STAT 598Z Course Project
Instructor: Prof. S V N Vishwanathan **Jiajie Huang**
Due April 23th, 2013 **huang147@purdue.edu**

Problem 1

- The three different functions are as below:

$$J_1(x) = x_1^2 + x_2^2 + x_1 + x_2$$

$$J_2(x) = x_1^2 + 2x_2^2 + x_1 + x_2$$

$$J_3(x) = x_1^2 + 10x_2^2 + x_1 + x_2$$

  Tables 1, 2, and 3 shows the visualization of the three different functions $J_1(x)$, $J_2(x)$, and $J_3(X)$ respectively.

- I start from $[5, 5]$, then use gradient descent to find out the minimum of the three functions above. The algorithm of gradient descent is:

Algorithm 1 Gradient Descent

Input: Initial point w_0, gradient norm tolerance ϵ
Output: w_t
Set $t = 0$
while $\|\nabla J(w_t)\| \geq \epsilon$ then **do**
$w_{t+1} = w_t - \eta_t \nabla J(w_t)$
$t = t + 1$
end while

  Therefore, we can change the values of ϵ and η and see how the trajectory of points using gradient descent is changing. Each red line in the four panels in Tables 1, 2, and 3 is showing the trajectory of points using different ϵ and η, with the starting and ending points shown as well.

- This three functions are convex functions, which means if we can find a local minimum by gradient descent algorithm, that point is also the global minimum of that function. See summary results below of the three functions with different values of ϵ and η (Please note that I used h below instead of η in my Python code).

  First of all, the variation in tolerance ϵ will make a difference in the number of steps needed to get to the minimum. The smaller ϵ is, the more expensive it is to get the minimum, but the result is also more accurate.

  Secondly, the choice of η is the key in getting the minimum. The three functions differ in their variances as $var(J_1) < var(J_2) < var(J_3)$. In tracing the minimum in functions with a larger variance, which have more spread-out points, thus they have a larger gradient descent at a specific point, compared to the functions with smaller variances. Given the same η, and start from the same point, the step size $\eta_t \nabla J(w_t)$ is larger, and we need to be very careful about this. That's because with a large step size it is very possible that we miss the minimum by stepping over it. For function J_3, I first set $\eta = 0.1$, and that doesn't give me a converged result at all; instead, it keeps producing very large norm of gradients $\|\nabla J(w_t)\|$ which never go smaller than my ϵ (either 0.1 or 0.01). So for function J_3 I ended up setting η less than 0.01. A smaller η produces smaller steps; although it prevents us from missing the minimum point, but it also takes more steps to reach that point, which is more computational expensive.

```
J1(x):
Converged in 33 iterations
from [ 5.  5.] to [-0.49651396 -0.49651396]
J(x) min=-0.499975695062at[-0.49651396 -0.49651396]
 e=0.01 h=0.1

Converged in 364 iterations
from [ 5.  5.] to [-0.4964791 -0.4964791]
J(x) min=-0.499975206463at[-0.4964791 -0.4964791]
 e=0.01 h=0.01

Converged in 23 iterations
from [ 5.  5.] to [-0.46753373 -0.46753373]
J(x) min=-0.49789188268at[-0.46753373 -0.46753373]
 e=0.1 h=0.1

Converged in 250 iterations
from [ 5.  5.] to [-0.46477252 -0.46477252]
J(x) min=-0.497518048899at[-0.46477252 -0.46477252]
 e=0.1 h=0.01


J2(x):
Converged in 32 iterations
from [ 5.  5.] to [-0.49564245 -0.24999958]
J(x) min=-0.374981011767at[-0.49564245 -0.24999958]
 e=0.01 h=0.1

Converged in 347 iterations
from [ 5.  5.] to [-0.49503624 -0.2499963 ]
J(x) min=-0.374975361039at[-0.49503624 -0.2499963 ]
 e=0.01 h=0.01

Converged in 22 iterations
from [ 5.  5.] to [-0.45941716 -0.2499309 ]
J(x) min=-0.373353023794at[-0.45941716 -0.2499309 ]
```

```
 e=0.1 h=0.1

Converged in 233 iterations
from [ 5.  5.] to [-0.45033639 -0.24961153]
J(x) min=-0.372533223559at[-0.45033639 -0.24961153]
 e=0.1 h=0.01


J3(x):
Converged in 347 iterations
from [ 5.  5.] to [-0.49503624 -0.05      ]
J(x) min=-0.274975361066at[-0.49503624 -0.05      ]
 e=0.01 h=0.01

Converged in 3499 iterations
from [ 5.  5.] to [-0.4950097 -0.05     ]
J(x) min=-0.274975096906at[-0.4950097 -0.05     ]
 e=0.01 h=0.001

Converged in 233 iterations
from [ 5.  5.] to [-0.45033639 -0.05      ]
J(x) min=-0.272533525376at[-0.45033639 -0.05      ]
 e=0.1 h=0.01

Converged in 2348 iterations
from [ 5.  5.] to [-0.45001109 -0.05      ]
J(x) min=-0.272501109279at[-0.45001109 -0.05      ]
 e=0.1 h=0.001
```

