Molecular Conformation and Numerical Optimization

Breindy Berger (With Pranav and Mekhluli)

Stern College for Women/The Katz School

Numerical Methods

Professor Gidea

5/06/2025

Project code can be found here.

Molecular Conformation and Numerical Optimization

The function of a protein follows its form: The knobs and creases of the molecular shapes enable the bindings and blockings that are integral to their roles. The forces that govern the **conformation**, or folding, of amino acids into proteins are due to bonds between individual atoms and to weaker intermolecular interactions between unbound atoms such as electrostatic and Van der Waals forces. For densely packed molecules such as proteins, the latter are especially important.

One current approach to predicting the conformations of the proteins is to find the minimum potential energy of the total configuration of amino acids. The Van der Waals forces are modeled by the Lennard-Jones potential

$$U(r) = \frac{1}{r^{12}} - \frac{2}{r^6},$$

where r denotes the distance between two atoms. Figure 13.7 shows the energy well that is defined by the potential. The force is attractive for distances $r > 1$, but turns strongly repulsive when atoms try to come closer than $r = 1$. For a cluster of atoms with positions $(x_1, y_1, z_1), \ldots, (x_n, y_n, z_n)$, the objective function to be minimized is the sum of the pairwise Lennard-Jones potentials

$$U = \sum_{i<j} \left(\frac{1}{r_{ij}^{12}} - \frac{2}{r_{ij}^{6}} \right)$$

over all pairs of atoms, where

$$r_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$$

denotes the distance between atoms i and j. The variables in the optimization problem are the rectangular coordinates of the atoms.

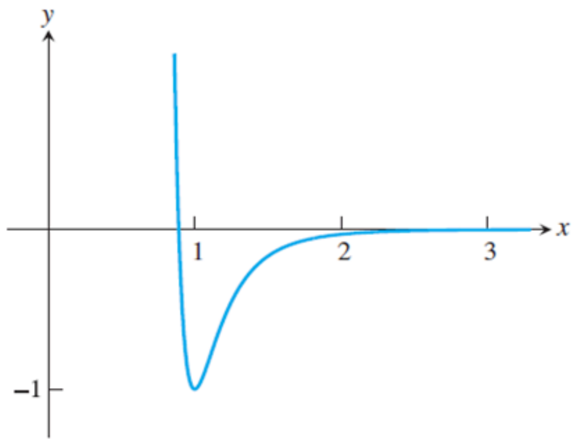


Figure 13.7 The Lennard-Jones potential $U(r) = r^{-12} - 2r^{-6}$. The energy minimum is -1, achieved at $r = 1$.

There are translational and rotational symmetries to consider: The total energy does not change if the cluster is moved in a straight line or rotated. To deal with the symmetries, we will limit the possible configurations by fixing the first atom at the origin $v_1 = (0, 0, 0)$ and requiring the second atom to lie on the z-axis at $v_2 = (0, 0, z_2)$. The remaining position variables $(x_3, y_3, z_3), \ldots, (x_n, y_n, z_n)$ are left to be arranged in a configuration that minimizes the potential energy U.

With the help of Figure 13.7, it is simple to arrange four or fewer atoms at the lowest possible Lennard-Jones energy. Note that the minimum of the single potential has the value -1 at $r = 1$. Thus, two atoms can sit exactly one unit from another, so that the energy is exactly at the bottom of the trough. Three atoms can sit in a triangle whose side is the same common distance, and a fourth atom can be placed at the same distance from the three vertices, say, above the triangle, creating an equilateral tetrahedron. The total energy U for the $n = 2$, 3, and 4 cases is -1 times the number of interactions, or -1, -3, and -6, respectively.

Placement of the fifth atom, however, is not so obvious. There is no point equidistant from the tetrahedron vertices of the $n = 4$ case, and a new technique is needed—numerical optimization.

Project activities

1. Write a function file that returns the potential energy. Apply Nelder–Mead to find the minimal energy for $n = 5$. Try several initial guesses until you are convinced you have the absolute minimum. How many steps are required?

2. Plot the five atoms in the minimum energy configuration as circles, and connect all circles with line segments to view the conformed molecule.

3. Extend the function in Step 1 so that it returns f and the gradient vector ∇f. Apply different versions of the Gradient Descent method for the $n = 5$ case. Find the minimum energy as before.

4. Apply the previous methods to $n = 6$. Rank the methods according to reliability and efficiency.

5. Determine and plot minimum-energy conformations for larger n. Information on minimum-energy Lennard-Jones clusters for n up to several hundred is posted at several Internet sites, so your answers can be readily checked.

I. Abstract/Introduction[1]

The folding of amino acids into proteins determines its structure and function. In turn, the folding is determined by bonds between individual atoms as well as intermolecular forces such as electrostatic and Van der Waals forces. One can predict protein conformation by finding the minimum potential energy of the the total configuration of amino acids using the Lennard Jones potential:

$$U(r) \;=\; 1/r^{12} \;-\; 1/r^{6}$$

where r is the distance between atoms. The force is attractive when r is less than 1 and repulsive when r >= 1. To find the minimum potential energy of a group of atoms, we need to minimize the sum of the pairwise Lennard-Jones potentials:

$$U \;=\; \sum_{i<j}\left(\frac{1}{r_{ij}^{12}} \;-\; \frac{2}{r_{ij}^{6}}\right)$$

where r_{ij} is the pairwise distance between atoms i and j. The variables in the equations are the (x, y, z) coordinates of each atom in three dimensional space.

Arranging 3 or 4 atoms in space to minimize potential energy is easy: when atoms sit exactly one unit from each other, potential energy is negative one times the number of interactions. However, placing five or more atoms is more complicated. This project attempts to find configurations of 5 or more atoms that will minimize total potential energy using numerical methods. Specifically, it utilizes the Nelder-Mead and Gradient Descent methods. To do this, we wrote a function that returns the total potential energy and an atom configuration of a given number of atoms, as well as a gradient function that provides a gradient vector of the potential energy function. We then applied a version of Nelder-Mead and two versions of Gradient-Descent to solve the minimization problem for various values of n. Moreover, the total energy remains constant if the cluster is moved in a straight line or rotated. To avoid these translational and rotational symmetries, this project held constant p1 at (0,0,0) and p2 on the z-axis. The rest between 3 and n were arranged in a configuration to minimize the potential energy U. To do this, we collapsed the three dimensional point vectors into one dimension so that we could minimize only the changing values. Then, we rebuilt the point vectors with p1 and p2(x,y) included.

[1] Based on project notes

II. Nelder-Mead Method[2]

The Nelder-Mead method is a derivative-free minimization technique that works for functions of several variables. It is used in machine learning for parameter selection. One drawback of this method is that it may converge to a local minimum instead of the global minimum.

The method initializes a simplex of n+1 vertices in n dimensions. Then, it reshapes the simplex one vertex at a time in the hope of converging at a local maximum or minimum. The steps of the algorithm are as follows: first, the vertices are sorted from worst to best. Then, we compute the centroid of the best face of the simplex and reflect the worst point with respect to the centroid with a factor of some reflection rate. If the new reflected point lies between the new worst vertex and the best vertex, then we replace the old worst vertex with the new reflected vertex and begin again. If the new vertex is better than the old best, we attempt to expand in the same direction, and we replace the worst vertex with the better of the first reflection or the second reflection. If the reflected point remains the worst point, we contract. If the reflected point is even worse than the worst point we began with, we perform a shrink contraction, moving away from the worst points.

Our implementation of the Nelder-Mead method attempted to minimize the potential energy function, which takes an array of 3 dimensional points. The algorithm takes an initial guess of size n-2 and constructs a flat array of parameters that change, of size 1 + 3(n-2). Then, it initializes a simplex of the number of parameters +1 vertices in the number of parameters dimensions. The objective function is wrapped so that it can evaluate the potential energy of the free points and the fixed points. Then the function iterates until a set number of iterations or until it converges based on a given tolerance level. At each iteration, the function evaluates all the points, sorts them according to their value, and then applies the Nelder-Mead operations – reflection, expansion, contraction, and shrinking – depending on how the reflected point relates to the simplex. The resulting vector is then changed into a set of optimized 3D points, and the function returns the points, the function values, the minimum potential energy, and the number of iterations. Below is the results for one running of the Nelder-Mead optimization with 5 atoms:

[2] Based on class notes

Final simplex vertices:

```
 [[ 0.          0.          0.        ]
 [ 0.          0.          0.99788141]
 [ 0.05230386  0.86469299  0.50263118]
 [ 0.58997435  0.73490588 -0.32799862]
 [ 0.833069    0.23763733  0.50238524]]
```

