

Lab 3: Inverse Kinematics

Lab Report

6.1 Method

The first internal function we implemented that will help to approach the final kinematic solver is to represent the displacement vector from the origin of the current frame to the target frame and the vector pointing along the axis of rotation from the current frame to the target frame given the transformation matrices of the target end effector and the current end effector with respect to the world frame. After obtaining the homogenous transformation matrices for both frames, we could then further calculate the magnitude of the angle of rotation between the current and target frames, as well as the distance between their origins in the unit of meters. On the ground of the distance, the angle between the target and current frame, as well as the joint angles, we could construct the validation function to check if the joints configuration outcome is a valid solution by checking the joint angles with the joint limit and comparing the distance/angle between the frames with the tolerances.

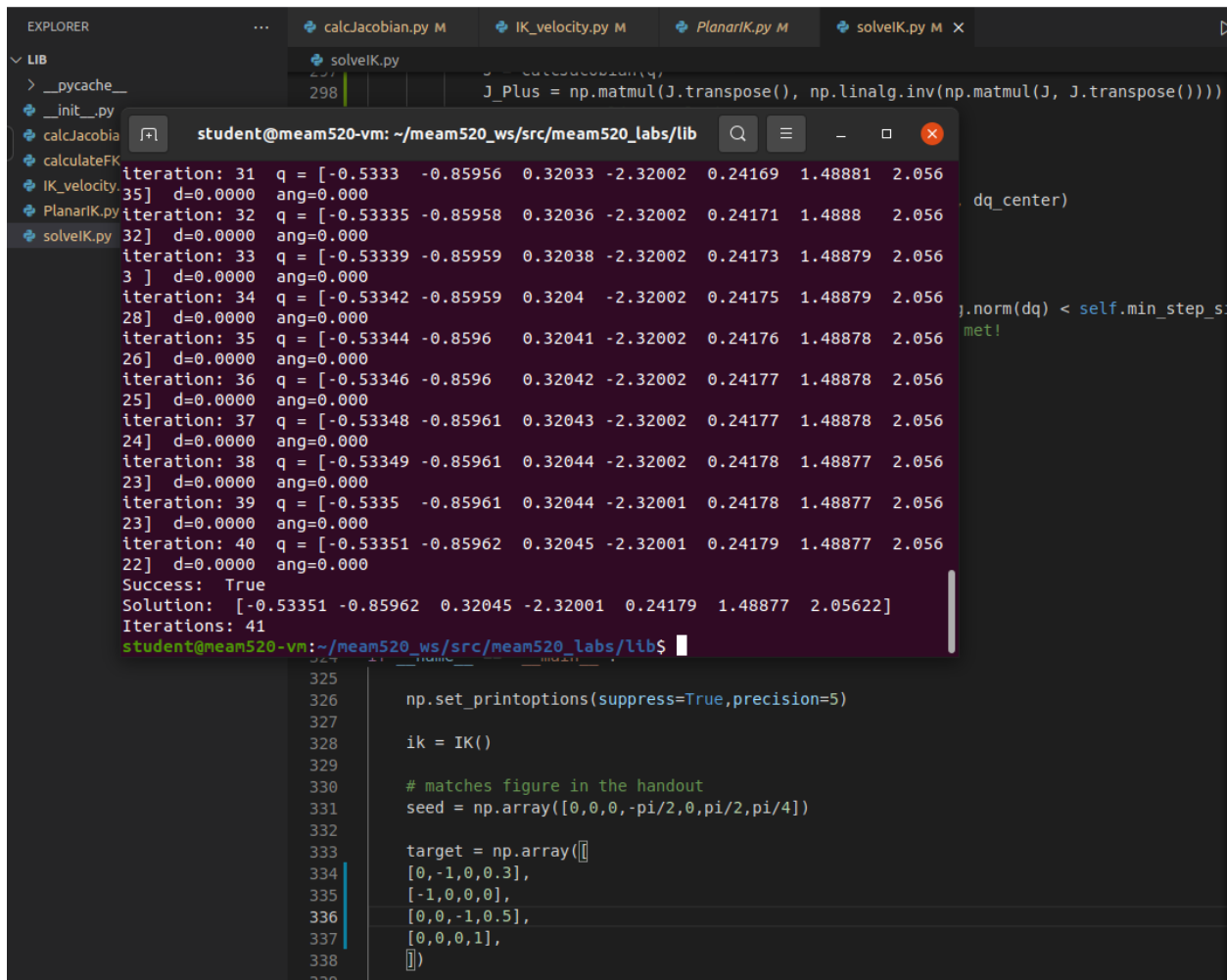
Since when approaching the position of the ‘desired’ configuration, the distance between origins of two frames and the vector pointing along the rotation between the current and target frame will gradually become zero, we could then use what we obtained from the first IK. displacement and axis function as well as the current joint position to get the desired joint velocity, dq . This ‘desired’ joint velocity is to fulfill task of reducing the error between the current and target end effector poses or called the primary task of our IK solver. Then we also computed again the joint velocity which will achieve the secondary task of our solver, which is to move the joints to the center of their motion ranges. The inputs of this task function are the joint configurations and the rate of how quick we want the center the joints.

