ChE 597 Computational Optimization

Homework 2

Due January 28th 11:59 pm

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** $f : \{\mathbf{x} \in \mathbb{R}^3 : x_1 + x_2 > 0, x_3 > 0\}$

(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.

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.

The below tutorial attempts to aid in understanding the method,
Quasi-Newton Methods: For the unconstrained, smooth convex optimization
$\min_x f(x)$
Where f is convex, twice differentable, and domain = $\mathbf{dom} f$.

Earlier we had seen:

	Algorithm
Gradient-Descent	$x^+ = x - t\nabla f(x)$
Newton's Method	$x^+ = x - t(\nabla^2 f(x))^{-1}\nabla f(x)$ or $x^+ = x - ts$ Where two main steps are to, Compute Hessian $\nabla^2 f(x)$ Solve the system $\nabla^2 f(x)s = -\nabla f(x)$
Now, Quasi-Newton	$x^+ = x - ts$ Where the direction s is defined by the linear system $Bs = -\nabla f(x)$ for some approximation B of $\nabla^2 f(x)$

With the information so far, we now state a Quasi-Newton template:

Let $x^{(0)} \in \mathbf{dom} f$, $B^{(0)} \succ 0$. For $k = 1, 2, 3, \ldots$, repeat:
1. Solve $B^{(k-1)}s^{(k-1)} = -\nabla f(x^{(k-1)})$
2. Update $x^{(k)} = x^{(k-1)} + t_k s^{(k-1)}$
3. Compute $B^{(k)}$ from $B^{(k-1)}$

Where t_k is calculated as per Backtracking-Line Search.

The highlight of the Quasi-Newton method is to evaluate $B^{(k)}$ from $B^{(k-1)}$, with different quasi-newton methods implementing this step differently.

At this point, we take a brief detour to discuss the linear-algebra of working on a general matrix with help of some well-structured or a closely related known matrix.
Let's say we have a matrix A which has some desirable structure, but we actually want to do computations on matrix B, in such a case, we write matrix B in terms of A and some low rank update (rank $k = 1, 2, ..$).
For example, a rank 1 update is written as:

$$B = A + uv^T$$

The expression uv^T represents the outer product of vectors u and v. This operation is often referred to as a low-rank update because uv^T forms a **Low-Rank** matrix, specifically of rank 1. However, it is also possible to generalize this to higher ranks, such as $k > 1$. For concreteness of how such a update helps us for working with matrix B, we take a basic example of solving a linear system $Bx = b$ for x. We exploit the low-rank update of B in terms of A using Sherman-Morrison formula, which tells us that:

$$B^{-1}b = (A + uv^T)^{-1}b = A^{-1}b - \frac{v^T A^{-1} b}{1 + v^T A^{-1} u} A^{-1} u$$

Thus we can complete a solve with B so long as we know how to complete a few solves with A.

The Low-Rank update theory directly helps us with our task of evaluating $B^{(k)}$ from $B^{(k-1)}$. Following updates can be done in order to motivate $B^{(k)}$:

$$\text{Rank 1 update : } B^+ = B + auu^T$$

$$\text{Rank 2 update : } B^+ = B + auu^T + bvv^T$$

The rank 1 update leads to the symmetric rank-one (SR1) update formula and rank 2 gives us the Broyden-Fletcher-Goldfarb-Shanno (BFGS) update formula.

Now we need some kind of mapping for $B^{(k+1)}$ or B^+ in terms of parameters associated with the recent updates other than hessian approximation, for us to use the low-rank update trick leading us to write B^+ in terms of B. We do that by forming the following quadratic model of the objective function at the current iterate x_k:

$$m_k(p) = f_k + \nabla f_k^T p + \frac{1}{2} p^T B_k p$$

Here, B_k is an $n \times n$ symmetric positive definite matrix that will be revised or updated at every iteration. Note that the function value and gradient of this model at $p \approx 0$ match f_k and ∇f_k, respectively. The minimizer p_k of this convex quadratic model, which we can write explicitly as:

$$p_k = -B_k^{-1} \nabla f_k$$

is used as the search direction, and the new iterate is

$$x_{k+1} = x_k + \alpha_k p_k$$

Where the step length is α_k. This iteration is quite similar to the line search Newton method; the key difference is that the approximate Hessian B_k is used in place of the true Hessian. Instead of computing B_k afresh at every iteration, let's generate a new iterate x_{k+1} and construct a new quadratic model, of the form

