Variational Methods and Optimal Control
Numerical Questions

Matthew Roughan
<matthew.roughan@adelaide.edu.au>

Some numerical questions and examples:

1. **Approximation:** In many applications, we want to approximate a piece-wise continuous function $R(x)$ with a smooth curve. For instance:

   - it is common to compile histograms of a set of data, but a typical pltted histogram can look artificially "blocky";
   - in many signal processing applications we will have a signal (often sampled at discrete time intervals), that contains noise, and one method for removing white noise is to smooth the data.

   We can approximate a piecewise constant curve arbitrarily closely, but the cost is the slope of the approximating function may become arbitrarily large. The resulting approximation won't look any smoother than the original.

   To obtain a "smoother" approximation, we need to constrain the derivative of the approximation function in some way, but a fixed constraint $|y'| \leq c$ would be rather rigid. Instead let us solve the optimization problem to minimize

   $$F\{y\} = \int_a^b \gamma^2 (y - R)^2 + {y'}^2 \, dx,$$

   over the interval $[a, b]$, with $y(x)$ free at each end point, where the parameter γ allows a trade off between degree of "smoothness" against a closer fit to the original.

   Solution: The Euler-Lagrange equations are

   $$\frac{d}{dx}\frac{\partial f}{\partial y'} - \frac{\partial f}{\partial y} = 2y'' - 2\gamma^2(y - R) = 0.$$

   So to find our approximation we need to find the solution to

   $$y'' - \gamma^2 y = -\gamma^2 R(x),$$

   for piece-wise constant function $R(x)$. The solution to the homogeneous DE $y'' - \gamma^2 y = 0$ is

   $$y_h = Ae^{\gamma x} + Be^{-\gamma x}.$$

   Break the interval $[a, b]$ into the segments where $R(x)$ is constant, i.e., take a series of points $a = x_0 < x_1 < \cdots < x_n = b$ where $R(x) = c_i$ for all $x \in [x_i, x_{i+1})$, then on each segment the DE will have particular solution $y_i = c_i$ and so on that segment we get

   $$\begin{aligned} y_i(x) &= a_i e^{\gamma x} + b_i e^{-\gamma x} + c_i, \\ y_i'(x) &= a_i \gamma e^{\gamma x} - b_i \gamma e^{-\gamma x}, \end{aligned}$$

   and the constants a_i and b_i will be determined by the end-points (which are not known for each segment *a priori*).

   To find the actual approximating curve we have to piece together a series of such segments so that

   (a) the end-point values all match so the curve is continuous,

(b) the derivatives match at the end points, so the curve is smooth, and

(c) the natural boundary conditions (where here are equivalent to $y' = 0$ at the boundary) are also satisfied.

The result will be our smooth interpolation of the piecewise constant curve.

In detail, at each point x_i for $i = 1, 2, \ldots, n-1$ we get a pair of conditions

$$\begin{aligned} y_{i-1}(x_i) &= y_i(x_i) \\ y'_{i-1}(x_i) &= y'_i(x_i) \end{aligned}$$

which is $2(n-1)$ conditions + the two natural boundary conditions where we have $2n$ unknowns (the (a_i, b_i) for $i = 0, \ldots, n-1$). Using the form of the solution y_i in our answer we get

$$\begin{aligned} a_{i-1}e^{\gamma x_i} + b_{i-1}e^{-\gamma x_i} - a_i e^{\gamma x_i} - b_i e^{-\gamma x_i} &= c_i - c_{i-1} \\ \gamma\left[a_{i-1}e^{\gamma x_i} - b_{i-1}e^{-\gamma x_i} - a_i e^{\gamma x_i} + b_i e^{-\gamma x_i}\right] &= 0 \end{aligned}$$

for $i = 1, 2, \ldots, n-1$, and the natural boundary conditions

$$\begin{aligned} a_0 e^{\gamma a} - b_0 e^{-\gamma a} &= 0 \\ a_{n-1}e^{\gamma b} - b_{n-1}e^{-\gamma b} &= 0 \end{aligned}$$

We have $2n$ linear equations in $2n$ unknowns. Solving gives the co-efficients we need to obtain the solution.

For an example, see Figure 1, which shows the approximations for three different γ values.

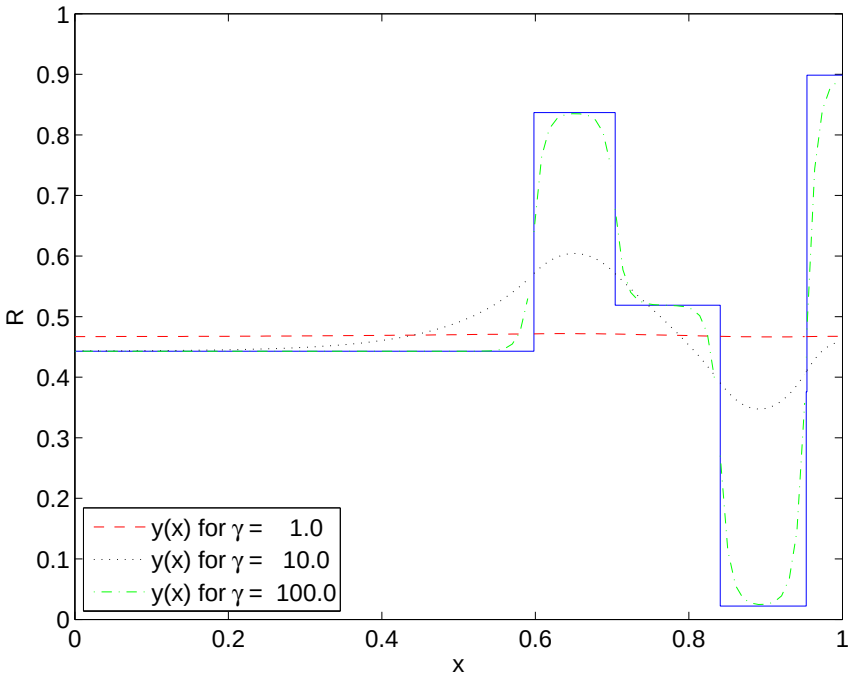


Figure 1: Approximations for three different γ values.

Remarks: We can obviously generalize this method to deal with approximation of curves that are piecewise linear, or for interpolating a function which we only know at some points.

Moreover, the method above is also a special case of a more general class of smoothing and approximation algorithms. In general, we are seeking to minimise some *distance* between our approximation and the original function, while also meeting some smoothness criteria. In general we might write this as find the curve y that minimises

$$F\{y\} = \gamma d_z\{y\} + I\{y\},$$

where d is a distance metric between the original function z and the approximation y, and $I(\cdot)$ is a function that measures the irregularity, or lack of smoothness of y. In the case above, both of these are integrals:

$$\begin{aligned} d_z\{y\} &= \int_a^b (y-z)^2\,dx, \\ I\{y\} &= \int_a^b {y'}^2\,dx. \end{aligned}$$

However, we could replace these with alternatives.

- We could measure irregularity by minimizing curvature of the aprpoximating functiun, as measured by

  $$I\{y\} = \int_a^b y''^2\,dx,$$

  or with higher-order derivatives, or different powers, or some combination thereof.
- We can use different powers p in the distance function (corresponding to the ℓ_p-norm distance. We could also measure the distance only on a set of sample points x_i, e.g., we could take a function

  $$d_z\{y\} = \sum_i \big(y(x_i) - y_i\big)^2.$$

- We can replace either of the two optimization criteria with hard constraints, e.g., $|y'| \le 1$, or $y(x_i) - y_i = 0$, which can be incorporated into the problem in the usual way.