With all the functions and outputs from above, we could finally construct our IK solver function. In this function, we still try to fulfill our primary task (reduce pose error) and the secondary task (center joints), but since we cannot compute a final result while achieving both goals simultaneously, we would like the arm to try its best to fulfill the secondary task while ensuring to achieve the primary task completely. In this function, we would like to put in the ‘target’ transformation matrix, which represents the transformation matrix from the desired end effector to the world frame, as well as the ‘seed’ which is our initial guess of the joint angles to start our optimizations. What we are expecting to obtain from the function is the set of joint angles, q , and the outcome of our validation function to justify if the solution is a success. If the function returns true, which means our optimization process is a success, the outcome, q , is regarded as the optimized solution to this IK problem; if not a success, q is the best guess to solve the problem based on our algorithm.

6.2 Evaluation

To evaluate the accuracy of our solution, we can first run the self-checking environment at the bottom of solveIK script. It provides one single testing condition for the solution with the ‘seed’ and the ‘target’ pose. Once we passed the single test, we could implement more complicated test in series by running the visualize.py script. It provides bunch of different end effector poses and we could go through each pose and see how our optimization process approaches the acceptable results with numbers of iterations. By observing the visualized result of how close the end effector frame is to the target frame, we should be able to decide if our optimization solution is acceptable upon passing all the tests. Finally, the test on Gradescope is another tool for us to conduct testing. If our solution code passed all the tests

above and we observed no obvious deviation between the current frame we generated and the target frame, we should be able to say that our solution is correct and free of bugs.



```
student@meam520-vm: ~/meam520_ws/src/meam520_labs/lib
iteration: 31 q = [-0.5333 -0.85956 0.32033 -2.32002 0.24169 1.48881 2.056
35] d=0.0000 ang=0.000
iteration: 32 q = [-0.53335 -0.85958 0.32036 -2.32002 0.24171 1.4888 2.056
32] d=0.0000 ang=0.000
iteration: 33 q = [-0.53339 -0.85959 0.32038 -2.32002 0.24173 1.48879 2.056
3 ] d=0.0000 ang=0.000
iteration: 34 q = [-0.53342 -0.85959 0.3204 -2.32002 0.24175 1.48879 2.056
28] d=0.0000 ang=0.000
iteration: 35 q = [-0.53344 -0.8596 0.32041 -2.32002 0.24176 1.48878 2.056
26] d=0.0000 ang=0.000
iteration: 36 q = [-0.53346 -0.8596 0.32042 -2.32002 0.24177 1.48878 2.056
25] d=0.0000 ang=0.000
iteration: 37 q = [-0.53348 -0.85961 0.32043 -2.32002 0.24177 1.48878 2.056
24] d=0.0000 ang=0.000
iteration: 38 q = [-0.53349 -0.85961 0.32044 -2.32002 0.24178 1.48877 2.056
23] d=0.0000 ang=0.000
iteration: 39 q = [-0.5335 -0.85961 0.32044 -2.32001 0.24178 1.48877 2.056
23] d=0.0000 ang=0.000
iteration: 40 q = [-0.53351 -0.85962 0.32045 -2.32001 0.24179 1.48877 2.056
22] d=0.0000 ang=0.000
Success: True
Solution: [-0.53351 -0.85962 0.32045 -2.32001 0.24179 1.48877 2.05622]
Iterations: 41
student@meam520-vm:~/meam520_ws/src/meam520_labs/lib$

325
326 np.set_printoptions(suppress=True,precision=5)
327
328 ik = IK()
329
330 # matches figure in the handout
331 seed = np.array([0,0,0,-pi/2,0,pi/2,pi/4])
332
333 target = np.array([
334 [0,-1,0,0.3],
335 [-1,0,0,0],
336 [0,0,-1,0.5],
337 [0,0,0,1],
338 ])
339
```

