

# HOMWORK 4

## DUALITY & SECOND-ORDER METHODS \*

10-425/10-625 INTRODUCTION TO CONVEX OPTIMIZATION  
<http://425.mlcourse.org>

OUT: 10/23/23

DUE: 11/02/23

TAs: Asad & Tiancheng

### Instructions

- **Collaboration Policy:** Please read the collaboration policy in the syllabus.
- **Late Submission Policy:** See the late submission policy in the syllabus.
- **Submitting your work:** You will use Gradescope to submit answers to all questions and code.
  - **Written:** You will submit your completed homework as a PDF to Gradescope. For each problem, please clearly indicate the question number (e.g. 3.2). Submissions can be handwritten, but must be clearly legible; otherwise, you will not be awarded marks. Alternatively, submissions can be written in  $\text{\LaTeX}$ . You may use the  $\text{\LaTeX}$  source of this assignment (included in the handout .zip) as your starting point. For multiple choice / select all questions, simply write the letter(s) (e.g. A, B, C) corresponding to your chosen answer.
  - **Programming:** You will submit your code for programming questions to Gradescope. There is no autograder. We will examine your code by hand and may award marks for its submission.
- **Materials:** The data and reference output that you will need in order to complete this assignment is posted along with the writeup and template on the course website.

Question	Points
Randomized Kaczmarz	15
Duality in Linear Programs	19
Lagrangian Duality	5
Programming	33
Collaboration Questions	2
Total:	74

---

\*Compiled on Monday 23<sup>rd</sup> October, 2023 at 17:52

## 1 Randomized Kaczmarz (15 points)

In this question we'll explore an example of a stochastic gradient descent algorithm which exhibits a very fast (linear) rate of convergence. As we discussed in lecture this is often not the case (and the example is very special).

Here is the setting we'll work in. We're interested in solving a very large linear system:  $Ax = b$  where  $A \in \mathbb{R}^{n \times d}$ . We're given  $\{(a_1, b_1), \dots, (a_n, b_n)\}$  and would like to find a solution  $x^*$  such that,  $a_i^T x^* = b_i$ , for  $i \in \{1, \dots, n\}$ .

1. We'll suppose that  $A$  is a full-rank matrix, and that  $n \gg d$ , and that the system of equations is consistent i.e. there is some  $x^*$  that satisfies all the equations.
2. We'll also suppose that each of the  $a_i$  have unit norm, i.e.  $\|a_i\|_2 = 1$  (you can always rescale to ensure this).

To solve this system we just use SGD on the least squares objective, i.e. we'll try to minimize:

$$f(x) = \frac{1}{2n} \sum_{i=1}^n (a_i^T x - b_i)^2.$$

- 1.1. (3 points) It will turn out that SGD (where we sample an equation uniformly at random) on the above objective with a step-size of 1 will work (i.e. we don't need to tune it) – and this algorithm is known as the Randomized Kaczmarz algorithm. Write out the iterates of the algorithm, i.e. express  $x^{t+1}$  as a function of  $x^t$ .

- 1.2. (3 points) Show that if at time-step  $t$  we select the  $i_t$ -th equation, then

$$\|x^{t+1} - x^*\|_2^2 = \|x^t - x^*\|_2^2 - (a_{i_t}^T x^t - b_{i_t})^2.$$

- 1.3. (4 points) Take appropriate expectations to argue that,

$$\mathbb{E}[\|x^{t+1} - x^*\|_2^2 | x^t] = \|x^t - x^*\|_2^2 - \frac{1}{n} \|Ax^t - b\|_2^2.$$

- 1.4. (5 points) Let  $\lambda_{\min}$  denote the smallest eigenvalue of  $(A^T A)/n$ . Then argue that after  $k$ -iterations,

$$\mathbb{E}\|x^k - x^*\|_2^2 \leq (1 - \lambda_{\min})^k \|x^0 - x^*\|_2^2.$$

In nice cases, when the rows of  $A$  are well-spread out – the minimum eigenvalue will be some small constant – and in such cases the randomized Kaczmarz algorithm has very rapid linear convergence. In these cases, the number of iterations you will need to get to  $\epsilon$ -error will not even depend on the number of equations you have. This is a nice/surprising property to have, and should be contrasted with properties that a standard (Gaussian elimination type) method would have.

It's also worth reflecting on why SGD had linear convergence here (and doesn't usually) – notice that the noise in the stochastic gradients gets very small as  $x^k$  gets close to  $x^*$ .

## 2 Duality in Linear Programs (19 points)

In this question, you'll derive the famous duality between the max flow and min cut LPs. (You can find this example worked out in almost any book/resource on LP, but you should try to solve the problem first before looking at any resources.)