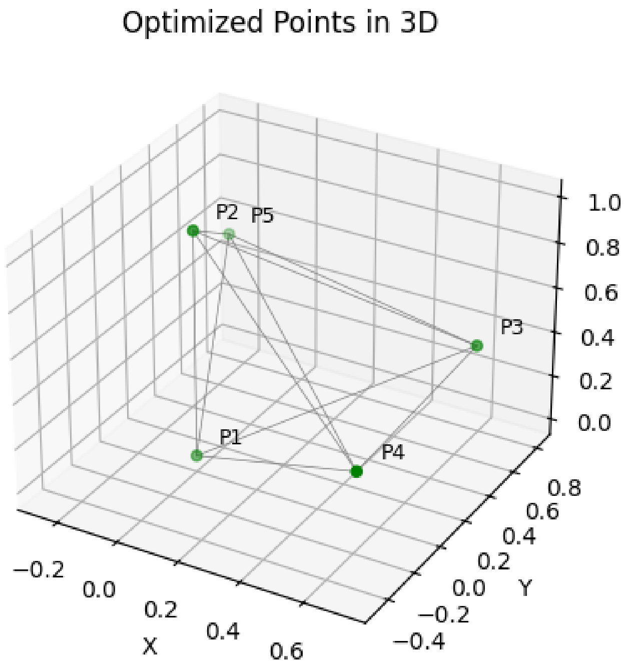
Function values at vertices:

```
[-9.10385147 -9.10385141 -9.10385127
-9.10385115 -9.10385108 -9.10385094
-9.10385081 -9.10385058 -9.10385057
-9.10385053 -9.10385049]
```

potential energy at minimum:

-9.103851466510415

Number of steps: 1357



III. Gradient-Descent Method

The Gradient-Descent method finds the minimum faster. It works by computing the gradient vector, which is orthogonal to the level sets. The gradient thus represents the direction of the steepest increase, and the negative gradient is the direction of the steepest descent. At a minimum point, maximum point, or a point of inflection, the gradient is equal to zero.

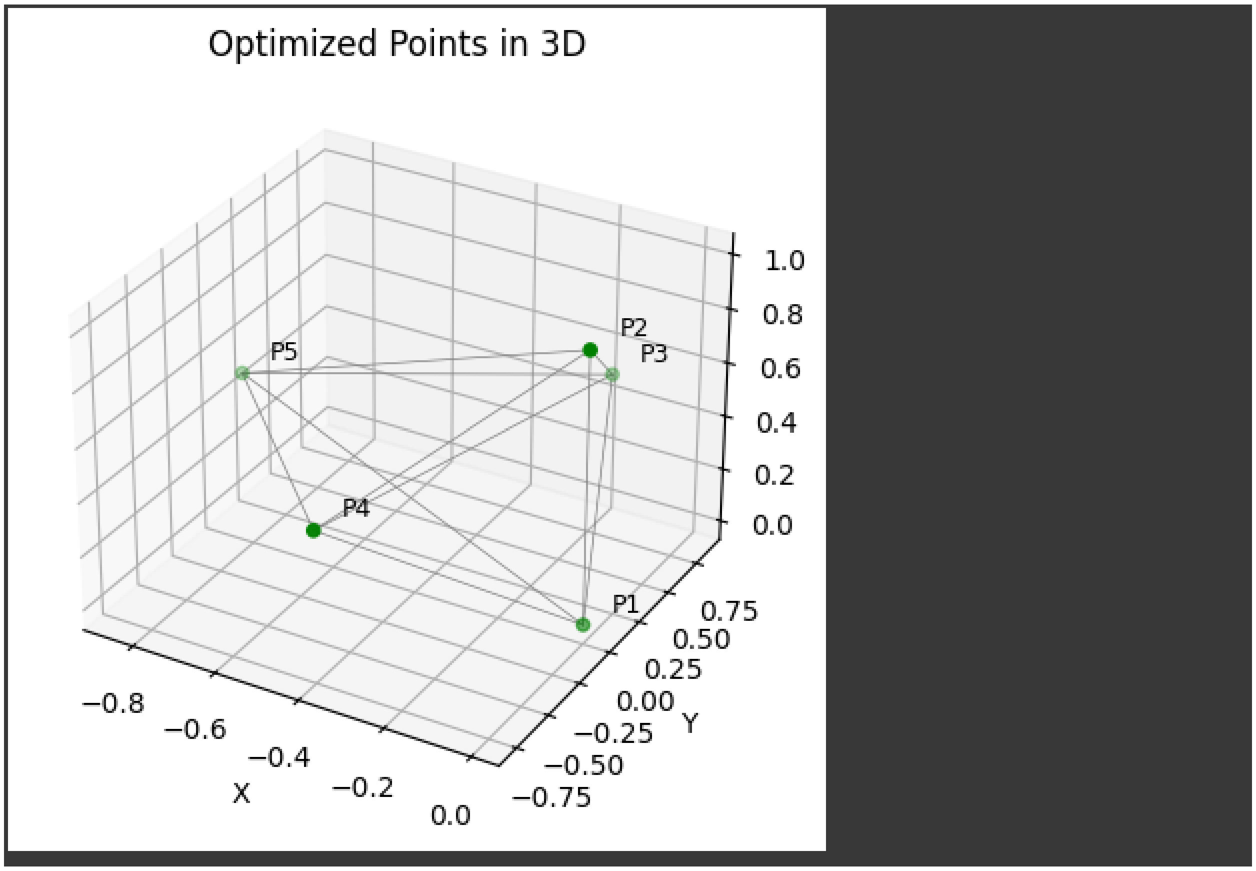
The gradient descent method is an iterative process that starts at an initial point. Then, each ensuing point is chosen by multiplying the gradient at the first point by some learning rate and subtracting it from the original point. This continues until the answer is close enough, as determined by an established tolerance level.

Gradient descent can be modified to include backtracking. It begins with a very large eta. It also defines other hyperparameters alpha and beta. Then, while $f(x - \eta\nabla f(x)) > f(x) - \alpha\eta||\nabla f(x)||2$, we change η to $\beta\eta$. This allows us to decrease the step size when necessary.

Our implementation takes a gradient function, an initial guess, a learning rate, a maximum number of iterations, and a tolerance for convergence. The algorithm iteratively updates the parameter vector by stepping in the opposite direction of the gradient, scaled by the learning rate. After each update, it recalculates the gradient and checks if its norm is below the convergence threshold. If the gradient norm becomes smaller than the tolerance level before reaching the maximum number of steps, the method successfully converges. Otherwise, it returns -1 as the step count to indicate failure to converge. The function returns the final estimate of the minimum, the number of steps taken, and the final gradient norm.

The backtracking function works similarly to the gradient descent function, but it adds a step where it recomputes the step size at every iteration based on the rules above. This helps prevent overly large steps when that will cause us to overshoot where the local minimum is. Below are the results from our implementation of Gradient Descent when n=5:

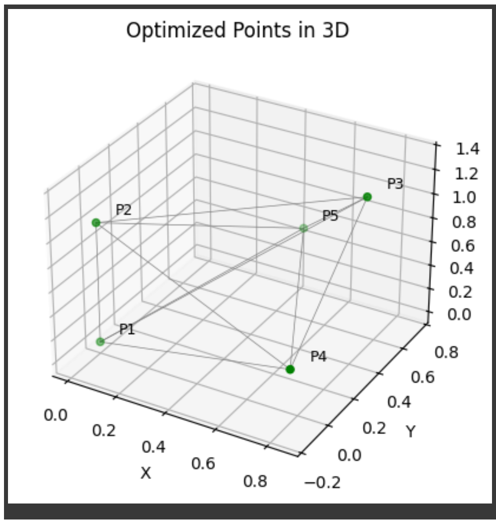
Simple Gradient Descent:
Initial energy: 45.35161913489518
Initial gradient norm: 1386.9785364813197
Final simplex vertices:
```
 [ 1.00147347 -0.17548036  0.84515064  0.50074512 -0.39725335 -0.7663307
  0.50074512 -0.85917177  0.11823972  0.50074635]
```
Potential energy at minimum: -9.10385238706892
Number of steps: 5689
Norm of gradient: 0.000993473737314662983



Gradient Descent with Backtracking:
Final simplex vertices:

```
 [ 0.99790033  0.85599402  0.39445944  1.32577648  0.8515571  -0.15890101
  0.49537118  0.43240181  0.750617    0.49539349]
```

Potential energy at minimum: -9.103852397746724
Number of steps: 176
Norm of gradient: 0.0007803869611007267