Figure 6.2.1 Self-checking test with solveIK.py

New Poses

POSE 1

End Effector Translation Vector from World Frame: $[0.5, 0, 0.5]$

End Effector Euler Angle with respect to World Frame: $[0, \pi/2, \pi/2]$

Result:

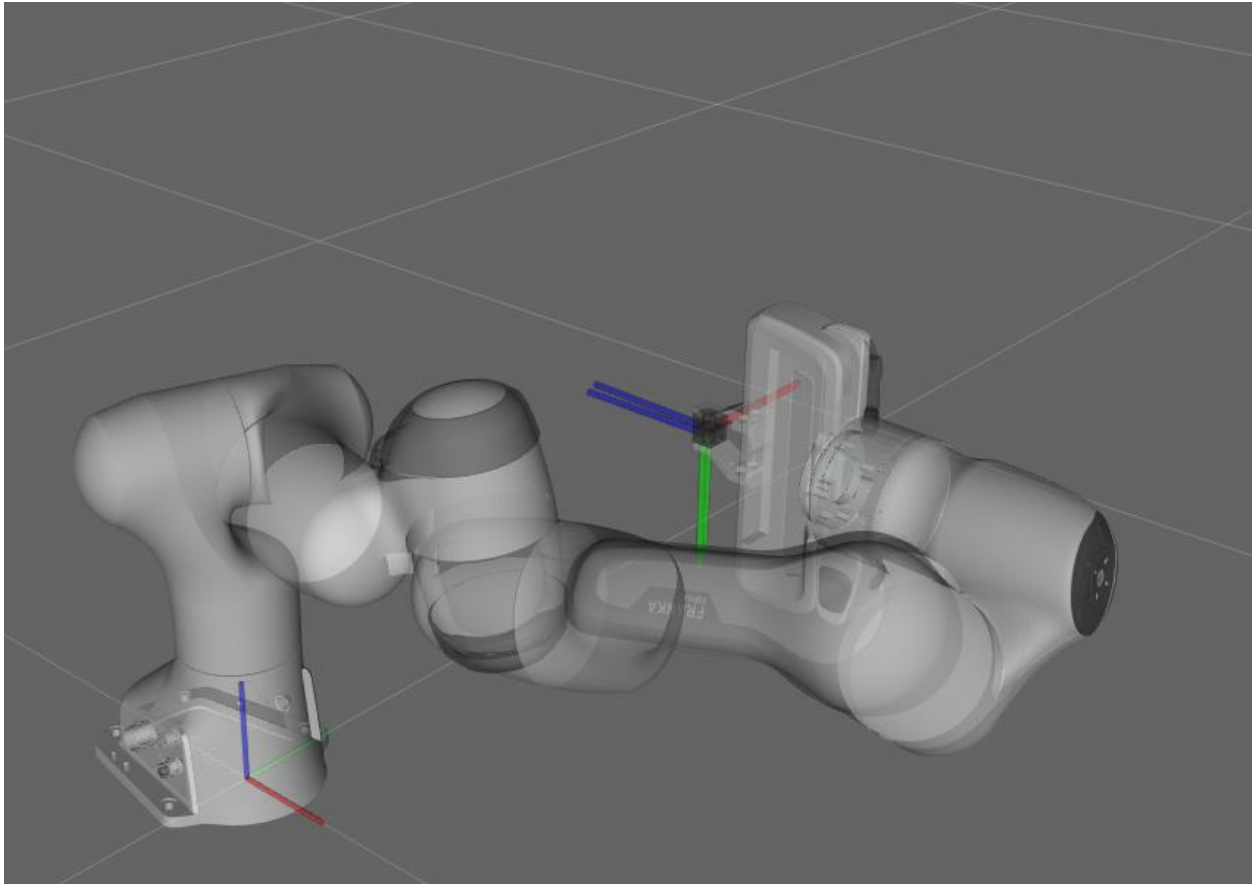


Figure 6.2.2 POSE 1 Result

POSE 2

End Effector Translation Vector from World Frame: $[0.5, 0, 0.5]$

End Effector Euler Angle with respect to World Frame: $[0, \pi, \pi/2]$

Result:

