EC ENGR 133A, Spring 2021: Kellen Cheng (905155544)
Nonlinear Least Squares Project
Due 4 June, 2021 at 11:59 PM

1 Nonlinear Least Squares Problem Formulation

(a) **Solution:** In our wireless sensor network consisting of N nodes, we know the location of K sensor nodes, but unfortunately, we do not know the exact location of the other $M = N - K$ sensor nodes. However, we can work around this by predicting the locations of the unknown M nodes through solving a nonlinear least squares problem.

(b) **Solution:** First, represent the distances between points as an undirected graph with node coordinates as follows:

$$p_1 = (u_1, v_1), p_2 = (u_2, v_2), ..., p_N = (u_N, v_N)$$

In addition, in a graph of L edges between points, we can represent the each edge distance as follows (with ϵ_k being measurement error):

$$\rho_k = \|p_{i_k} - p_{j_k}\| + \epsilon_k, k = 1, 2, ..., L$$

(c) **Solution:** Next, we can formulate our minimizing function as follows:

$$\sum_{k=1}^{L} (\sqrt{(u_{i_k} - u_{j_k})^2 + (v_{i_k} - v_{j_k})^2} - \rho_k)^2$$

Thus, we can conclude that our function $f(x^{(k)})$ is as follows:

$$f(x^{(k)}) = (\sqrt{(u_{i_k} - u_{j_k})^2 + (v_{i_k} - v_{j_k})^2} - \rho_k)$$

As a result, utilizing the Levenberg-Marquardt method, we should try to minimize the following:

$$\|f(x^{(k)}) + Df(x^{(k)})(x - x^{(k)})\|^2 + \lambda^{(k)}\|x - x^{(k)}\|^2$$

(d) **Solution:** Thus, from our minimizing function in (c), we can formulate the unknown positions as a nonlinear least squares problem with variables $u_1, u_2, ..., u_{N-K}$ and $v_1, v_2, ..., v_{N-K}$. In addition, we know that the solution is computed as follows:

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

In this particular case, we will terminate when $\nabla g(x^{(k)}) = 2A^T f(x^{(k)}) \leq 10^{-5}$.

2 Levenberg-Marquardt Jacobian Matrix

(a) **Solution:** A crucial step in executing the Levenberg-Marquardt method involves the computation of the Jacobian matrix. Recall the definition of the Jacobian matrix as follows:

$$Df(z) = \begin{bmatrix} \frac{df_1}{dx_1}(z) & \frac{df_1}{dx_2}(z) & \dots & \frac{df_1}{dx_n}(z) \\ \frac{df_2}{dx_1}(z) & \frac{df_2}{dx_2}(z) & \dots & \frac{df_2}{dx_n}(z) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{df_n}{dx_1}(z) & \frac{df_n}{dx_2}(z) & \dots & \frac{df_n}{dx_n}(z) \end{bmatrix}$$

In our wireless sensor node network, $x_1, x_2, ..., x_n$ refer to $u_1, u_2, ..., u_{N-K}, v_1, v_2, ..., v_{N-K}$, with our function f(z) being defined as follows (using the points at the k-th iteration):

$$f(x^{(k)}) = (\sqrt{(u_{i_k} - u_{j_k})^2 + (v_{i_k} - v_{j_k})^2} - \rho_k)$$

Quickly, it is evident that our matrix $Df(z) \in \mathbb{R}^{\mathrm{Lx2(N-K)}}$, as we have a row for each of the L edges, and we have $2(N-K)$ variables.