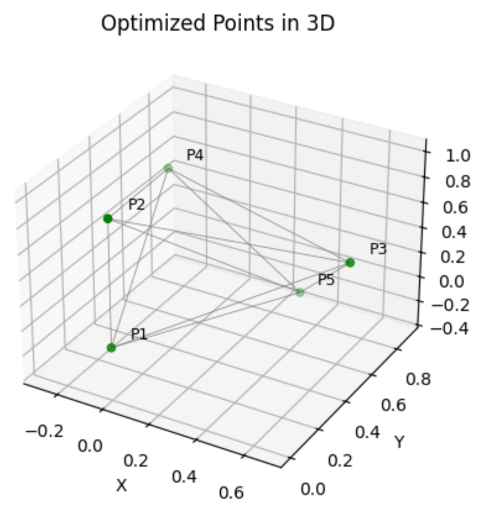
Results for N=6:
Nelder-Mead:
Final simplex vertices:

```
 [[ 0.          0.          0.        ]
 [ 0.          0.          0.99797151]
 [ 0.68099272  0.53542583  0.50258224]
 [-0.27916685  0.82000813  0.50268793]
 [ 0.26781238  0.90369963 -0.32774348]]
```

Function values at vertices:

```
 [-9.10385172 -9.1038515  -9.10385129 -9.10385126
-9.1038512  -9.10385114
 -9.10385114 -9.10385111 -9.10385087 -9.10385086
-9.10385084]
```

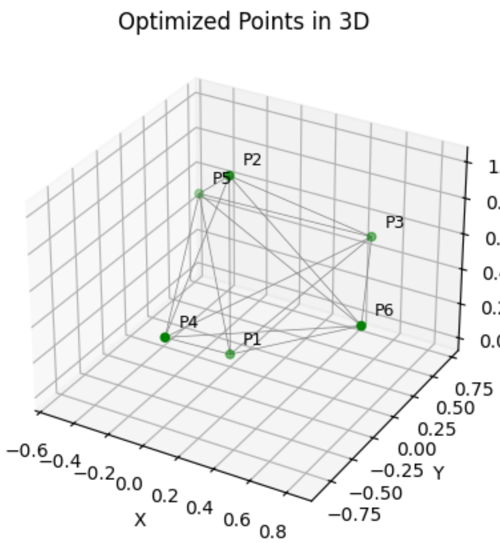
potential energy at minimum: -9.103851721282108
Number of steps: 551



Gradient-Descent:
Initial energy: 44.07103459057907
Initial gradient norm: 1386.98091793431
Final simplex vertices:

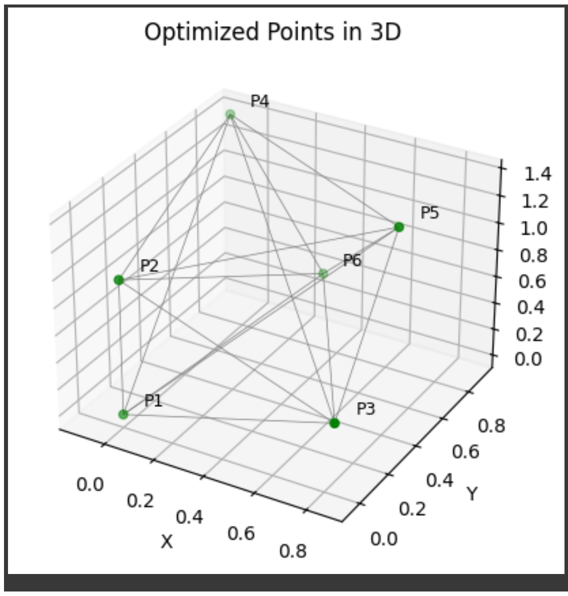
```
 [ 1.00805282  0.46965962  0.72228089  0.50402165  0.07658453 -0.85429192
```

```
  0.50402204 -0.5287991   0.67531576  0.50402204
0.83674888 -0.20523437
  0.50402165]
Potential energy at minimum: -12.302927476873355
Number of steps: 5298
Norm of gradient: 0.000996777021810333
```



gradient-descent-with-backtracking:

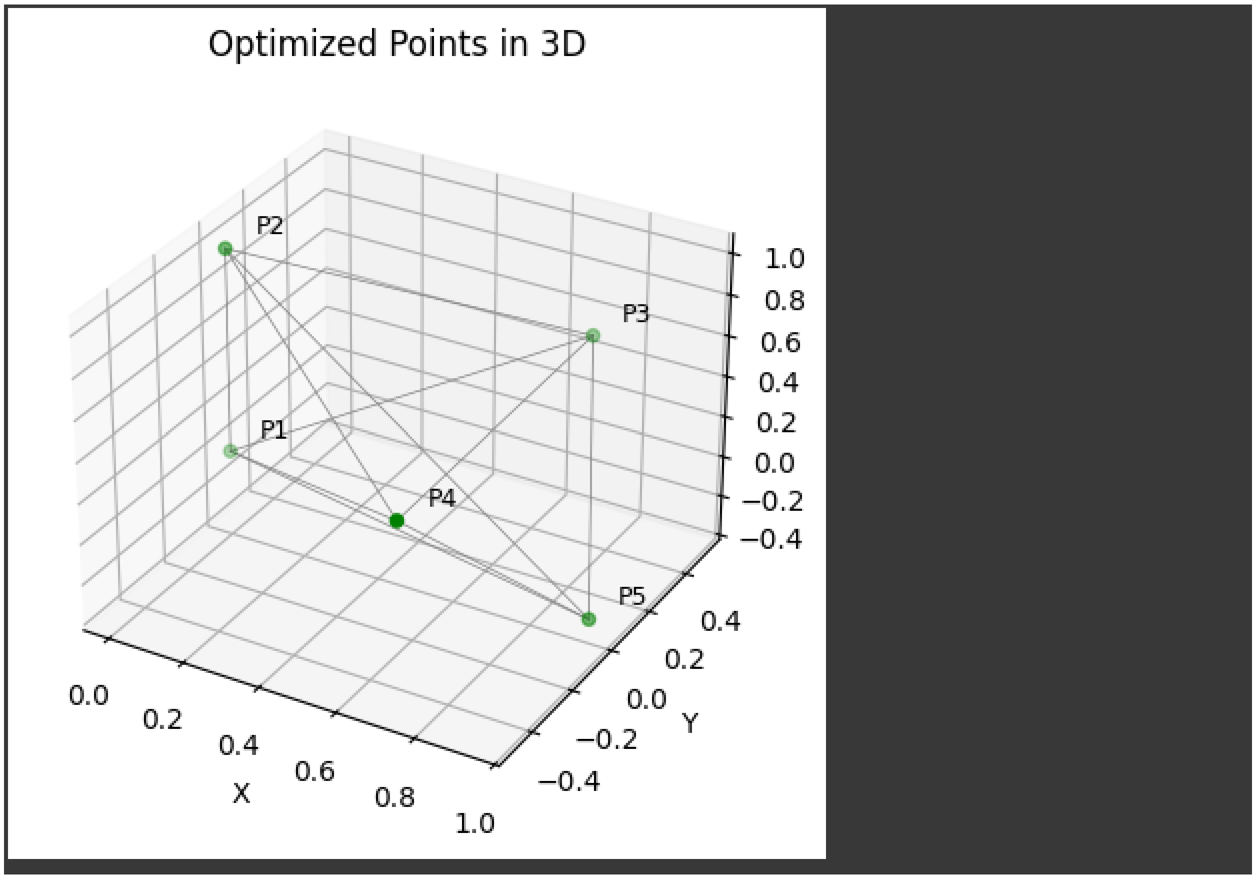
```
Final simplex vertices:
 [ 0.99485398  0.86473631 -0.05011962  0.49883758
-0.11544831  0.91812616
  1.36013405  0.79473431  0.50659699  1.32358357
0.34095558  0.7994337
  0.48415393]
Potential energy at minimum: -12.302927453761779
Number of steps: 218
Norm of gradient: 0.000982183786030248
```