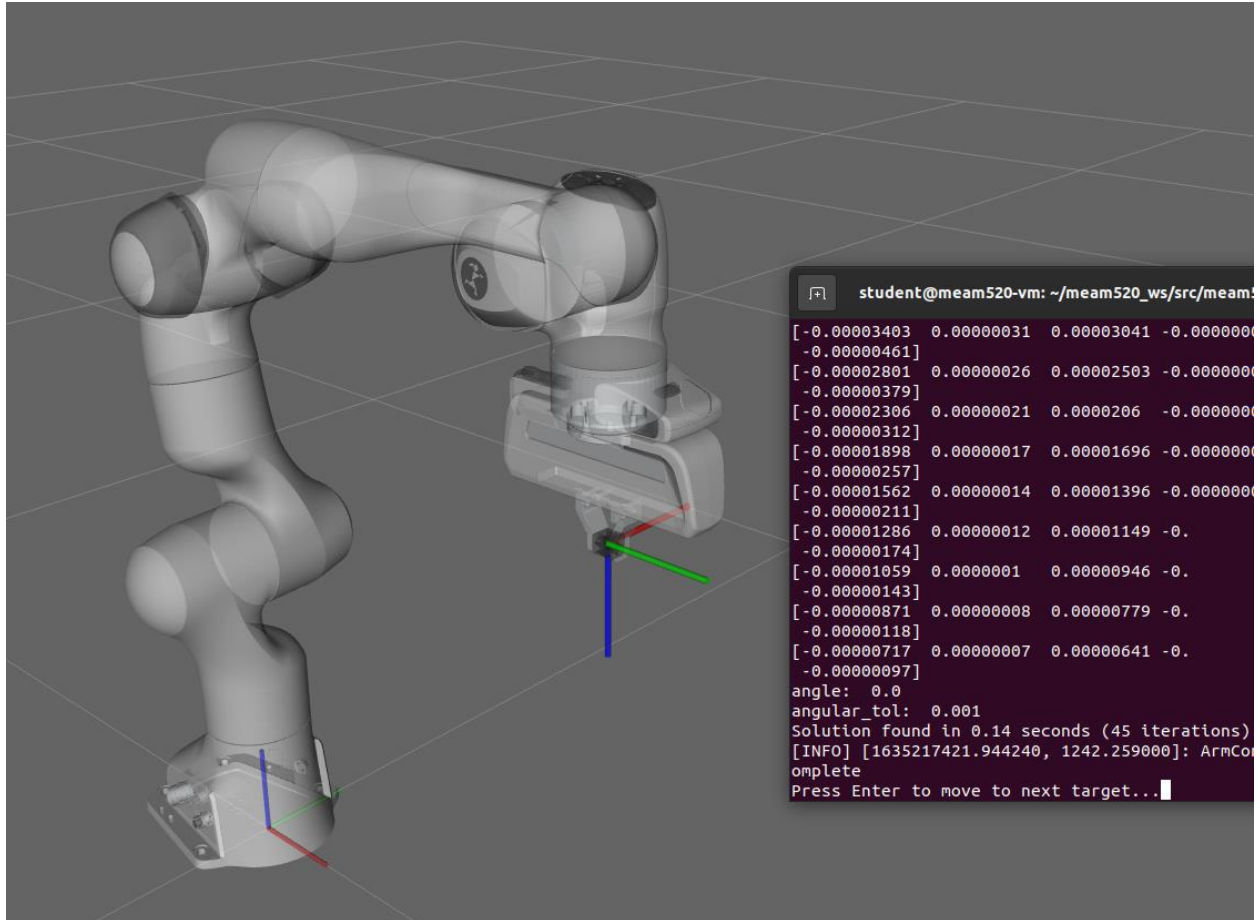


Figure 6.2.2 POSE 2 Result

POSE 3

End Effector Translation Vector from World Frame: [0.5, 0.5, 0.5]

