MTAT.03.227 Machine Learning (Spring 2015)
Exercise session V: Optimization basics

Konstantin Tretyakov

March 10, 2015

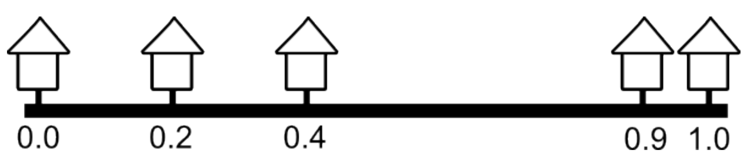
The aim of this exercise session is to get acquainted with the basics of optimization methods. In particular, we are going to explore the following topics:

- Analytic optimization of a quadratic function.
- Gradient descent for one- and two-dimensional functions.
- Newton's method.
- Stochastic descent.
- Non-differentiable functions.
- Constrained gradient descent.

For that we shall go through 30 small exercises, each worth 0.5 points and presumably doable in under 5-10 minutes even by the slowest of us. For each exercise you typically need to write up a short piece of code (usually about 1-2 lines), and 1-2 sentences of your opinion about what you did and saw. You can submit your whole solution as a single R file with comments, provided it is formatted to be sequentially readable. Exercises marked with a * are suggested as bonus tasks. As usual, the nominal point count is 10, but I'm sure doing all 15 points would not hurt.

We shall use the following toy example in our exercises (which is, as you might figure out, a trivialized version of a clustering problem).

Description of the problem The village of Optimia consists of a single kilometer-long straight road with five houses built along it. The road has distance markings. The first house is built at marking 0, the second – at marking 0.2, the third – 0.4, fourth – 0.9 and fifth – 1.0.



The villagers wish to embrace the marvels of the Internet and want to build a data center somewhere along their central road. The question they face is – at what position along the road should they build it in order to optimize cabling costs.

The data center will be connected with a separate cable to each of the five houses. The price of each cable piece is somehow dependent on the distance between the house and the center. That is, if the data center is built at position w along the road, the cabling company will charge the villagers a total of

$$\ell(|w-0.0|)+\ell(|w-0.2|)+\ell(|w-0.4|)+\ell(|w-0.9|)+\ell(|w-1.0|),$$

where $\ell(d)$ is the price of a cable of length d (this will vary between the exercises).

Exercise 1 (0.5pt). Suppose the price of each cable segment is $\ell(d)=d^2$. Find the optimum position for the data center analytically. *Fermat's rule*

Hint: This was done on the lecture.

Exercise 2 (0.5pt). Define an R function `f` which takes position of the data centre w as input and produces the resulting total cost of the cable. Plot the function using the following code: *Objective function*

```
ws = seq(0,1,0.01)
plot(ws, Vectorize(f)(ws))
```

Exercise 3 (0.5pt). Define an R function `df` which takes w as input and outputs $\nabla f(w)$. Plot it. What is the important y-value to look for on that plot? *Gradient*

Exercise 4 (0.5pt). Implement the function `gradient_descent(f, df, x0, u, nsteps)`, which takes in the function to be optimized `f`, the gradient of the function `df`, the initial guess `x0`, the step size `u`, the number of steps `nsteps`, and returns the result of running the basic gradient descent optimization algorithm. Run the function using `f` and `df` defined above, with `x0 = 0, u=0.01, nsteps=100`. Does it produce the correct answer? *Gradient descent*

Hint: $\Delta x_i = -\mu\nabla f(x_i)$, remember?

Hint: The whole body of the function should not be longer than 5 lines. In fact, just 3 suffices.

Exercise 5 (0.5pt). Study the following code: *Step size*

```
gd_by_step_size = Vectorize(function(step) {
                      gradient_descent(f, df, 0, step, 20)
                  })
step_sizes = seq(0,0.3,0.01)
output = gd_by_step_size(step_sizes)
```

Try to guess, without running it, what will be in the `output` array. Verify your guess. If you guessed correctly, explain how you did it. If you did not, understand what is happening and explain.

Exercise 6-7 (1.0pt). Modify the function gradient_descent so that it would output, along with the correct answer, the whole trajectory of the solution. In particular, make the last lines of your function the following:

Convergence properties

```
  ...
  result = list(xi, traj)
  names(result) = c("ans", "traj")
  result
}
```

Now do plot(gradient_descent(f, df, 0, step_size, 20)$traj) for step_size equal to 0.01, 0.08, 0.1, 0.15, 0.19, 0.20, 0.21. What different types of behaviour do you observe?

Exercise 8-9 (1.0pt). Define a function ddf which outputs $\nabla^2 f$. Then, define the function newton_descent(f, df, ddf, x0, nsteps) which implements the Newton's algorithm. Make sure it outputs the trajectory in the same way as gradient_descent does. Study the trajectories for various starting points. What do you observe? Why is it happening? Compare the value of $\nabla^2 f(w)^{-1}$ with the step sizes you studied in the previous exercise.

Newton's algorithm

Hint: $\Delta x_i = -H^{-1}c$, remember? This means $\Delta x = -\frac{\nabla f(x)}{\nabla^2 f(x)}$.

Exercise 10 (0.5pt). Next we want to implement a *stochastic* version of gradient descent. Which function (f, df, gradient_descent) needs modification to achieve it (only one does)? Modify the necessary function appropriately and study the trajectories of the stochastic gradient descent algorithm for various step sizes (as in Exercise 6-7). What differences do you see? Could we also use a *stochastic Newton's descent*, what do you think?

