Homework #2: Unconstrained Optimization — ME 575

due 2/4/2015 before class via Learning Suite — **50 possible points**

2.1 [5 pts] Read Chapter 3 (Gradient-Based Optimization).

2.2 [45 pts] Write your own unconstrained (gradient-based) optimization algorithm. You can use any method(s) you want, but you must develop your own code. Your code will be tested on 6 different unconstrained optimization problems. Three of the problems are specified below and the other three will not be disclosed until after your submission. We will have a miniature class competition to compare performance. You will *not* be graded on your algorithm's relative performance, just on having completed the expected tasks. The goal of this exercise is learning—the competition is just for fun. There will, however, be bonus points awarded for top-performing algorithms. In order to display top rankings for each problem in an anonymous manner please **supply a psuedoname** that you would like to be identified by.

In order to facilitate automated testing, your code submission must use the expected function signature. On Learning Suite I have uploaded an *.m file and a *.py template file (for Matlab and Python respectively) that you must use and should not change the function signatures.

Be sure to include the following elements in your report:

- [3 pts] Describe your approach and why/how you selected that methodology.
- [8 pts] Show convergence plots for each test problem.
- [8 pts] For the 2D problems show the iteration history on a contour plot, and for the Brachistochrone problem show the shape change every n iterations (where n is some user-chosen value just so the plot isn't too overwhelming).
- [10 pts] Implement at least one variation to your method and compare its performance (e.g., more than one line search approach, and/or more than one search direction approach).
- [5 pts] Compare your performance with an existing optimization algorithm such as Matlab's fminunc.
- [5 pts] For the Brachistochrone problem study the effect of increased problem dimensionality. Try dimensions of $4, 8, 16, 32, 64, 128, \ldots$ up to the highest number you can reasonably manage. Plot and discuss the increase in computational expense with problem size (example metrics include things like major iterations, functional calls, wall time, etc.).
- [3 pts] Try warm-starting the Brachistochrone problem with a better solution (e.g., solve the problem of size 64 by warm-starting with the solution from size 32). Compare the performance versus starting with the straight line initial condition. You only need to test one case.
- [3 pts] Discuss the main challenges you faced and areas where you think your approach could most effectively be improved.

Test Cases:

(a) Matyas function

$$f(x) = 0.26\left(x_1^2 + x_2^2\right) - 0.48 x_1 x_2 \tag{1}$$

This is a simple quadratic problem with one optimum at $x^* = (0, 0)$, $f^* = 0$. A good place to start.

(b) Rosenbrock function

$$f(x) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2 \tag{2}$$

The Rosenbrock function is 2-dimensional problem with a long curved valley. There is only one optimum, located at $x^* = (1, 1)$, $f^* = 0$. This problem is a good test case because it is easily visualized (2D) but is challenging due to the steepness of the slopes and the long, flat, curved valley.

(c) The Brachistochrone Problem (with friction). The Brachistochrone problem seeks to find the minimum-time path between two points for a particle acted on only by gravity (think of a bead on a wire). Bernoulli posed this problem to the mathematics community (some interesting history here). This problem can be solved in infinite dimensions using calculus of variations—theory which resulted in part from the discussion around this problem. We will solve the problem discretely using nonlinear optimization. Additionally, we will add friction to the problem.

The bead starts at some y-position H and begins from rest. For convenience we define the starting point at $x = 0$. From conservation of energy we can then find the velocity of the bead at any other location.

$$\begin{aligned} E_0 &= E \\ mgH &= \frac{1}{2}mv^2 + mgy + \int_0^x \mu_k mg\cos\theta ds \\ 0 &= \frac{1}{2}v^2 + g(y - H) + \mu_k g x \\ v &= \sqrt{2g(H - y - \mu_k x)} \end{aligned} \tag{3}$$

where the last term is the energy equation is the dissipative work from friction acting along the path length.

The time taken to traverse an infinitesimal element is

$$\begin{aligned} dt &= \int_{x_i}^{x_i+dx} \frac{ds}{v(x)} \\ &= \int_{x_i}^{x_i+dx} \frac{\sqrt{dx^2 + dy^2}}{\sqrt{2g(H - y(x) - \mu_k x)}} \\ &= \int_{x_i}^{x_i+dx} \frac{\sqrt{1 + \left(\frac{dy}{dx}\right)^2}\, dx}{\sqrt{2g(H - y(x) - \mu_k x)}} \end{aligned} \tag{4}$$

Consider discretizing the domain into linear segments. The slope $s_i = (\Delta y/\Delta x)_i$ is then a constant along a given segment, and $y(x) = y_i + s_i(x - x_i)$. Making these substitutions results in

$$\Delta t_i = \frac{\sqrt{1 + s_i^2}}{\sqrt{2g}} \int_{x_i}^{x_{i+1}} \frac{dx}{\sqrt{H - y_i - s_i(x - x_i) - \mu_k x}} \tag{5}$$