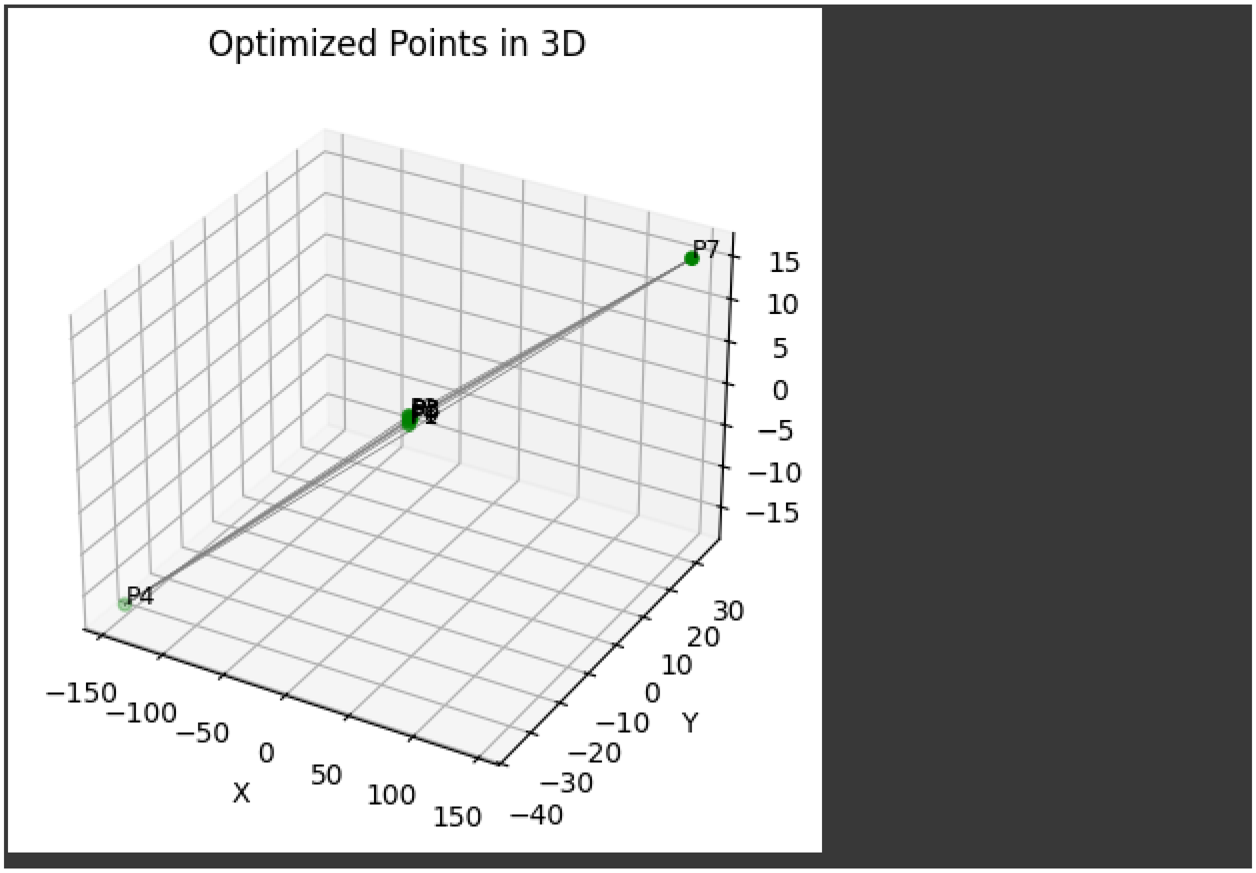
For various n:

For n = 7

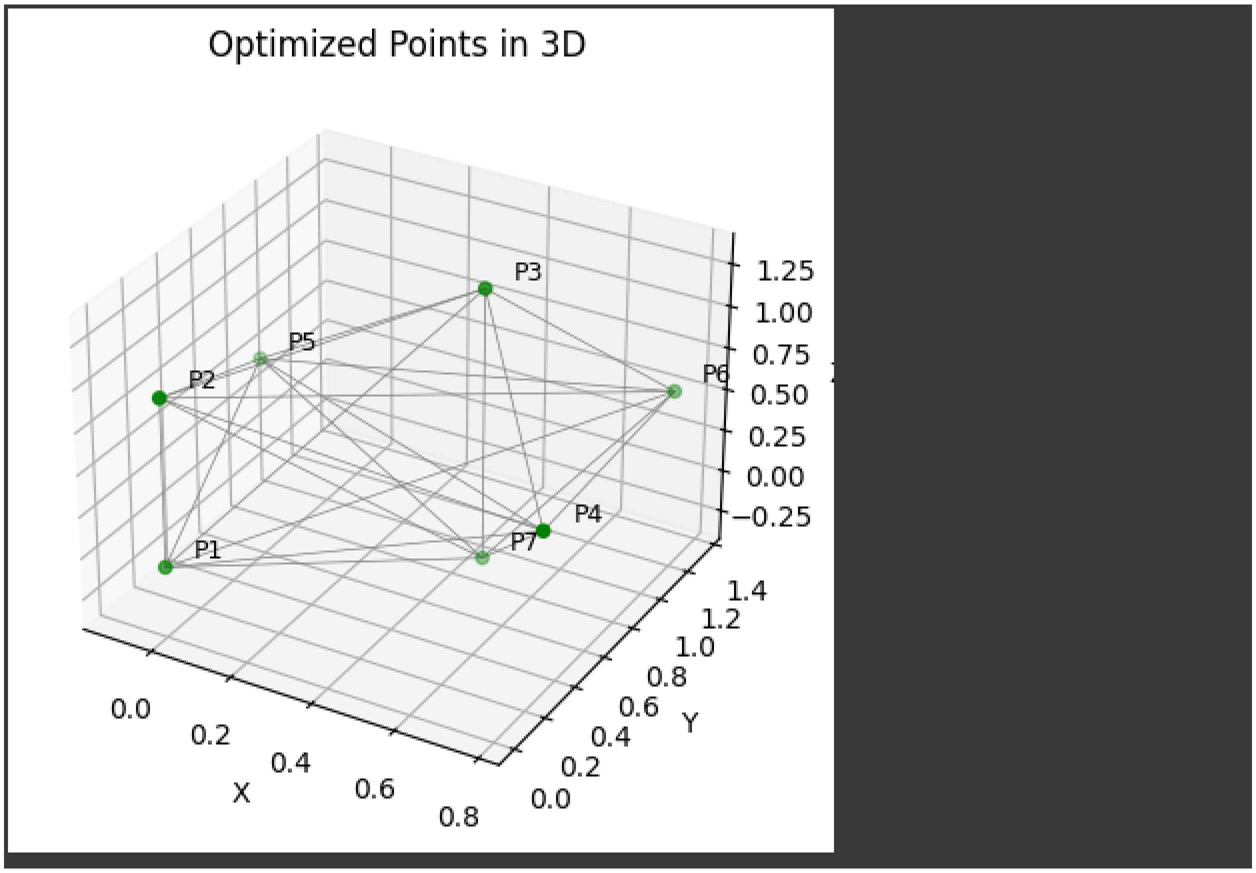
```
Nelder-Mead:
Final simplex vertices:
 [[ 0.          0.          0.        ]
 [ 0.          0.          0.99803567]
 [ 0.69668329  0.51475151  0.5024951 ]
 [ 0.71666994 -0.4865172   0.50252253]
 [ 0.94235352  0.01883702 -0.32776337]]
Function values at vertices:
 [-9.1038511  -9.10385084 -9.10385081 -9.10385062 -9.10385061 -9.10385051
 -9.10385045 -9.10385043 -9.1038503  -9.10385029 -9.1038501 ]
potential energy at minimum: -9.103851096728885
Number of steps: 1502
```



```
Gradient-Descent:
Initial energy: 5981.73462063458
Initial gradient norm: 211095.98139828496
Final simplex vertices:
 [   1.00147343    0.5189683     0.69485879    0.5007463  -144.08583284
 -35.44721201  -16.62952283   -0.478671      0.71829457    0.50074508
   0.82462127   -0.25509367    0.50074508  144.94634435   34.18507999
  15.30730641]
Potential energy at minimum: -9.103852387213053
Number of steps: 2632
Norm of gradient: 0.0009910015910541655
```

