ECE 133A Project

Nevin Liang

May 30, 2021

Abstract

In this project we determine the location of $M = N - K$ points in the plane where each point (x, y) satisfies $x, y \in [0, 1]$. We are given the location of K points with known positions and the inexact distances between certain pairs of points. The majority of this code is implementation of the Levenberg–Marquardt method.

1 Overview of the Algorithm

Algorithm 1: network loc

function: network loc(N, E, pos anchor, rho) := pos free
initialize x to a random $2N - 2K$ vector
for k = 1:10000 **do**
  calculate $fxk = f(x)$ and $A = Df(x)$
  calculate the solution to $\hat{x} = x - (A^T A + \lambda^{(k)} I)^{-1} A^T fxk)$
  if $||f(\hat{x}^{(k)})||^2 < ||fxk||^2$ **then**
    $x = \hat{x}$
    $\lambda^{(k+1)} = \beta_1 \lambda^{(k)}$
  else
    $\lambda^{(k+1)} = \beta_2 \lambda^{(k)}$
  end if
  if $2A' \cdot fxk < 1e - 5$ **then**
    break
  end if
end for

1.1 Data Structures Involved

For my algorithm, the first question I had to ask was how to represent the variable x, in terms of the variables given in the equation. I decided to go with a $2N - 2K$ vector representing

$$u_1, u_2, u_3, \cdots, u_{N-K}, v_1, v_2, v_3, \cdots, v_{N-K}$$

. This way it would be easy to ravel and unravel from the original 'pos_anchor' and final 'pos_free' values. It also made it simple to loop over to calculate the Jacobian and the cost function, f. We initialize the value of x to a vector of random numbers between 0 and 1.

1.2 Algorithms to calculate cost function and Jacobian

To calculate $f(x^{(k)})$, all we had to do was calculate

$$||p_{ik} - p_{jk}|| - \rho_k$$

for every k. The actual value of the cost function was just the norm of $f(x^{(k)})$ squared.

To calculate the Jacobian, a little more work was required. Since the majority of elements in this matrix are 0, we first initialize the L by $2N-2K$ matrix to all 0's. Then, for each row, either 2 or 4 of the elements are non-zero because either one of the points is fixed (in which case the partial derivatives are 0), or both are variable (one of the first N-K points), in which case the partial derivatives are nonzero.

For two points (u_i, v_i) and (u_j, v_j). The partial derivatives of the kth row of f with respect to u_i and v_i are

$$\frac{\partial f_k}{\partial u_i} = ||f(x^{(k)})||^{-1} \cdot (u_i - u_j)$$

. The other ones can be calculated through symmetry. (NOTE: all have the same coefficient, though, which was already calculated so we can reuse the value). The reason I use

$$||f(x^{(k)})||^{-1}$$

for the coefficient is because when we do calculate the partial derivative for the Jacobian, we end up with

$$((u_i - u_j)^2 + (v_i - v_j)^2)^{-1/2}$$

as the coefficient. This can be rewritten as $||f(x^{(k)})||^{-1}$.

If either of the two points i or j are fixed, however, we make sure that element in the matrix is 0.

1.3 Every Iteration

1.3.1 Calculation of $\hat{x}$

Once we have calculated the cost function as well as the matrix A, all that is left is to calculate the new value of x to update our variable. To do this we calculate the solution to the linear least squares problem

$$\hat{x} = x - (A^T A + \lambda^{(k)} I)^{-1} A^T f(x^{(k)})$$

. We can do this in two ways, the fast of which is to use QR factorization or the \ operator in Matlab. The way I did it was to calculate the solution to the compacted form of our least squares problem:

$$\left|\left| \begin{bmatrix} A \\ \sqrt{\lambda} I \end{bmatrix} \Delta x + \begin{bmatrix} f(x^{(k)}) \\ \mathbf{0} \end{bmatrix} \right|\right|^2$$

. We can then use $M \backslash b$ where M and b are the two terms in the above least squares representation. Then, knowing Δx we can figure out $\hat{x}$.

1.3.2 Updating Values

Every iteration we have to update values of x and λ. To do so, we first see if our cost function has improved. If it does, we update x to $\hat{x}$ and decrease λ. Otherwise, we don't update x and increase λ. The exact equation is shown in the algorithm at the top. I increased λ by either a factor of 2.0 or decreased λ by a factor of 0.8 every iteration.

1.3.3 Exit Conditions

When ∇g is small enough, where

$$\nabla g = 2A^T f(x^{(k)})$$

we are able to exit. I used the value of 10^{-5} in my code to test for exit conditions.

2 Test Case

Below, I show the test case for $N = 50$ and $R = 0.4$ with an s-value of $s = 0.05$.