Particular combinations of these two result in well-known special cases:

- If we require that $y(x_i) = y_i$, e.g., we require the curve to match $R(x)$ at the grid points x_i, and minimize curvature of the approximation should be minimal, then the Euler-Poisson equations are

  $$y^{(3)} = 0$$

  subject to the constraints, so the curves will be piece-wise cubic. This leads naturally to interpolating splines.
- Smoothing splines arise if we aim to minimize

  $$F\{y\} = \gamma \sum_i \big(y(x_i) - y_i\big)^2 + \int_a^b y''^2\,dx.$$

  As γ approaches 1, we approach the interpolative splines above, whereas, when γ approaches zero we obtain a linear-regression to the data.
- De Boor extended this to allow for weights on each data point, and higher-order derivatives, e.g.,

  $$F\{y\} = \gamma \sum_i \left(\frac{y(x_i) - y_i}{\delta_i}\right)^2 + \int_a^b {y^{(m)}}^2\,dx.$$

The critical point is that nearly all of these "smoothers" can be derived from the calculus of variations.

The choice of smoothing parameter γ is a hard problem in general and is often a little arbitrary, though in specific cases there may be methods to choose it quantitatively, based on known issues such as the level of noise in a dataset.

Matlab code: for performing calculating the co-efficients for an arbitrary piece-wise constant function is included below:

```
function [a,b,c] = approx_function(x_0, x_n, x_i, R_i, gamma)
% file:       approx_function.m, (c) Matthew Roughan, Thu Oct 28 2010
% created:  Thu Oct 28 2010
% author:   Matthew Roughan, matthew.roughan@adelaide.edu.au
%
% R is a piece-wise constant function which we wish to approximate
% with minimal deviation, and slope, i.e., minimize
%    \[ F\{ y \} = \int_{a}^{b} \gamma^2 (y-R)^2 + y'^2 \, dx , \]
%
% input:
%    x_0
%    x_n = the function R(x) is defined on [x_0, x_n]
%    x_i = a nx1 vector of the x-values where the function R changes
%              x_i(1) = x_0
%            but x_n is not included in the vector x_i
%    R_i = a nx1 vector of the values of R(x) on each sub-interval
%    gamma = the tradeoff parameter between fidelity to the function and the
%              allowed slope of the approximation
%
% outputs:
%    the optimal approximation function takes the form
%          y_i(x) =  a_i e^{\gamma x} + b_i e^{-\gamma x} + c_i
%    on each interval [x_i(i), x_i(i+1)]. The outputs of the function are vectors of
%    the coefficients a_i and b_i
%        a = a nx1 vector of the coefficients of a_i in the above approximation
%        b = a nx1 vector of the coefficients of b_i in the above approximation
%        c = a nx1 vector of the coefficients of c_i in the above approximation (here c_i = R_i)
%
n = length(x_i);

% check inputs
sx = size(x_i);
sR = size(R_i);
if (sx(1) == 1 & sx(2) > 1)
  x_i = x_i';
elseif (sx(1) > 1 & sx(2) > 1)
  error('sx should be  a nx1 vector');
end
if (sR(1) == 1 & sR(2) > 1)
  R_i = R_i';
elseif (sR(1) > 1 & sR(2) > 1)
  error('sR should be  a nx1 vector');
end
if (sx(1) ~= sR(1) | sx(2) ~= sR(2))
  error('x and R should be the same size');
end
if (x_0 ~= x_i(1))
  error('we should have x_i(1)=x_0');
end

% temporary variables
c = R_i;
d = exp(gamma*x_i);
e = exp(-gamma*x_i);

% set up continuity equations at (n-1) joins x_i(2:end)
%  a_{i-1} e^{\gamma x_i} + b_{i-1} e^{-\gamma x_i} - a_{i} e^{\gamma x_i} - b_{i} e^{-\gamma x_i}  & = & c_{i
%   matrix A1 are coefficients for the a_i
%   matrix B1 are coefficients for the b_i
%   y1 are the right-hand side terms of the equations (c_{i} - c_{i-1)
A1 = zeros(n-1,n);
B1 = zeros(n-1,n);
for i=1:n-1
  A1(i,i) = d(i+1);
```

```
  A1(i,i+1) = -d(i+1);
  B1(i,i) = e(i+1);
  B1(i,i+1) = -e(i+1);
end
y1 = [diff(c)];

% set up continuity equations for derivatives at (n-1) joins x_i(2:end)
% \gamma \left[ a_{i-1}  e^{\gamma x_i} - b_{i-1} e^{-\gamma x_i} - a_i   e^{\gamma x_i} + b_i e^{-\gamma x_i}
%   matrix A2 are coefficients for the a_i
%   matrix B2 are coefficients for the b_i
%   y2 are the right-hand side terms of the equations (zero in this case)
A2 = zeros(n-1,n);
B2 = zeros(n-1,n);
for i=1:n-1
  A2(i,i) = d(i+1);
  A2(i,i+1) = -d(i+1);
  B2(i,i) = -e(i+1);
  B2(i,i+1) = e(i+1);
end
y2 = zeros(n-1,1);

% complete set of constraints at the joins
A = [[A1, B1]
     [A2, B2]];
y = [y1; y2];

% add in natural boundary conditions at the edges
%    natural boundary says y'=0 at x_0 and x_n
A = [[exp(gamma*x_0), zeros(1,n-1), -exp(-gamma*x_0), zeros(1,n-1)];
     A;
     [zeros(1,n-1), exp(gamma*x_n), zeros(1,n-1), -exp(-gamma*x_n)];
    ]
y = [0;y;0]

% solve the equations to find 'coefficients', which consists of
%      coefficients = [a; b]
% where a = [a_i], and b=[b_i] and    A*coefficients = y
coefficients = A \ y;
a = coefficients(1:n)
b = coefficients(n+1:end)
```

2. **Catenary:**

The shape of a hanging chain of length L was presented as the solution of the problem of minimizing potential energy

$$W_p\{y\} = mg \int_{x_0}^{x_1} y\sqrt{1+y'^2}\,dx,$$

under the isoperimetric constraint

$$G\{y\} = \int_{x_0}^{x_1} \sqrt{1+y'^2}\,dx = L.$$

assuming the (given) heights of the pylons $y_i = y(x_i) > 0$.

We determined that the solution to this problem took the form

$$y \quad = \quad c_1 \cosh\left(\frac{x-c_2}{c_1}\right) - \lambda,$$

where the constants λ, c_1 and c_2 are determined by the length L of the chain, and the end conditions, i.e., the heights of the poles $y(x_0) = x_0$ and $y(x_1) = x_1$.

Determine:

(a) the infimum of possible lengths of chain, and

(b) the maximum length before the chain drags on the ground (at height $y = 0$).

Solution:

(a) The minimum length of the chain is not well defined because although the minimum distance (along a straight line) is well defined, a chain cannot take this shape (except in the limit as tension in the chain goes to infinity). Hence we use the term infimum, which can be thought of as the *greatest lower bound*.