(b) **Solution:** Regarding the actual matrix entries themselves, there are four possible derivatives we could require in each row, with the rest of the entries in that row being 0. The four possible derivative expressions are listed as below:

$$\begin{bmatrix} \frac{df(x^{(k)})}{du_{i_k}} \\ \frac{df(x^{(k)})}{du_{j_k}} \\ \frac{df(x^{(k)})}{dv_{i_k}} \\ \frac{df(x^{(k)})}{dv_{j_k}} \end{bmatrix} = \begin{bmatrix} \frac{(u_{i_k} - u_{j_k})}{\sqrt{(u_{i_k} - u_{j_k})^2 + (v_{i_k} - v_{j_k})^2}} \\ \frac{-(u_{i_k} - u_{j_k})}{\sqrt{(u_{i_k} - u_{j_k})^2 + (v_{i_k} - v_{j_k})^2}} \\ \frac{(v_{i_k} - v_{j_k})}{\sqrt{(u_{i_k} - u_{j_k})^2 + (v_{i_k} - v_{j_k})^2}} \\ \frac{-(v_{i_k} - v_{j_k})}{\sqrt{(u_{i_k} - u_{j_k})^2 + (v_{i_k} - v_{j_k})^2}} \end{bmatrix}$$

However, we can clearly notice that we have four possible conditions: both nodes are anchors, one of the two nodes are anchors (either the first node in the edge or the second node), and neither of the two nodes are anchors.

If the i-th edge contains nodes that are both anchors, our Jacobian matrix row is all zeros, since both variables are known (assuming points at the k-th iteration):

$$\left[\nabla f_i(x^{(k)})^T\right] = \begin{bmatrix} 0 & 0 & \dots & 0 \end{bmatrix}$$

If the i-th edge contains one anchor node, our Jacobian matrix row will contain the derivatives with respect to the unknown node, with the rest of the entries in that row being 0 (assuming points at the k-th iteration):

$$\left[\nabla f_i(x^{(k)})^T\right] = \begin{bmatrix} 0 & 0 & \dots & 0 & \frac{df(x^{(k)})}{du_{i_k}} & 0 & \dots & 0 & \frac{df(x^{(k)})}{dv_{i_k}} & 0 & \dots & 0 \end{bmatrix}, \text{node 2 as anchor}$$

$$\left[\nabla f_i(x^{(k)})^T\right] = \begin{bmatrix} 0 & 0 & \dots & 0 & \frac{df(x^{(k)})}{du_{j_k}} & 0 & \dots & 0 & \frac{df(x^{(k)})}{dv_{j_k}} & 0 & \dots & 0 \end{bmatrix}, \text{node 1 as anchor}$$

Our last of the four conditions occurs if the i-th edge contains no anchor nodes, in which our Jacobian matrix row will contain the derivatives with respect to both of the unknown nodes (i.e. a potential of four non-zero elements in the row), all assuming points at the k-th iteration:

$$\left[\nabla f_i(x^{(k)})^T\right] = \begin{bmatrix} 0 & \dots & \frac{df(x^{(k)})}{du_{i_k}} & \dots & \frac{df(x^{(k)})}{du_{j_k}} & \dots & \frac{df(x^{(k)})}{dv_{i_k}} & \dots & \frac{df(x^{(k)})}{dv_{j_k}} & \dots & 0 \end{bmatrix}$$

3 Levenberg-Marquardt MATLAB Algorithm

(a) **Solution:** The MATLAB code (see section 4) executes the Levenberg-Marquardt algorithm to predict the estimated coordinates of the unknown sensor nodes in our wireless network. The algorithm will continue to run until our termination condition, $\|\nabla g(x)\| \leq 10^{-5}$, is satisfied.

(b) **Solution:** In each iteration of our algorithm, the MATLAB code computes first $f(x^{(k)})$, and then the Jacobian matrix $Df(x^{(k)})$ row by row, with each row being evaluated for one of the four conditions described in section 2.

