IC-32 **Optimal Control** **Winter 2006**

Benoît Chachuat
ME C2 401, Ph: 33844, benoit.chachuat@epfl.ch

Problem Set #4 (With Corrections)

1. Consider the problem of finding the smooth curve $y(x), x_A \leq x \leq x_B$, in the vertical plane (x, y), joining given points $A = (x_A, y_A)$ and $B = (x_B, y_B)$, $x_A < x_B$, and such that a material point sliding along $y(x)$ without friction from A to B, under gravity and with initial speed $v_A \geq 0$, reaches B in a minimal time (see Fig. 1).

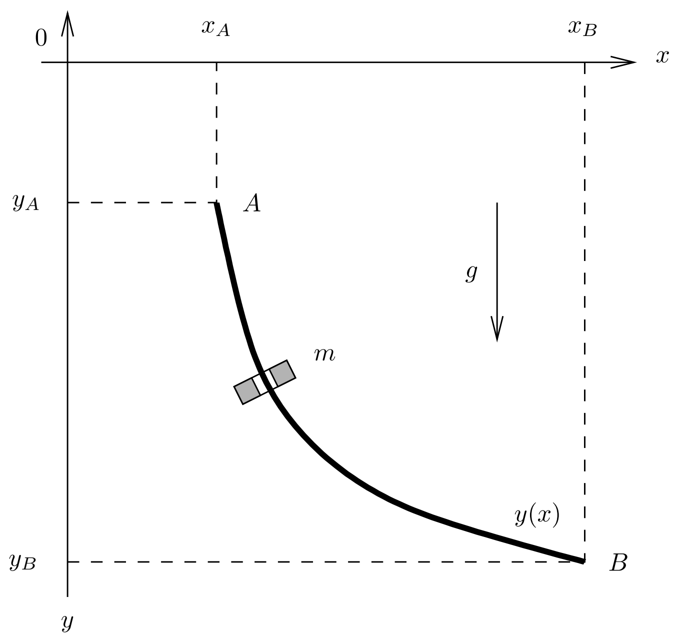

Figure 1: Brachistochrone problem.

(a) Consider the case where $x_A = y_A = 0$. Show that the problem can be expressed in mathematical terms as the following problem of the calculus of variations:

$$\text{minimize:} \quad \mathcal{J}(y) = \int_0^{x_B} \sqrt{\frac{1 + \dot{y}(x)^2}{v_A^2 - 2gy(x)}} \mathrm{d}x \tag{1}$$

$$\text{subject to:} \quad y \in \mathcal{D} := \{y \in \mathcal{C}^1[0, x_B] : y(0) = 0, y(x_B) = y_B\},$$

where g denotes the gravity acceleration.

Solution. The objective function $\mathcal{J}(y)$ is the time required for the point to travel from $(0, 0)$ to (x_B, y_B) along the curve $y(x)$:

$$\mathcal{J}(y) = \int_0^{x_B} \mathrm{d}t = \int_0^{x_B} \frac{\mathrm{d}s}{v(x)},$$

where s denotes the Jordan length of $y(x)$, defined by $\mathrm{d}s = \sqrt{1+\dot{y}(x)^2}\mathrm{d}x$, and v, the velocity along $y(x)$. Since the point is sliding along $y(x)$ without friction, energy is conserved,

$$\frac{1}{2}m\left(v(x)^2 - v_A^2\right) - mgy(x) = 0,$$

with m being the mass of the point, and g, the gravity acceleration. That is, $v(x) = \sqrt{v_A^2 - 2gy(x)}$, and

$$\mathcal{J}(y) = \int_0^{x_B} \sqrt{\frac{1+\dot{y}(x)^2}{v_A^2 - 2gy(x)}}\mathrm{d}x.$$

(b) We wish to calculate an approximate solution to this problem by parameterizing the curve $y(x)$ using Lagrange polynomials of order $N \geq 1$:

$$Y(\tilde{\mathbf{y}};x) \ := \ \sum_{k=0}^{N} \tilde{y}_k \prod_{\substack{j=0\\ j\neq k}}^{N} \frac{x - X_j}{X_k - X_j},$$

where $\tilde{\mathbf{y}} = (\tilde{y}_0, \ldots, \tilde{y}_N)^\mathsf{T}$; and $X_0, \ldots, X_N$ are $N+1$ points in $[0, x_B]$. Here, we shall consider equally spaced points:

$$X_0 := 0, \quad \text{and} \quad X_k := X_{k-1} + \frac{x_B}{N}, \quad k = 1, \ldots, N.$$