Suppose we have a graph  $G = (V, E)$  (i.e. with some set of vertices  $V$  and edges in  $E$ ). We label one of the nodes the source node  $s$ , and one of them the sink/target node  $t$ . We are additionally given some capacity constraints, i.e. each edge  $(i, j)$  can support no more than  $c_{ij}$  amount of flow. The capacities are all positive,  $c_{ij} > 0$ .

Now, a flow is simply a vector  $f$ , with one entry  $f_{ij}$  per edge in the graph, that satisfies these constraints:

1. Non-negative and capacity bounded: i.e.  $c_{ij} \geq f_{ij} \geq 0$
2. Conservation of flow except at the source and sink nodes, i.e. for any  $k \in V \setminus \{s, t\}$  we have that,

$$\sum_{(i,k) \in E} f_{ik} = \sum_{(k,j) \in E} f_{kj}.$$

- 2.1. (4 points) Write out the maximum flow LP, i.e. write out a linear program to determine the flow which maximizes the total flow emanating from the source node  $s$ .

A different problem on graphs is to partition the graph (into two pieces). We'll focus on what are called  $s - t$  cuts, i.e. partitions of the vertices where the source node  $s$  is one partition and the target node  $t$  is in the other partition. An  $s - t$  cut is simply a vector  $x \in \{0, 1\}^n$  where  $n$  denotes the number of nodes in the graph, where we require  $x_s = 1, x_t = 0$ .

For every edge we'll associate a variable  $b_{ij}$  such that  $b_{ij} \geq (x_i - x_j)$  and  $b_{ij} \geq 0$  (and eventually we'll try to make the  $b_{ij}$  small). This will have the effect that if  $i$  is in the same partition as  $s$ , and  $j$  is in the same partition as  $t$ , then  $b_{ij}$  will be 1 (and 0 otherwise). This means that  $b_{ij}$  will be one if our partition *cuts* the edge between  $i$  and  $j$ . The cost to cut the edge  $(i, j)$  is given to you (we are given some positive costs  $c_{ij}$  for each edge  $(i, j)$ ).

- 2.2. (5 points) Write out an optimization problem to find the  $s-t$  cut with minimal cost (use the variables  $b, x$  and the given costs  $c_{ij}$ ). This is not an LP because we have some additional integer constraints that require  $x_i \in \{0, 1\}$ . Just drop these integer constraints to form an LP *relaxation* to the min cut problem (i.e. an LP with lower objective value than the original problem we wanted to solve.)
- 2.3. (5 points) Derive the dual of the max flow LP.
- 2.4. (5 points) Now argue that (after some basic algebraic manipulations/observations similar to ones we did in lecture for the optimal transport LP) the dual of the max flow LP is identical to the min cut LP.

### 3 Lagrangian Duality (5 points)

Consider the following optimization problem, herein called the log barrier problem:

$$\min_x c^T x - \tau \sum_i \log(x_i) \text{ such that } Ax = b \quad (1)$$

The second term in the objective is sometimes called the log barrier function, and acts as a ‘soft’ inequality constraint, because it will tend to positive infinity as any of the  $x_i$  tend to zero from the right.

Throughout, assume that  $A$ ,  $b$ , and  $c$  have been chosen such that  $\{x : x > 0, Ax = b\}$  and  $\{y : A^T y > -c\}$  are non-empty, i.e. the primal LP and its dual are both strictly feasible for both the original LP and the second problem.

3.1. (5 points) Derive the dual of the log barrier problem in (1).

## 4 Programming (33 points)

In this section, you will implement two algorithms: (damped) Newton's method<sup>1</sup> for unconstrained optimization and the Log-Barrier Method for inequality constrained optimization.

You should start by implementing Newton's method, since it will be called as a subroutine in your Log-Barrier Method implementation—exactly how you do this is up to you, the APIs are designed so that you might be able to reuse your Newton's Method implementation directly within your Log-Barrier method implementation; but because of the various quantities we wish you to print out, it may be easier to keep them separate and duplicate some code.

The starter code for this section contains four files: `models.py`, `optimizer.py`, `plot.py`, `run.py`.

**`models.py`:** Use this file for your implementation of the mathematical program for the Log-Barrier Method implementation. This file contains two classes corresponding to a logistic regression model (i.e. an `UnconstrainedProgram`) and a simple 2D constrained mathematical program (i.e. a `ConstrainedProgram`). These classes should be considered as containing the full definition of the mathematical program, but should not store state about the current parameters over which we are optimizing. Any necessary information from these models can be called directly in your optimizer implementations.

**`optimizer.py`:** Use this file for your optimizer implementations. Implement your optimizers in the `optimize` method of the classes `NewtonMethod` and `LogBarrierMethod`. Please include your implementation where the code is marked with `#TODO`. It may be helpful to define helper functions.

