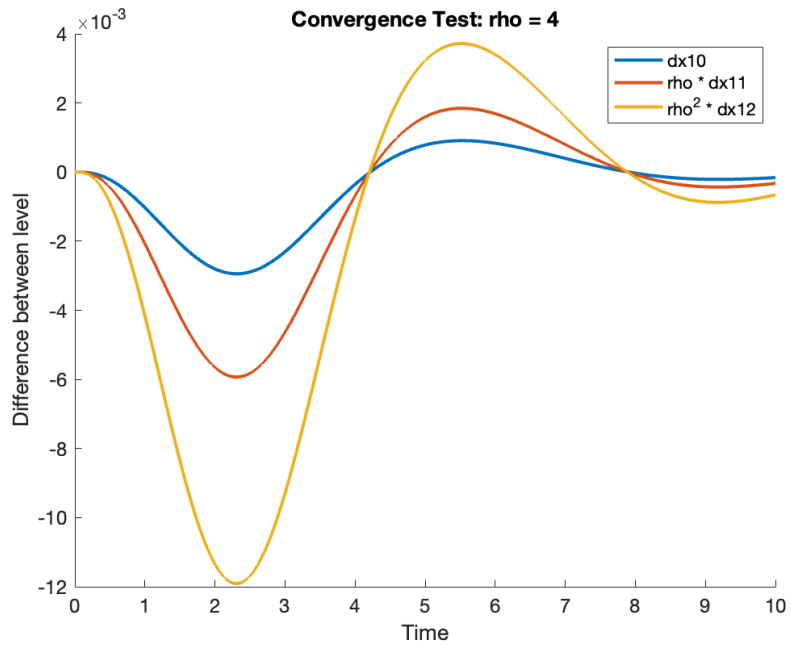


Figure 2: Second order convergence test

plotv.m output

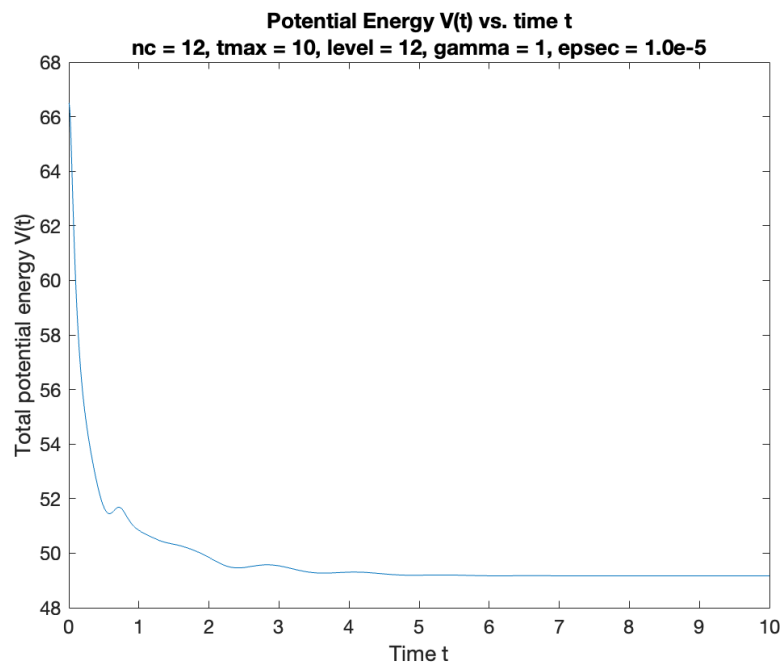


Figure 3: Output of `plotv.m` - electrostatic potential energy vs. time

survey.m output

2	0.5000000000
3	1.7320508076
4	3.6742346142
5	6.4746914947
6	9.9852813742
7	14.4529797059
8	19.6752878612
9	25.7599865313
10	32.7169494601
11	40.5964505082
12	49.1652530576
13	58.8532306117
14	69.3063632966
15	80.6702441143
16	92.9116553025
17	106.0504048286
18	120.0844674475
19	135.0894679270
20	150.8815683338
21	167.6416258950
22	185.2875361493
23	203.9301906629
24	223.3470740518
25	243.8127606026
26	265.1333263174
27	287.3026150330
28	310.4915423582
29	334.6344403086

30 359.6039459038
 31 385.5308380633
 32 412.2612746505
 33 440.2040582989
 34 468.9048532813
 35 498.5734541438
 36 529.1224112279
 37 560.6279730637
 38 593.0489435380
 39 626.3890090168
 40 660.6752788346
 41 695.9167443419
 42 732.0781075437
 43 769.1908491730
 44 807.1742630853
 45 846.1884022990
 46 886.1671136392
 47 927.0592706797
 48 968.7134553438
 49 1011.5571826536
 50 1055.1823147263
 51 1099.8192903189
 52 1145.4219806346
 53 1191.9222906171
 54 1239.3614747292
 55 1287.7890572384
 56 1337.0987274148
 57 1387.4303724833
 58 1438.6381050033
 59 1490.7743860781
 60 1543.8350995985

