# Project 2: Backtracking and Heavy ball

January 25, 2019

## 1 Guideline

Here is a guideline for your project.

- The project is designed to help understand the topic, so please do not copy and paste.
- You are welcome to work in groups (no more than 3 in one group), but you should write it up for your submission/exercises on your own.
- The standard submission of project contains the following documents:
  - The code files
  - A `README` file
  - A brief writeup document for the project

In your `README` file, there are several things you need to declare:

- What language are you using to write the program;
- How to run the code;
- Contributors.

You can add additional information such as possible issues, future plan in the `README` file as well. For the writeup document, you will be answering the questions in the project.

## 2 Project: Backtracking algorithm and Heavy ball

In this project, you will write program to solve some optimization problems with the `Backtracking Algorithm` to select step length and compare with other line search strategies in Project 1. The test functions are similar to the ones in Project 1, so you do not have to redo everything.

### 2.1 Project Description of backtracking

The `Backtracking Algorithm` is a simple algorithm to select the step length for line search methods. It only respects the sufficient decrease condition (Armijo) and might violate the curvature condition. However, it has excellent performance on a variety of problems. In this project, you will compare this algorithm with the ones in your Project 1.

### 2.1.1 Task 1: Implement backtracking

In this task, you will implement the backtracking algorithm for selection of $\alpha_k$. The algorithm is stated as the following:

1. Choose $\alpha = \alpha_0$ as the initial guess for the step length, also choose $\rho \in (0,1)$ and choose $c_1 \in (0,1)$ for Wolfe condition.
2. If the sufficient decrease condition

$$f(x_k + \alpha p_k) \leq f(x_k) + c_1 \alpha p_k^T \nabla f(x_k) \tag{1}$$

   is not violated, then return $\alpha$, otherwise $\alpha \leftarrow \rho\alpha$. Goto step 2.

**Task**: You will implement a function called `backtracking` in your favorite programming language. The signature should be like

$$\texttt{function alpha = backtracking}(f,\ \nabla f,\ p_k,\ x_k,\ c_1,\ \rho) \tag{2}$$

or

$$\texttt{def backtracking}(f,\ \nabla f,\ p_k,\ x_k,\ c_1,\ \rho)\texttt{:} \tag{3}$$

returning the chosen $\alpha$. We choose the starting step length $\alpha_0 = 1$.

### 2.1.2 Task 2: Backtracking Steepest Descent Method

In this task, you will implement the inexact line search steepest descent method with backtracking. The only difference between this method and the exact line search is to replace the $\alpha$ selection routine. The algorithm is omitted here.

**Task**: You will be using your algorithm to solve the unconstrained optimization problem $\min_{x=(x_1,x_2)} f(x)$ for the objective function $f$ as the following quadratic function:

$$f(x_1, x_2) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$$

The solution of above optimization problem is $(1, 3)$. For your implementation, you will use starting point from $\hat{x} = (0, 0)$ and the tolerance $\delta = 10^{-9}$ and $c_1 = 10^{-4}$, $\alpha_0 = 1$, $\rho = 0.1$.

- Compare your result (e.g. iteration number, running time, etc.) with the other steepest descent methods in project 1. Is the backtracking step length selection method better than them?
- What if you change the $\rho$ to be 0.5 or 0.9, what do you observe?

### 2.1.3 Task 3: Backtracking Newton's method

The Newton's method is a good choice for low dimensional problems. The line search Newton's method is the following

$$x_{k+1} = x_k - \alpha_k [\nabla^2 f(x_k)]^{-1} \nabla f(x_k), \tag{4}$$

When step length is fixed as $\alpha_k \equiv 1$, the above iteration is the standard Newton's method.

**Task**: In this task, you will have to implement the Newton's method with your `backtracking`

routine, and then test your implementations of backtracking steepest descent method ($\rho = 0.1$) and Newton's method ($\rho = 0.9$) on the following Rosenbrock function:

$$f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$$

You will try the following starting points (other parameters are fixed as the previous tasks) and answer the questions below:

1. starting point $\hat{x} = (1.2, 1.2)$, find out how many iterations have the above methods used, what are the convergence types (Q-linear, Q-superlinear, Q-quadratic)?
2. starting point $\hat{x} = (-1.2, 1)$, find out how many iterations have the above methods used, what are the convergence types (Q-linear, Q-superlinear, Q-quadratic)?

### 2.2 Project Description of Heavy ball

If you have tried steepest descent on the quadratic function like $f(x, y) = x^2 + 10y^2$, then your iteration will follow a zig-zag path. The heavy ball method is used to reduce this ping-pong effect. The idea is simple: instead of the bouncing back and forth, why not let the point keep its way a little further. This idea is also referred as a momentum method, the trend of the point moving from $x_{k-1}$ to $x_k$ is characterized as the vector $x_k - x_{k-1}$, so during our iteration, we can let the point keep this movement a little bit and also combine gradient descent direction.

#### 2.2.1 Task 4: Heavy ball method

The heavy ball iteration is

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k) + \beta_k (x_k - x_{k-1}) \tag{5}$$

the first part on right hand side is steepest descent, the last term is called momentum. Let's choose $\alpha_k \equiv 10^{-3}$ and $\beta_k \equiv 0.9$ for the Rosenbrock function. Caution your first point is chosen without using $\beta_k$, which is

1. If $k = 0$, $x_{k+1} = x_k - \alpha_k \nabla f(x_k)$.
2. If $k > 0$, $x_{k+1} = x_k - \alpha_k \nabla f(x_k) + \beta_k (x_k - x_{k-1})$.

- Test your heavy ball function on Rosenbrock function, what is the iteration number? What is the convergence type (Q-linear? Q-superlinear? Q-quadratic?).
- Compare this result against your previous steepest descent methods (exact line search, fixed step length, backtracking), which one performs better on Rosenbrock? Take the starting point as $(-1.2, 1)$.

## 3 Starter code kit

The starter code kit is in Python (MATLAB users should have no difficulty to understand.) It is up to you to following the starter code kit's style or not. As a reminder, the kit only fills out framework, there are some details you will have to write on your own.

## 4 Submission

If you prefer to use the GitHub or Bitbucket, etc. to store your code, then it is OK to simply submit the repository's link on a text file (if your repository is public), otherwise, please submit all files in a zip file through Canvas.