$$m_{k+1}(p) = f_{k+1} + \nabla f_{k+1}^T p + \frac{1}{2} p^T B_{k+1} p$$

Now we discuss the requirements that should be imposed on B_{k+1}, based on the knowledge gained during the latest step. One reasonable requirement is that the gradient of m_{k+1} should match the gradient of the objective function f at the latest two iterates x_k and x_{k+1}. Since $\nabla m_{k+1}(0)$ is precisely ∇f_{k+1}, the second of these conditions is satisfied automatically. The first condition can be written mathematically as

$$\nabla m_{k+1}(-\alpha_k p_k) = \nabla f_{k+1} - \alpha_k B_{k+1} p_k = \nabla f_k.$$

By rearranging, we obtain

$$B_{k+1} \alpha_k p_k = \nabla f_{k+1} - \nabla f_k.$$

To simplify the notation it is useful to define the vectors

$$s_k = (x_{k+1} - x_k) = \alpha_k p_k, \quad y_k = (\nabla f_{k+1} - \nabla f_k),$$

Which upon substitution in the previous equation leads us to write,

$$B_{k+1} s_k = y_k$$

This formula is known as the *secant equation.*

Further, in addition to the secant equation, we want:

- B^+ to be symmetric
- B^+ to be "close" to B
- $B \succ 0 \Rightarrow B^+ \succ 0$

As mentioned earlier, a rank one update leads to the SR1 update formula. Here, for the sake of brevity we discuss only the rank two update leading to BFGS formula. In general, SR1 is simple and cheap, but has a key shortcoming of not preserving positive definiteness in contrast to BFGS, which makes the latter one widely used and more sophisticated. Now, we move to deriving the formula:

$$B^+ = B + auu^T + bvv^T.$$

The secant equation $y = B^+ s$ yields

$$y - Bs = (au^T s)u + (bv^T s)v$$

Putting $u = y, v = Bs,$ and solving for a, b, we get

$$B^+ = B - \frac{Bss^T B}{s^T Bs} + \frac{yy^T}{y^T s}$$

This update is called the Broyden-Fletcher-Goldfarb-Shanno (BFGS) update (where $+$ means $(k+1)^{th}$ iteration and $(k)^{th}$ iteration otherwise.

Now with the update step known, we can motivate the following optimisation algorithm:

Algorithm 1 BFGS Method

```
Given starting point $x_0$, convergence tolerance $\varepsilon > 0$, starting matrix $B_0$;
$k \leftarrow 0$;
while $\|\nabla f_k\| > \varepsilon$ do
    Compute search direction, by solving;
        $B_k p_k = -\nabla f_k$
    Set $x_{k+1} = x_k + \alpha_k p_k$ where $\alpha_k$ is computed from a line search procedure;
    Define $s_k = x_{k+1} - x_k$ and $y_k = \nabla f_{k+1} - \nabla f_k$;
    Compute $B_{k+1}$ using BFGS;
    $k \leftarrow k + 1$;
end while
```

There is no magic formula that works well for choosing a B_0 in all cases but we can use specific information about the problem, for instance by setting it to an approximate Hessian calculated by finite differences at x_0. Otherwise, we can simply set it to be the identity matrix, or a multiple of the identity matrix, where the multiple is chosen to reflect the scaling of the variables.

Another version of BFGS algorithm which is frequently used is by simply applying Sherman–Morrison formula (as stated earlier) to the derived B^+ update. This leads us to state and proceed by an initial assumption of inverse of the Hessian and without any requirement of solving a linear system in order to find the direction. Here we state the update for the alternate version without providing the derivation:

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

Where, $\rho_k = \frac{1}{y_k^T s_k}$, with s_k and y_k as defined previously.

Final Note on Implementation:

- We recommend implementation of the algorithm using latter version, since the other variant requires the system $B_k p_k = -\nabla f_k$ to be solved for the step p_k thereby increasing the cost of the step computation to $O(n^3)$ making the implementation tricky and not very efficient.
- During the practical implementation of the algorithm in a compiler, the numerical value of $y_k^T s_k$ can get very small (since y_k involves difference of ∇f) which can lead to error of $1/0$ form while calculating ρ_k. Thus, it can be a good idea to set ρ_k as a constant after $y_k^T s_k$ gets smaller than some certain ε say 10^{-5}.
- Take inital $H_0 = I$ (Identity matrix)

