ChE 597 Computational Optimization

Homework 2 Solutions

January 15th, 2024

1. Consider the following function

$$f(x_1, x_2, x_3) = x_3 \log\left(e^{\frac{x_1}{x_3}} + e^{\frac{x_2}{x_3}}\right) + (x_3 - 2)^2 + e^{\frac{1}{x_1+x_2}}$$

Where the function $f : \mathbb{R}^3 \mapsto \mathbb{R}$ has its domain as **dom** $f : \{\mathbf{x} \in \mathbb{R}^3 : x_1 + x_2 > 0, x_3 > 0\}$ and log is the natural log.

(a) Show that this function is convex

(b) Implement different numerical methods in Python to optimize the given convex function

i. Gradient descent with backtracking line search. Use $t_{init} = 1, \alpha = 0.4, \beta = \frac{1}{2}$

ii. Newton's method (use same values for line search as part i)

Hints:
1. You can either calculate the gradient/Hessian analytically or use the finite difference method, i.e., $f'(x) = \frac{f(x+h)-f(x)}{h}$ where h is a small number like 10^{-5}. We suggest using numpy in Python for the implementation. You need to learn how to create a vector/matrix, vector-matrix production, and matrix inversion. `https://numpy.org/doc/stable/user/quickstart.html`
2. Be **cautious of the domain** while implementing the numerical methods.
Take tolerance of 10^{-5} for the gradient norm wherever necessary.

Solution: For the given function $f(x) : \mathbb{R}^3 \mapsto \mathbb{R}$, we first check for the convexity:

a. We write $f(x) = f_1(x) + f_2(x) + f_3(x)$, and evaluate the convexity of the individual function, knowing that sum of convex functions is convex.

- $f_1(x) = x_3 \log\left(e^{\frac{x_1}{x_3}} + e^{\frac{x_2}{x_3}}\right)$

  The function $f_1(x)$ can be written is terms of perspective of function $g(x) = \log\left(e^{x_1} + e^{x_2}\right)$, where $g(x) : \mathbb{R}^2 \mapsto \mathbb{R}$ and $f_1(x) : \mathbb{R}^{(2+1)} \mapsto \mathbb{R}$ as:

  $f_1(x) = x_3 g\left(\frac{x}{x_3}\right)$ with domain
  dom $f_1(x) = \{(x, x_3) \mid x/x_3 \in \textbf{dom } g, x_3 > 0\}$
  Now, since $g(x)$ is a log-sum-exp function, thus is convex and because perspective operation preserves convexity, making $f_1(x)$ convex.

  Elaborating the convexity of Log-Sum-Exp function:
  The Hessian of the log-sum-exp function is given by:
  $\nabla^2 f(x) = \frac{1}{(\mathbf{1}^T z)^2}\left(\left(\mathbf{1}^T z\right) \text{diag}(z) - zz^T\right)$
  where $z = (e^{x_1}, \ldots, e^{x_n})$. To verify that $\nabla^2 f(x) \succeq 0$, we must show that for all v,

$v^T\nabla^2 f(x)v \geq 0$, i.e.,

$$v^T\nabla^2 f(x)v = \frac{1}{(\mathbf{1}^T z)^2}\left(\left(\sum_{i=1}^{n} z_i\right)\left(\sum_{i=1}^{n} v_i^2 z_i\right) - \left(\sum_{i=1}^{n} v_i z_i\right)^2\right) \geq 0$$

But this follows from the Cauchy-Schwarz inequality $\left(\sum_{i=1}^n a_i b_i\right)^2 \leq \left(\sum_{i=1}^n a_i^2\right)\left(\sum_{i=1}^n b_i^2\right)$ applied to the vectors with components $a_i = v_i\sqrt{z_i}$ and $b_i = \sqrt{z_i}$.