- Table 4 is showing the visualization of the objective function

  $$J(w) = \frac{\lambda}{2}\|w\|^2 + \sum_i (w^T x_i - y_i)^2$$

  with different values of λ's. The first panel is showing $\lambda = 0.001$.

- See the red lines in Table 4 for different trajectories of points with λ's 0.001, 1.0, 1000.0, 100000.0, 0.0001, and 0.0. Starting and ending points are marked in each panel.

- This function $J(w)$ is a combination of quadratic functions of w, we know that quadratic function is a convex function, thus $J(W)$ is also a convex function, which means if we can find a local minimum by gradient descent algorithm, that point is also the global minimum of that function.

  First of all, the choice of ϵ determines the accuracy of results, which is similar to the functions J_1, J_2, and J_3 above.

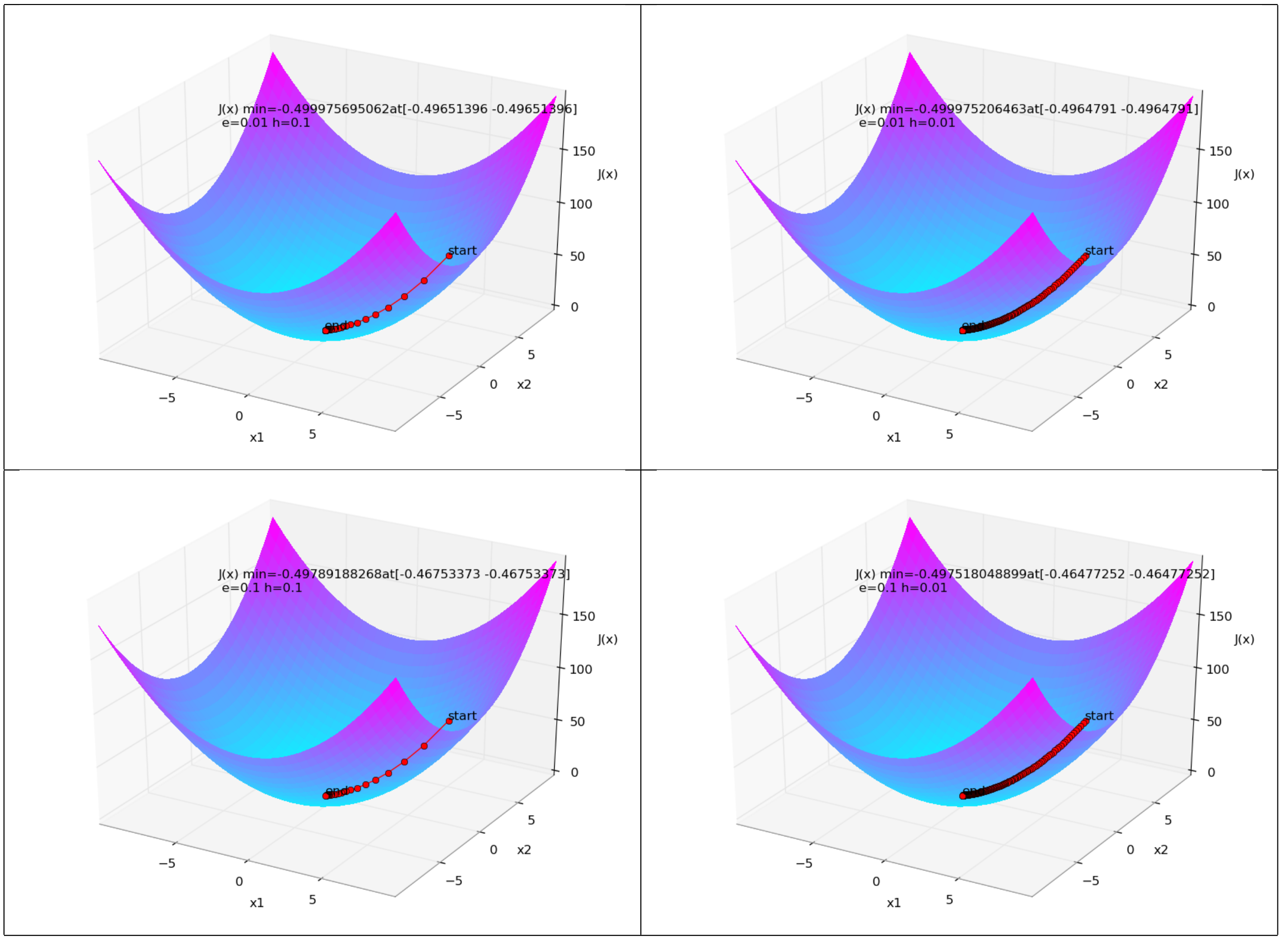

Table 1: Visualization of $J_1(x)$ and Trajectory of Points Using Gradient Descent

Secondly, the choice of η is determined by the $2,000$ data points we have. This big amount of data points makes the step size much larger than the cases in the previous functions J_1, J_2, or J_3. I tested on several different η's, and found that $\eta = 1e^{-5}$ is the optimal value in getting the minimum point in reasonable step numbers (I set the max iteration number to be 1000).

Lastly, the choice of λ makes a difference in the minimum of $J(w)$ only when λ is huge. From

$$J(w) = \frac{\lambda}{2}\|w\|^2 + \sum_i (w^T x_i - y_i)^2$$

we know that the gradient of this function can be vectorized and put in Python code (see my code for reference):

$$\nabla J(w) = \lambda w + 2X^T(Xw - Y)$$

Here X is a 2000×2 matrix of the $2,000$ data points; Y is a 1000×1 vector of the labels (either $+1$ or -1) of the $2,000$ dat points. When λ is small, the first part λw is dominated by