End Effector Euler Angle with respect to World Frame: [0, π , $\pi/2$]

Result:

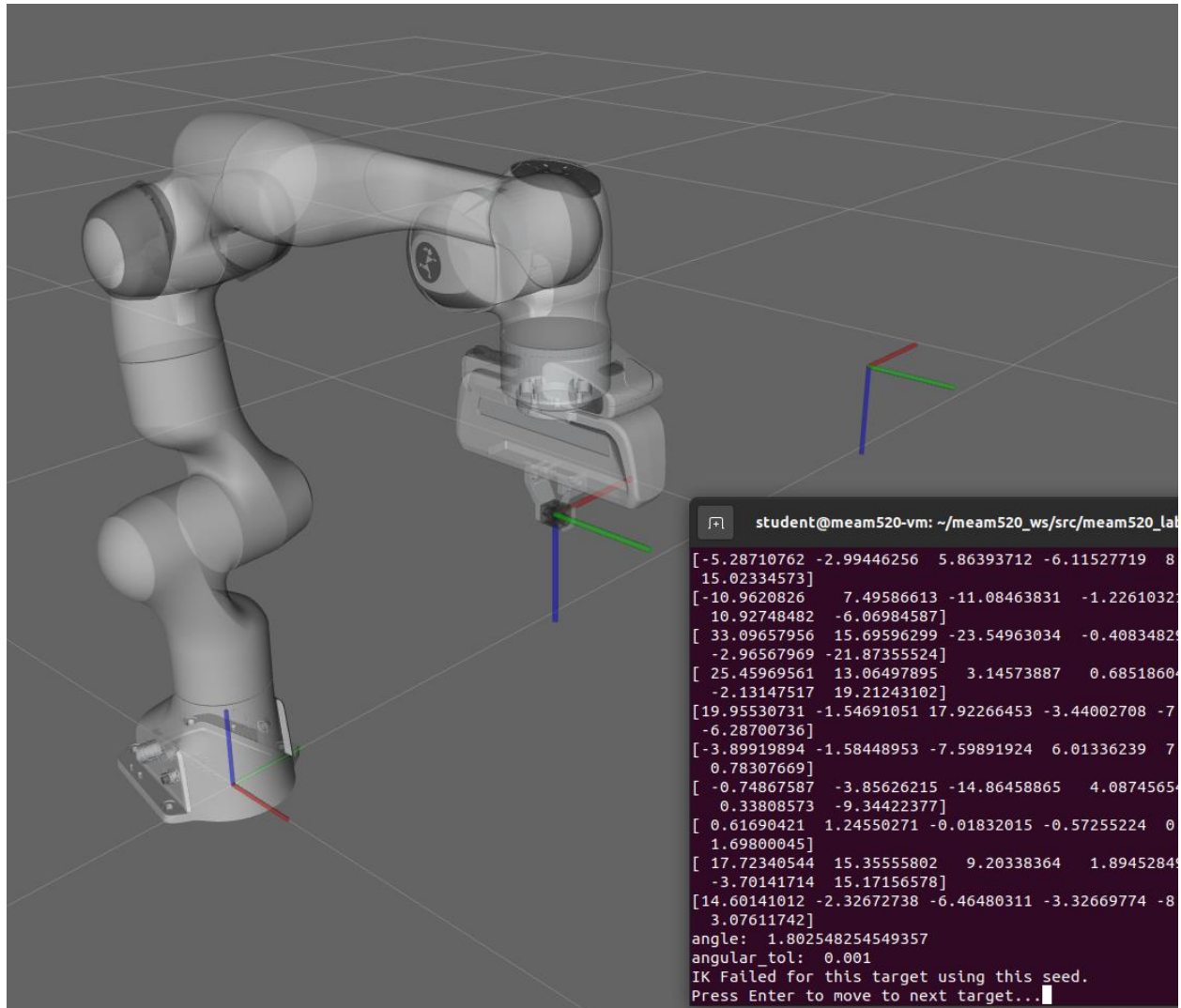


Figure 6.1.3 POSE 3 Result

Performance Statistics

| [Translation Vector], [Euler Angle] of end effector with respect to world frame | Elapsed time(s) | Number of Iterations performed | Success or not |
|---|-----------------|--------------------------------|----------------|
| [0.5, 0, 0.5]; [0, pi/2, pi/2] | 0.05 | 27 | Success |
| [0.5, 0, 0.5]; [0, pi, pi/2] | 0.13 | 45 | Success |
| [0.5, 0.2, 0.5]; [0, pi, pi/2] | 0.14 | 43 | Success |
| [0.7, 0.2, 0.5]; [0, pi, pi/2] | N/A | Max. out | Failure |
| [-0.2, 0.5, 0.5]; [0, pi, pi] | 0.15 | 45 | Success |
| [-0.2, 0, 0.5]; [0, pi, pi/2] | N/A | Max. out | Failure |
| [0.2, -0.3, 0.5]; [0, pi/2, pi/2] | 0.17 | 54 | Success |
| [0.2, -0.6, -0.5]; [pi/2, pi/2, pi] | N/A | Max. out | Failure |
| [0.5, 0, 0]; [pi/2, pi/2, pi/2] | N/A | Max. out | Failure |
| [0.5, 0, 0.5]; [pi/2, pi, pi/2] | 0.11 | 46 | Success |
| Average | 0.125 | 43.33 | 60% |
| Medium | 0.135 | 45 | |
| Maximum | 0.17 | 54 | |

6.3 Analysis

Uniqueness of Solutions

The solutions will always be different if we choose different seeds to approach for the same target configuration. The reason is that our optimization solution depends much on our initial set of the starting configurations to generally approach the acceptable results. Once we change the seed, the number of iterations, time for approaching the result and the solution itself will all be different based on our initial guess. Here is a prove for such finding. Let's take the seed and target in the solveIK.py script as an example, when the target is set and the seed is set as [0,0,0, -pi/2,0, pi/2, pi/4], the result turns out to be the following:

```
Solution: [-0.53351 -0.85962 0.32045 -2.32001 0.24179 1.48877 2.05622]
Iterations: 41
```

However, if we change the seed to something else, for instance [pi/2, pi/4,0, -pi/2,0, pi/2, pi/4], now the result becomes:

```
Success: False
Solution: [ 2.43049 -1.27024 -3.49694 -4.97217 1.738 0.34375 -2.99934]
Iterations: 103
```

It is obvious that in the second case, we cannot yield a solid solution for the same target since we changed the seed value, which proves that we do not yield the same result if changing the seed for a same target frame.