- $f_2(x) = (x_3 - 2)^2$, is a quadratic function with positive x^2 coefficient, thus is convex.
- $f_3(x) = e^{\frac{1}{x_1+x_2}}$, has both of its eigen-values non-negative for its Hessian given the domain: $\{x_1 + x_2 > 0\}$; making it is convex (check yourself!).

Thus, we conclude that $f(x)$ is convex.

b. Optimizing the function using:

i. Gradient descent with backtracking line search

Algorithm: Backtracking Line Search
Given a descent direction Δx for f at $x \in \mathbf{dom} f, \alpha \in (0, 0.5), \beta \in (0, 1)$.
Set $t := 1$.
While $f(x + t\Delta x) > f(x) + \alpha t \nabla f(x)^T \Delta x$, set $t := \beta t$.

The descent is implemented as:

Input: Starting point $x \in \operatorname{dom} f$
repeat
  $\Delta x := -\nabla f(x)$;
  Line search: Choose step size t via backtracking line search;
  Update: $x := x + t\Delta x$;
until *stopping criterion is satisfied*;

Stopping Criteria: Tolerance of $\varepsilon = 10^{-5}$ for the norm of gradient.

The above stated implementation is in general true when the domain of function is defined on all of $\mathbb{R}^3$, but since we have certain restrictions on domain, we first have to multiply t by β until $x + t\Delta x \in \operatorname{dom} f$ and then we start to check whether the inequality $f(x + t\Delta x) \leq f(x) + \alpha t \nabla f(x)^T \Delta x$ holds.

Thus, the practical implementation of the algorithm would be:
Algorithm: Backtracking Line Search
Given a descent direction $\Delta x = -\nabla f(x)$ for f at $x \in \mathbf{dom} f, \alpha \in (0, 0.5), \beta \in (0, 1)$.
Set $t := 1$.
While $x + t\Delta x \notin \mathbf{dom} f$, set $t := \beta t$
While $f(x + t\Delta x) > f(x) + \alpha t \nabla f(x)^T \Delta x$, set $t := \beta t$.

Finding the Hessian analytically for such a function can be tedious. However, calculating the gradient is fairly straightforward, and is given as:

$\nabla f = [\partial f/\partial x_1 \; \partial f/\partial x_2 \; \partial f/\partial x_3]^T$

$$\implies \nabla f = \begin{Bmatrix} \frac{e^{\frac{x_1}{x_3}}}{e^{\frac{x_1}{x_3}}+e^{\frac{x_2}{x_3}}} - \frac{e^{\frac{1}{x_1+x_2}}}{(x_1+x_2)^2} \\ \\ \frac{e^{\frac{x_2}{x_3}}}{e^{\frac{x_1}{x_3}}+e^{\frac{x_2}{x_3}}} - \frac{e^{\frac{1}{x_1+x_2}}}{(x_1+x_2)^2} \\ \\ log(e^{\frac{x_1}{x_3}}+e^{\frac{x_2}{x_3}}) - \frac{x_1 e^{\frac{x_1}{x_3}}+x_2 e^{\frac{x_2}{x_3}}}{x_3(e^{\frac{x_1}{x_3}}+e^{\frac{x_2}{x_3}})} + 2(x_3-2) \end{Bmatrix}$$

Implementation:

- We take starting points as $\mathbf{x} = [3;4;5]^T$ and follow above stated algorithms.

The link to the Jupyter notebook is:
`https://github.com/li-group/ChE-597-Computational-Optimization/blob/main/HW%202/HW2%20Q1a.ipynb` **ii. Newton's method** (damped/guarded, $t \neq 1$)

Algorithm:

Input: Starting point $x \in \text{dom} f$, tolerance $\varepsilon > 0$

Repeat:

(a) Compute the Newton step and decrement.
$\Delta x_{nt} := -\nabla^2 f(x)^{-1}\nabla f(x)$; $\lambda^2 := \nabla f(x)^T \nabla^2 f(x)^{-1}\nabla f(x)$.

(b) Stopping criterion. **quit** if $\lambda^2/2 \leq \varepsilon$.