**`run.py`:** This supplementary file is provided for your convenience for training/testing your various optimizer implementations. You will need to right your own code to properly run the experiments described below.

**`plot.py`:** This supplementary file is provided to help with generating plots for the tasks that require them.

---

<sup>1</sup>Throughout this writeup, we will may refer to “Newton's Method”, but by that we mean *damped* Newton's method.

## Newton's Method

To start, you will implement Newton's method for an unconstrained optimization problem. The Newton's method algorithm is shown below.

---

### Algorithm 1 Newton's Method

---

- 1: Given a stopping threshold  $\epsilon$ , maximum number of iterations  $T$ , and unconstrained optimization problem  $\min_x f(x)$ .
  - 2: Initialize  $\theta^{(0)}$
  - 3: **for**  $t = 0, 1, 2, \dots, T$  **do**
  - 4:   Compute gradient  $g = \nabla f(\theta^{(t)})$
  - 5:   Compute the Hessian  $H = \nabla^2 f(\theta^{(t)})$
  - 6:   Compute Newton's step  $v = -H^{-1}g = -(\nabla^2 f(\theta^{(t)}))^{-1}\nabla f(\theta^{(t)})$
  - 7:   Compute Newton's decrement  $\lambda^2 = g^T H^{-1}g = \nabla f(\theta^{(t)})^T (\nabla^2 f(\theta^{(t)}))^{-1} \nabla f(\theta^{(t)})$
  - 8:   Select a step size  $\eta$  using backtracking line search in the direction  $v$
  - 9:   Update parameters  $\theta^{(t+1)} = \theta^{(t)} + \eta v$
  - 10:   Stop early if  $\lambda^2/2 \leq \epsilon$
  - 11: **return**  $\theta^{(t)}$
- 

You will implement Newton's method with backtracking line search. Below is the backtracking line search algorithm where we assume that  $v$  is the line search direction, which may naturally differ from the negative gradient direction.

---

### Algorithm 2 Backtracking line search

---

- 1: Let  $v$  be the line search direction,  $\theta$  be the current parameter value,  $\nabla f(\theta)$  the gradient at  $\theta$ ,  $\tau_0$  the initial step size, and  $\alpha, \beta$  the hyperparameters for the line search.
  - 2:  $\eta = \tau_0$
  - 3: **while**  $f(\theta - \eta v) \geq f(\theta) - \eta \alpha \nabla f(\theta)^T v$  **do**
  - 4:    $\eta \leftarrow \eta * \beta$
  - 5: **return**  $\eta$
- 

Other important details:

- To invert the Hessian, you can use `np.linalg.inv`.
  - Before inverting, you should add a scaled identity matrix to the Hessian for stability, i.e. you should add  $\xi I$  where  $\xi = 0.001$ . See `np.eye`.
  - For all questions below, use the line search hyperparameters  $\tau_0 = 1.0$ ,  $\alpha = 0.7$ , and  $\beta = 0.9$ .
- 4.1. (5 points) Run Newton's method on the logistic regression model (see the class `LogisticRegression`) with  $\epsilon = 0.0001$  and  $T = 100$  and  $\theta^{(0)} = 0$ . Plot the training loss  $f(\theta)$  and testing accuracy for each epoch—these should be *two separate plots*. Both axes must be log-scale. (Hint: See `plot_log_log_convergence()`.)
  - 4.2. (5 points) Fill in the table below by running Newton's method on the logistic regression model for  $T = 100$  and  $\theta^{(0)} = 0$  and the given values of  $\epsilon$ . For each  $\epsilon$ , report the accuracy on test dataset and the number of epochs that completed before the early stopping condition was reached.

$\epsilon$	Number of Epochs	Test Accuracy
0.1		
0.01		
0.001		
0.0001		

- 4.3. (3 points) Which choice of  $\epsilon$  gives the best training loss? Which choice of  $\epsilon$  gives the best test accuracy? If these are the same, why are they the same; if they are different, why might they differ?

## Log-Barrier Method

Now you will implement the Log-Barrier Method for an inequality constrained optimization problem of the form:

$$\begin{aligned} \min_x \quad & f(x) \\ \text{subject to} \quad & h_i(x) \leq 0, \quad \forall i = 1, \dots, m \end{aligned}$$

The Log-Barrier Method algorithm is given below. The idea is to repeatedly solve a sequence of optimization problems of the form below for increasing values of  $\rho > 0$ :

$$\begin{aligned} \min_x \quad & \rho f(x) + \phi(x) \\ \text{where} \quad & \phi(x) = - \sum_{i=1}^m \log(-h_i(x)) \end{aligned}$$

We will use Newton's method to solve each successive problem.