Using this parameterization, (1) is transformed into the following (finite-dimensional) constrained NLP problem:

$$\begin{aligned}
\text{mininimize:} \quad & f(\tilde{\mathbf{y}}) := \int_0^{x_B} \sqrt{\frac{1+Z(\tilde{\mathbf{y}};x)^2}{v_A^2 - 2gY(\tilde{\mathbf{y}};x)}}\mathrm{d}x \\
\text{subject to:} \quad & Y(\tilde{\mathbf{y}};0) \ = \ 0 \\
& Y(\tilde{\mathbf{y}};x_B) \ = \ y_B \\
& \tilde{y}_k \leq \frac{v_A^2}{2g}, \quad k = 0, \ldots, N,
\end{aligned}$$

with

$$\begin{aligned}
X_k \ &:= \frac{k}{N}x_B, \quad k = 0, \ldots, N \\
Y(\tilde{\mathbf{y}};x) \ &:= \sum_{k=0}^{N} \tilde{y}_k \prod_{\substack{j=0\\ j\neq k}}^{N} \frac{x - X_j}{X_k - X_j} \\
Z(\tilde{\mathbf{y}};x) \ &:= \sum_{k=0}^{N} \tilde{y}_k \sum_{\substack{i=0\\ i\neq k}}^{N} \frac{1}{X_k - X_i} \prod_{\substack{j=0\\ j\neq i,k}}^{N} \frac{x - X_j}{X_k - X_j}.
\end{aligned}$$

Compute an optimal solution $\tilde{\mathbf{y}}^\star$ to this problem using the function `fmincon` in MATLAB®'s Optimization Toolbox:

- Consider $N = 10$ stages, and take the values $x_B = 1$, $y_B = -0.5$ and $v_A = 1$ for the parameters;
- Specify the **initial guess** as $\tilde{y}_k^0 = \frac{k}{N}y_B$, for $k = 0, \ldots, N$;
- Make sure that the medium-scale **SQP algorithm**, with Quasi-Newton update and line-search, is the selected solver in `fmincon`;
- Set the solution point tolerance, function tolerance and constraint tolerance to 10^{-7} in `fmincon`;

- Use the function quad or quadl to compute the value of the integral cost, with tolerance set to 10^{-7};
- For simplicity, let fmincon calculate a **finite-difference approximation** for the gradients of the objective function and constraints. In particular, set the minimum change in variables for finite differencing to 10^{-6};
- M-files calculating Lagrange polynomials (lagrange.m) and their derivative (dlagrange.m) can be retreived from the class website.

Solution. A possible implementation is as follows (remark that increments start from 1 in the implementation, not 0):

brachist-main.m

```
clear all

% Number of Stages
N = 11;

% Parameters
xB = 1.;
yB = -.5;
vA = 1.;
g  = 10.;
for k = 1:N
    y0(k) = (k-1.)/(N-1.)*yB;
    yu(k) = vA^2/(2*g);
    xk(k) = (k-1.)/(N-1.)*xB;
end

% Optimization Options
options = optimset('GradObj', 'off', 'GradConstr', 'off', 'Display', 'iter',...
                   'LargeScale', 'off', 'HessUpdate', 'bfgs', 'Diagnostics', 'on',...
                   'TolX', 1e-7, 'TolFun', 1e-7, 'TolCon', 1e-7, 'MaxIter', 1000,...
                   'MaxFunEval', 1000, 'DiffMinChange', 1e-6 );


% Solve NLP Problem
[yopt, fopt, iout] = fmincon( @(yk)brachistFun(yk,xk,xB,vA,g), y0, [], [], ...
                              [], [], [], yu, @(yk)brachistCtr(yk,xk,xB,yB), ...
                              options );

% Plot Results
clf;
figure(1);
x = [0:0.01:xB];
[dum,P] = size(x);
for k = 1:P
    y(k) = lagrange(yopt,xk,x(k));
end
plot(x,y,'r');
```

brachistFun.m

```
%%%%%%%%%%%%%%% FUNCTION TO BE MINIMIZED %%%%%%%%%%%%%%%
% Brachistochrone Cost:
%      f = int_0^xB sqrt((1-dy^2)/(vA^2/(2*g)-y) dx
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [f] = brachistFun(yk,xk,xB,vA,g)
    qtol = 1.e-7;
    disp = 0;
    f = f + quadl( @(x)brachistInt(x,yk,xk,vA,g), 0, xB, qtol, disp );
end
```