(c) **Solution:** Next, within the iteration, the code computes both $\|\nabla g(x)\|$ and $\hat{x}$, in order to evaluate $f(\hat{x})$ in comparison with $f(x^{(k)})$. Depending on this calculation, the code updates $x^{(k+1)}$ and $\lambda^{(k+1)}$ as follows:

$$x^{(k+1)} = \hat{x} \text{ and } \lambda^{(k+1)} = \beta_1 \lambda^{(k)}, \text{ if } \|f(\hat{x})\|^2 < \|f(x^{(k)})\|^2$$

$$x^{(k+1)} = x^{(k)} \text{ and } \lambda^{(k+1)} = \beta_2 \lambda^{(k)}, \text{ otherwise}$$

(d) **Solution:** If $\nabla g(x^{(k)}) = 2A^T f(x^{(k)}) \leq 10^{-5}$, then the algorithm will terminate out of the loop and return our calculated estimated coordinates of our unknown sensor nodes.

4 Levenberg-Marquardt MATLAB Function

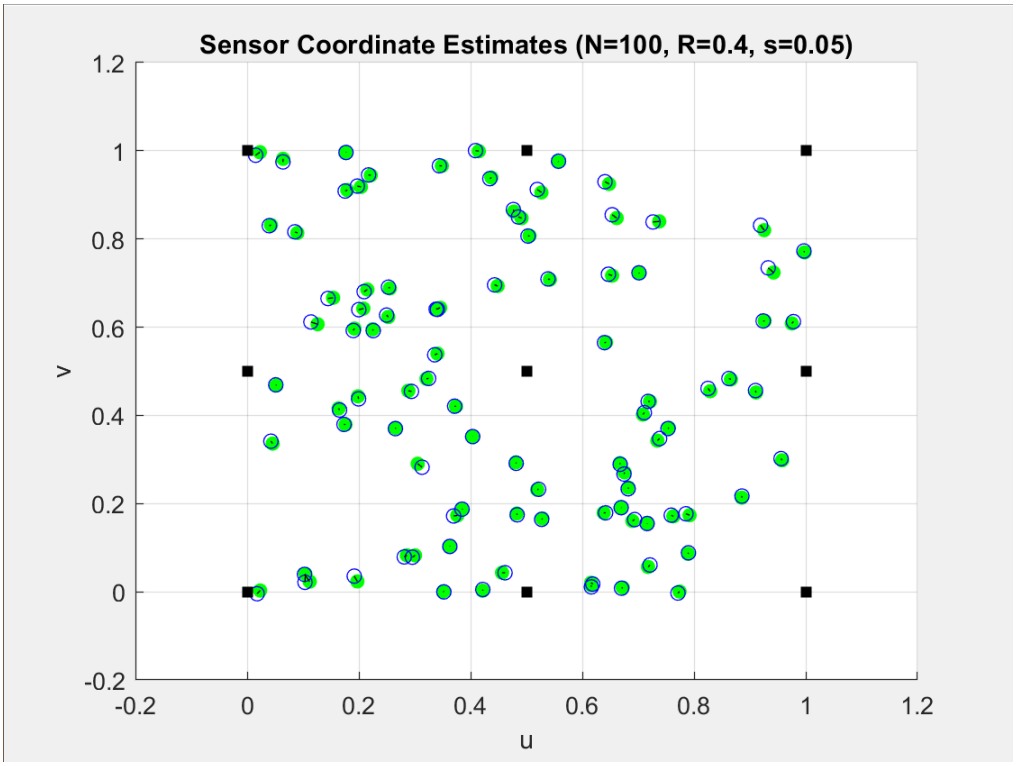
```
% network_loc.m
function pos_free = network_loc(N, E, pos_anchor, rho)
%% Parameter Initialization
beta1 = 0.8; beta2 = 2.0; lambda = 10^-5; constraint = 10^-5;
L = size(E, 1);

M = N - size(pos_anchor, 1); % number of free points
pos_free = rand(M, 2); % random initialization
x_k = [pos_free(:, 1); pos_free(:, 2)]; % unraveled coordinates
loc_total = [pos_free; pos_anchor]; % all node coordinates
grad = 1 / 0; % begin with infinite gradient magnitude

%% Levenberg-Marquardt Method
k = 1; % iteration number
lambdas = []; % stores lambda values per iteration
costs = []; % stores cost function values per iteration
k_vals = []; % stores iteration numbers per iteration

while grad > constraint
    % evaluate f(x^(k))
    f_vals = zeros(L, 1);
    f_vals(:, 1) = sqrt( sum((loc_total(E(:, 1), :) ...
        - loc_total(E(:, 2), :)).^2, 2) ) - rho(:, 1);

    % compute derivative matrix
    df = zeros(L, 2*M);
    for i = 1:L
        % 1) both nodes known, i.e. entire row is zero
        if (E(i, 1)) > M && (E(i, 2)) > M
            continue
        % 2) node 1 known
        elseif (E(i, 1)) > M
            uv_i = pos_anchor(E(i, 1) - M, :);
            uv_j = pos_free(E(i, 2), :);