Formally, the *infimum* of a subset S of some partially ordered set T is the greatest element of T that is less than or equal to all elements of S. Here, the set T is the set of all curves $y(x)$ with two continuous derivatives, that satisfy the end-point constraints. The ordering is based on the lengths of the curves, and the subset S is the set of all catenaries. The infimum of the lengths will simply be the length of the straight line between the two end-points, because we know that we can get a catenary arbitraily close to this line. Hence

$$L_{\inf} = \sqrt{(x_1-x_0)^2 + (y_1-y_0)^2}.$$

That is, even though we can't have a catenary which is a straight line, it forms the greatest lower bound on the lengths of the catenaries, i.e., the infimum.

(b) We can calculate the maximum length, by noting that the chain will only drag on the ground if its minimum height $y_{\min} < 0$, so the maximum chain length will occur when $y_{\min} = 0$. Note that

$$\begin{aligned} y &= c_1 \cosh\left(\frac{x-c_2}{c_1}\right) - \lambda \\ y' &= \sinh\left(\frac{x-c_2}{c_1}\right), \end{aligned}$$

which has a zero at $x = c_2$. So there are three possible locations for the mimimum – the two edges, or the stationary points $y' = 0$.

$$x_{\min} = \begin{cases} x_0, & \text{if } c_2 \le x_0, \\ c_2, & \text{if } x_0 \le c_2 \le x_1, \\ x_1, & \text{if } c_2 \ge x_1. \end{cases} \tag{1}$$

In these cases we get

$$y_{\min} = \begin{cases} y_0, & \text{if } c_2 \le x_0, \\ c_1 - \lambda, & \text{if } x_0 \le c_2 \le x_1, \\ y_1, & \text{if } c_2 \ge x_1. \end{cases} \tag{2}$$

However, in order that we have the maxmium chain length $y_{\min} = 0$, so we know that $x_0 \leq c_2 \leq x_1$, and

$$c_1 = \lambda.$$

The form of the catenary is therefore

$$y = c_1 \left[\cosh\left(\frac{x - c_2}{c_1} \right) - 1 \right]$$

Note that $\cosh^2(x) - \sinh^2(x) = 1$, so

$$\begin{aligned} c_1 \sinh((x - c_2)/c_1)) &= \text{sign}(x - c_2) \sqrt{c_1^2 \cosh^2((x - c_2)/c_1)) - c_1^2} \\ &= \text{sign}(x - c_2) \sqrt{(y_1 + \lambda)^2 - c_1^2}. \end{aligned}$$

Hence

$$\begin{aligned} L\{y\} &= c_1 \left[\sinh((x - c_2)/c_1)) \right]_{x_0}^{x_1} \\ &= \left[\text{sign}(x - c_2) \sqrt{(y + \lambda)^2 - c_1^2} \right]_{x_0}^{x_1} \\ &= \begin{cases} \sqrt{(y_1 + \lambda)^2 - c_1^2} - \sqrt{(y_0 + \lambda)^2 - c_1^2}, & \text{if } c_2 < x_0, \\ \sqrt{(y_1 + \lambda)^2 - c_1^2}, & \text{if } c_2 = x_0, \\ \sqrt{(y_1 + \lambda)^2 - c_1^2} + \sqrt{(y_0 + \lambda)^2 - c_1^2}, & \text{if } x_0 < c_2 < x_1, \\ \sqrt{(y_0 + \lambda)^2 - c_1^2}, & \text{if } c_2 = x_1, \\ -\sqrt{(y_1 + \lambda)^2 - c_1^2} + \sqrt{(y_0 + \lambda)^2 - c_1^2}, & \text{if } c_2 > x_1. \end{cases} \end{aligned}$$

but as we know for the maximum length chain that $\lambda = c_1$ and $x_0 \leq c_2 \leq x_1$, we can write

$$L\{y\} = \sqrt{y_1(y_1 + 2c_1)} + \sqrt{y_0(y_0 + 2c_1)}.$$

We can determine c_1 and c_2 now by numerical solution of the end-point equations. We solve by constructing a function which is the square of the deviation from the two end-point constraints: *i.e.*, we define

$$\begin{aligned} g_1(c_1, c_2) &= y_0 - c_1 \cosh\left(\frac{x_0 - c_2}{c_1} \right) - c_1, \\ g_2(c_1, c_2) &= y_1 - c_1 \cosh\left(\frac{x_1 - c_2}{c_1} \right) - c_1, \end{aligned}$$

We then use Matlab's optimization toolbox function `fminsearch` to find the minimum of

$$g(c_1, c_2) = g_1^2 + g_2^2.$$

Then we use these to determine $L_{\max}$. Notice also that as $\cosh(x) \geq 1$ for all x, and is equal to 1 only at $x = 0$, the point at which the curve touches the ground will be c_2.

Matlab code: for performing estimating the catenary parameters is included below.

```
function [L_max, L_min, c_1, c_2, lambda] = catenary_max_length(y_0, y_1, x_0, x_1)
%
% file:         catenary_max_length.m, (c) 2012 Matthew Roughan
% author:   Matthew Roughan
% email:    matthew.roughan@adelaide.edu.au
%
%
% CATENARY_SOLVER: calculate the maximum (and min) length of a hangling chain before it dangles on the
%                  ground (is drawn taught), where it takes catenary shape
%                y = c_1*cosh((x-c_2)/c_1) - c_1
%         with fixed length
%                L = c_1.*( sinh((x_1-c_2)./c_1) - sinh((x_0-c_2)./c_1) );
%         but we need to work out the constants of integration c_1, c_2 and lambda
%
% INPUTS:
%         y_0       = height of the left pylon
%         y_1       = height of the right pylon
%         x_0       = left pylon position
%         x_1       = right pylon position
%
% OUTPUTS:
%         L_max     = the maximum length of the chain
%         L_min     = the infimum length of the chain
%         c_1,c_2,lambda = parameters of maximal catenary
%
%
%
%
L_min =  sqrt( (x_1 - x_0).^2 + (y_1 - y_0).^2 );

% create a function which we will minimize to find the solution
%   g1 is the left end-point constraint
%   g2 is the right end-point constraint
%   a = [c_1, c_2], lambda = c_1
g1 = @(a) ( y_0 - a(1)*cosh( (x_0 - a(2))/a(1) ) + a(1) ).^2;
g2 = @(a) ( y_1 - a(1)*cosh( (x_1 - a(2))/a(1) ) + a(1) ).^2;
g = @(a) g1(a) + g2(a);
a_est = [100, (x_0+x_1)/2];
options = optimset('fminsearch');
options = optimset(options, 'MaxFunEvals', 100000, 'MaxIter', 1000);
[a, fval, exitflag, output] = fminsearch(g, a_est, options);
c_1 = a(1); % must be > 0
c_2 = a(2); % must be between x_0 and x_1 for the maximum
lambda = -c_1; % for the maximum

g_val = [g1(a), g2(a)];

% compute the maximum length
L_max = c_1.*( sinh((x_1-c_2)./c_1) - sinh((x_0-c_2)./c_1) );
```

3. **Catenary (numerical solution of the constants):**

   The shape of a hanging chain of length L was presented as the solution of the problem of minimizing potential energy

   $$W_p\{y\} = mg \int_{x_0}^{x_1} y\sqrt{1+y'^2}\,dx,$$

   under the isoperimetric constraint

   $$G\{y\} = \int_{x_0}^{x_1} \sqrt{1+y'^2}\,dx = L.$$

   We determined that the solution to this problem took the form

   $$y = c_1 \cosh\left(\frac{x-c_2}{c_1}\right) - \lambda,$$

   where the constants λ, c_1 and c_2 are determined by the length L of the chain, and the end conditions, i.e., the heights of the poles $y(x_0) = x_0$ and $y(x_1) = x_1$.

