Convex Optimization - Homework 3

Report theoretical results in a pdf or scanned file. Send this file and your code with all requested functions together with a main script reproducing your experiments, collected in a zip file called `lastname-firstname-DM3.zip` to `dm.daspremont@gmail.com`. Please use either Julia, Python or MATLAB.

1 Interior Points Method

The interior point algorithm uses the combination of self-concordant barrier and Newton method to efficiently solve many convex problems. In this homework, we focus on the general quadratic problem

$$\min_x \phi(x) = \frac{1}{2}x^TQx + p^Tx \quad \text{s.t. } Ax \leq b, \tag{QP}$$

where Q is a symmetric semi-definite matrix and $x \in \mathbb{R}^d$. The goal of the interior point algorithm is to transform the constrained problem into

$$\min_x \phi_t(x) = t\left(\frac{1}{2}x^TQx + p^Tx\right) + \mathcal{B}(b - Ax)$$

where $\mathcal{B}(b - Ax)$ is a self-concordant barrier for the set $Ax \leq b$ and t the parameter of the barrier (see lecture notes). In this homework, we will use the logarithmic barrier

$$\mathcal{B}(x) = -\sum_i \log(x_i).$$

Note: The sum of a quadratic function and a self-concordant function is a self-concordant function.

Questions

- Compute $\nabla\phi_t(x)$ and $\nabla^2\phi_t(x)$.
- Implement the functions `phi(x,t,Q,p,A,b)`, `grad(x,t,Q,p,A,b)` and `hess(x,t,Q,p,A,b)` which return respectively the function value, gradient and hessian of $\phi_t(x)$ at point x.

2 Newton Method

For minimizing the function $\phi_t(x)$ we will use the Newton method. However, when the Newton decrements

$$\lambda(x) = (\nabla\phi_t(x))^T(\nabla^2\phi_t(x))^{-1}\nabla\phi_t(x)$$

is too large, the Newton step

$$x_{ns} = x - (\phi_t(x))^{-1}\nabla\phi_t(x)$$

is no longer guaranteed to be feasible. However, it is possible to overcome this problem using a backtracking line-search. We will explore here another solution, called the *damped Newton method*, written

$$x_{dns} = x - \frac{1}{1+\lambda(x)}(\phi_t(x))^{-1}\nabla\phi_t(x). \qquad \text{(dNm)}$$

Questions

- Implement the function `[xnew,gap] = dampedNewtonStep(x,f,g,h)`, which compute the damped Newton step at point x. Assume the argument `f` is a function which takes input x and returns $\phi_t(x)$ (the same for `g` and `h` for the gradient and the hessian). The output `gap` is the estimated gap between $\phi_t(\texttt{xnew})$ and $\min_x \phi_t(x)$ (see lecture notes).
Note: this function should call **one and only one** time `g(x)` and `h(x)`.
- Implement the function `[xstar,xhist] = dampedNewton(x0,f,g,h,tol)` which minimizes the function f starting at `x0` using the damped Newton algorithm. The output `xstar` is the estimated minimum, which satisfies $\phi_t(\texttt{xstar}) - \min_x \phi_t(x) \leq \texttt{tol}$. The output `xhist` contains the history of all Newton steps.
Warning: For theoretical reasons, the parameter `tol` should be **smaller** than $\frac{3-\sqrt{5}}{2}$.
- Implement a function `[xstar,xhist] = newtonLS(x0,f,g,h,tol)` which minimizes the function f starting at `x0` using the Newton algorithm with backtracking line-search (see lecture notes).