            temp = 1 / (sqrt( sum((uv_i - uv_j).^2) ));

            % input derivatives w/ respect to node 2
            df(i, E(i, 2)) = temp * -(uv_i(:, 1) - uv_j(:, 1));
            df(i, E(i, 2) + M) = temp * -(uv_i(:, 2) - uv_j(:, 2));
        % 3) node 2 known
        elseif (E(i, 2)) > M % node 2 is known
            uv_i = pos_free(E(i, 1), :);
            uv_j = pos_anchor(E(i, 2) - M, :);
            temp = 1 / sqrt( sum((uv_i - uv_j).^2) );
```

```
        % input derivatives w/ respect to node 1
        df(i, E(i, 1)) = temp * (uv_i(:, 1) - uv_j(:, 1));
        df(i, E(i, 1) + M) = temp * (uv_i(:, 2) - uv_j(:, 2));
    % 4) neither node known
    else
        uv_i = pos_free(E(i, 1), :);
        uv_j = pos_free(E(i, 2), :);
        temp = 1 / sqrt( sum((uv_i - uv_j).^2) );

        % input derivatives w/ respect to node 1 and node 2
        df(i, E(i, 1)) = temp * (uv_i(:, 1) - uv_j(:, 1));
        df(i, E(i, 1) + M) = temp * (uv_i(:, 2) - uv_j(:, 2));
        df(i, E(i, 2)) = temp * -(uv_i(:, 1) - uv_j(:, 1));
        df(i, E(i, 2) + M) = temp * -(uv_i(:, 2) - uv_j(:, 2));
    end
end

% compuate iteration gradient and iteration gradient magnitude
temp = 2 * df' * f_vals;
grad_temp = norm(temp, 1);

% compute x_hat using backslash operator
x_hat = x_k - (df' * df + lambda * eye(2 * M)) \ (temp / 2);

% store the potential new coordinates
pred_E = [x_hat(1:M, 1), x_hat(1+M:size(x_hat, 1))];
tmp_loc = [pred_E; pos_anchor];

% evaluate f(x_hat)
f_x_vals = zeros(L, 1);
f_x_vals(:, 1) = sqrt( sum((tmp_loc(E(:, 1), :) ...
    - tmp_loc(E(:, 2), :)).^2, 2) ) - rho(:, 1);

% store cost function, lambda parameter, and iteration number values
lambdas(end + 1) = lambda;
costs(end + 1) = norm(f_vals)^2;
k_vals(end + 1) = k;

% update parameters on next iteration
if norm(f_x_vals)^2 < norm(f_vals)^2
    x_k = x_hat;
    lambda = lambda * beta1;

    pos_free = [x_k(1:M), x_k(1+M:2*M)];
    loc_total = [pos_free; pos_anchor];
else
    lambda = lambda * beta2;
end
```

