

Lab 2: Minimization using Conjugate Gradient / Steepest Descent Method

Shreyansh Agarwal : B23PH1024

Aim :- Given 10 particles ($m=1$) in a box of $10 \times 10 \times 10$ units under influence of gravitation potential energy $V(r_{ij}) = \frac{-1}{r_{ij}}$. What is the minimized position of the 10 particles using conjugate gradient and steepest descent method.

Theory:- 1. Steepest Descent Algorithm:- Given $f(x)$ s.t. $x \in \mathbb{R}^n$ and $f(x)$ is differentiable,

$$f(x+d) = f(x) + \nabla f(x)^T d$$

$$\text{where } d = -\nabla f(x).$$

Algorithm :-

I. Given x^0 , set $k:=0$

II. $d^k = -\nabla f(x^k)$. If $d^k=0$ stop.

III. Solve $\min (f(x^k + \alpha d^k))$ fro stepsize α^k

IV. $x^{k+1} \leftarrow x^k + \alpha^k d^k$, $k \leftarrow k+1$. Go to Step II.

2. Conjugate Gradient Algorithm:- Given $f(x) = \frac{1}{2} x^T A x - b^T x$

$$\nabla f(x) = Ax - b$$

Solution of $\min f(x)$ is equivalent to
 $\nabla f(x) = 0 \Rightarrow Ax - b = 0$.

Algorithm:-

- i. $x^0; k=0$
- ii. While $k \leq \text{num_Iterations}$:
- iii. $r_k = b - Ax_k$
- iv. If $r_k < \text{threshold}$ return x_0
- v. $p_k := r_k$
- vi. $k := 0$
- vii. $\alpha_k = r_k^T r_k / p_k^T A p_k$
- viii. $x_{k+1} = x_k + \alpha_k p_k$

Procedure :-

- i. Initialize 10 particles in a box of length $10 \times 10 \times 10$.
- ii. Use steepest descent algorithm & CG algorithm to get final positions.

III. Some points:-

a) Use $V(r_{ij}) = \frac{-1}{r_{ij} + \epsilon}$ where ϵ is a small padding

used to avoid division by zero. It is equivalent to minimum allowed distance between 2 particles.

b) $f(x) = \sum_{i,j} V(r_{ij}) \rightarrow$ gives total energy of system, the function to be minimized.

$$c) \nabla_i f(x) = \frac{\partial}{\partial x_i} \sum_{j \neq i} \frac{-1}{r_{ij} + \epsilon} = \sum_{j \neq i} \frac{1}{(r_{ij} + \epsilon)^2} \frac{\partial}{\partial x_i} (r_{ij} = \|x_i - x_j\|)$$

$$= \sum_{j \neq i} \frac{(x_i - x_j)}{(r_{ij} + \epsilon)^3}$$

where $x_i \in \mathbb{R}^3$, $x_k \in X$ $X \in \mathbb{R}_3^{10}$

IV. Line Search Procedure to get α_k for SD :-

a) Initialize $\alpha = 1, \beta = 0.5, C = 10^{-4}$

b) $E_{\text{curr}} = E(x_k)$

c) While $\alpha > 0$:

$$x_{k+1} = x_k + \alpha^* d$$

$$E_{\text{new}} = E(x_{k+1})$$

if $E_{\text{new}} \leq E_{\text{curr}} + C^* \alpha^* \text{Grad.} d$

return α

$$\alpha = \alpha * \beta$$

V. Use non-linear CG :-

a) $g_0 = \nabla f(x_0), d_0 = -g_0 : k := 0$

b) while $k \leq \text{num_iterations}$:

c) $\alpha_k = \min_d f(x_k + \alpha_k d_k)$

d) $x_{k+1} = x_k + \alpha_k d_k$

e) $g_{k+1} = \nabla f(x_{k+1})$

f) $\beta_{k+1} = \frac{\|g_{k+1}\|^2}{\|g_k\|^2} \rightarrow \text{Fletcher Reeves}$