Optional questions

- Using the following inequality (true for any self-concordant functions),

$$\phi_t(y) \leq \phi_t(x) + (\nabla\phi_t(x))^T(y-x) + \omega_*\Big((y-x)^T\big(\nabla^2\phi_t(x)\big)(y-x)\Big),$$

where $\omega_*(\tau) = -\tau - \ln(1-\tau)$, show that the damped Newton method (dNm) is optimal.
- Using the same inequality, show that the decrements is at least

$$\phi_t(x) - \phi_t(x_{dns}) \geq \lambda(x).$$

3 Support Vector Machine Problem

In the previous homework, we introduced the Data Separation problem (ex. 3), more commonly known as Support Vector Machine (SVM) problem. Its common formulation is the following. Given n data points $x_i \in \mathbb{R}^d$ with labels $y_i \in \{-1, 1\}$ and a regularization parameter $\tau > 0$, the SVM reads

$$\begin{array}{ll} \text{minimize} & \frac{1}{\tau n}\sum_{i=1}^{n} z_i + \frac{1}{2}\|w\|_2^2 \\ \text{subject to} & y_i(w^T x_i) \geq 1 - z_i, \quad i = 1, \ldots, n \\ & z \geq 0 \end{array} \tag{SVM-P}$$

and its dual is written

$$\begin{array}{ll} \text{maximize} & \mathbf{1}^T\lambda - \frac{1}{2}\left\|\sum_{i=1}^{n} \lambda_i y_i x_i\right\|_2^2 \\ \text{subject to} & 0 \leq \lambda \leq \frac{1}{\tau n} \end{array} \tag{SVM-D}$$

1. Give strictly feasible points for primal and dual.

2. Implement the barrier method (using log barrier) to solve (SVM-P) and (SVM-D) given a data matrix $X = (x_1, \ldots, x_n) \in \mathbb{R}^{d \times n}$, a vector of labels $y \in \{-1, 1\}^n$ and a regularization parameter $\tau > 0$ by

   - coding the generic functions `[Q,p,A,b] = transform_svm_primal(tau,X,y)` and `[Q,p,A,b] = transform_svm_dual(tau,X,y)`, which writes the primal and the dual SVM problem as particular instances of the quadratic problem (QP).
   - coding a function `[x_sol,xhist] = barr_method(Q,p,A,b,x_0,mu,tol)` which implements the barrier method to solve QP given the inputs (Q, p, A, b) and the initial point `x_0` (which should be strictly feasible). The input `mu` is the increment of the barrier at each iteration (see lecture notes). The output `x_sol` must be feasible and satisfies

     $$\phi(\texttt{x\_sol}) - \phi(x^*) \leq \texttt{tol}$$

     where x^* is the solution of (QP). The function also outputs `xhist`, the history of all Newton steps.

3. Use the Iris dataset (https://archive.ics.uci.edu/ml/datasets/iris) to test your code and try to separate the last two classes, i.e., Iris-versicolor versus Iris-virginica. Try different values of τ and measure out-of-sample performance by learning the classifier on 80% of the dataset. *Add one dimension to your data points in order to account for the offset if your data is not centered.*
   Optional: In addition, randomly generate another data set.

4. Plot the duality gap versus Newton iterations (using the damped Newton Method) in semilog-scale for different values of the barrier method parameter $\mu = 2, 15, 50, 100$ and comment the results. Compare the performance with the line-search strategy.

Tips

You can declare *anonymous functions*. It will help you to use the code you have done in Question 1 and 2.

```matlab
% Matlab Implementation
Q=10; p=1; A=2; b=-1; % Declare some parameters
t=0.001; % Set the barrier parameter

% Declare f as an anonymous function which takes one
% input x, and returns phi(x,t,Q,p,A,b)
f = @(x) phi(x,t,Q,p,A,b);
g = @(x) grad(x,t,Q,p,A,b); % same
h = @(x) hess(x,t,Q,p,A,b); % same

% Perform a Damped Newton Step at x=-1.
x = -1;
[x_dns,gap] = dampedNewtonStep(x,f,g,h);
```

```python
# Python Implementation
Q=10; p=1; A=2; b=-1; # Declare some parameters
t=0.001; # Set the barrier parameter

# Declare f as an anonymous function which takes one
# input x, and returns phi(x,t,Q,p,A,b)
f = lambda x: phi(x,t,Q,p,A,b);
g = lambda x: grad(x,t,Q,p,A,b); # same
h = lambda x: hess(x,t,Q,p,A,b); # same

# Perform a Damped Newton Step at x=-1.
x = -1;
(x_dns,gap) = dampedNewtonStep(x,f,g,h);
```