Stochastic descent

Hint: You'll need to use R's sample function.

Hint: $\Delta x_i = -\mu \nabla f_j(x_i)$.

Exercise 11 (0.5pt). Now the cable company proposes new cabling prices. Namely, $\ell(d) = |d|$. Define f, df and ddf appropriately and plot them (figure out a natural way to handle non-differentiable points).

Handling non-differentiable points

Exercise 12* (0.5pt). The optimal w for the previous cable prices (the $\ell(d) = d^2$ case) was the mean. What is the optimal w for the new scheme? Study the plot of df to guess an answer, then prove the general result.

Nice-to-know

Exercise 13 (0.5pt). Repeat Exercise 5 for the new function. Do you see conceptual differences from the previous case? Explain their causes. You should see that non-differentiability of f at just a couple of points, although it can be handled in an ad-hoc manner, may cause convergence problems for gradient descent. It also makes it hard to devise a reliable convergence criterion.

Non-differentiability issues

Also, what about using the Newton's algorithm here?

Exercise 14 (0.5pt). Implement stochastic gradient descent for the new objective function. Contemplate the conceptual simplicity of the resulting algorithm and feel happy about it.

Multidimensional optimization

Exercise 15-16 (1.0pt). Later on, the inhabitants of Optimia decided that they could have two data centers instead of one. Each house would only need to set up a single cable link to the closer of the two data centers. The price of the cabling is $\ell(d) = |d|$, thus the overall cabling price would be:

$$f(w_1, w_2) = \sum_{i=1}^{5} \min(|w_1 - h_i|, |w_2 - h_i|),$$

and the new task is to find the optimal locations for two data centers w_1 and w_2.

Implement the new objective function `f(w)` where `w = c(w1, w2)` will be treated as a vector. Plot f using the following code:

```
x = seq(0,1,by=0.05)
y = seq(0,1,by=0.05)
fvals = outer(x, y, Vectorize(function(w1,w2){ f(c(w1,w2)) }))
contour(x,y,fvals)
image(x,y,fvals)
```

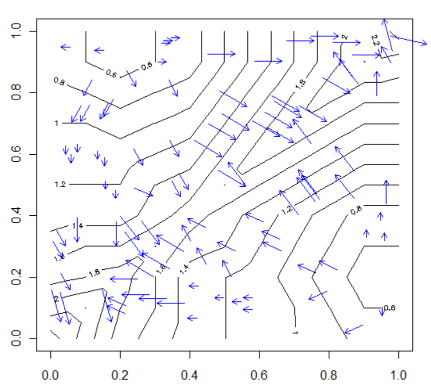
Hint: `mapply(min, dists_to_1, dists_to_2)`

Multidimensional gradient

Exercise 17-19 (1.5pt). Implement the gradient `df` for the above mentioned function. Note that the gradient must return a vector of two elements. Handle non-differentiable points naturally.

Visualizing multidimensional gradient

Exercise 20-21* (1.0pt). A multidimensional gradient is a *vector field* and cannot be visualized using contour or image plots. Find a way to visualize the gradient as shown below:



Hint: You'll need `arrow.plot` from the `fields` package.

Exercise 22 (0.5pt). Make sure your gradient descent implementation works with the new multidimensional functions. If you did Exercise 4 correctly, chances are your algorithm works with the new multidimensional functions without any modification. The only place which might require changes is the trajectory compilation. If before you used something like `traj = c(traj, xi)` then simply rewrite it to `traj = rbind(traj, xi)`.

Multidimensional gradient descent

Test your algorithm by running

```
result = gradient_descent(f, df, runif(2), 0.01, 100)
contour(x, y, fvals) # From Ex. 15-16
points(result$traj[,1], result$traj[,2], col='red')
```

Exercise 23 (0.5pt). Implement a stochastic version of 2-dimensional gradient descent. Run it and plot the results as in previous exercise.

Multidimensional stochastic gradient descent

Exercise 24 (0.5pt). In practice you would rarely need to implement your own optimization algorithm. Use the R's built-in `optim` function to find a solution to your current optimization problem. Show the most basic way of invoking this function.

Library functions

Exercise 25* (0.5pt). The villagers just found out that their two data centers may not be located further away from each other than 0.5km. I.e. the locations of the two data centers must satisfy

Constrained optimization

$$|w_1 - w_2| \leq 0.5$$

Do(es) the current solution(s) satisfy this requirement? Which points on the contour plot satisfy it? Is it a convex set of points?

Exercise 26-28* (1.5pt). Let us implement constrained stochastic gradient descent to help the villagers. Note that as our objective is not convex, this approach may not always work, but in this case it actually will. You need to change your gradient descent implementation by adding a constraint check and a projection step, i.e, you'll have something like:

Constrained gradient descent

```
for (i in 1:nsteps) {
  xi = xi - u*df(xi);
  if (xi does not satisfy constraints) xi = project(xi)
  traj = rbind(traj, xi)
}
```

The non-obvious part is the orthogonal projection onto the set $\{(w_1, w_2) : |w_1 - w_2| \leq 0.5\}$.

Multidimensional Newton's descent

Exercise 29-30* (1.0pt) Finally, it might be interesting to implement the multidimensional Newton's algorithm. As you should know by now, it will not work with the $\ell(d) = |d|$ case, and it will work somewhat trivially for $\ell(d) = d^2$ case. The next simplest choice is $\ell(d) = |d|^3$. Implement the corresponding functions `f`, `df`, `ddf` (which will produce a matrix this time!), fix up your `newton_descent` appropriately and see how it works.