CE191: Civil and Environmental Engineering Systems Analysis

Lab 5: Nonlinear Optimization

Due: Wednesday 12/09/09 1:00pm

Professor A. Bayen *Fall 09*

1 Lab Overview

In this lab, you will see how nonlinear optimization techniques you have learned can be applied in practice. Since this is the last lab of the class, you will apply it to a linear program, for which you will express the constraints using a logarithmic barrier. In Section 2 you will transform a constrained linear program into an unconstrained nonlinear optimization problem, and compute its gradient and Hessian. In Section 3, you will implement Newton's method with a backtracking line search. Finally, in Section 4 , you will analyze the performance of the algorithm. You may find MATLAB's Symbolic Math Toolbox helpful during this lab. For some examples of how to use symbolic math in MATLAB, see the `sym_example.m` script, available on the bSpace site.

2 Preliminary Computations

Question 2.1 Let us consider the following linear program:

$$\begin{array}{ll} \textbf{max:} & x_1 + 2x_2 \\ \textbf{s.t.} & x_1 \geq 0 \\ & x_2 \geq 0 \\ & x_1 + x_2 \leq 3 \\ & x_1 \geq x_2 - 1 \end{array} \tag{1}$$

Using what you have learned since lecture 1 of CE191, solve this problem graphically, explaining your solution briefly. Write the matrix A, the vectors b and c so that you

can solve the problem using MATLAB's `linprog(c,A,b)`. Check that your result is in agreement with your graphical solution.

Question 2.2 Rewrite the linear program of question 2.1 as seen in class, i.e. as an unconstrained optimization program, in which you have put the constraints in the objective function, in logarithmic form. Your answer should be in the following form:

$$\begin{array}{ll} \textbf{min:} & (\cdots) - \varepsilon \log(\cdots) - \varepsilon \log(\cdots) \cdots - \varepsilon \log(\cdots) \\ \textbf{s.t.} & \text{no constraints} \end{array} \tag{2}$$

Question 2.3 Compute the gradient of the augmented cost function, i.e. if you call $\mathcal{J}(x_1, x_2) = (\cdots) - \varepsilon \log(\cdots) - \varepsilon \log(\cdots) \cdots - \varepsilon \log(\cdots)$ where $\mathcal{J}(x_1, x_2)$ is the objective function derived in the previous question, compute its partial derivatives. Your answer should be in the following form:

$$\frac{\partial \mathcal{J}}{\partial x_1} = (\cdots)$$
$$\frac{\partial \mathcal{J}}{\partial x_2} = (\cdots)$$

Question 2.4 Compute the Hessian matrix of the augmented cost function, as derived in class. Your answer should be in the following form:

$$\begin{pmatrix} \frac{\partial^2 \mathcal{J}}{\partial x_1^2} & \frac{\partial^2 \mathcal{J}}{\partial x_1 \partial x_2} \\ \frac{\partial^2 \mathcal{J}}{\partial x_2 \partial x_1} & \frac{\partial^2 \mathcal{J}}{\partial x_2^2} \end{pmatrix} = \begin{pmatrix} (\cdots) & (\cdots) \\ (\cdots) & (\cdots) \end{pmatrix} \tag{3}$$

Question 2.5 Given a matrix

$$H = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{4}$$

Let us call H_1 the following matrix:

$$H_1 = \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} \tag{5}$$

Compute $H \cdot H_1$. What is H^{-1}, i.e. the inverse of H? Your answer should be in the following form:

$$H^{-1} = \begin{pmatrix} (\cdots) & (\cdots) \\ (\cdots) & (\cdots) \end{pmatrix} \tag{6}$$

3 Implementation of Newton's Method

Question 3.1 Using all previous results, implement a descent method with logarithmic barrier, as seen in class. For this, you will need to break this question into several tasks. In particular, you will need to create an outer loop (weight ε in front of the log barrier), and an inner loop (Newton + backtracking). In the inner loop, you will need to compute the Newton step or search direction, i.e. the vector

$$\Delta x_{\text{nt}} = -H^{-1}(x_1, x_2)\nabla\mathcal{J}(x_1, x_2)$$

and then implement backtracking to determine the step size t.