   We determined a method to calculate the constants when the problem is symmetric, though this required numerical solution of a non-linear equation. In this more general case, we can make use of identity such as

   $$\begin{aligned} L\{y\} &= \int_{x_0}^{x_1} \sqrt{1+y'^2}\,dx \\ &= \int_{x_0}^{x_1} \cosh((x-c_2)/c_1)\,dx \\ &= c_1\left[\sinh((x-c_2)/c_1))\right]_{x_0}^{x_1}. \end{aligned}$$

   Given a length $L > D$ where D is the distance between the two pylon ends, write code to numerically determine the constants (c_1, c_2, λ).

   Solution: Given a length $L > D$ where D is the distance between the two pylon ends, there will always be a valid catenary, because we know we could hang a chain between these two points (ignoring the possibility that it would drag on the surface).

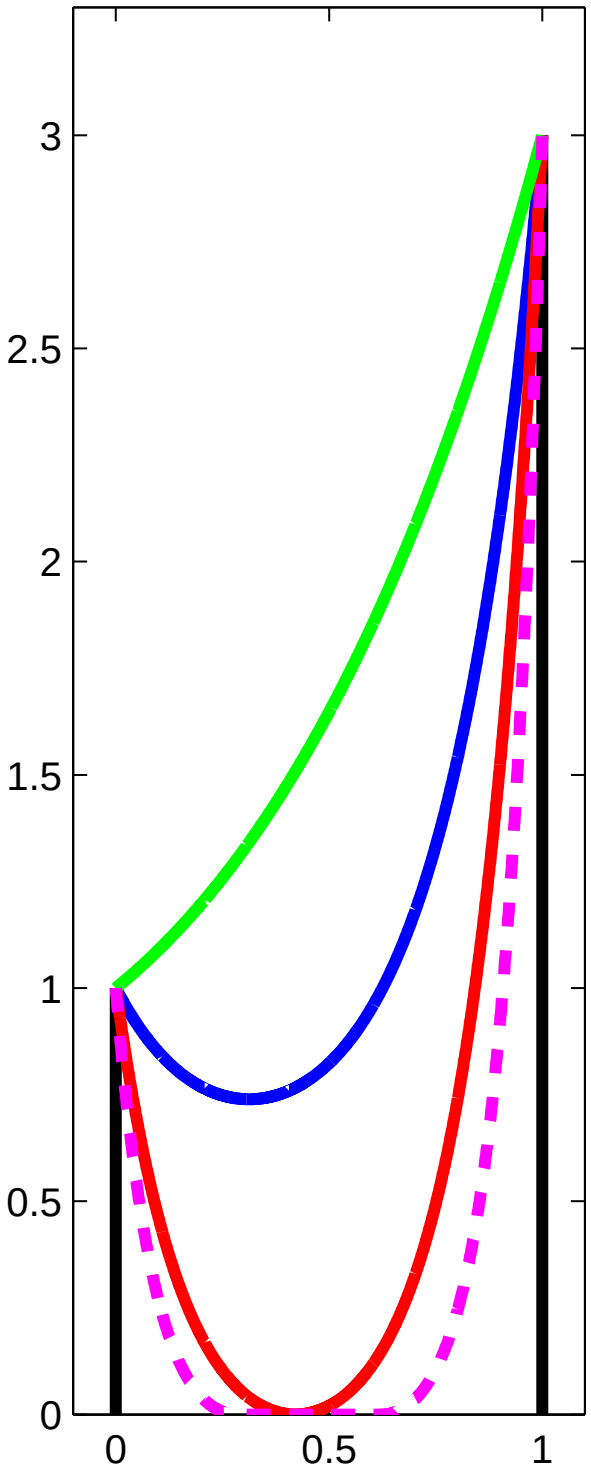
   We solve by constructing a function which is the square of the deviation from the three available constraints: *i.e.*, we define

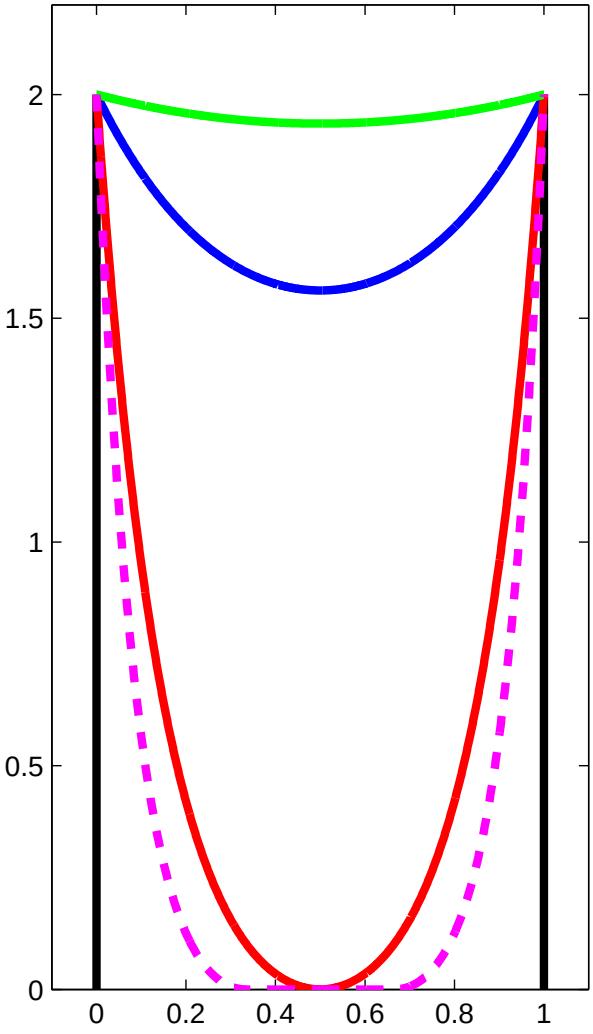
   $$\begin{aligned} g_1(c_1,c_2,\lambda) &= y_0 - c_1\cosh\left(\frac{x_0-c_2}{c_1}\right) - \lambda, \\ g_2(c_1,c_2,\lambda) &= y_1 - c_1\cosh\left(\frac{x_1-c_2}{c_1}\right) - \lambda, \\ g_3(c_1,c_2,\lambda) &= L - c_1\left[\sinh((x-c_2)/c_1))\right]_{x_0}^{x_1}. \end{aligned}$$

   We then use Matlab's optimization toolbox function `fminsearch` to find the minimum of

   $$g(c_1,c_2,\lambda) = g_1^2 + g_2^2 + g_3^2.$$

   Matlab code is provided below, as are some example results.