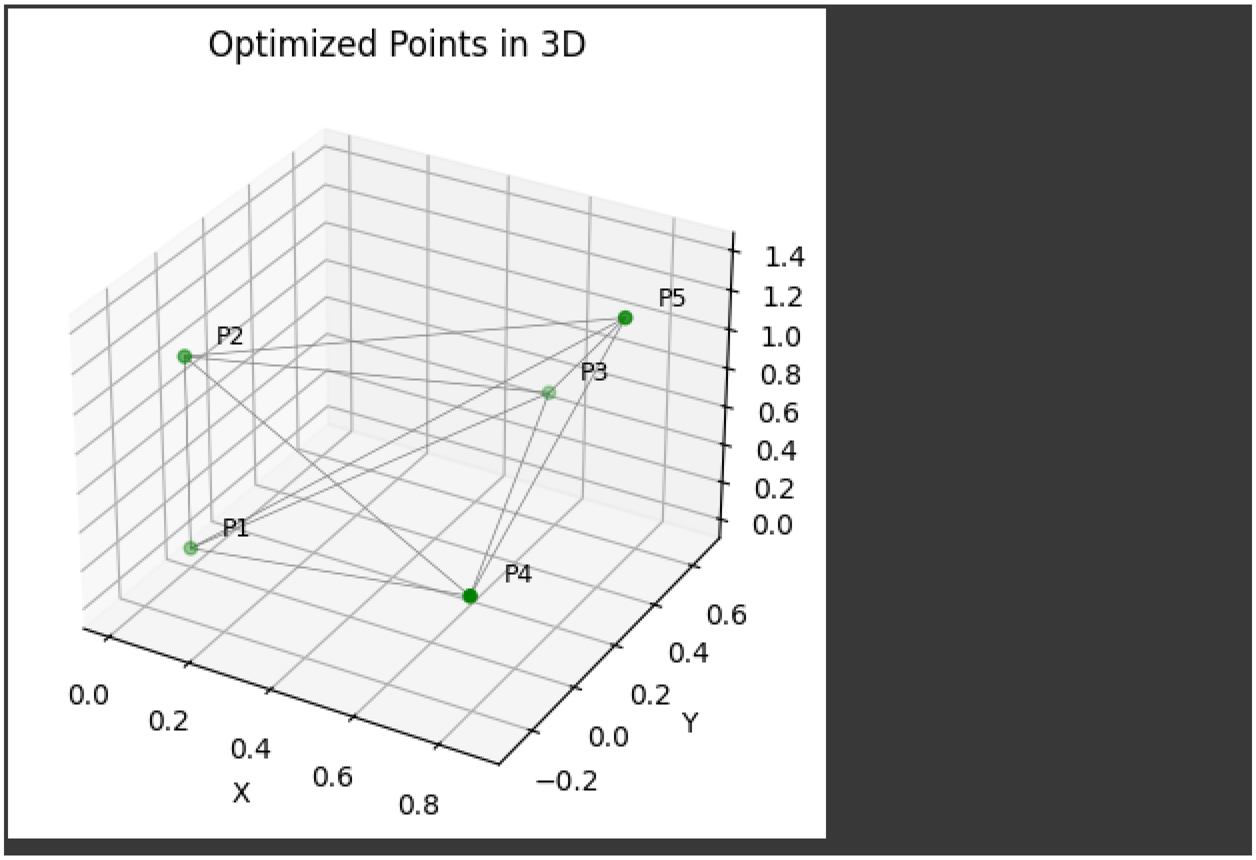


```
gradient-descent-with-backtracking:
Final simplex vertices:
 [ 1.00143683  0.46609545  0.83061742  1.31086783  0.78312604  0.35083734
  0.50069495 -0.10859707  0.8512226   0.50069436  0.75414323  1.34393837
  0.50065876  0.46607538  0.83057916 -0.30950925]
Potential energy at minimum: -16.50538412669684
Number of steps: 120
Norm of gradient: 0.0008612876047898031
```

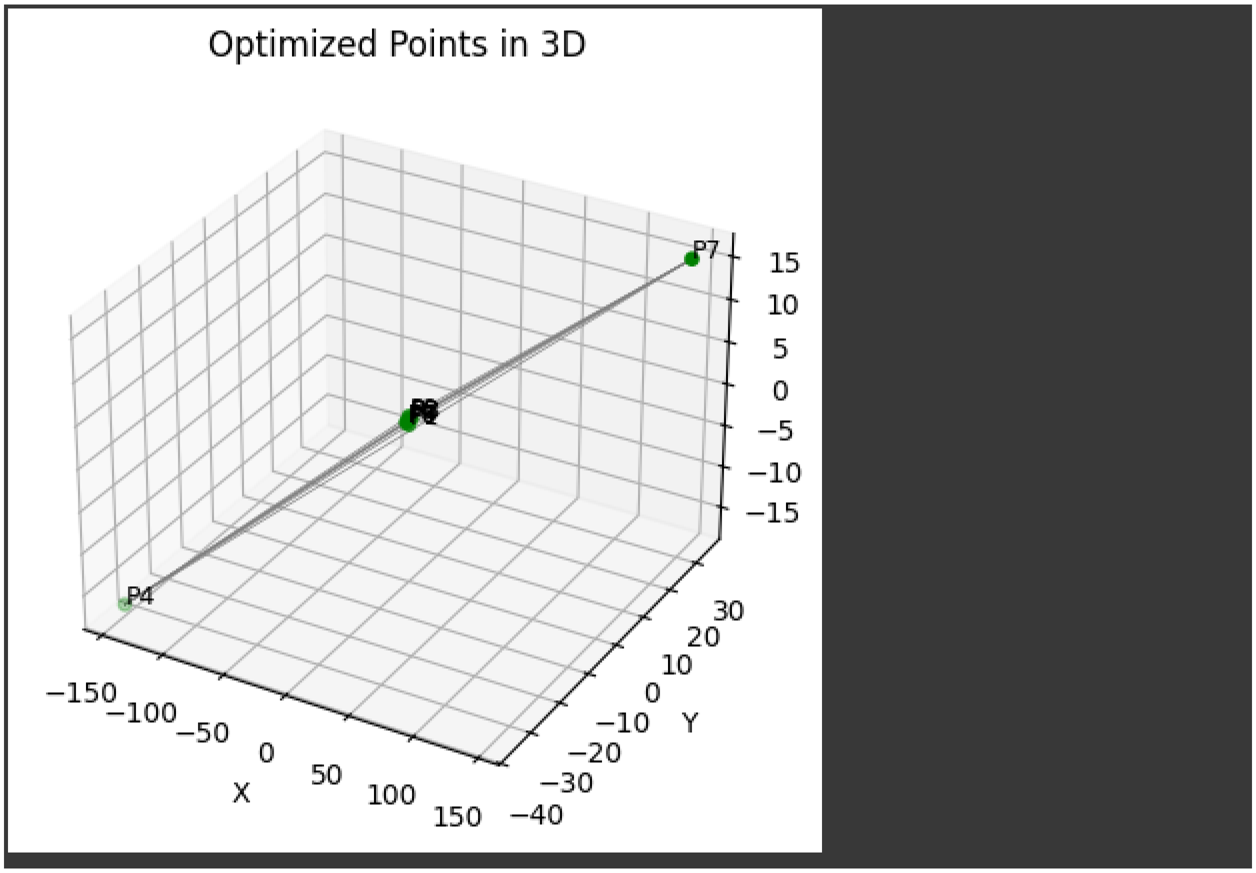


For n = 8

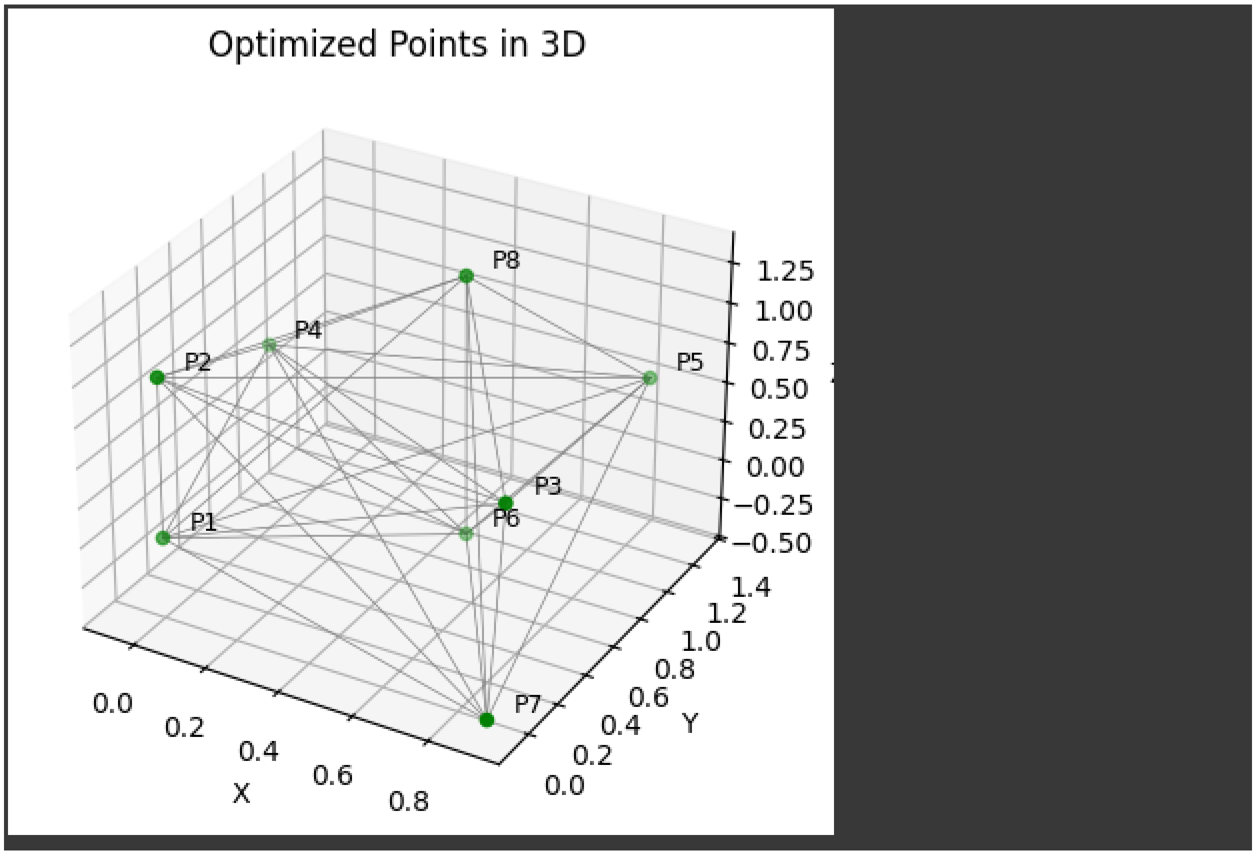
```
Nelder-Mead:
Final simplex vertices:
 [[ 0.          0.          0.        ]
 [ 0.          0.          1.0024434 ]
 [ 0.54314693  0.67494403  0.50207684]
 [ 0.83477946 -0.28139776  0.59896332]
 [ 0.8761102   0.34177864  1.38835057]]
Function values at vertices:
 [-8.93310321 -8.93310318 -8.93310311 -8.93310272 -8.93310268 -8.93310265
 -8.93310245 -8.93310244 -8.93310237 -8.93310226 -8.93310223]
potential energy at minimum: -8.93310321495112
Number of steps: 2489
```



```
Gradient-Descent:
Initial energy: 6001.482636044272
Initial gradient norm: 211097.09702376058
Final simplex vertices:
[   1.0080529     0.21489619    0.83036103    0.50402209 -144.08768451
  -35.44655106  -16.62787847   -0.71915113    0.47443799    0.5040217
    0.27116352   -0.81372609    0.50402209  144.94573081   34.18516687
   15.30763059   -0.68503205   -0.52249483    0.5040217 ]
Potential energy at minimum: -12.302927477161996
Number of steps: 1534
Norm of gradient: 0.000994063959163936
```