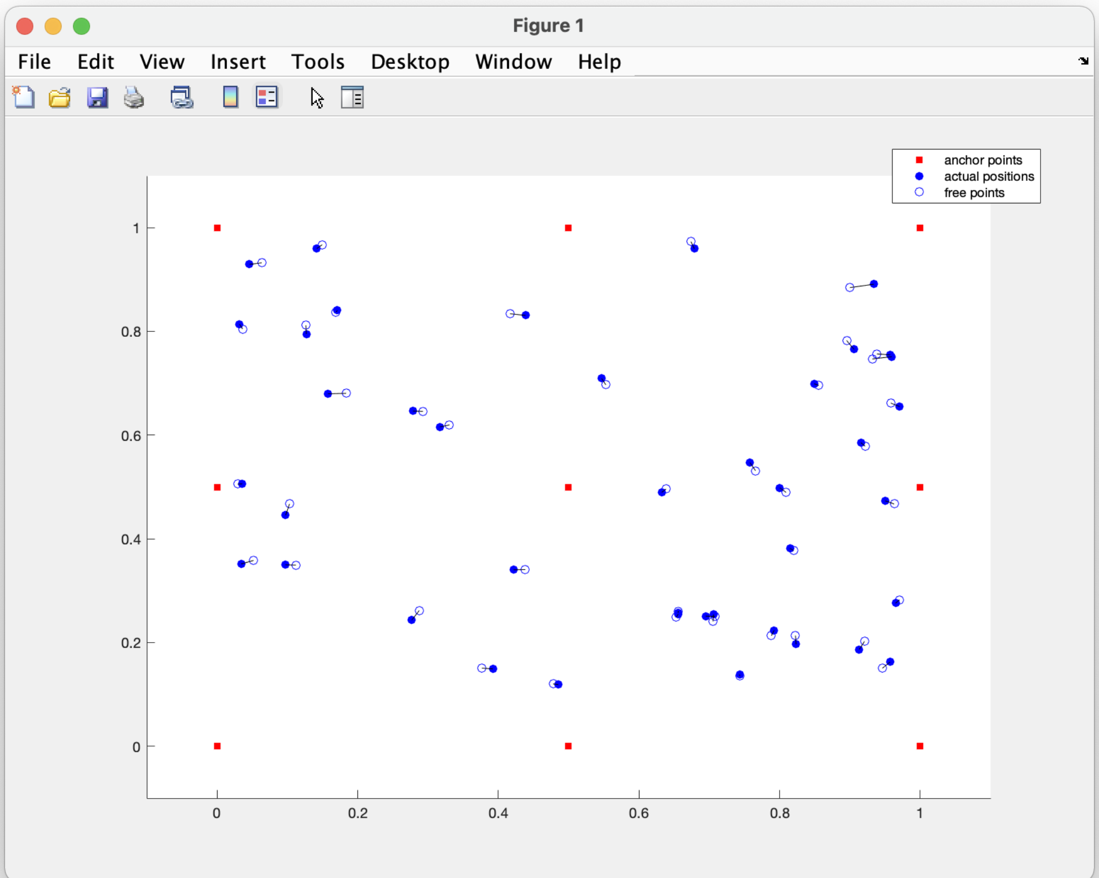
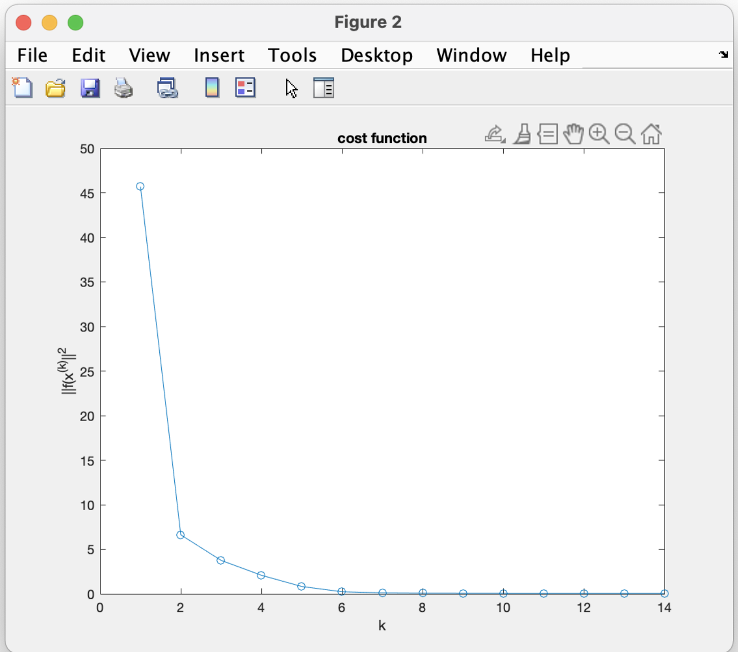


Figure 1: Plot of the cost function.



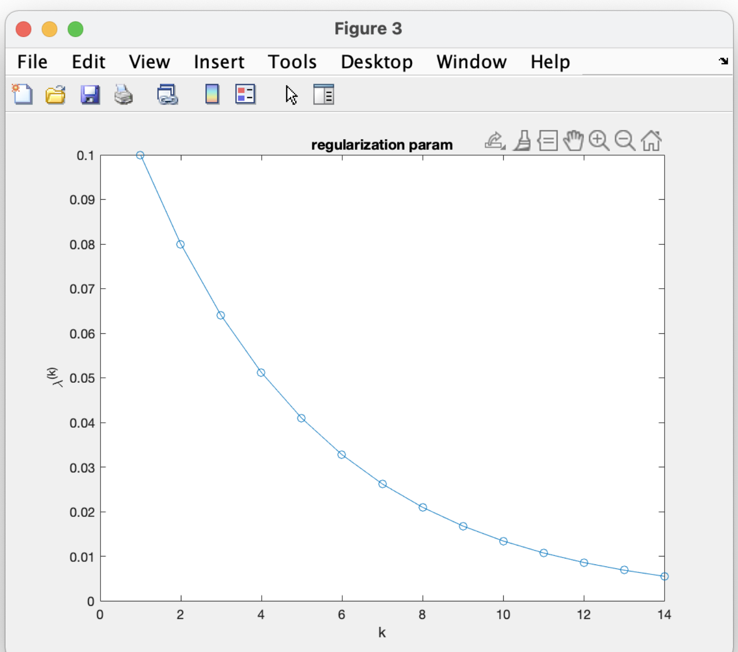


Figure 2: Plot of the cost function (top); Plot of the regularization parameter λ (bottom)