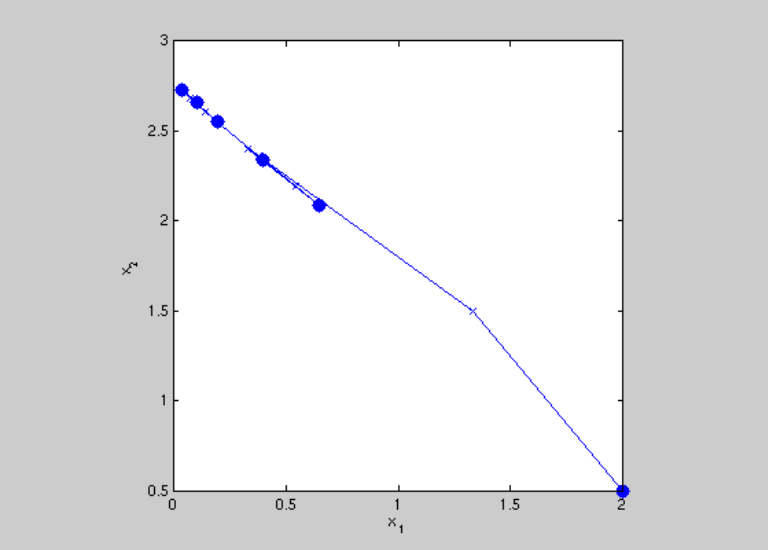
Hints:

- Two parameters (α and β) will be needed to implement backtracking. While you can choose any value within the bounds given in the lecture notes, $\alpha = 0.25$ and $\beta = 0.9$ have been tested successfully on the present example.
- You may find it useful to write $\mathcal{J}(x_1, x_2)$, $\nabla\mathcal{J}(x_1, x_2)$, and $H^{-1}(x_1, x_2)$, as a function of x_1, x_2 and ε using MATLAB's symbolic math functions, then using `subs` to find numerical values for these functions.

4 Analysis

Question 4.1 Run your code, starting from the guess $(x_1^0, x_2^0) = (2, 0.5)$. Run it a second time, starting from a different guess.

For both runs, plot the successive points (x_1, x_2) that the algorithm goes through. Make sure to plot a line between consecutive points, so that the progression of the algorithm clearly appears. Optional: to make the plot more interesting, you can plot the first point of each outer loop iteration in a different manner. For example, on the following plot, the first point of each outer loop was drawn as a large dot, while all other points are drawn as `x`'s. Note that this example does not correspond to the optimization problem of the lab, so it is a good sign if your results are different!



Example of progression of the algorithm, starting at guess (2,0.5)

Hints:

- To draw a large dot at $(x_1, x_2) = (1, 2)$, you can use the following command:
  `plot(1,2,'.k','markersize',30)`
  The `.` indicates that a dot is to be drawn, and `k` specifies it is black. The command `markersize`, followed by the number 30, specifies the size of the dot.

- To draw a line between $(x_1^1, x_2^1) = (1, 2)$ and $(x_1^2, x_2^2) = (4, 5)$, as well as draw an `x` at each one of these two points, you can use the following command:
  `plot([1,4],[2,5],'x-k')`
  The `x` indicates that `x` has to be drawn, `-` that a line has to be drawn, and `k` specifies its color (black). Note the order of the coordinates of the two points: first a vector with the coordinates of both points along the horizontal axis, then a vector with the coordinates of both points along the vertical axis.

- If you put a `hold on` statement at the beginning of your code, you can add points and lines to your plot one by one. You can then just add `plot` statements into your loops and the plot will get built as your loops run.