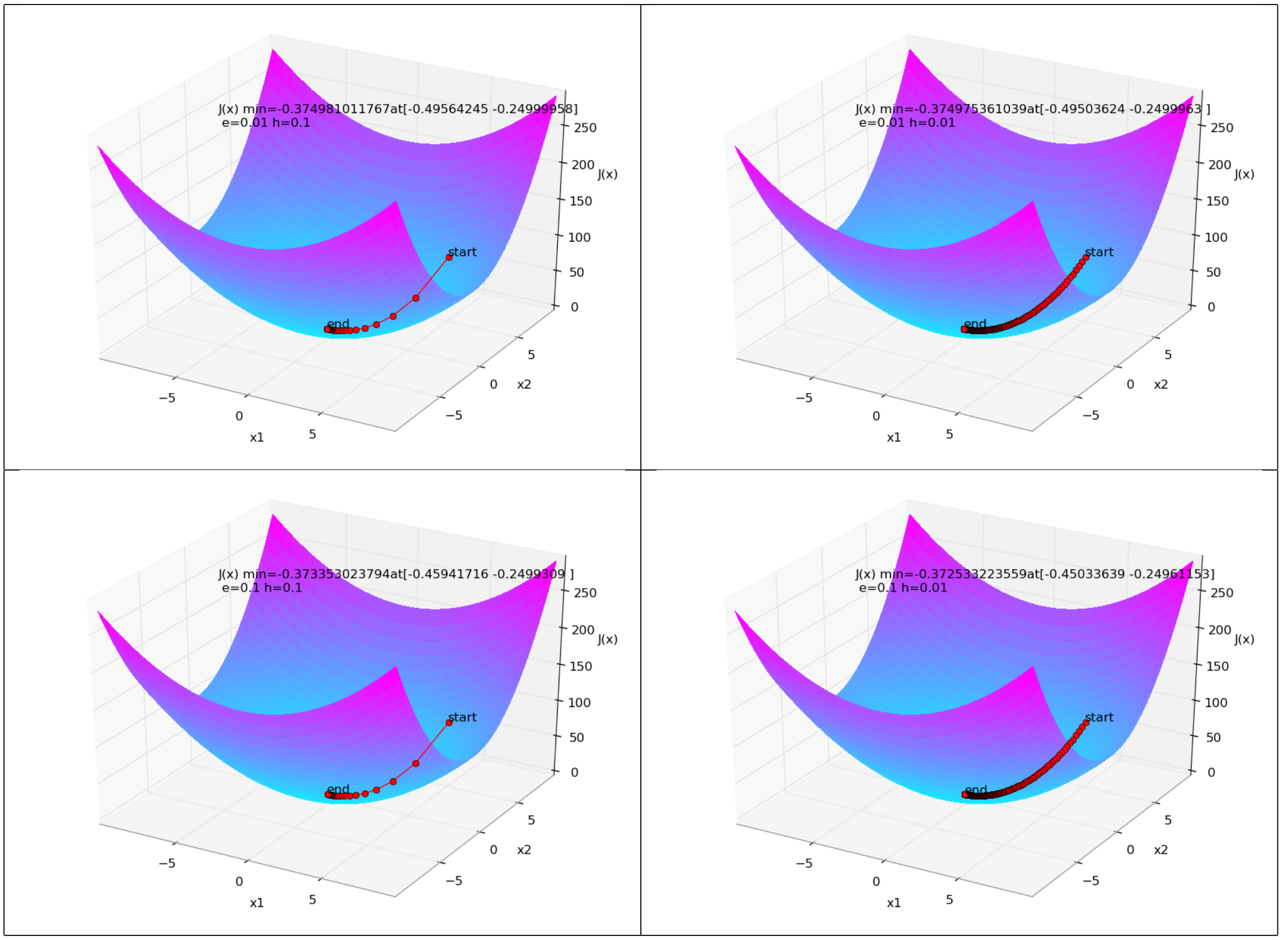

Table 2: Visualization of $J_2(x)$ and Trajectory of Points Using Gradient Descent

the second part $2X^T(Xw - Y)$, as X and Y are huge matrices compared to the $2-d$ vector w; therefore in this case λ almost has no impact on the gradient descent. However, when λ gets larger, w gets significantly important in determining $\nabla J(w_t)$. See summary results below of changes of $J(w)$ with different values of λ (Please note that I used h below instead of η in my Python code). When $\lambda = 100000.0$, the minimum moves from $[-0.11526332, -0.07674558]$ $(\lambda = 0.001)$ to $[5.94202596e + 17, 5.96931252e + 17]$.

```
Converged in  230 iterations
from [[ 1.]
 [ 1.]] to [[-0.11526332]
 [-0.07674558]]
J(w) min=1039.21825972at[[-0.11526332]
 [-0.07674558]] e=0.01 h=1e-05lamb=0.001

Converged in  230 iterations
from [[ 1.]
```