Performing the integration and simplifying (many steps omitted here) results in

$$\Delta t_i = \sqrt{\frac{2}{g}} \frac{\sqrt{\Delta x^2 + \Delta y^2}}{\sqrt{H - y_{i+1} - \mu_k x_{i+1}} + \sqrt{H - y_i - \mu_k x_i}} \tag{6}$$

where $\Delta x = (x_{i+1} - x_i)$ and $\Delta y = (y_{i+1} - y_i)$. The objective of the optimization is to minimize the total travel time, so we need to sum up the travel time across all of our linear segments

$$T = \sum_{i=1}^{n-1} \Delta t_i \tag{7}$$

Minimization is unaffected by multiplying by a constant, so we can remove the multiplicative constant for simplicity (we see that the magnitude of the acceleration of gravity has no affect on the optimal path)

$$\text{minimize} \quad \sum_{i=1}^{n-1} \frac{\sqrt{\Delta x^2 + \Delta y^2}}{\sqrt{H - y_{i+1} - \mu_k x_{i+1}} + \sqrt{H - y_i - \mu_k x_i}} \tag{8}$$

The design variables are the n positions of the path parameterized by y_i. Note that x is a parameter, meaning that it is fixed. You could space the x_i any reasonable way and still find the same optimal curve, but it is easiest to just use uniform spacing.

As the dimensionality of the problem increases, the solution becomes more challenging. You may want to test out different problem sizes, friction coefficients, and starting points, but for the purposes of our competition we will use the following specifications:

- starting point: $(x, y) = (0, 1)$
- ending point: $(x, y) = (1, 0)$
- kinetic coefficient of friction $\mu_k = 0.3$
- number of points from x_{start} to x_{end} (evenly spaced): $n = 60$ (which means there are 58 design variables)
- starting point: linear line from starting point to ending point

The analytic solution for the case with friction is more difficult to derive, but the analytic solution for the *frictionless* case with our starting and ending points is:

$$\begin{aligned} x &= a(\theta - \sin(\theta)) \\ y &= -a(1 - \cos(\theta)) + 1 \end{aligned} \tag{9}$$

for $a = 0.572917$ and $\theta = 0 \ldots 2.412$. Again, this is not the analytic solution for the case with friction, but is provided just as a check that you are on the right track.

Test Procedure:

Each problem will be tested from 10 randomly generated starting points, with the same random points used for all participants. The Brachistochrone problem is an exception and will only use one starting point, a straight line, but seven different discretizations (4, 8, 16, 32, 64, 128, and 256 points). Most importantly, the algorithm should be accurate. This will be measured by

$$||g(x)||_\infty < \tau \tag{10}$$

where τ is a requested tolerance that will generally be 1×10^{-6} (a larger tolerance of 1×10^{-5} will be used for the Brachistochrone problem). A trial, one starting point for one problem, that fails to arrive at the correct solution with the requested tolerance will receive 0 points. If the trial satisfies the accuracy check, then the total number of function calls will be recorded. The score for that trial will be $n_{f_{nominal}}/n_f$, where n_f is the total number of function calls and $n_{f_{noiminal}}$ is a typical number of iterations for that problem used to help normalize the problems relative to each other. In other words, faster algorithms (that meet the minimum accuracy requirement) will be awarded higher scores. Your score for each problem will be the average of your scores for each starting point. (Again, these scores have nothing to do with your grade on the homework).

Notes:

- Be careful that you do not to try to optimize the end points of the path. Those should be fixed.
- Belegundu presents an alternative setup for solving the discretized Brachistochrone problem. I have posted a copy on Learning Suite for those interested. It also has some helpful figures to visualize the problem. The mathematical derivation is very simple and has only a couple steps after the energy balance. None of the mathematical gymnastics used here were required. However, the resulting optimization problem is much more difficult to solve, especially as the dimensionality increases. Their methodology appears to depend on g (but of course really does not and can cause serious scaling issues), the arguments in their square roots can become negative (and they sometimes do during the course of an optimization which causes the solution to fail), and unlike this method where each segment can be solved independently their derivation requires each segment to be computed sequentially (which has efficiency implications). I note this because it is *very* common to need to rearrange and reconsider your problem formulation so that it is more amenable to optimization. This can make a *huge* difference, and is a skill that is underdeveloped by many optimization users. We will discuss this concept more throughout the course.
- You should not get the impression from this assignment that writing your own algorithms is superior to using existing implementations. It is not. There are many excellent optimization packages out there that are robust and have great performance. Development of these tools takes many years and lots of expertise. The reason for this exercise is that I have found that there is a big difference between users who know how to connect their code to an optimization algorithm as compared to users who really know how to use optimization algorithms because they've spent time trying to implement one themselves. We can talk about the theory all we want, but it is hard to really understand what is going on without wrestling with the details.