(c) Line search. Choose step size t by backtracking line search; ensuring update $:= x + t\Delta x$ lies in **dom**f throughout.

(d) Update. $x := x + t\Delta x_{nt}$.

We follow what is essentially the general descent method described using the Newton step as search direction. The only minor difference is that the stopping criterion is checked after computing the search direction, rather than after the update.

Implementation:

- We take starting points as $\mathbf{x} = [3;4;5]^T$.
- We calculate the Hessian wherever required numerically using the forward difference with the help of $\nabla f(x)$ function (Note: $\nabla^2 f(x)$ would be 3x3 matrix i.e. difference w.r.t. each variable)
- Further, it is worth mentioning that the direction of search used in backtracking line search here would be taken as $\Delta x = -\nabla^2 f(x)^{-1}\nabla f(x)$, in contrast to the previous part wherein gradient descent $\Delta x = -\nabla f(x)$ was used.

The link to the Jupyter notebook is:

Optima Achieved: Due to the symmetricality of the function w.r.t. x_1 and x_2 i.e. $f(x_1,x_2,x_3) = f(x_2,x_1,x_3)$, it can be said by observation that any optimum value would be

a point where $x_1 = x_2$. We indeed get such a optimum point:
$x_{opt} = [0.924; 0.924; 1.653]$ with $f_{opt} = 3.908$

`https://github.com/li-group/ChE-597-Computational-Optimization/blob/main/HW%202/HW2%20Q1b.ipynb`

2. Optimize the Problem 1 in Python using one of the Quasi-Newton methods mentioned in Lecture 3, namely BFGS (Broyden-Fletcher-Goldfarb-Shanno) method.

Solution: We follow the directions as given in the tutorial to implement the below algorithm for the pupose of optimisation:

BFGS Algorithm: Inverse Hessian Approximation

(a) Given starting point $x_0 = [2,3,5]^T$, convergence tolerance $\varepsilon > 0$, starting matrix H_0: Identity Matrix;
$k \leftarrow 0$;

(b) While $\|\nabla f_k\| > \varepsilon$:

   i. Compute search direction by solving:

$$p_k = -H_k \nabla f_k$$

   ii. Set $x_{k+1} = x_k + \alpha_k p_k$ where α_k is computed from a line search procedure;

   iii. Define $s_k = x_{k+1} - x_k$ and $y_k = \nabla f_{k+1} - \nabla f_k$;

   iv. Compute H_{k+1} using BFGS;

   v. $k \leftarrow k+1$;

Where,

$$H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T$$

and $\rho_k = \frac{1}{y_k^T s_k}$.

Finally, as mentioned in the implementation note to set ρ_k as a constant after $y_k^T s_k$ gets smaller than some certain ε say 10^{-5}. We use $\varepsilon = 10^{-4}$ and set the value for $\rho_k = 10^4$.

The link to the Jupyter notebook is:

`https://github.com/li-group/ChE-597-Computational-Optimization/blob/main/HW%202/HW2%20Q2.ipynb`

3. Convert the following LP to one in standard form. Write the result in matrix-vector form, giving x, c, A, b (corresponding to the general standard form discussed in lecture).

$$\text{Minimize: } z = 2x_1 + 3x_2 - x_3 + 4x_4 + x_5$$

Subject to:

$$\begin{aligned}
x_1 - x_2 + 2x_3 &\leq 5 \\
3x_1 + 2x_2 + x_4 &= 10 \\
2x_3 - x_5 &\geq 7 \\
2x_1 + 20x_4 + x_5 &\leq 15 \\
x_1, x_3, x_5 &\geq 0 \\
x_2 \text{ and } x_4 &\text{ are URS}
\end{aligned}$$

