Homework 6: Nonlinear Systems and Optimization

CS 205A: Mathematical Methods for Robotics, Vision, and Graphics (Winter 2018)
Stanford University
Due Thursday, March 1, 11:59pm

Textbook problems: 8.7 (20 points), 9.2 (20 points), 9.3 (12 points), 9.5 (18 points)
9.3a erratum: the denominator of the last term should be $2(x_2 - x_1)(y_2 - y_3) - 2(x_2 - x_3)(y_2 - y_1)$.

Julia Programming: (30 points) Let's model a horizontal elastic string pinned at anchor points $(0, 0)$ and $(1, 0)$. We can model this string as $N + 1$ consecutive springs connecting N freely-moving interior nodes, and each node j is located at $\vec{x}_j = (x_j, y_j)$. The rest length L_S of each spring is set to $\frac{1}{N+1}$ so that when there are no external forces, the string is entirely loose, yet straight with no extra slack. Under gravity, the string will stretch downward into a catenary shape. We can express a vector containing all the forces, $\vec{f} = (\vec{f}_1, ..., \vec{f}_N)$, as a function of all the interior node positions, $\vec{x} = (\vec{x}_1, ..., \vec{x}_N)$:

$$\vec{f}_j = -k\left[(\vec{x}_{j(j-1)} - L_S\hat{x}_{j(j-1)}) + (\vec{x}_{j(j+1)} - L_S\hat{x}_{j(j+1)})\right] + m\vec{g},$$

$$\text{where } \vec{x}_{ij} = \vec{x}_i - \vec{x}_j, \hat{x}_{ij} = \frac{\vec{x}_{ij}}{\|\vec{x}_{ij}\|_2}, \vec{x}_0 = \langle 0, 0\rangle, \vec{x}_{N+1} = \langle 1, 0\rangle, \text{ and } \vec{g} = \langle 0, -9.8\rangle.$$

This function has been implemented for you in the starter code, and your goal is to find the equilibrium configuration $\vec{x}$ such that $\vec{f}(\vec{x}) = \vec{0}$.

Do the following:

J1. (10 points) Implement Newton's method, using `ForwardDiff.jacobian()` to retrieve the Jacobian at every step. Test it on 50 and 100 interior nodes, and run each configuration for 10 iterations. Plot both the final configuration and the convergence curve, for 50 and 100 interior nodes, and report the runtimes. (The plotting and runtime reporting are implemented for you in the starter code. Implement Newton's method in the provided space.)

J2. (10 points) Broyden's method can avoid calling `jacobian()` every iteration. Implement Broyden's method, initializing J_0 to the identity. Test this on 10, 50, and 100 interior nodes, and run each configuration for enough iterations until it converges. (Note: expect to spend several minutes per run for $N = 100$.) Plot both the final configuration and the convergence curve, for each run, and report the runtimes. Then repeat these runs with a new Broyden implementation that initializes J_0 to the first result from `ForwardDiff.jacobian()`.

J3. (10 points) The Broyden Inverse Method allows us to update the inverse Jacobian directly, instead of solving a linear system at every iteration. Implement the Broyden Inverse Method, initializing J_{inv} to the identity. Test this on 10, 50, and 100 interior nodes, and run for enough

iterations in each run until it converges. Plot both the final configuration and the convergence curve, for each run, and report the runtime. Then repeat these runs with a new implementation that initializes J_{inv} to the inverse of `ForwardDiff.jacobian()`.