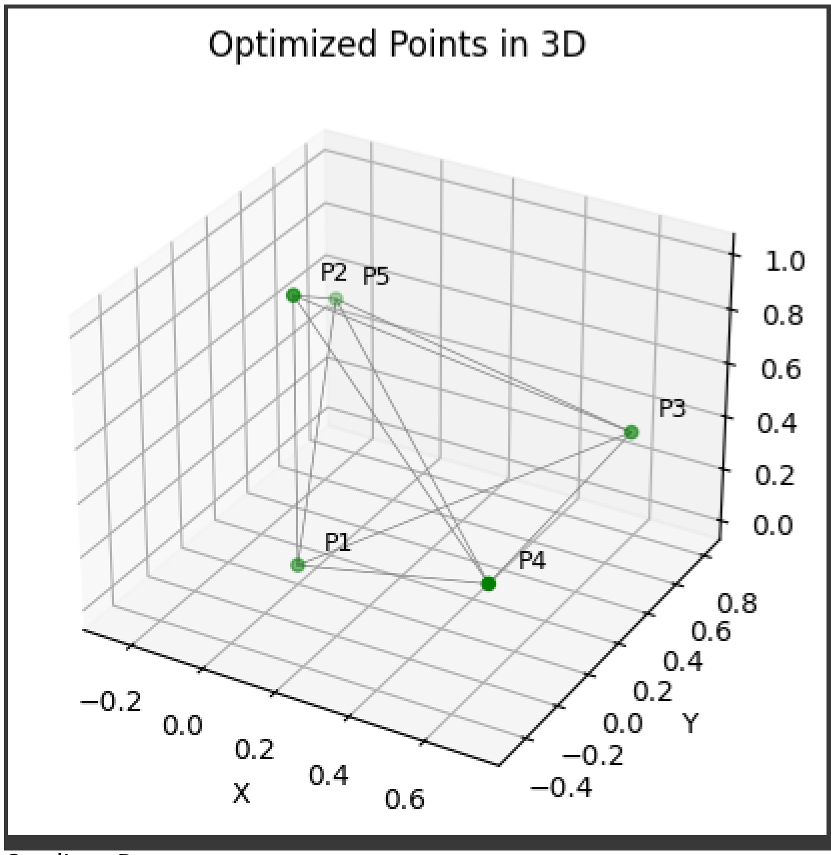
```
gradient-descent-with-backtracking:
Final simplex vertices:
 [ 0.99969484  0.79738682  0.31844207  0.50625044 -0.07017017  0.85472729
  0.49725134  0.81413379  1.30856114  0.49862041  0.5034959   0.80942848
 -0.30993602  0.91300745 -0.09552781 -0.38720818  0.49668241  0.81367405
  1.31101886]
Potential energy at minimum: -19.821489115808404
Number of steps: 203
Norm of gradient: 0.0009469926079402057
```



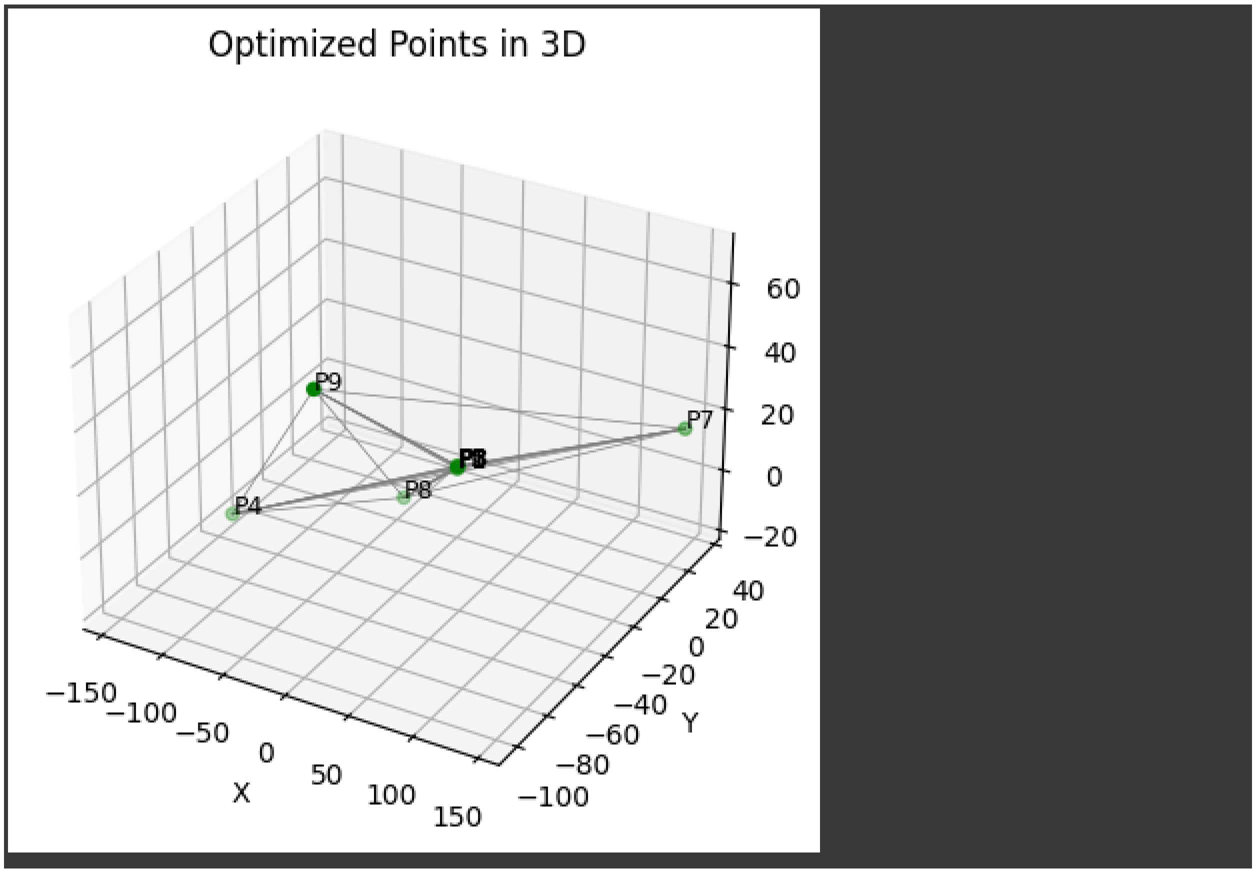
For n = 9

Nelder-Mead:

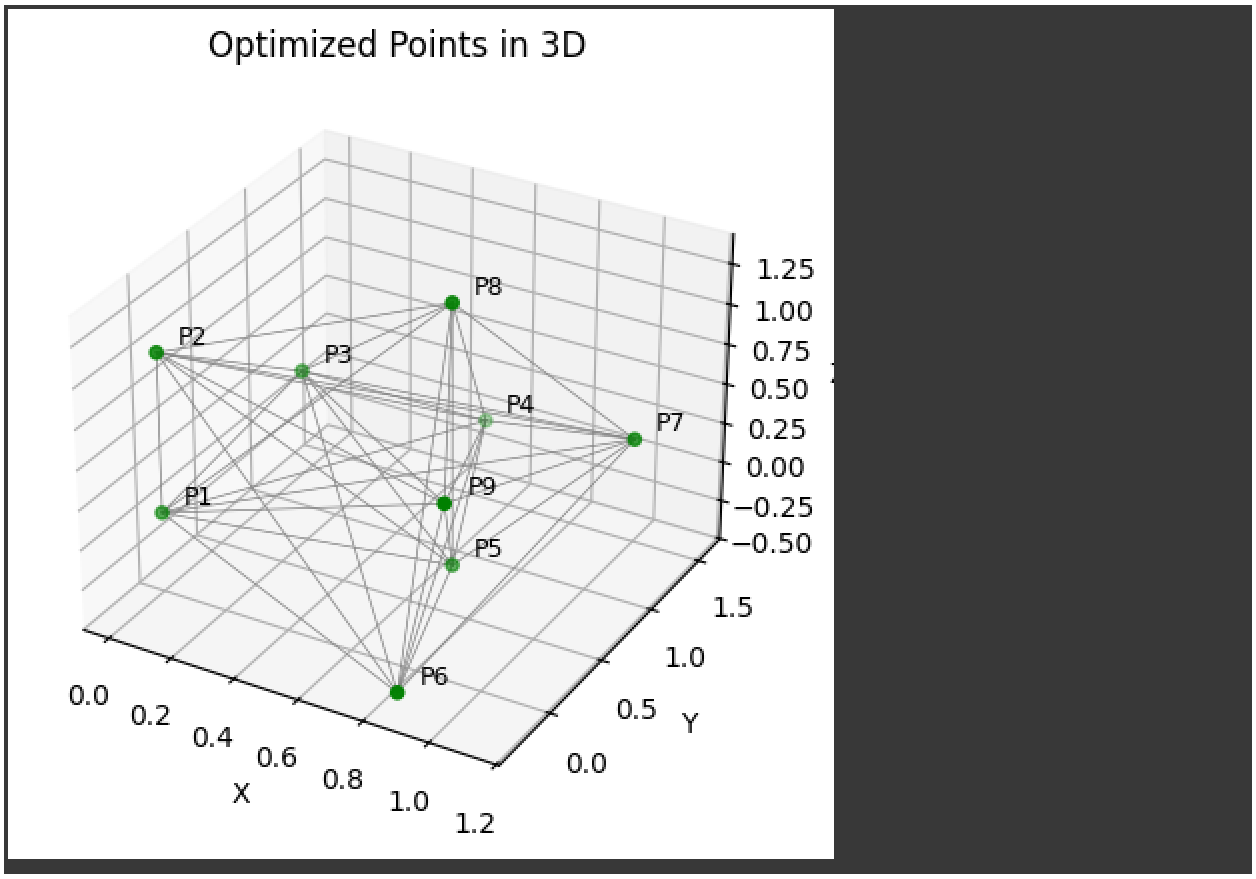
```
Final simplex vertices:
 [[ 0.          0.          0.        ]
 [ 0.          0.          1.00142679]
 [ 0.68944552  0.52621481  0.50068527]
 [ 0.72332049 -0.47106842  0.50063338]
 [-0.26359176  0.82189835  0.50079114]]
Function values at vertices:
 [-9.1038517  -9.1038515  -9.10385149 -9.10385136 -9.10385123 -9.10385121
 -9.10385112 -9.10385105 -9.10385081 -9.10385074 -9.10385071]
potential energy at minimum: -9.103851700747422
Number of steps: 1659
```



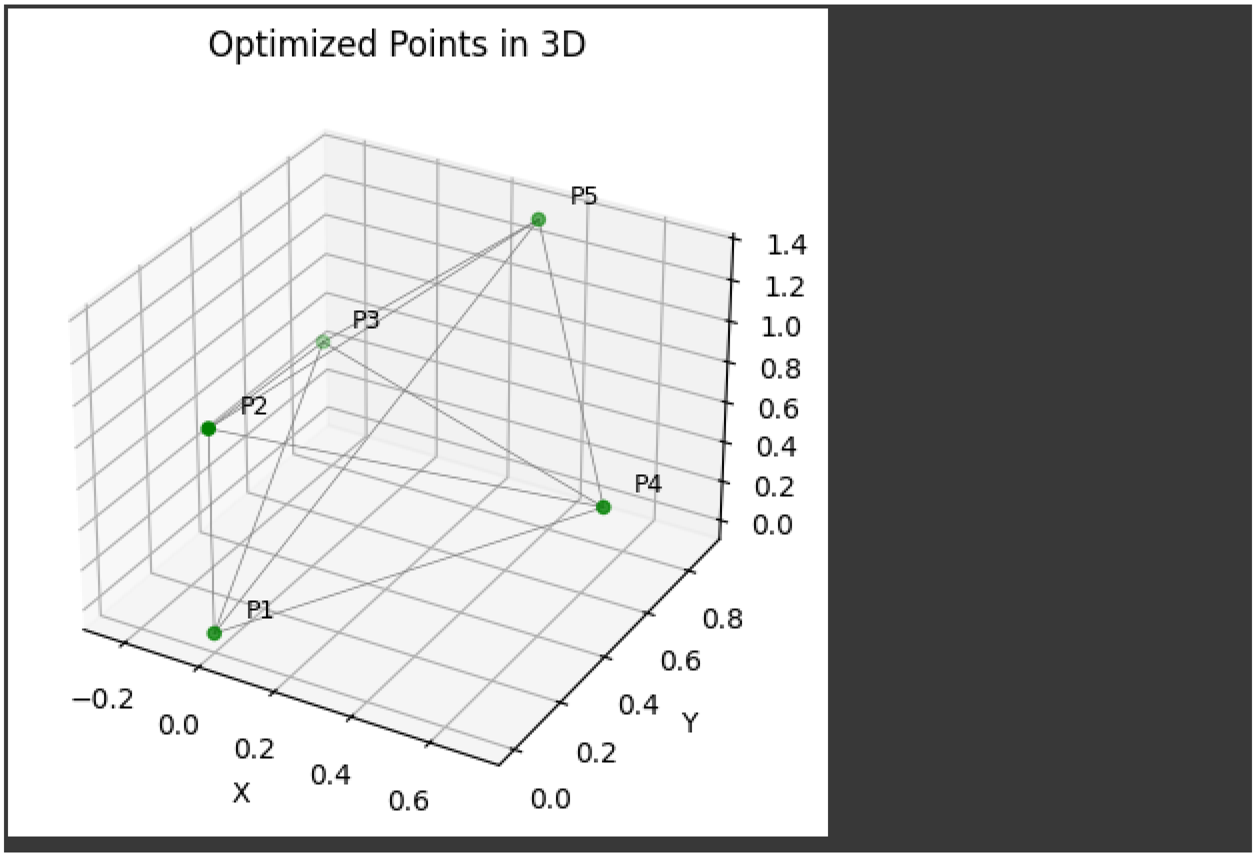
```
Gradient-Descent:
Initial energy: 12793.686607181391
Initial gradient norm: 249198.28093299523
Final simplex vertices:
 [   1.00147349    0.51902148    0.694819      0.50074637 -144.08785446
  -35.44641323  -16.62741377   -0.47861604    0.71833115    0.50074514
    0.82460167   -0.25515689    0.50074514  144.94415659   34.18547696
   15.30932088  -46.36679206    1.75128336  -15.81526029   12.96333397
 -101.75543897   69.14061739]
Potential energy at minimum: -9.103852387728418
Number of steps: 2634
Norm of gradient: 0.0009941839886725278
```