11

Matlab code: for performing estimating catenary parameters is included below.

```
function [x, y, c_1, c_2, lambda, Lest, Fest, Lest_check, Fest_check, ...
  f_val, exitflag, output] = catenary_solver_gen(n, y_0, y_1, x_0, x_1, L)
%
% file:       catenary_solver_gen.m, (c) 2012 Matthew Roughan
% author:   Matthew Roughan
% email:    matthew.roughan@adelaide.edu.au
%
%
% CATENARY_SOLVER: solves the shape of a hanging chain, which we know will be
%                y = c_1*cosh((x-c_2)/c_1) + lambda
%         with fixed length
%                L = c_1.*( sinh((x_1-c_2)./c_1) - sinh((x_0-c_2)./c_1) );
%         but we need to work out the constants of integration c_1, c_2 and lambda
%
% INPUTS:
%         n         = number of points at which to calculate the curve
%         y_0       = height of the left pylon (must be > 0)
%         y_1       = height of the right pylon (must be > 0)
%         x_0       = left pylon position
%         x_1       = right pylon position
%         L         = length of chain
%
% OUTPUTS:
%         [x, y]    = n (x,y) points along the shape of the catenary
%         c_1,c_2     = constants of integration
%         lambda    = Lagrange multiplier
%         Lest      = estimated length, to be used in debugging
%         Fest      = an estimate of the functional which gives the potential energy of the chain
%         Lest_check = a check based on estimated (x,y) positions (only valid for large n)
%         Fest_check = a check based on estimated (x,y) positions (only valid for large n)
%         [f_val, exitflag, output] = output from the optimization used to find the solution
%

if (y_0 < 0)
    error('y_0 must be >= 0');
end

if (y_1 < 0)
    error('y_1 must be >= 0');
end

if (x_1 <= x_0)
  error('x_1 should be > x_0');
end

if (nargin == 6) % L defined

  [x, y, c_1, c_2, lambda, Lest, Fest, Lest_check, Fest_check, ...
   f_val, exitflag, output] = catenary_new_b(y_0, y_1, x_0, x_1, L);


elseif (nargin == 5) % compute the natural catenary
  lambda = 0;

  % create a function which we will minimize to find the solution
  %    g1 is the left end-point constraint
  %    g2 is the right end-point constraint
  % remembering for this case, there may be one, zero, or two solutions
  g1 = @(a) ( y_0 - a(1)*cosh( (x_0 - a(2))/a(1) )).^2;
  g2 = @(a) ( y_1 - a(1)*cosh( (x_1 - a(2))/a(1) )).^2;
  g = @(a) g1(a) + g2(a);

  a_est = [0.1, (x_1+x_0)/2]; % case with smaller c_1 is usually the min
  options = optimset('fminsearch');
  options = optimset(options, 'MaxFunEvals', 100000);
```

```
  [a, fval1, exitflag1, output1] = fminsearch(g, a_est, options);
  c_11 = a(1);
  c_21 = a(2);
  Fest_1 =  c_11*(x_1 - x_0)/2 + ...
    c_11^2*(sinh(2*(x_1 - c_21)/c_11)-sinh(2*(x_0 - c_21)/c_11))/4;

  a_est = [100, (x_1+x_0)/2]; % case with larger c_1 is important to check for as well
  options = optimset('fminsearch');
  options = optimset(options, 'MaxFunEvals', 100000);
  [a, fval2, exitflag2, output2] = fminsearch(g, a_est, options);
  c_12 = a(1);
  c_22 = a(2);
  lambda = 0;
  Fest_2 =  c_12*(x_1 - x_0)/2 + ...
    c_12^2*(sinh(2*(x_1 - c_22)/c_12)-sinh(2*(x_0 - c_22)/c_12))/4;
  f_val = [fval1, fval2];

  if (fval1 > 1.0e-6 & fval2> 1.0e-6)
    % no solutions
    c_1 = NaN;
    c_2 = NaN;
    exitflag=exitflag1;
    output=output1;
  elseif (Fest_1  < Fest_2)
    c_1 = c_11;
    c_2 = c_21;
    exitflag=exitflag1;
    output=output1;
  else
    c_1 = c_12;
    c_2 = c_22;
    exitflag=exitflag2;
    output=output2;
  end


end

% check the length is correct
Lest = c_1.*( sinh((x_1-c_2)./c_1) - sinh((x_0-c_2)./c_1) );

%
% now calculate points on the curve
%
x = x_0:(x_1 - x_0)/n:x_1;
y = c_1*cosh((x-c_2)/c_1) + lambda;

% second check of the length
Lest_check = sum(sqrt(diff(x).^2 + diff(y).^2));


% calculate the potential a few different ways to compare
Fest_1 =  sum(y(1:end-1).*sqrt(diff(x).^2 + diff(y).^2));
Fest_2 =  c_1*(x_1 - x_0)/2 + ...
  c_1^2*(sinh(2*(x_1 - c_2)/c_1)-sinh(2*(x_0 - c_2)/c_1))/4 - ...
  lambda*Lest;
% Fest_3 =  c_1*(x_1 - x_0)/2 + ...
%    c_1*((y_1+lambda)*sinh((x_1 - c_2)/c_1) - (y_0+lambda)*sinh((x_0 - c_2)/c_1))/2 - ...
%    lambda*Lest;
% Fest_4 =  c_1*(x_1 - x_0)/2 + ...
%    c_1*(y_1*sinh((x_1 - c_2)/c_1) - y_0*sinh((x_0 - c_2)/c_1))/2 - ...
%    lambda*Lest/2;

Fest = Fest_2;
Fest_check = Fest_1;
```

Remarks:

and

$$\begin{aligned}
F\{y\} &= \int_{x_0}^{x_1} (c_1\cosh((x-c_2)/c_1)-\lambda)\sqrt{1+\sinh^2((x-c_2)/c_1)}\,dx \\
&= \int_{x_0}^{x_1} c_1\cosh^2((x-c_2)/c_1)\,dx - \lambda L \\
&= \int_{x_0}^{x_1} \frac{c_1}{2}\left[1+\cosh(2(x-c_2)/c_1)\right]\,dx - \lambda L \\
&= \frac{c_1}{2}(x_1-x_0) + \left[\frac{c_1^2}{4}\sinh(2(x-c_2)/c_1)\right]_{x_0}^{x_1} - \lambda L \\
&= \frac{c_1}{2}(x_1-x_0) + \left[\frac{c_1^2}{2}\sinh((x-c_2)/c_1)\cosh((x-c_2)/c_1)\right]_{x_0}^{x_1} - \lambda L \\
&= \frac{c_1}{2}(x_1-x_0) + \left[\frac{c_1}{2}\sinh((x-c_2)/c_1)(y+\lambda)\right]_{x_0}^{x_1} - \lambda L \\
&= \frac{c_1}{2}(x_1-x_0) + \frac{c_1}{2}\left[y(x)\sinh((x-c_2)/c_1)\right]_{x_0}^{x_1} - \frac{\lambda L}{2}
\end{aligned}$$

which were given in lectures.

$$\sinh(2x) = 2\sinh(x)\cosh(x), \quad \text{and } \cosh(2x) = 2\cosh^2(x)-1,$$

If $y_1 = y_2$ we get the symmetric case, which has $c_2 = (x_1+x_2)/2$, and from above

$$\begin{aligned}
F\{y\} &= \frac{c_1}{2}(x_1-x_0) + \frac{c_1}{2}\left[y(x)\sinh((x-c_2)/c_1)\right]_{x_0}^{x_1} - \frac{\lambda L}{2} \\
&= \frac{c_1}{2}(x_1-x_0) + \frac{y_1-\lambda}{2}L
\end{aligned}$$

When $\lambda = 0$, we get the natural catenary, and so this must give the lowest value for $F\{y\}$ for any L, and hence, $\lambda < 0$ for all other cases.

The first question we might therefore wish to answer is the rough location of c_2.

We can just try each of the cases above, and see which one works ...