Solution: In order to standardise, we convert the inequality constraints to equality by introducing slack variables:
The deficit equations are equalised by adding some $s_i \geq 0$, and on the other hand, the surplus ones are equalised by subtracting some $y_i \geq 0$.
The URS variables can be written as:
$x_2 = x_2^+ - x_2^-$; $x_2^+ \geq 0$ and $x_2^- \geq 0$ and,
$x_4 = x_4^+ - x_4^-$; $x_4^+ \geq 0$ and $x_4^- \geq 0$

Thus, in our problem we introduce x_2^+, x_2^-, x_4^+ and x_4^-, s_1, s_2 and y_1 to modify the constraints as:

$$\begin{aligned}
x_1 - x_2^+ + x_2^- + 2x_3 + s_1 &= 5 \\
3x_1 + 2x_2^+ - 2x_2^- + x_4^+ - x_4^- &= 10 \\
2x_3 - x_5 - y_1 &= 7 \\
2x_1 + 20x_4^+ - 20x_4^- + x_5 + s_2 &= 15 \\
x_1, x_2^+, x_2^-, x_3, x_4^+, x_4^-, x_5, s_1, s_2, y_1 &\geq 0
\end{aligned}$$

The A matrix is then given as:

$$A = \begin{bmatrix} 1 & -1 & 1 & 2 & 0 & 0 & 0 & 1 & 0 & 0 \\ 3 & 2 & -2 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & -1 & 0 & 0 & -1 \\ 2 & 0 & 0 & 0 & 20 & -20 & 1 & 0 & 1 & 0 \end{bmatrix}$$

For the $\mathbf{x} = \begin{bmatrix} x_1 & x_2^+ & x_2^- & x_3 & x_4^+ & x_4^- & x_5 & s_1 & s_2 & y_1 \end{bmatrix}^T$

and $b = \begin{bmatrix} 5 \\ 10 \\ 7 \\ 15 \end{bmatrix}$

Finally, objective function $z = 2x_1 + 3x_2 - x_3 + 4x_4 + x_5$ becomes $z = 2x_1 + 3x_2^+ - 3x_2^- - x_3 + 4x_4^+ - 4x_4^- + x_5$

which can be written as $z = c^T x$ for: $c = \begin{bmatrix} 2 & 3 & -3 & -1 & 4 & -4 & 1 & 0 & 0 & 0 \end{bmatrix}^T$

The system can be compactly written as:

$$\min \; c^T x$$

subject to:

$$\begin{aligned} &Ax = b \\ &x \geq 0 \end{aligned}$$

4. Solve the multi-commodity flow problem discussed in the slides (Lecture 4) using Pomo.

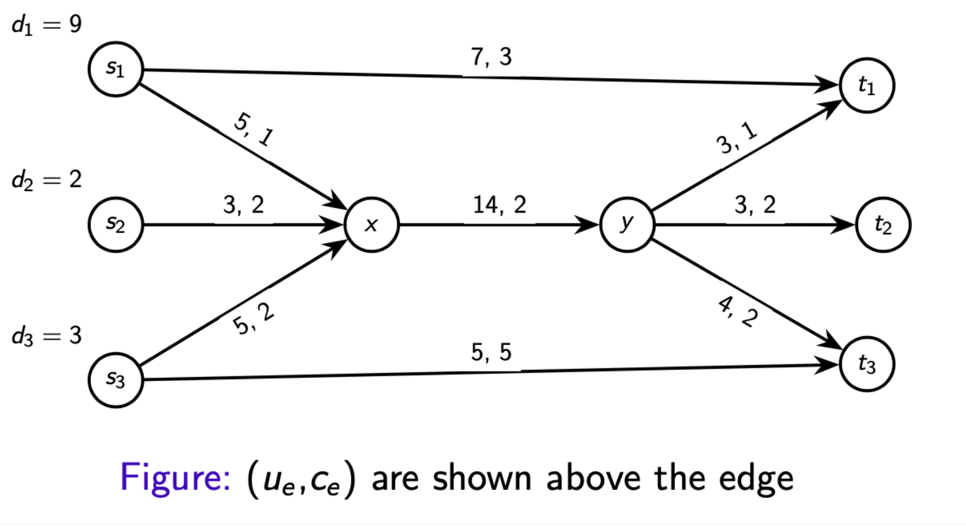

Figure 4: Directed Graph

Recall: We were given the network as above represented as a directed graph $G = (V, E)$ with multiple commodities needed to be transported across the network. Each edge $e \in E$ has a capacity u_e and a cost c_e per unit commodity and each commodity k has a demand d_k from a source s_k to a sink t_k.