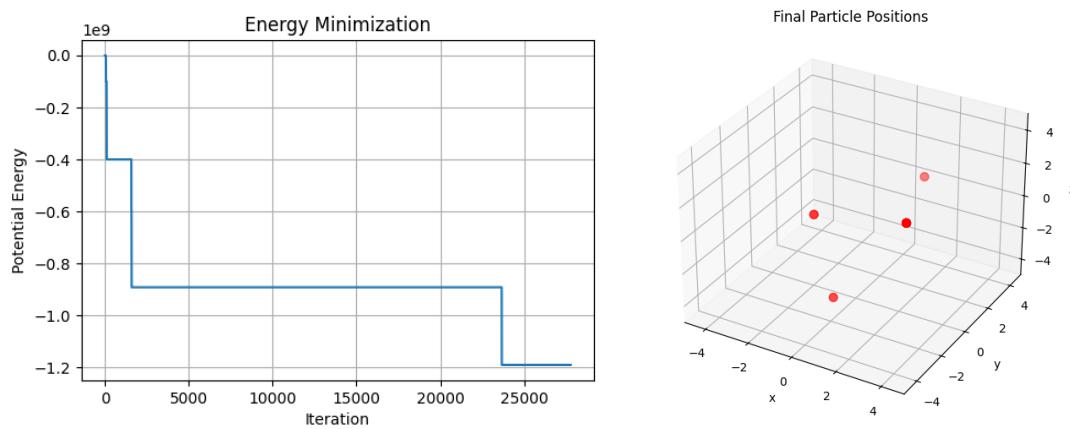
g) $d_{k+1} = -g_{k+1} + \beta_{k+1} d_k$

Result:

1. Steepest Descent:

Final Energy = -1191025259.17

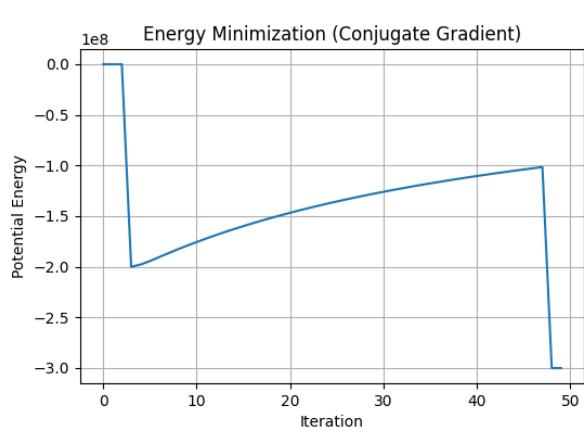
X	Y	Z
0.64573321	4.55737053	-0.1424616
4.98886103	-4.98880342	4.98545668
4.98886103	-4.98880342	4.98545668
4.98886103	-4.98880342	4.98545668
0.68656355	-2.94197096	-2.53177161
-4.98188663	4.98884991	-4.98529682
-4.98188663	4.98884991	-4.98529682
-4.98188663	4.98884991	-4.98529682
4.98886103	-4.98880342	4.98545668



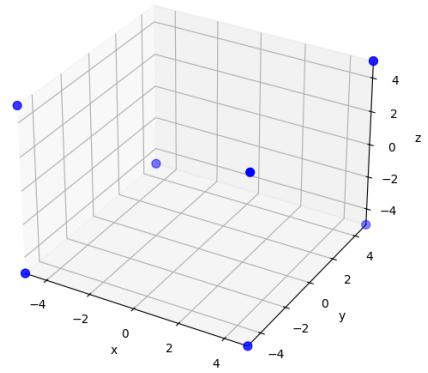
2. Conjugate Gradient:

Final Energy = -300000003.37

X	Y	Z
5	-5	-5
-5	5	-5
5	5	5
-5	-5	-5
-5	-5	-5
5	-5	5
-5	-5	5
-5	5	-5
5	5	5
5	5	-5



Final Particle Positions (CG)