Algorithmic Completeness

Failing to find a solution for some end effector poses does not mean that either the robot arm cannot achieve that end effector pose, nor that we have coded our algorithm incorrectly. There are several scenarios that failure might happen. In our code we set the upper/lower constraints for each joint as their limits. If some end effector pose ends up outside those constraints that some joints cannot reach a specific configuration, the result would turn out to be FALSE. Though the pose is within the workspace, the arm will not be able to reach that space with our constraints. One other reason could be that sometimes the value we set as the seed is a bad guess with respect of the end effector pose. In this case, the arm might not be able to make it up to the final pose within the maximum steps. Or in another case the step set to be too large that the result cannot converge to that local point, instead, it 'jumps' back and forth between two values outside of the tolerance but close to our solutions. It does not mean that the arm cannot reach the position, nor that our algorithm is incorrect. The algorithm cannot fully cover the efficiency and accuracy at the same time, what we could do is to improve it and achieve a balance for most cases.

Warm Start

To decrease the total runtime/iterations necessary to compute IK solutions for all these poses, one potential solution is to create an adaptive step size for the optimization instead of a fixed step. For example, each time after we set our initial guess, we start with some big values of step size. As the guesses gradually approach the converging point, the optimization process will adjust the step size regarding the gradient to keep the result accurate enough while improving the processing speed. This idea is rather similar to another type of optimization algorithm, which is called the 'Stochastic Gradient Descent'. It could obviously reduce the iteration numbers and time to get to the solution, while in trade of a relatively lower rate of convergence. If we have a high demand of computation speed instead of convergence rate, we could improve our algorithm in this way.