---

### Algorithm 3 Log-Barrier Method

---

- 1: Let  $\rho^{(1)}$  be the initial temperature,  $\mu > 1$  the rate at which we adjust the temperature, and  $\delta$  our stopping threshold.
  - 2: Initialize  $x^{(0)}$
  - 3: **for**  $t = 1, 2, \dots, T$  **do**
  - 4:   Let  $g_t(x) = \rho^{(t)} f(x) + \phi(x)$
  - 5:   Obtain  $x^{(t)} = \min_x g_t(x)$  using Newton's method initialized to  $x^{(t-1)}$
  - 6:   Stop if  $m/\rho^{(t)} \leq \delta$
  - 7:   Update  $\rho^{(t+1)} = \mu \rho^{(t)}$
  - 8: **return**  $\theta^{(t)}$
- 

The stopping condition comes from our bound on the duality gap  $f(x^{(t)}) - f(x^*) \leq m/\rho$ .

You will apply the Log-Barrier Method to the following 2D inequality constrained optimization problem, so that you can visualize the progress of the algorithm as it runs.

$$\begin{aligned} \min_{x_1, x_2} \quad & (10x_1^2 + (x_2 + 30)^2)/2 + 5 \log(1 + \exp(-x_1 - (x_2 + 15))) \\ \text{subject to} \quad & x_2 \leq 0.5x_1 + 15 \quad [\text{top left of pentagon}] \\ & x_2 \leq -x_1 + 15 \quad [\text{top right of pentagon}] \\ & x_2 \geq -3x_1 - 30 \quad [\text{bottom left of pentagon}] \\ & x_2 \geq 2x_1 - 30 \quad [\text{bottom right of pentagon}] \\ & x_2 \geq 0.4x_1 - 10 \quad [\text{bottom of pentagon}] \end{aligned}$$

Other important details:

- You should implement the constrained optimization problem in `Simple2DProgram`.
- Before writing any code, you should call `plot_objective_constraints_iterates()` to see a visualization of the 2D optimization problem.
- For all questions below, use the Newton's method hyperparameters  $\epsilon = 0.1$  and  $T = 100$ ; use the line search hyperparameters  $\tau_0 = 1$ ,  $\alpha = 0.7$ , and  $\beta = 0.9$ .



- 4.4. (5 points) Run the Log-Barrier Method on the 2D problem above for each of the following starting points:  $(x_1, x_2) \in [(13, 0), (-10, 7)]$ . Let  $\mu = 2$ ,  $t^{(0)} = 1$ ,  $\delta = 0.1$ . Plot the learning curve, i.e. the value of the objective function  $f(x)$  for each Newton's Method step. (Important Note: the x-axis should *not* be  $t$  from the Log-Barrier Method algorithm above, it should be the  $t$  from the Newton's Method algorithm, which is ever increasing on every call to Newton's Method for the purposes of plotting). Put the learning curves for each starting point all on the same plot. Both axes must be log-scale. (Hint: See `plot_log_log_convergence()`.)
- 4.5. (5 points) Using the same hyperparameter settings and starting points as in the previous question: Plot the iterates  $x^{(1)}, x^{(2)}, \dots, x^{(T)}$  of the algorithm over time, i.e. one iterate per Newton's step. Your plot should also include a contour plot of the function  $f$  and the inequality constraints  $h$ . Put the sequence of iterates for each starting point all on the same plot. (Hint: See `plot_objective_constraints_iterates()`.)
- 4.6. (5 points) Run the Log-Barrier Method on the 2D problem above for the starting point  $(x_1, x_2) = (-10, 7)$ ,  $\mu = 2$ ,  $\delta = 0.1$ , for differing values of  $t^{(0)} \in [0.1, 1, 10]$ . Plot the learning curve, i.e. the value of the objective function  $f(x)$  for each Newton's Method step. Put the learning curves for each  $t^{(0)}$  all on the same plot. Both axes should be in log-scale. (Hint: See `plot_log_log_convergence()`.)
- 4.7. (5 points) Run the Log-Barrier Method on the 2D problem using the same hyperparameters as in the previous question. But now plot the upper bound on the duality gap  $m/\rho^{(t)}$  for each Newton's method step. *Only the vertical axis should be in log-scale.*

## 5 Collaboration Questions (2 points)

After you have completed all other components of this assignment, report your answers to these questions regarding the collaboration policy. Details of the policy can be found in the syllabus.

- 5.1. (1 point) Did you collaborate with anyone on this assignment? If so, list their name or Andrew ID and which problems you worked together on.
- 5.2. (1 point) Did you find or come across code that implements any part of this assignment? If so, include full details.