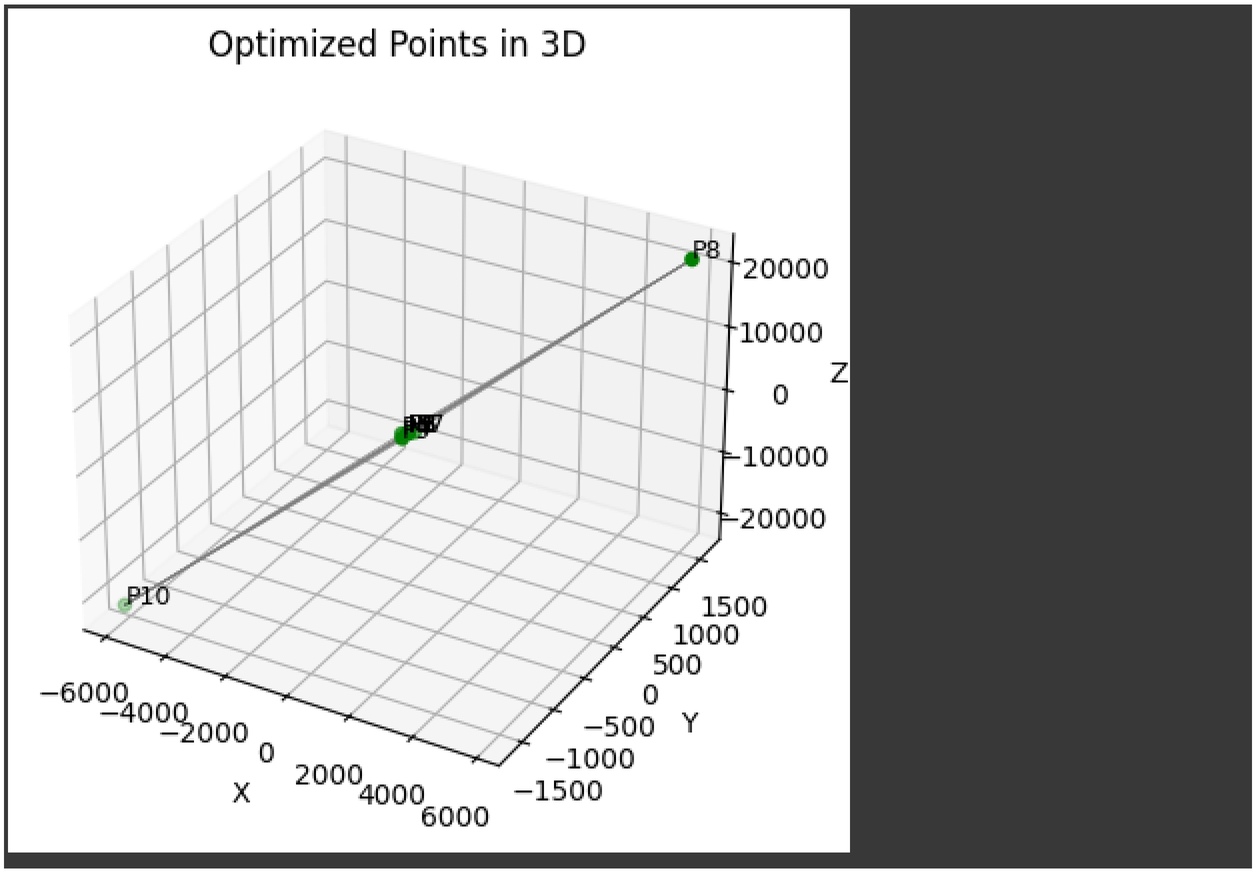
```
gradient-descent-with-backtracking:
Final simplex vertices:
 [ 1.00105926  0.15179937  0.84064458  0.49670681  0.49619002  1.58654097
 -0.05801514  0.69875313  0.64545206 -0.31059638  0.8573792  -0.33448175
 -0.38139012  1.12657714  1.05155845  0.49770454  0.6874748   0.66399372
  1.30953728  0.85185132  0.10245402  0.50841119]
Potential energy at minimum: -23.170759922976035
Number of steps: 217
Norm of gradient: 0.0009109155414098372
```



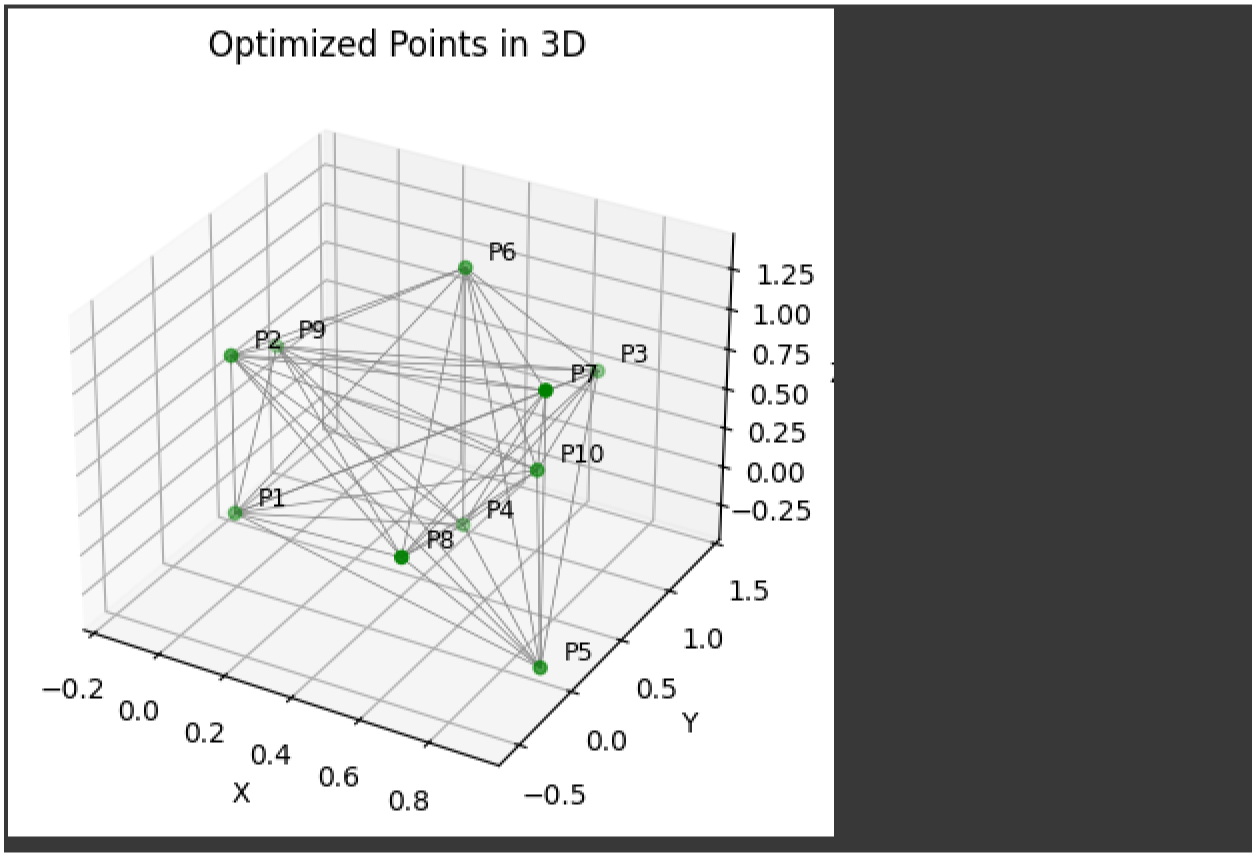
```
For n = 10
Nelder-Mead:
Final simplex vertices:
 [[ 0.          0.          0.        ]
 [ 0.          0.          0.99790427]
 [-0.24204857  0.83175926  0.49551786]
 [ 0.70430377  0.5044323   0.49535364]
 [ 0.3082655   0.89037868  1.32606686]]
Function values at vertices:
 [-9.10384992 -9.10384982 -9.10384974 -9.10384962 -9.10384932 -9.10384924
 -9.1038492  -9.10384911 -9.10384902 -9.10384899 -9.10384893]
potential energy at minimum: -9.103849923273563
Number of steps: 1413
```



```
Gradient-Descent:
Initial energy: 619020.6122258276
Initial gradient norm: 31232477.39639582
Final simplex vertices:
 [ 1.00147353e+00  5.18884281e-01  6.94921439e-01  5.00746401e-01
 -1.44089676e+02 -3.54458731e+01 -1.66265790e+01 -4.78757882e-01
  7.18236599e-01  5.00745172e-01  8.24651997e-01 -2.54994129e-01
  5.00745172e-01  1.44943370e+02  3.41855675e+01  1.53095303e+01
  5.84100222e+03  1.62374968e+03  2.12211149e+04  1.32086080e+01
 -1.01756207e+02  6.93465067e+01 -5.88836399e+03 -1.62243004e+03
 -2.12374147e+04]
Potential energy at minimum: -9.103852386929766
Number of steps: 2636
Norm of gradient: 0.0009959644041579977
```



```
gradient-descent-with-backtracking:
Final simplex vertices:
 [ 0.9888312   0.65856591  1.40421426  0.49434057  0.42398464  0.8462405
 -0.32159669  0.92149278 -0.03900483 -0.35256963  0.42402587  0.84627861
  1.3103306   0.92152926 -0.0389605   1.34132709  0.67734066 -0.54414198
  0.49440244 -0.15350104  0.8424962   0.49439146  0.76569289  0.42966363
  0.49437215]
Potential energy at minimum: -28.42253182969044
Number of steps: 436
Norm of gradient: 0.0009796894878484007
```



Based on this, Gradient Descent with backtracking works in significantly fewer steps than the other methods. In our implementation, Gradient Descent without backtracking did not always converge faster than Nelder-Mead. However, both gradient descent methods generally found smaller minimums for higher n, with Gradient Descent with Backtracking being the best.