Assignment #1: Intro to Optimization — ME 575

due 1/22/2020 before midnight via Learning Suite — 50 possible points

Overview: These two problems involve using existing optimization algorithms. The goal is to help you become familiar with the process before we start diving into the details. The first problem is unconstrained, and the second adds constraints.

1.1 Unconstrained Brachistochrone Problem: Solve the Brachistochrone Problem (with friction) using an unconstrained optimizer (e.g., `fminunc` in Matlab). The Brachistochrone problem seeks to find the fastest path, meaning minimum time, between two points for a particle acted on only by gravity (think of a bead on a wire). Details on this problem are at the end of this document.

Complete the following:

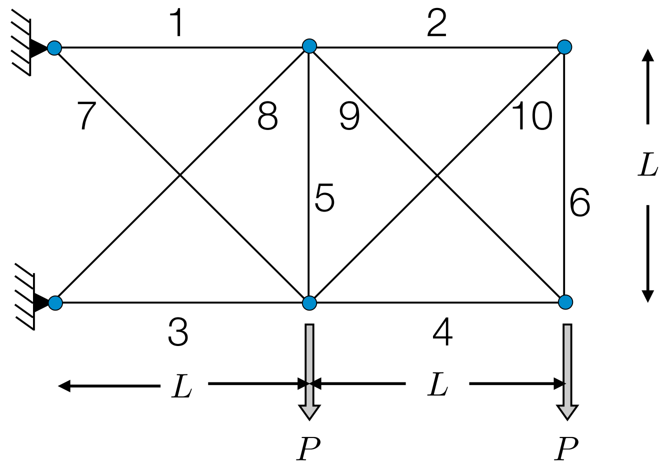
(a) Plot the optimal shape with decent resolution (i.e., not 4 points).

(b) Report the travel time between the two points. Don't forget to put $g = 9.81$, the acceleration of gravity, back in.

(c) Study the effect of increased problem dimensionality. Start with 4 points and double the dimension each time up to the highest you can manage without waiting too long (i.e., 4, 8, 16, 32, 64, ...). Plot and discuss the increase in computational expense with problem size. Example metrics include things like major iterations, functional calls, wall time, etc. You might want to explore 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) as compared to starting with the straight line initial condition.

1.2 Constrained Truss Problem: In ME EN 372 you "optimized" a truss, using ANSYS, by manually changing cross-sectional areas, positions, and materials. We are going to solve a related problem using actual optimization methods. You will need to use a nonlinear constrained optimization method (e.g., `fmincon` in Matlab). For this problem, we will find the optimal cross-sectional areas for the 10-bar truss shown below.

I've written a truss finite element code for you and set it up for this particular configuration (`truss.m`). You just need to use the provided function `[mass, stress] = truss(A)`. Further details are explained in the matlab file.



The objective of the optimization is to minimize the mass of the structure, subject to the constraints that every segment does not yield in compression or tension. The yield stress of all elements is 25×10^3

psi, except for member 9 which uses a stronger alloy with a yield stress of 75×10^3 psi. Mathematically the constraint is (where the absolute value is needed to handle tension and compression):

$$|\sigma_i| \leq \sigma_{y_i} \quad \text{for } i = 1 \ldots 10 \tag{1}$$

Absolute values are not differentiable at 0 and so should be avoided in gradient-based optimization if possible. Put this in a mathematically equivalent form that avoids absolute value. Each element should have a cross-sectional area of at least 0.1 in^2 for manufacturing reasons. This type of constraint is called a bound constraint. In solving this optimization problem you may need to consider scaling the objective and constraints.

Report the following:

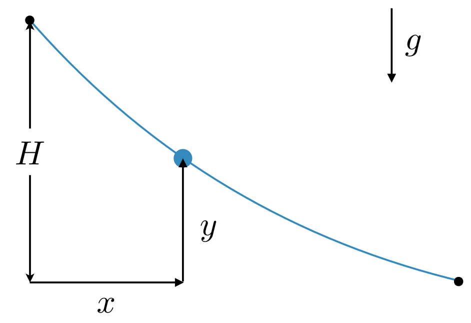
(a) the optimal mass and corresponding cross-sectional areas

(b) a convergence plot: x-axis should be some measure of computational time (e.g., major iterations, function calls) on a linear scale, the y-axis should be what Matlab calls firstorderopt on a log scale. We will learn later what firstorderopt means.

(c) the number of function calls required to converge (functions calls from the `truss` function)

Discuss your findings.