For more information, Refer:
Numerical Optimization (Jorge Nocedal and Stephen J. Wright), Springer, 2006. (Chapter 6)

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:

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

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

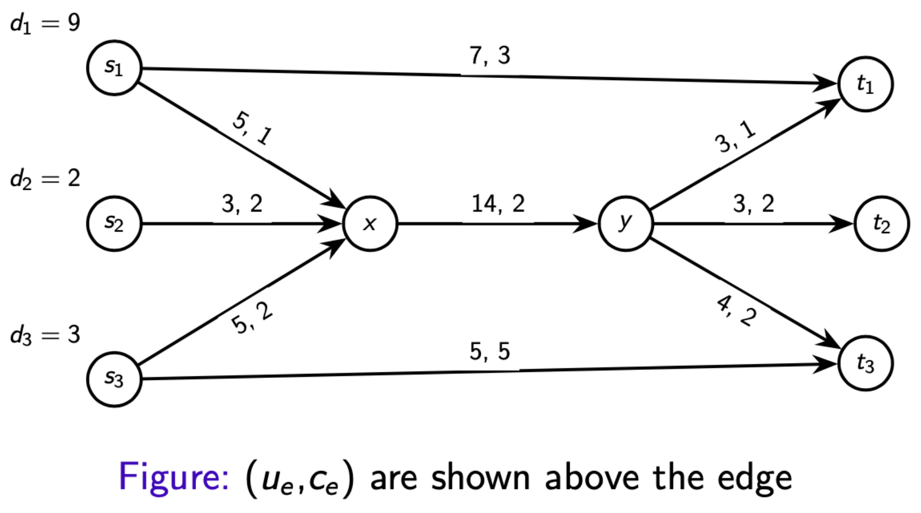

Figure 4: Directed Graph

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.

5. Consider a well-stirred chemical reactor depicted in Figure 5, where an irreversible, first-order reaction $A \to B$ occurs in the liquid phase. The feed to the reactor consists of pure A at flowrate F_0, temperature T_0, and molar concentration c_0. Note that the reactor temperature can be regulated with external cooling with temperature T_c.

