CS168, Spring 2015

Mini-Project #8

Due by 11:59 PM on Tuesday, May 26

Instructions

- You can work individually or with one partner. If you work in a pair, both partners will receive the same grade.
- If you've written code to solve a certain part of a problem, or if the part explicitly asks you to implement an algorithm, you must also include the code in your pdf submission. See the problem parts below for instructions on where in your writeup to put the code.
- Make sure plots you submit are easy to read at a normal zoom level.
- Detailed submission instruction can be found on the course website (`http://cs168.stanford.edu`) under the "Coursework - Assignment" section. If you work in pairs, only one member should submit all of the relevant files.
- a) Use 12pt or higher font for your writeup. b) Code marked as "Deliverable" gets pasted into the relevant section, rather than into the appendix (though feel free to put it in both). Keep variable names consistent with those used in the problem statement, and with general conventions. No need to include import statements and other scaffolding, if it is clear from context. Also, please use the `verbatim` environment to paste code in LaTeX from now on, rather than the listings package:

  ```
  def example():
  print "Your code should be formatted like this."
  ```

- **Reminder:** No late assignments will be accepted, but we will drop your lowest mini-project grade when calculating your final grade.

Part 1: Gradient Descent

Goal: In this exercise you will get some experience with different variations of Gradient Descent and get some appreciation for functions that Gradient Descent works well for.

Description: Recall that Gradient Descent updates a solution $\mathbf{x}$ at time step j for objective function f, with step size α according to the following rule:

$$\mathbf{x}^{(j+1)} = \mathbf{x}^{(j)} - \alpha \cdot \nabla f(\mathbf{x}^{(j)})$$

So for each coordinate i we have $x_i^{(j+1)} = x_i^{(j)} - \alpha \cdot \frac{d}{dx_i} f(\mathbf{x}^{(j)})$. Accelerated Gradient Descent additionally takes the current momentum into account:

$$\mathbf{x}^{(j+1)} = \mathbf{x}^{(j)} - \alpha \cdot \nabla f(\mathbf{x}^{(j)}) + \beta \cdot \left(\mathbf{x}^{(j)} - \mathbf{x}^{(j-1)}\right)$$

where for convenience we define $\mathbf{x}^{(-1)} = \mathbf{x}^{(0)}$. The parameter β indicates how much the momentum plays a role — note that $\beta = 0$ yields the vanilla Gradient Descent update rule.

(a) We will minimize the following 10-dimensional objective function

$$\sum_{i=1}^{10} \left(1 + \frac{i-1}{10}\right) x_i^2 \tag{1}$$

using Gradient Descent. What stopping condition should you typically use for Gradient Descent? Is there an easier stopping rule that would work for this particular objective function?

(b) Minimize the objective function given in part (a) using step sizes $0.01, 0.02, ..., 0.50$. Use a random start position in $[-100, 100]^{10}$, and repeat this 10 times. Plot the results with the time step on the x-axis, and the average number of iterations on the y-axis (as always, properly label your axes). Which value works best for this setting? Why does increasing the step size beyond that value yield a larger number of iterations?

(c) We now focus on a different objective function:

$$\sum_{i=1}^{10} 2^{i/2} \cdot x_i^2 \tag{2}$$

Plot the number of iterations for the same step sizes as part (b). How do the results compare to part (b)? What happens when the step size α is too large; Why? [Caution: be careful about the stopping criterion that you use. A step size that is too large for the problem may not lead to a solution, but terminate because one of the numbers became infinity (or NaN).]

(d) Now implement Accelerated Gradient Descent and rerun the experiments of part (b) and (c) for $\beta = 0, 0.1, 0.25, 0.5, 0.75$. Generate two figures, one for the experiments minimizing (1), and one for minimizing (2). In each plot, plot 5 lines corresponding to the different choices of β, and as always make sure to include a legend. For the plot minimizing (2), run the experiments for $\alpha = 0.002, 0.004, ..., 0.1$.

Discuss the results: does Accelerated Gradient Descent improve the number of iterations? Under what conditions?

Deliverables: Discussion for part (a). Code, plot, (approximately) optimal value for α and discussion for part (b) and (c). Code, two plots and discussion for part (d).

Part 2: QWOP[1]

Goal: In this exercise you will get some hands-on experience solving a tricky optimization problem that captures some of the difficulties of some real-world robotics problems.

Description: In 2010, the QWOP game[2] took the Internet by storm. The purpose of the game is to get your avatar to run the 100-meter event at the Olympic Games by controlling their thigh and knee angles. If you haven't played the game, give it a try—this assignment will be much more amusing if you have first spent a few minutes with the flash game; even getting your runner to fall forwards rather than backwards can be quite a challenge.

Rather than learn how to play the game, you'll just write a computer program to do it for you! The TAs have supplied implementations of the QWOP physics engine, implemented in Python and MATLAB. These functions take a list of 40 floating point numbers as input (corresponding to 20 instructions for the angle of the thighs and 20 for the angle of the knees). The output of the function is a single number, representing the final $x-$position of the head of the avatar. The more efficiently you get the avatar to run, the further it will go and the larger the output number. The goal of this problem is for you to find a near-optimal input that makes the avatar go as far as possible—namely find a length 40 vector that results in the QWOP code evaluating to as large a number as possible.

[1] This part of the assignment, and the implementations of the QWOP physics engine, are based on an assignment developed by Paul Valiant at Brown University, and we thank him for sharing his code, etc.

[2] `http://www.foddy.net/Athletics.html`

For your amusement, the MATLAB and Python implementations also provide an animation of the movements of the avatar on any given input.

(a) *[do not hand in]* Make sure you can call either the python `sim(plan)` function, or the MATLAB `quop(plan)` functions with a `plan` consisting of 40 floating point numbers. The MATLAB and Python implementations come with a visualization of the game, and we highly encourage you to look at what actually happens.

(b) Optimize QWOP. Your score to this part will be the distance you get your avatar to go, with anything above 10 points regarded as bonus.. (In the MATLAB implementation, you can suppress the animation by including an additional argument of "0" in the function call, ie `qwop(plan, 0)`.)

Note: this is a non-convex optimization problem with lots of local optima, and vanilla gradient descent on the objective function value (ie the distance travelled) might not work too well. Feel free to unleash the full arsenal of tools that you have learned and your creativity to solve this problem. [You might consider many variants of gradient descent, included the MCMC/simulated annealing variant that we saw in assignment 3; if it seems too tricky to do gradient descent on the whole vector, you might consider a divide-and-conquer approach, if you think it might help, feel free to take a close look at the physics engines—its just 30 lines of MATLAB code, or a little more python code....feel free to play around with this.] Please do NOT use optimization code that you found online.

(c) Describe which techniques (or hacks) you tried, and whether or not they were successful.

Deliverables: For part (b), the code, input and a link to a YouTube video of the run (the Python code gives the option to save to an mp4 video; you may need to install ffmpeg for the code to work; if you are using MATLAB or do not want to install the extra python package, just record the animation on your phone and upload it). The input `plan` *has* to be formatted as either a MATLAB vector or Python list: e.g. `[0.100, 2.3000, ..., -2.111]`. Discussion for part (c).