brachistInt.m

```
%%%%%%%%%%%%%%%%%%%%%%% INTEGRAND %%%%%%%%%%%%%%%%%%%%%%
% Brachistochrone Integrand:
%          l = sqrt((1-dy^2)/(vA^2/(2*g)-y)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function l = brachistInt(x,yk,xk,vA,g)
    [dum,P] = size(x);
    for i = 1:P
        l(i) = sqrt((1+dlagrange(yk,xk,x(i))^2) ...
             / (vA^2-2.*g*lagrange(yk,xk,x(i))));
    end
end
```

brachistCtr.m

```
%%%%%%%%%%%%%%%%%%%%%% CONSTRAINTS %%%%%%%%%%%%%%%%%%%%%
% Brachistochrone end_Points:
%      geq(1) = y(0)  - 0.
%      geq(2) = y(xB) - yB
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [gin,geq] = brachistCtr(yk,xk,xB,yB)
  gin = [];
  geq(1) = lagrange(yk,xk,0.);
  geq(2) = lagrange(yk,xk,xB) - yB;
end
```

lagrange.m

```
% y = lagrange(wk,xk,x)
% arguments:
%   wk : row vector of size N+1 containing the interpolation coefficients
%   xk : row vector of size N+1 containing the interpolation points, xk(1)<...<xk(N+1)
%   x  : points at which the Lagrange polynomial is to be evaluated
% output:
%   y  : value of the Lagrange polynomial at x
function y = lagrange(wk,xk,x)
    [dum,N] = size(wk);
    y = 0.;
    for k = 1:N
        prod = wk(k);
        for j = 1:N
            if j ~= k
                prod = prod * (x-xk(j)) / (xk(k)-xk(j));
            end
        end
        y = y + prod;
    end
```

```
end
```

dlagrange.m

```
% dy = dlagrange(wk,xk,x)
% arguments:
%   wk : row vector of size N+1 containing the interpolation coefficients
%   xk : row vector of size N+1 containing the interpolation points, xk(1)<...<xk(N+1)
%   x  : points at which the Lagrange polynomial is to be evaluated
% output:
%   y  : value of the Lagrange polynomial at x
function dy = dlagrange(wk,xk,x)
    [dum,N] = size(wk);
    dy = 0.;
    for k = 1:N
        sum = 0.;
        for i = 1:N
            if i ~= k
                prod = 1.;
                for j = 1:N
                    if j ~= k && j ~= i
                        prod = prod * (x-xk(j)) / (xk(k)-xk(j));
                    end
                end
                sum = sum + prod / (xk(k)-xk(i));
            end
        end
        dy = dy + wk(k) * sum;
    end
end
```

The optimal solution found for $N = 10$ is:

$$\tilde{\mathbf{y}}^\star \approx \begin{pmatrix} 0 \\ -0.1751 \\ -0.2753 \\ -0.3485 \\ -0.4029 \\ -0.4446 \\ -0.4748 \\ -0.4945 \\ -0.5056 \\ -0.5072 \\ -0.5000 \end{pmatrix},$$

and the corresponding minimum time for the point to slide from A to B is:

$$f(\tilde{\mathbf{y}}^\star) \approx 0.463436.$$

A plot of the optimal curve for $N = 10$ is shown in Fig. 2 below. Note that, for the moment, we do not have any way of checking whether the resulting curve is close to the true optimal curve for the original problem (i.e., without using a parameterization of the curve $y(x)$ using Lagrange polynomials). We just hope that by increasing the number N of stages, the solution will get closer to that of the original problem.

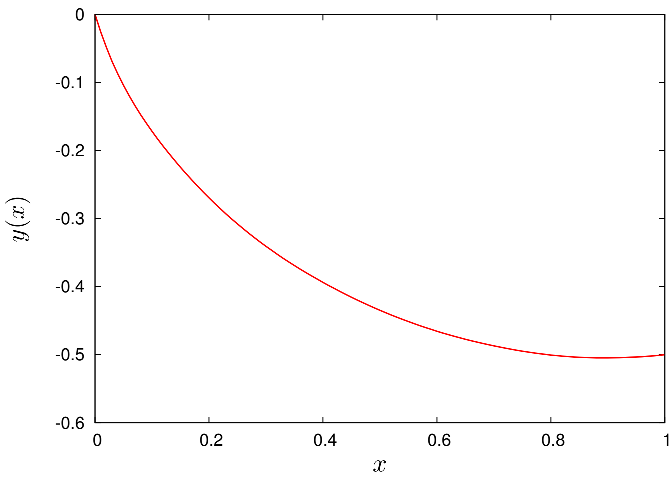

Figure 2: Optimal curve for the parameterized brachistochrone problem, with $N = 10$ stages, $x_B = 1$, $y_B = -0.5$, $v_A = 1$ and $g = 10$.