$y_{min} = c_1 - \lambda$ occurs when $x = c_2$, and the derivative will be zero – could be in the middle, or the left or right of the interval of interest

Start with y and approximate $\cosh$ using first two terms of Taylor series:

$$\begin{aligned}
\sinh(x) &= x + \frac{x^3}{6} + \frac{x^5}{120} + ... \\
\cosh(x) &= 1 + \frac{x^2}{2} + \frac{x^4}{24} + ...
\end{aligned}$$

so that

$$\begin{aligned}
y+\lambda &= c_1\cosh\left(\frac{x-c_2}{c_1}\right), \\
&= c_1 + \frac{(x-c_2)^2}{2c_1} + \cdots
\end{aligned}$$

Taking the values at x_0 and x_1, and subtracting we get

$$\begin{aligned} y_1 - y_0 &= \frac{(x_1 - c_2)^2}{2c_1} - \frac{(x_0 - c_2)^2}{2c_1} + \cdots \\ 2c_1(y_1 - y_0) &\simeq (x_1 - c_2)^2 - (x_0 - c_2)^2 \\ &\simeq (x_1^2 - x_0^2) - 2(x_1 - x_0)c_2 \\ c_1 &\simeq \frac{(x_1^2 - x_0^2) - 2(x_1 - x_0)2c_2}{2(y_1 - y_0)} \\ &\simeq \alpha + \beta c_2 \end{aligned}$$

so there is an approximately linear relationship between c_1 and c_2. Substitute this into the equation for L, and we get a single non-linear equation which we can solve to get an approximate value for c_2, and hence c_1 (by the above) and hence λ by the edge equations.

This gives us an initial guess as to the values of c_1, c_2 and λ which we can refine to get an optimal value.

4. **Direct Approximate Solution:** One of the classic problems in the CoV is the catenary problem, which we solved analytically early in the course. The problem reduces to minimising potential energy

$$W_p\{y\} = mg\int_0^L y(s)ds = mg\int_{x_0}^{x_1} y\sqrt{1+y'^2}dx,$$

subject to the end-point conditions (the fixed heights of the pylons), and the length constraint:

$$G\{y\} = \int_0^L y(s)ds = \int_{x_0}^{x_1} \sqrt{1+y'^2}\,dx = L.$$

Ignoring the constant mg, we solved this using Lagrange multipliers, instead seeking extremals of

$$H\{y\} = \int_{x_0}^{x_1} (y+\lambda)\sqrt{1+y'^2}\,dx.$$

Now we consider the numerical approximation and solution of this problem. We could apply Euler's finite differences, or Ritz, but instead consider a chain of length L made of n pieces of length ℓ (where $n\ell = L$). Assume each chain link i starts at the point (x_i, y_i) and ends at (x_{i+1}, y_{i+1}), for $i = 0, 1, \ldots, n-1$, where $y_i \geq 0$ and $x_i \in [x_a, x_b]$. The length constraint on each link means that

$$\ell^2 = (x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2, \;\; \forall i = 0, 1, 2, \ldots, n-1.$$

The start and end points of the chain are fixed at $(x_0, y_0) = (x_a, y_a)$ and $(x_n, y_n) = (x_b, y_b)$ respectively, and we assume that $D = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2} < L$ to allow for a solution.

The objective function that we aim to optimize is to minimize the potential energy of the chain, where the potential of each link will be given by the height of its mid-point, times a constant (mg), i.e.,

$$W = mg\sum_{i=0}^{n-1} \frac{y_i + y_{i+1}}{2}.$$

Note that the constant factors are irrelevant, and the sum concertinas so that we can rewrite it

$$W = \sum_{i=1}^{n-1} y_i,$$

where we ommit the values of the end points because these are fixed.

Use a numerical optimization algorithm to find the shape of the curve that solves this problem, and show numerically that it approaches the analytic solution as $n \to \infty$.

Solution:

We can solve the problem by including a Lagrange multiplier for each of the constraints, i.e.,

$$f(\mathbf{x}, \mathbf{y}, \lambda) = \sum_{i=1}^{n-1} y_i + \sum_{i=0}^{n-1} \lambda_i \left[\ell^2 - (x_{i+1} - x_i)^2 - (y_{i+1} - y_i)^2\right].$$

for some Lagrange mutipliers λ_i.

To apply Steepest Descent, starting at some point $\mathbf{v}_0$, we need to calculate the gradient. The gradient of this function is derived as follows:

$$\begin{aligned}
\frac{\partial f}{\partial x_i} &= 2\lambda_i(x_{i+1} - x_i) - 2\lambda_{i-1}(x_i - x_{i-1}),\\
\frac{\partial f}{\partial y_i} &= 1 + 2\lambda_i(y_{i+1} - y_i) - 2\lambda_{i-1}(y_i - y_{i-1}),\\
\frac{\partial f}{\partial \lambda_i} &= \left[\ell^2 - (x_{i+1} - x_i)^2 - (y_{i+1} - y_i)^2\right],
\end{aligned}$$

except for $i = 0$ and $i = n$, where the end points are fixed so $\partial f/\partial x_i = \partial f/\partial y_i = 0$. The Hessian matrix is hence almost tridiagonal, though we don't need this for the Steepest Descent Algorithm.

Take the case with n even, and then the starting point can have the chain links in a zig-zag. Construct it as follows. Draw the line interpolating between (x_a, y_a) and (x_b, y_b), i.e.,

$$y = \frac{y_b - y_a}{x_b - x_a}(x - x_a) + y_a,$$

which has length

$$D = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2}.$$

Now distribute a set of points x_i' evenly along a line of length

$$x_i' = iD/n.$$

and alternate the y_i' positions to satisfy the length constraint on each segment:

$$y_i' = \begin{cases} y_a, & \text{if } i = 2k, \\ y_a + d & \text{if } i = 2k + 1, \end{cases}$$

where d is chosen to enforce the length constraint on each segment, i.e.,

$$d = \sqrt{\ell^2 - (D/n)^2}.$$

The points $(x_i'.y_i')$ are then rotated by θ about (x_a, y_a) to meet the other end-point constraint (x_b, y_b), where the rotation takes the form

$$\begin{pmatrix} x_i \\ y_i \end{pmatrix} = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} x_i' - x_a \\ y_i' - y_a \end{pmatrix} + \begin{pmatrix} x_a \\ y_a \end{pmatrix}$$

where

$$\begin{aligned} \cos(\theta) &= \frac{x_b - x_a}{D} \\ \sin(\theta) &= \frac{y_b - y_a}{D} \end{aligned}$$

Matlab code: for performing the steepest descent is provided below:

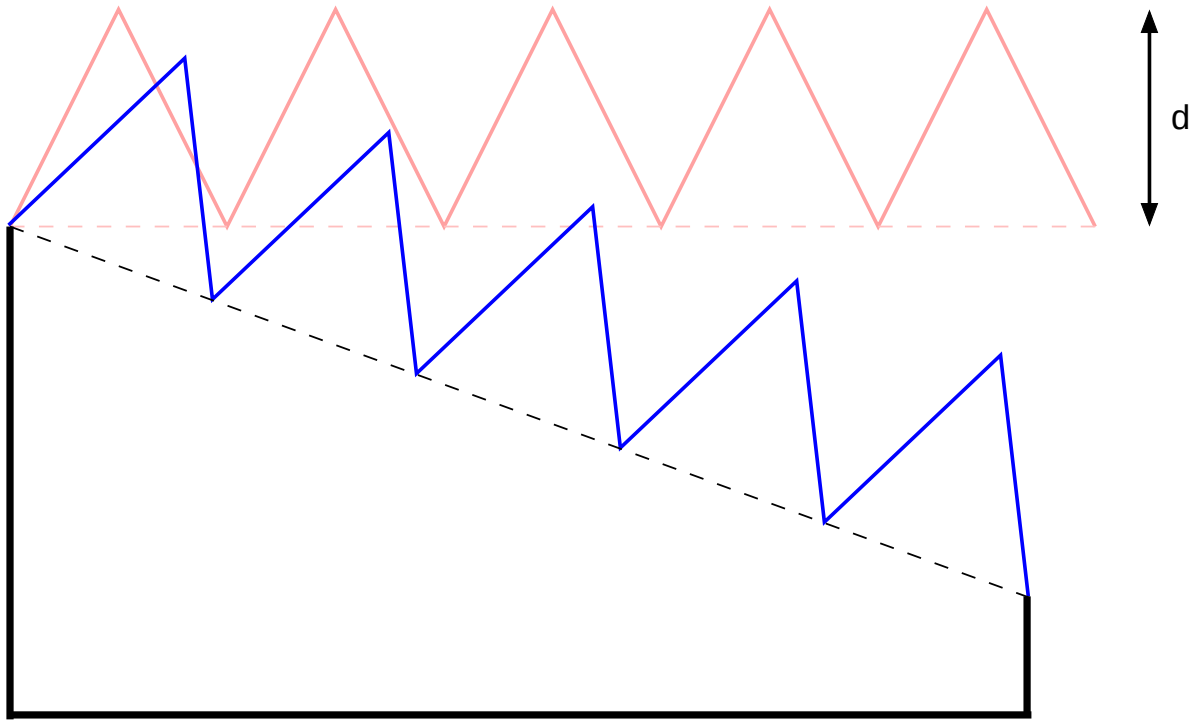


Figure 3: .

5. **Rayleigh-Ritz:**

Consider the following problem — find the extremal curves of the following functional:

$$F\{y\} = \int_0^1 {y'}^2 - 2g(x)y\,dx,$$

for twice differentiable function $g(\cdot)$, and $y(0) = y(1) = 0$.

(a) Solve this problem using the Euler-Lagrange equations in general, and in detail for the special cases

i. $g(x) = \sin(2\pi x)$.

ii. $g(x) = x$.

(b) Now use the Rayleigh-Ritz method to solve the same problems using the approximation

$$\tilde{y}_N = b_0 + \sum_{n=1}^{N} a_n \sin(2\pi nx) + b_n \cos(2\pi nx).$$

Solution:

(a) The Euler-Lagrange equations are

$$\frac{d}{dx}\frac{\partial f}{\partial y'} - \frac{\partial f}{\partial y} = \frac{d}{dx}2y' + 2g(x) = 2y'' + 2g(x) = 0.$$

The solution is just to integrate twice, resulting in

$$\begin{aligned} y' &= -\int g(u)\,du + c_1 \\ y &= -\iint g(u)\,du\,dv + c_1 x + c_2, \end{aligned}$$

where c_1 and c_2 are chosen to satisfy $y(0) = y(1) = 0$.

i. So for the particular case $g(x) = \sin(2\pi x)$, where the double integral will be $-\sin(2\pi x)/4\pi^2$, which is zero at the end-points, the extremal takes the form

$$y(x) = \frac{1}{4\pi^2}\sin(2\pi x).$$

ii. For the case $g(x) = x$, the double integral will be $x^3/6$, so the extremals are

$$y(x) = -\frac{x^3}{6} + c_1 x + c_2.$$

Using the end-point conditions $y(0) = y(1) = 0$ we get

$$y(x) = -\frac{x^3}{6} + \frac{x}{6} = \frac{x(1-x^2)}{6}.$$

(b) Given the approximation, we get

$$\begin{aligned} \tilde{y}_N &= b_0 + \sum_{n=1}^{N} a_n \sin(2\pi nx) + b_n \cos(2\pi nx), \\ \tilde{y}'_N &= 2\pi \sum_{n=1}^{N} n a_n \cos(2\pi nx) - n b_n \sin(2\pi nx), \\ {\tilde{y}'_N}^2 &= 4\pi^2 \sum_{n=1}^{N}\sum_{m=1}^{N} nm\big[a_n a_m \cos(2\pi nx)\cos(2\pi mx) - a_n b_m \cos(2\pi nx)\sin(2\pi mx) + b_n b_m \sin(2\pi nx)\sin(2\pi mx)\big]. \end{aligned}$$

When we integrate, we can use

$$\begin{aligned}\int_0^1 \cos(2\pi nx)\,\cos(2\pi mx)\,dx &= \frac{1}{2}\delta_{mn}\\ \int_0^1 \sin(2\pi nx)\,\sin(2\pi mx)\,dx &= \frac{1}{2}\delta_{mn}\\ \int_0^1 \sin(2\pi nx)\,\cos(2\pi mx)\,dx &= 0\end{aligned}$$

So the "cross-terms" in the above integrals vanish, and we are left with

$$\begin{aligned}F\{y\} &= \int_0^1 {y'}^2 + 2g(x)y\,dx\\ F(\mathbf{a},\mathbf{b}) &= 2\pi^2\sum_{n=1}^N n^2(a_n^2+b_n^2) - 2\sum_{n=1}^N a_n\int_0^1 g(x)\sin(2\pi nx)\,dx - 2\sum_{n=0}^N b_n\int_0^1 g(x)\cos(2\pi nx)\,dx\\ &= 2\pi^2\sum_{n=1}^N n^2(a_n^2+b_n^2) - \sum_{n=1}^N a_nA_n - \sum_{n=0}^N b_nB_n,\end{aligned}$$

where $a_n = 2\int g(x)\sin(2\pi nx)\,dx$ and $B_n = 2\int g(x)\cos(2\pi nx)\,dx$ are the Fourier series coeeficients for $g(x)$ (assuming we repeat the function on $[0,1]$ periodically). We can now take the derivatives, and for $n = 1, 2, \ldots, N$ we get

$$\begin{aligned}\frac{d}{da_n}F(\mathbf{a},\mathbf{b}) &= 4\pi^2n^2a_n - A_n\\ &= 0,\\ \frac{d}{db_n}F(\mathbf{a},\mathbf{b}) &= 4\pi^2n^2b_n - B_n\\ &= 0,\end{aligned}$$

or

$$\begin{aligned}a_n &= \frac{A_n}{4\pi^2n^2},\\ b_n &= \frac{B_n}{4\pi^2n^2}.\end{aligned}$$

Note that the b_n terms don't depend on n, so we simply expand until we have enough of these terms to converge. We don't get a condition for b_0, but we have the end-points conditions to consider, and given $\cos(0) = \cos(2\pi) = 1$, and we require $y(0) = y(1) = 0$, we choose

$$b_0 = -\sum_{n=1}^N b_n.$$

In the specific cases of interest:

i. When $g(x) = \sin(2\pi x)$, the Fourier coefficients are

$$A_n = \begin{cases} 1, & \text{for } n = 1,\\ 0, & \text{otherwise,}\end{cases} \qquad \text{and} \quad B_n = 0,$$

so for all n we get the exact solution:

$$\tilde{y}_n(x) = \frac{1}{4\pi^2}\sin(2\pi x).$$

ii. For the case $g(x) = x$, we consider the Fourier coefficients for the "saw-tooth" function, i.e.,

$$\begin{aligned} A_n &= 2\int_0^1 x\sin(2\pi nx)\,dx = -\frac{1}{\pi n}, \\ B_n &= 2\int_0^1 x\cos(2\pi nx)\,dx = -\frac{1}{2\pi n^2} actually 0, \end{aligned}$$

$$\tilde{y}_n(x) = b_0 + \sum_{n=1}^{n} \frac{1}{4\pi^2 n^3}\sin(2\pi nx). + \sum_{n=1}^{n} \frac{1}{8\pi^2 n^3}\cos(2\pi nx).$$

The following figure shows this approximation for $N = 1, 2, 3, \ldots$ versus the actual extremal curve:
If we just took a Fourier expansion of $x(1-x^2)$ we get

$$\begin{aligned} \int_0^1 x(1-x^2)\,dx &= 1/4 \\ a_n = 2\int_0^1 x(1-x^2)\sin(2\pi nx)\,dx &= -\frac{3}{2\pi^3 n^3} \\ b_n = 2\int_0^1 x(1-x^2)\cos(2\pi nx)\,dx &= -\frac{3}{2\pi^2 n^2} \end{aligned}$$

6. **Another Problem (not finished yet):**
Consider the following problem — find the extremal curves of the following functional:

$$F\{y\} = \int_0^1 y'^2 - y^2\, dx,$$

for $y(0) = 1$ and $y(1) = e$.