Code:

Lab2-CG_SD.py

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 class SteepestDescent:
5     def __init__(self, particles, n_particles, box_length, tol=1e-12, max_iter=100000,
6      min_particle_dist=1e-8):
7         self.n = n_particles
8         self.L = box_length
9         self.tol = tol
10        self.max_iter = max_iter
11        self.Eplison = min_particle_dist
12        self.Energies = []
13        self.positions = particles
14
15    def potential_energy(self, positions=None):
16        if positions is None:
17            positions = self.positions
18        U = 0.0
19        for i in range(self.n):
20            for j in range(i+1, self.n):
21                rij = np.linalg.norm(positions[i] - positions[j])
22                U += -1.0 / (rij + self.Eplison)
23        return U
24
25    def gradient(self, positions=None):
26        if positions is None:
27            positions = self.positions
28        grad = np.zeros_like(positions)
29        for i in range(self.n):
30            for j in range(self.n):
31                if i != j:
32                    rij_vec = positions[i] - positions[j]
33                    rij = np.linalg.norm(rij_vec)
34                    if rij > 1e-8:
35                        grad[i] += rij_vec / ((rij + self.Eplison)**3)
36        return grad
37
38    def line_search(self, direction, grad):
39        alpha = 1.0
40        beta = 0.5
41        c = 1e-4
42
43        current_energy = self.potential_energy()
44        while alpha > 1e-8:
45            new_positions = self.positions + alpha * direction
46            new_positions = np.clip(new_positions, -self.L/2, self.L/2)
47            new_energy = self.potential_energy(new_positions)
48
49            # Armijo condition
50            if new_energy <= current_energy + c * alpha * np.sum(grad * direction):
51                return alpha
52            alpha *= beta
```

```

52     return alpha
53
54     def minimize(self):
55         prev_energy = self.potential_energy()
56
57         if self.Energies == []:
58             self.Energies.append(prev_energy)
59
60         for step in range(self.max_iter):
61             grad = self.gradient()
62             direction = -grad
63
64             if np.linalg.norm(grad)<=self.tol:
65                 print(f"Converged at step {step}, Energy = {self.Energies[-1]:.6f}")
66                 break
67
68             alpha = self.line_search(direction, grad)
69             self.positions += alpha * direction
70             self.positions = np.clip(self.positions, -self.L/2, self.L/2)
71
72             energy = self.potential_energy()
73             self.Energies.append(energy)
74             prev_energy = energy
75
76     return self.positions, self.Energies[-1]
77
78     def energy_plot(self):
79         plt.figure(figsize=(6,4))
80         plt.plot(self.Energies)
81         plt.xlabel("Iteration")
82         plt.ylabel("Potential Energy")
83         plt.title("Energy Minimization")
84         plt.grid(True)
85         plt.show()
86
87     def final_positions_plot(self):
88         fig = plt.figure(figsize=(6,6))
89         ax = fig.add_subplot(111, projection='3d')
90         ax.scatter(self.positions[:,0], self.positions[:,1], self.positions[:,2], c='red',
s=50)
91         ax.set_xlim([-self.L/2, self.L/2])
92         ax.set_ylim([-self.L/2, self.L/2])
93         ax.set_zlim([-self.L/2, self.L/2])
94         ax.set_title("Final Particle Positions")
95         ax.set_xlabel("x")
96         ax.set_ylabel("y")
97         ax.set_zlabel("z")
98         plt.show()
99
100    class ConjugateGradient:
101        def __init__(self, particles, n_particles, box_length, tol=1e-12, max_iter=100000,
min_particle_dist=1e-8):
102            self.n = n_particles
103            self.L = box_length

```