Solution: We use the formulation as discussed in the lecture:

Variables

- Let $x_{e,k} \geq 0$ represent the flow of commodity k through edge e

Constraints:

- Capacity constraints: $\sum_k x_{e,k} \leq u_e$ for all $e \in E$
- Flow conservation: For each node v and each commodity k,

$$\sum_{(u,v)\in E} x_{u,v,k} - \sum_{(v,w)\in E} x_{v,w,k} = \begin{cases} d_k, & \text{if } v = s_k \\ -d_k, & \text{if } v = t_k \\ 0, & \text{otherwise} \end{cases}$$

Objective Function:

- minimize the total cost: $\min \sum_{e\in E} \sum_k c_e x_{e,k}$

The achieved minimum cost with the given constraints comes out to be: 56.

The link to the Jupyter notebook is:

`https://github.com/li-group/ChE-597-Computational-Optimization/blob/main/HW%202/HW2%20Q4.ipynb`

5. Develop the LP model for the optimal control of the CSTR.

Solution: According to the problem statement, the optimization model can be derived as

$$\begin{aligned}
\min_{u(0),u(1),u(2)} \quad & \sum_{k=0}^{k=3} (|y_1(k)| + |y_2(k)|) \\
s.t. \quad & x(k+1) = Ax(k) + Bu(k) \quad k = 0,1,2 \\
& y(k) = Cx(k) \quad k = 0,1,2,3 \\
& x(0) = \begin{bmatrix} -0.03 \\ 0 \\ 0.3 \end{bmatrix} \\
& \begin{bmatrix} -0.05 \\ -5 \\ -0.5 \end{bmatrix} \leq x(k) \leq \begin{bmatrix} 0.05 \\ 5 \\ 0.5 \end{bmatrix} \quad k = 0,1,2,3 \\
& \begin{bmatrix} -10 \\ -0.05 \end{bmatrix} \leq u(k) \leq \begin{bmatrix} 10 \\ 0.05 \end{bmatrix} \quad k = 0,1,2
\end{aligned}$$

Then, the LP formulation can be obtained by introducing new variable $z_1(k)$ and $z_2(k)$ as follows

$$\begin{aligned}
\min_{u(0),u(1),u(2)} \quad & \sum_{k=0}^{k=3} (z_1(k) + z_2(k)) \\
s.t. \quad & x(k+1) = Ax(k) + Bu(k) \quad k = 0,1,2 \\
& y(k) = Cx(k) \quad k = 0,1,2,3 \\
& x(0) = \begin{bmatrix} -0.03 \\ 0 \\ 0.3 \end{bmatrix} \\
& \begin{bmatrix} -0.05 \\ -5 \\ -0.5 \end{bmatrix} \leq x(k) \leq \begin{bmatrix} 0.05 \\ 5 \\ 0.5 \end{bmatrix} \quad k = 0,1,2,3 \\
& \begin{bmatrix} -10 \\ -0.05 \end{bmatrix} \leq u(k) \leq \begin{bmatrix} 10 \\ 0.05 \end{bmatrix} \quad k = 0,1,2 \\
& -z_1(k) \leq y_1(k) \leq z_1(k) \quad k = 0,1,2,3 \\
& -z_2(k) \leq y_3(k) \leq z_2(k) \quad k = 0,1,2,3
\end{aligned}$$

The link to the Jupyter notebook is:

`https://github.com/li-group/ChE-597-Computational-Optimization/blob/main/HW%202/HW2%20Q5.ipynb`