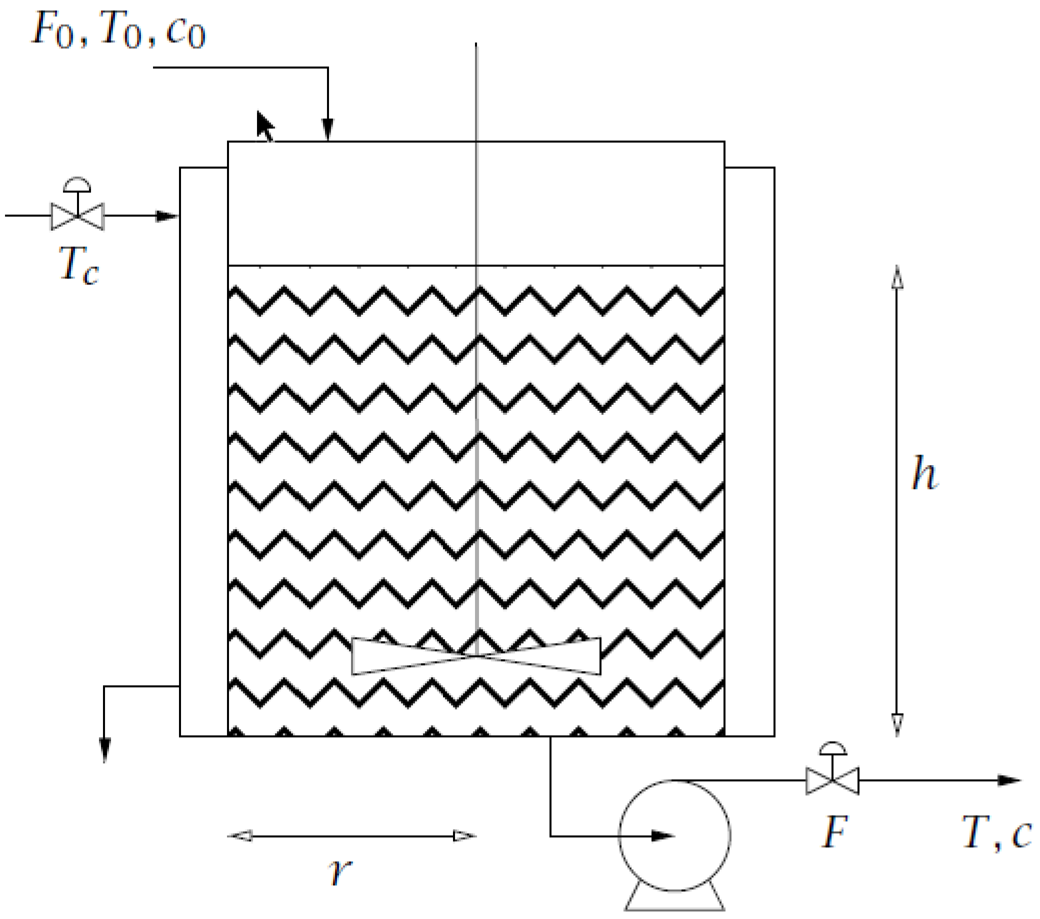

Figure 5: Schematic of the continuous stirred tank reactor.

Based on first principles, the following nonlinear state space model can be obtained which involves the mass and energy balances:

$$
\begin{aligned}
\frac{dc}{dt} &= F_0\left(c_0 - c\right) - k_0 \exp\left(-\frac{E}{RT}\right) c \\
\frac{dT}{dt} &= \frac{F_0\left(T_0 - T\right) - \Delta H}{\pi r^2 h} + \frac{k_0 \exp\left(-\frac{E}{RT}\right) c + \frac{2U}{r\rho c_p}\left(T_c - T\right)}{\pi r^2} \\
\frac{dh}{dt} &= \frac{F_0 - F}{\pi r^2}
\end{aligned}
$$

The controlled variables are the level of the tank, h, and the molar concentration of species A, denoted as c; the additional state variable is the reactor temperature, denoted as T, . While, the manipulated variables are the coolant liquid temperature, T_c, and the outlet flowrate, F. The open-loop stable steady-state operating conditions can be obtained with specified model parameters and are the following

$$c^s = 0.878\,\text{kmol/m}^3 \qquad T^s = 324.5\,\text{K} \qquad h^s = 0.659\,\text{m}$$
$$T_c^s = 300\,\text{K} \qquad F^s = 0.1\,\text{m}^3/\text{min}$$

Using a sampling time of 1 min, a linearized discrete state space model is obtained to approximate the complex process model (i.e., the set of ODE equations). It is also assumed that all the states are measurable without noise. Shifted by the given steady state, the state space variables are

$$x = \begin{bmatrix} c - c^s \\ T - T^s \\ h - h^s \end{bmatrix} \quad u = \begin{bmatrix} T_c - T_c^s \\ F - F^s \end{bmatrix} \quad y = \begin{bmatrix} c - c^s \\ T - T^s \\ h - h^s \end{bmatrix}$$

The corresponding linear model is

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) \end{aligned}$$

in which

$$A = \begin{bmatrix} 0.2681 & -0.00338 & -0.00728 \\ 9.703 & 0.3279 & -25.44 \\ 0 & 0 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} -0.00537 & 0.1655 \\ 1.297 & 97.91 \\ 0 & -6.637 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The goal in this case is to develop an optimization problem to drive the two controlled variables, c and h from the given initial value to the specified constant setpoints. Specifically, the considered time instants are from 0 to 3 (i.e., $k = 0, 1, 2, 3$), and the initial states and the setpoints for the two controlled variables are given as

$$x(k=0) = \begin{bmatrix} -0.03 \\ 0 \\ 0.3 \end{bmatrix} \quad c_{sp} = 0 \quad h_{sp} = 0$$

In this sense, the objective function is to minimize the deviation of the two controlled variables from their setpoints over the time horizon, which can be derived as follows

$$\min \sum_k (|y_1(k) - c_{sp}| + |y_3(k) - h_{sp}|)$$

Also, the bounds for the input and state variables are given as follows

$$\begin{bmatrix} -0.05 \\ -5 \\ -0.5 \end{bmatrix} \le x(k) \le \begin{bmatrix} 0.05 \\ 5 \\ 0.5 \end{bmatrix} \qquad \begin{bmatrix} -10 \\ -0.05 \end{bmatrix} \le u(k) \le \begin{bmatrix} 10 \\ 0.05 \end{bmatrix}$$

Please derive the LP problem for this purpose with the given information of the state space model, time horizon, initial state, setpoints, and bounds. Then, please use Pyomo to solve the problem.