```

104     self.tol = tol
105     self.max_iter = max_iter
106     self.Eplison = min_particle_dist
107     self.Energies = []
108     self.positions = particles
109
110     def potential_energy(self, positions=None):
111         if positions is None:
112             positions = self.positions
113         U = 0.0
114         for i in range(self.n):
115             for j in range(i+1, self.n):
116                 rij = np.linalg.norm(positions[i] - positions[j])
117                 U += -1.0 / (rij + self.Eplison)
118         return U
119
120     def gradient(self, positions=None):
121         if positions is None:
122             positions = self.positions
123         grad = np.zeros_like(positions)
124         for i in range(self.n):
125             for j in range(self.n):
126                 if i != j:
127                     rij_vec = positions[i] - positions[j]
128                     rij = np.linalg.norm(rij_vec)
129                     if rij > 1e-8:
130                         grad[i] += rij_vec / ((rij + self.Eplison)**3)
131         return grad
132
133     def line_search(self, direction, grad):
134         alpha = 1.0
135         beta = 0.5
136         c = 1e-4
137
138         current_energy = self.potential_energy()
139         while alpha > 1e-8:
140             new_positions = self.positions + alpha * direction
141             new_positions = np.clip(new_positions, -self.L/2, self.L/2)
142             new_energy = self.potential_energy(new_positions)
143
144             # Armijo condition
145             if new_energy <= current_energy + c * alpha * np.sum(grad * direction):
146                 return alpha
147             alpha *= beta
148         return alpha
149
150     def minimize(self):
151         grad = self.gradient()
152         direction = -grad
153         prev_energy = self.potential_energy()
154         self.Energies.append(prev_energy)
155
156         for step in range(self.max_iter):
157             if np.linalg.norm(grad) <= self.tol:

```

```

158         print(f"Converged at step {step}, Energy = {self.Energies[-1]:.6f}")
159         break
160
161     alpha = self.line_search(direction, grad)
162     self.positions += alpha * direction
163     self.positions = np.clip(self.positions, -self.L/2, self.L/2)
164
165     new_grad = self.gradient()
166     energy = self.potential_energy()
167     self.Energies.append(energy)
168
169     # Fletcher-Reeves beta update
170     beta = np.sum(new_grad*new_grad) / (np.sum(grad*grad) + 1e-12)
171     direction = -new_grad + beta * direction
172
173     grad = new_grad
174     prev_energy = energy
175
176     return self.positions, self.Energies[-1]
177
178 def energy_plot(self):
179     plt.figure(figsize=(6,4))
180     plt.plot(self.Energies)
181     plt.xlabel("Iteration")
182     plt.ylabel("Potential Energy")
183     plt.title("Energy Minimization (Conjugate Gradient)")
184     plt.grid(True)
185     plt.show()
186
187 def final_positions_plot(self):
188     fig = plt.figure(figsize=(6,6))
189     ax = fig.add_subplot(111, projection='3d')
190     ax.scatter(self.positions[:,0], self.positions[:,1], self.positions[:,2],
c='blue', s=50)
191     ax.set_xlim([-self.L/2, self.L/2])
192     ax.set_ylim([-self.L/2, self.L/2])
193     ax.set_zlim([-self.L/2, self.L/2])
194     ax.set_title("Final Particle Positions (CG)")
195     ax.set_xlabel("x")
196     ax.set_ylabel("y")
197     ax.set_zlabel("z")
198     plt.show()
199
200
201 # Example usage:
202 if __name__ == "__main__":
203     particles = np.random.uniform(-5, 5, (10,3))
204
205     sd = SteepestDescent(particles, n_particles=10, box_length=10)
206     final_positions, final_energy = sd.minimize()
207     print("Final positions SD:\n", final_positions)
208     print("Final energy:", final_energy)
209     sd.energy_plot()
210     sd.final_positions_plot()

```

```
211  
212     cg = ConjugateGradient(particles, n_particles=10, box_length=10)  
213     final_positions, final_energy = cg.minimize()  
214     print("Final positions CG:\n", final_positions)  
215     print("Final energy:", final_energy)  
216     cg.energy_plot()  
217     cg.final_positions_plot()  
218
```