2 2
 3 3
 4 4
 5 3 2
 6 6
 7 2 1 1 1 1 1
 8 8
 9 6 3
 10 8 2
 11 4 2 2 2 1
 12 12
 13 4 2 2 2 2 1
 14 12 2
 15 6 6 3
 16 12 4
 17 10 5 2
 18 8 8 2
 19 2 2 2 2 2 2 2 2 1
 20 6 6 6 2
 21 2 2 2 2 2 2 1 1 1 1 1 1
 22 12 6 4
 23 6 6 6 3 2
 24 24
 25 1
 26 2 2 2 2 2 2 2 2 2 2 2 2
 27 10 10 5 2

[illegible]

Video of sample evolution

The attached file `charges.mp4`

Conclusions

Appendix A - charges.m Code

```

1 %% 5.2 – Top-level function for simulation
2
3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 % charges: Top-level function for solution of charges-on-a-sphere
5 % problem.
6 %
7 % Input arguments
8 %
9 % r0: Initial positions (nc x 3 array, where nc is the number of
10 % charges)
11 % tmax: Maximum simulation time
12 % level: Discretization level
13 % gamma: Dissipation coefficient
14 % epsec: Tolerance for equivalence class analysis
15 %
16 % Output arguments
17 %
18 % t: Vector of simulation times (length nt row vector)
19 % r: Positions of charges (nc x 3 x nt array)
20 % v: Potential vector (length nt row vector)
21 % v_ec: Equivalence class counts (row vector with length determined
22 % by equivalence class analysis)
23 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
24 function [t, r, v, v_ec] = charges(r0, tmax, level, gamma, epsec)
25
26     nt = 2^level + 1;           % Number of timesteps
27     dt = tmax / (nt - 1);       % Time period between steps
28     t = linspace(0, tmax, nt); % Vector of simulation times
29
30     % Number of charges
31     nc = height(r0);
32
33     % Initialize array for locations of charges throughout all timesteps
34     r = zeros(nc, 3, nt);
35     r(:, :, 1) = r0; r(:, :, 2) = r0; % No initial velocity
36
37     % Array for storing the potential at each time step
38     v = zeros(1, nt);
39
40     % Compute for all time steps
41     for n = 1:nt
42         % Compute the position of each charge at the current time step
43         for i = 1:nc
44             % We already know all positions at the first two time steps
45             if n == 1 || n == 2
46                 continue
47             end
48
49             % Compute electrostatic forces on the current charge
50             es_disp = [0 0 0];
51             for j = 1:nc
52                 if j == i
53                     continue
54                 end
55                 es_disp = es_disp + (r(j, :, n-1) - r(i, :, n-1)) / ...
56                     norm(r(j, :, n-1) - r(i, :, n-1))^3;

```

```

57         end
58
59         % Displacement term depending on location of the charge from 1
           ts ago
60         one_ts_disp = (2 / dt^2) * r(i,:,n-1);
61         % Displacement term depending on location of the charge from 2
           ts ago
62         two_ts_disp = (gamma/(2 * dt) - 1/(dt^2)) * r(i,:,n-2);
63
64         % Combining all computed displacements prior to normalization
65         pos_before_norm = (two_ts_disp + one_ts_disp - es_disp) / ...
66             (1 / dt^2 - gamma/(2 * dt));
67
68         % Normalize the position
69         pos_norm = pos_before_norm / (norm(pos_before_norm));
70         % Add to r matrix
71         r(i,:,n) = pos_norm;
72     end
73
74     % Compute potential at the current time step
75     for i = 2:nc
76         for j = 1:i-1
77             v(n) = v(n) + 1/(norm(r(j,:,n) - r(i,:,n)));
78         end
79     end
80 end
81
82 % Compute equivalence classes
83
84 % Compute dij matrix
85 dij = zeros(nc);
86 for i = 1:nc
87     for j = 1:nc
88         dij(i, j) = norm(r(j,:,nt) - r(i,:,nt));
89     end
90 end
91 % Sort each row of matrix to be in ascending order
92 dij_sorted = sort(dij, 2);
93
94 % Vector for storing the number of charges in each equivalence class.
95 % The number of equivalence classes is the number of nonzero entries
96 v_ec = zeros(1, nc);
97
98 % Array of flags for if each row has been sorted into an equivalence
           class
99 rows_in_ec = zeros(1, nc);
100
101 for i = 1:nc
102     if rows_in_ec(i) == 1
103         continue
104     end
105
106     % If not already matched, create a new equivalence class
107     v_ec(i) = v_ec(i) + 1;
108     rows_in_ec(i) = 1;
109
110     for j = 1:nc
111         if j == i || rows_in_ec(j) == 1