Brachistochrone Problem



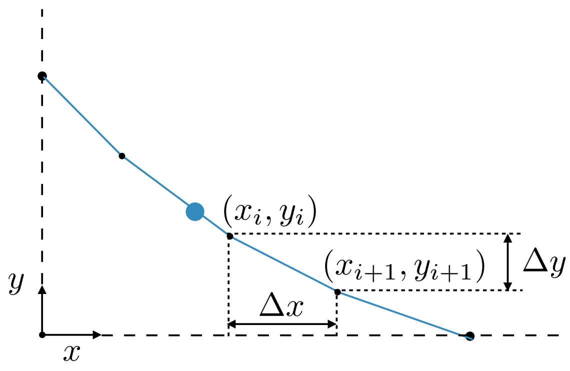
Bernoulli posed this problem to the mathematics community (some interesting history here). This problem can be solved in infinite dimensions using calculus of variations—theory which resulted in part from the discussion around this problem. We will solve the problem by discretizing the wire and using 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. The initial potential energy is converted to kinetic energy, potential energy, and dissipative work from friction acting along the path length:

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

Now that we know the speed of the bead as a function of x, we can compute the time it takes to traverse an infinitesimal element of length ds:

$$\begin{aligned} \Delta t &= \int_{x_i}^{x_i+dx} \frac{ds}{v(x)} \\ &= \int_{x_i}^{x_i+dx} \frac{\sqrt{dx^2+dy^2}}{\sqrt{2g(H-y(x)-\mu_k x)}} \\ &= \int_{x_i}^{x_i+dx} \frac{\sqrt{1+\left(\frac{dy}{dx}\right)^2}dx}{\sqrt{2g(H-y(x)-\mu_k x)}} \end{aligned} \tag{3}$$



To solve this discretely, consider dividing up the wire into linear segments. As an example, the above figure shows the wire divided into 4 linear segments (5 nodes) as an approximation of a continuous wire. 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

$$\Delta t_i = \frac{\sqrt{1+s_i^2}}{\sqrt{2g}} \int_{x_i}^{x_{i+1}} \frac{dx}{\sqrt{H - y_i - s_i(x-x_i) - \mu_k x}} \tag{4}$$

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

$$\Delta t_i = \sqrt{\frac{2}{g}} \frac{\sqrt{\Delta x_i^2 + \Delta y_i^2}}{\sqrt{H - y_{i+1} - \mu_k x_{i+1}} + \sqrt{H - y_i - \mu_k x_i}} \tag{5}$$

where $\Delta x_i = (x_{i+1} - x_i)$ and $\Delta y_i = (y_{i+1} - y_i)$. 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

$$T = \sum_{i=1}^{n-1} \Delta t_i \tag{6}$$

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)

$$\text{minimize} \quad J = \sum_{i=1}^{n-1} \frac{\sqrt{\Delta x_i^2 + \Delta y_i^2}}{\sqrt{H - y_{i+1} - \mu_k x_{i+1}} + \sqrt{H - y_i - \mu_k x_i}} \tag{7}$$

The design variables are the $n-2$ positions of the path parameterized by y_i. The end points must be fixed, otherwise the problem is ill-defined, which is why there are $n-2$ design variables instead of n. 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 underlying optimal curve, but it is easiest to just use uniform spacing.

As the dimensionality of the problem increases, the solution becomes more challenging. We will use the following specifications:

- starting point: $(x, y) = (0, 1)$ m
- ending point: $(x, y) = (1, 0)$ m
- kinetic coefficient of friction $\mu_k = 0.3$

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:

$$\begin{aligned} x &= a(\theta - \sin(\theta)) \\ y &= -a(1 - \cos(\theta)) + 1 \end{aligned} \tag{8}$$

with $a = 0.572917$ 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.

Notes:

- Again, do not to try to optimize the end points of the path. Those should be fixed. So if $n = 4$, you only have two design variables.
- 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 scaling issues),

the arguments in their square roots can become negative and they often do during the course of an optimization which causes the solution to fail. Also unlike this method where each segment can be solved independently their derivation requires each segment to be computed sequentially (which can have efficiency implications). I note this because it is 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.

- It is still possible with the present formulation for the arguments of the square roots to become negative, but we can address this. The simplest is to just ignore it. As long as you start with a reasonable starting point it shouldn't be an issue for this problem. A robust approach is to add a simple bound constraint, but if you want to do that you need to switch to a constrained solver. A crude approach (but effective on this problem), would be to add a penalty. For example, if you had a constraint like $y_i < 0.5$, you could use a penalty of the form $(y_i - 0.5)$ or $(y_i - 0.5)^2$ or some scalar multiple of those. You could probably even add a constant penalty (i.e., 10). A penalty will increase your objective and encourage a return to feasible space.