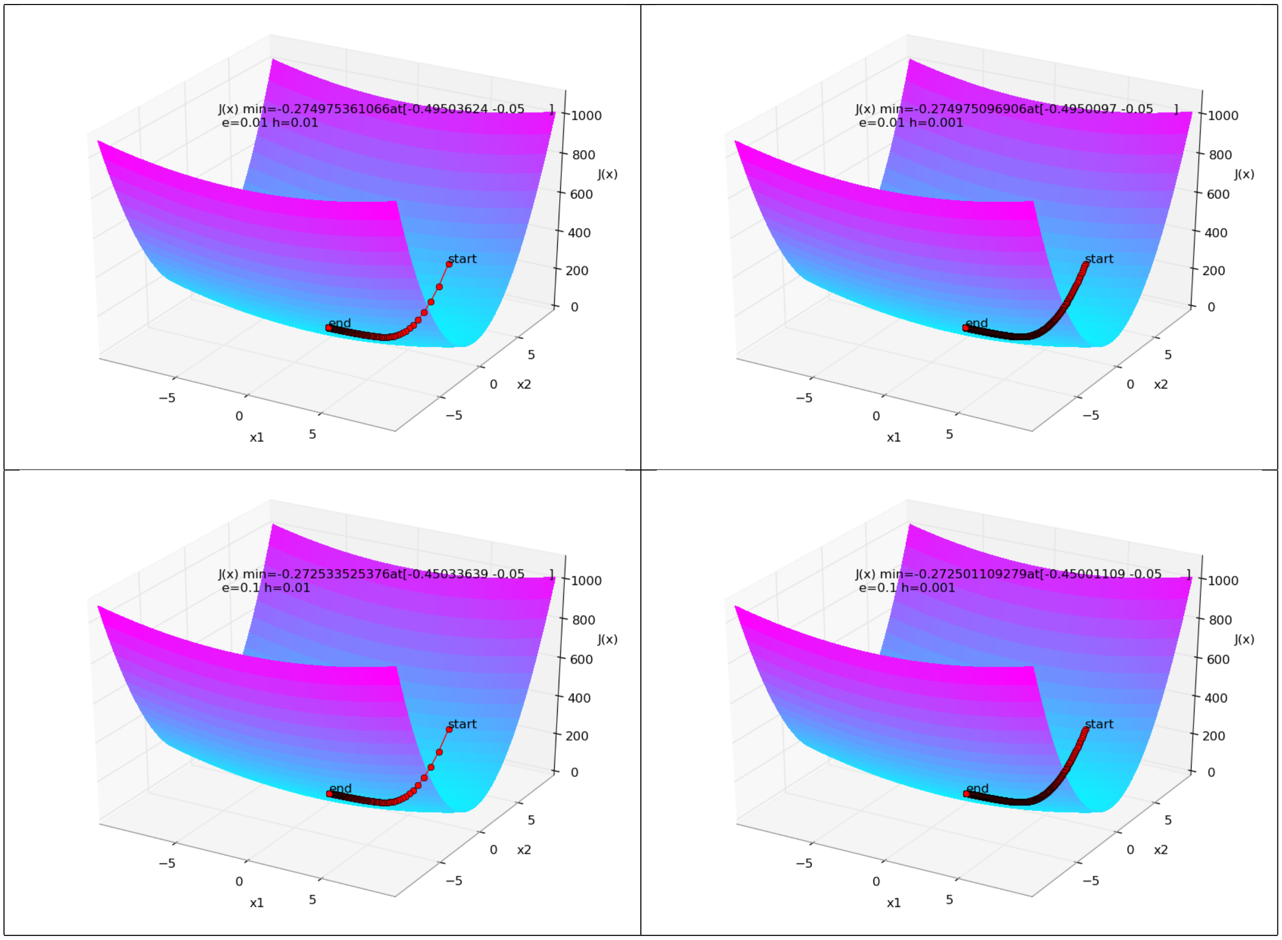

Table 3: Visualization of $J_3(x)$ and Trajectory of Points Using Gradient Descent

```
 [ 1.]] to [[-0.11525755]
 [-0.07674949]]
J(w) min=1039.22783777at[[-0.11525755]
 [-0.07674949]] e=0.01 h=1e-05lamb=1.0

Converged in  184 iterations
from [[ 1.]
 [ 1.]] to [[-0.11046043]
 [-0.07970587]]
J(w) min=1048.64285946at[[-0.11046043]
 [-0.07970587]] e=0.01 h=1e-05lamb=1000.0
from [[ 1.]
 [ 1.]] to [[  5.94202596e+17]
 [  5.96931252e+17]]
J(w) min=6.67348784196e+40at[[  5.94202596e+17]
 [  5.96931252e+17]] e=0.01 h=1e-05lamb=100000.0
```

```
Converged in  230 iterations
from [[ 1.]
 [ 1.]] to [[-0.11526333]
 [-0.07674558]]
J(w) min=1039.21825109at[[-0.11526333]
 [-0.07674558]] e=0.01 h=1e-05lamb=0.0001

Converged in  230 iterations
from [[ 1.]
 [ 1.]] to [[-0.11526333]
 [-0.07674558]]
J(w) min=1039.21825013at[[-0.11526333]
 [-0.07674558]] e=0.01 h=1e-05lamb=0.0
```