```

```

112         continue
113     end
114     % Check if rows are the same
115     if all(abs(dij_sorted(j,:) - dij_sorted(i,:)) < epsec)
116         v_ec(i) = v_ec(i) + 1;
117         rows_in_ec(j) = 1;
118     end
119 end
120 end
121
122 % Remove zero entries and sort in descending order
123 v_ec(v_ec == 0) = [];
124 v_ec = sort(v_ec, 'descend');
125
126 end

```

Appendix B - convtest.m Code

```
1 %% 6.1 - 4-level convergence test
2
3 close all; clear; clc;
4
5 % Simulation parameters as described in project description
6 nc = 4;
7 tmax = 10;
8 gamma = 1;
9 epsec = 1.0e-5;
10
11 % Initial conditions of charges
12 r0 = [[1, 0, 0]; [0, 1, 0]; [0, 0, 1]; (sqrt(3)/3) * [1, 1, 1]];
13
14 % Run computation at each discretization level
15 [t10, r10, v10, v_ec10] = charges(r0, tmax, 10, gamma, epsec);
16 [t11, r11, v11, v_ec11] = charges(r0, tmax, 11, gamma, epsec);
17 [t12, r12, v12, v_ec12] = charges(r0, tmax, 12, gamma, epsec);
18 [t13, r13, v13, v_ec13] = charges(r0, tmax, 13, gamma, epsec);
19
20 % x coordinate values of charge one at level 10
21 x10 = reshape(r10(1,1,:), size(t10));
22 x11 = reshape(r11(1,1,:), size(t11));
23 x12 = reshape(r12(1,1,:), size(t12));
24 x13 = reshape(r13(1,1,:), size(t13));
25
26 % Calculating the level-to-level differences, taking every second
27 % value of the larger length array
28 dx10 = downsample(x11, 2) - x10;
29 dx11 = downsample(x12, 2) - x11;
30 dx12 = downsample(x13, 2) - x12;
31
32 % First plot all curves for rho = 2
33 fig1 = figure(1);
34 rho = 2;
35 hold on
36 plot(t10, dx10, 'LineWidth', 2);
37 plot(t11, rho*dx11, 'LineWidth', 2);
38 plot(t12, rho^2*dx12, 'LineWidth', 2);
39 xlabel("Time");
40 ylabel("Difference between level");
41 legend('dx10', 'rho * dx11', 'rho^2 * dx12');
42 title("Convergence Test: rho = 2");
43 ax = gca;
44 ax.FontSize = 12;
45
46 % First plot all curves for rho = 4
47 fig2 = figure(2);
48 rho = 4;
49 hold on
50 plot(t10, dx10, 'LineWidth', 2);
51 plot(t11, rho*dx11, 'LineWidth', 2);
52 plot(t12, rho^2*dx12, 'LineWidth', 2);
53 xlabel("Time");
54 ylabel("Difference between level");
55 legend('dx10', 'rho * dx11', 'rho^2 * dx12');
56 title("Convergence Test: rho = 4");
```

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

Appendix C - plotv.m Code

```
1 %% 6.2 - Time evolution of potential for 12-charge calculation
2
3 close all; clear; clc;
4
5 % Simulation parameters
6 nc = 12;
7 tmax = 10;
8 level = 12;
9 gamma = 1;
10 epsec = 1.0e-5;
11
12 % Generate nc random initial locations for charges
13 r0 = 2*rand(nc,3) - 1;
14 for i = 1:nc
15     r0(i,:) = r0(i,)/(norm(r0(i,:)));
16 end
17
18 % Run simulation
19 [t, r, v, v_ec] = charges(r0, tmax, level, gamma, epsec);
20
21 % Plot V(t) vs. t
22 plot(t,v)
23 xlabel("Time t")
24 ylabel("Total potential energy V(t)")
25 title({"Potential Energy V(t) vs. time t", ...
26        "nc = 12, tmax = 10, level = 12, gamma = 1, epsec = 1.0e-5"})
27 ax = gca;
28 ax.FontSize = 12;
```


Appendix D - survey.m Code

```
1 %% 6.3 Survey of V(t_max; N) and v_ec(N) for various values of N
2
3 close all; clear; clc;
4
5 tmax = 500;
6 level = 12;
7 gamma = 2;
8 epsec = 1.0e-3;
9
10 % Creating file to be written to
11 fid_v = fopen('vsurvey.dat','w');
12 fid_ec = fopen('ecsurvey.dat','w');
13
14 for nc = 2:60
15     r0 = 2*rand(nc,3) - 1;
16     for i = 1:nc
17         r0(i,:) = r0(i,:)/(norm(r0(i,:)));
18     end
19     [t, r, v, v_ec] = charges(r0, tmax, level, gamma, epsec);
20
21     fprintf(fid_v, '%3d %16.10f\n', nc, v(end));
22     fprintf(fid_ec, '%3d ', nc);
23     fprintf(fid_ec, '%d ', v_ec);
24     fprintf(fid_ec, '\n');
25 end
26
27 fclose(fid_v);
28 fclose(fid_ec);
```