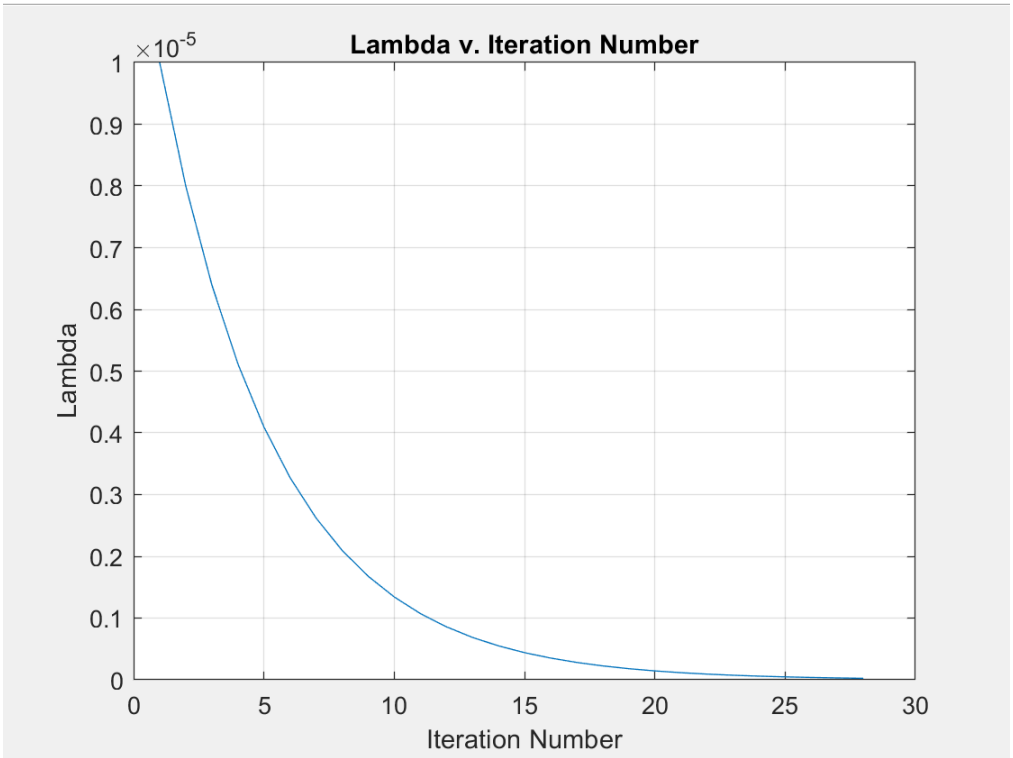
```
    % update iteration number and iteration gradient magnitude
    k = k + 1;
    grad = grad_temp;
end

%% Cost Function & Lambda v. Iteration
plot(k_vals, lambdas); title("Lambda v. Iteration Number"); grid on;
xlabel("Iteration Number"); ylabel("Lambda"); hold off;
figure;
plot(k_vals, costs); title("Cost Function v. Iteration Number"); grid on;
xlabel("Iteration Number"); ylabel("Cost Function");
figure;

%% Sensor Coordinates (Utilized in Calling Script)
% scatter(pos(1:N-K, 1), pos(1:N-K, 2), "go", "filled"); hold on;
% scatter(pos_free(:, 1), pos_free(:, 2), "bo");
% scatter(pos_anchor(:, 1), pos_anchor(:, 2), "s", "ko", "filled");
%
% for i=1:N-K
%     plot([pos_free(i, 1), pos(i, 1)], [pos_free(i, 2), pos(i, 2)], "k");
% end
%
% title("Sensor Coordinate Estimates (N=100, R=0.4, s=0.05)");
% xlabel("u"); ylabel("v"); grid on;
% xlim([-0.2, 1.2]); ylim([-0.2, 1.2]);
```

5 Levenberg-Marquardt Example ($N = 100$, $R = 0.4$, $s = 0.05$)

(a) **Solution:** Below is the plot of the estimated unknown sensor node coordinates (N.B. Anchors are plotted as black squares, actual coordinates are plotted as filled green circles, and predicted coordinates are plotted as blue hollow circles):



(b) **Solution:** Below (next page) is the plot of the regularization parameter λ values versus iteration number (for the same example as part (a)):



(c) **Solution:** Below is the plot of the cost function values versus iteration number (for the same example as part (a) and part (b)):