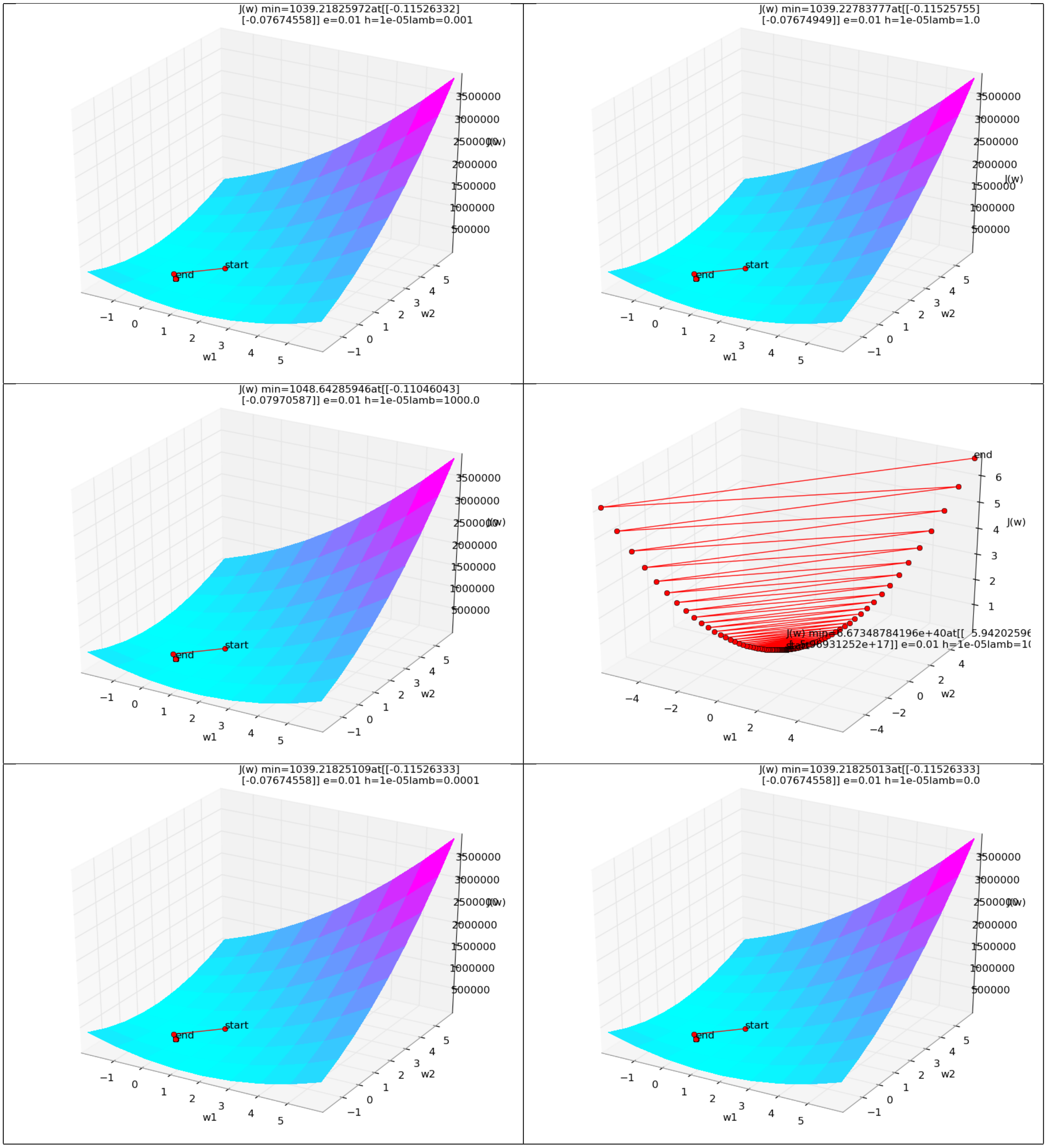

Table 4: Visualization of $J(w)$ and Trajectory of Points with λ = 0.001, 1.0, 1000.0, 100000.0, 0.0001, and 0.0