(a) Solve this problem using the Euler-Lagrange equations.

(b) Now use the Rayleigh-Ritz method to solve the problem using the trial functions

$$\tilde{y} = \sum_{i=0}^{\infty} a_i x^i.$$

Derive a set of conditions on the a_i and use your solution to the previous part to solve these conditions.

Solution:

(a) The LHS of the Euler-Lagrange equation is

$$\frac{d}{dx}\frac{\partial f}{\partial y'} - f = \frac{d}{dx}(2y') - 2y,$$

So the DE is

$$y'' - y = 0.$$

This is a linear, homogeneous DE, with characteristics equation $m^2 - 1 = 0$, so $m = \pm 1$, and therefore the solution is in the form

$$y(x) = Ae^t + Be^{-t}.$$

From the end-point conditions $y(0) = 1$ and $y(1) =$ we get the constraints

$$\begin{aligned} A + B &= 1, \\ Ae - B/e &= e. \end{aligned}$$

which has solution $A = 1$, $B = 0$, and therefore

$$y(x) = e^t.$$

(b) Use the an approximation function

$$y_n(x) = [1 + (e-1)x] + \sum_{i=1}^{n} a_i x^i (1-x).$$

$$y_n(x) = [1 + (e-1)x] + \sum_{i=1}^{n} a_i \sin(2\pi i x).$$

i. Given $\tilde{y} = \sum_{i=0}^{\infty} a_i x^i$ we get

$$\begin{aligned} y^2 &= \sum_{i,j=0}^{\infty} a_i a_j x^{i+j}, \\ y'^2 &= \sum_{i,j=1}^{\infty} ij a_i a_j x^{i+j-2} \\ &= \sum_{i,j=0}^{\infty} (i+1)(j+1) a_{i+1} a_{j+1} x^{i+j}, \end{aligned}$$

so the integral can be written

$$\begin{aligned} F\{y\} &= \int_0^1 \tilde{y}'^2 - \tilde{y}^2\,dx, \\ &= \int_0^1 \sum_{i,j=0}^{\infty} (i+1)(j+1)a_{i+1}a_{j+1}x^{i+j} - \sum_{i,j=0}^{\infty} a_i a_j x^{i+j}\,dx, \\ &= \sum_{i,j=0}^{\infty} [(i+1)(j+1)a_{i+1}a_{j+1} - a_i a_j] \int_0^1 x^{i+j}\,dx, \\ &= \sum_{i,j=0}^{\infty} \frac{(i+1)(j+1)a_{i+1}a_{j+1} - a_i a_j}{i+j+1} \\ &= \sum_{i,j:i\neq j=0}^{\infty} \frac{(i+1)(j+1)a_{i+1}a_{j+1} - a_i a_j}{i+j+1} + \sum_{i=0}^{\infty} \frac{(i+1)^2 a_{i+1}^2 - a_i^2}{2i+1}. \end{aligned}$$

Now to find the minimum, we set the partial derivatives to zero and we therefore get (for $i \geq 1$)

$$\begin{aligned} \frac{\partial F(\mathbf{a})}{\partial a_i} &= \sum_{j:j\neq i}^{\infty} \frac{i(j+1)a_{j+1}}{i+j} + \sum_{j:j\neq i}^{\infty} \frac{-a_j}{i+j+1} + \frac{i^2 a_i}{i} + \frac{-a_i}{i+1} && (3) \\ &= \sum_{j=0}^{\infty} \frac{i(j+1)a_{j+1}}{i+j} + \sum_{j=0}^{\infty} \frac{-a_j}{i+j+1} + \frac{i^2 a_i}{2(i-1)} - \frac{a_i}{2(i+1)} && (4) \\ &= \sum_{j=0}^{\infty} \frac{i(j+1)a_{j+1}}{i+j} + \sum_{j=0}^{\infty} \frac{-a_{j+1}}{i+j+2} - \frac{a_0}{i+2} + \frac{i^2 a_i}{2(i-1)} + \frac{-a_i}{2i} = 0. && (5) \end{aligned}$$

We also know from the end-point conditions that

$$a_0 = 1 \quad \text{and} \quad \sum_i a_i = e.$$

This represents a set of relationships between the a_i, which might be quite hard to solve, however, we already know that there is a solution in this form:

$$y = e^t = \sum_{i=0}^{\infty} \frac{x^i}{i!},$$

so if we take $a_i = 1/i!$ we should get a solution.
First note that with the above choice $a_0 = 1$, and

$$\sum_{i=0}^{\infty} a_i = \sum_{i=0}^{\infty} 1/i! = e.$$

Now substitute these a_i into the relation (5) and we get

ii. Do you get the same solution?

iii. Why did we choose the particular values of a_0 and a_1 as the first terms of the sequence?

7. **More on Rayleigh-Ritz:**

In R-R we use an approximation, typically of the form

$$y_n(x) = \phi_0(x) + \sum_{i=1}^{n} c_i \phi_i(x),$$

to approximate the function

$$F\{y\} = \int_{x_0}^{x_1} f(x, y, y')\, dx,$$

by

$$F(c_1, c_2, \ldots, c_n) = \int_{x_0}^{x_1} f(x, y_n, y_n')\, dx.$$

where ϕ_0 satisfies the boundary conditions on y and $\phi_i(x_i) = 0$.

Show ...

Solution:

The Leibniz integral rule says that for f and $\partial f/\partial x$ continuous over the region $[x_0, x_1] \times [y_0, y_1]$ we can differentiate an integral in the following form:

$$\frac{d}{dx}\int_{y_0}^{y_1} f(x, y)\, dy = \int_{y_0}^{y_1} \frac{\partial}{\partial x} f(x, y)\, dy.$$

$$\begin{aligned}
F(c_1, c_2, \ldots, c_n) &= \int_{x_0}^{x_1} f(x, y_n, y_n')\, dx \\
&= \int_{x_0}^{x_1} f\left(x, \phi_0 + \sum_{i=1}^{n} c_i\phi_i, \phi_0' + \sum_{i=1}^{n} c_i\phi_i'\right) dx \\
\frac{\partial}{\partial c_i} F(c_1, c_2, \ldots, c_n) &= \int_{x_0}^{x_1} \frac{\partial}{\partial c_i} f\left(x, \phi_0 + \sum_{i=1}^{n} c_i\phi_i, \phi_0' + \sum_{i=1}^{n} c_i\phi_i'\right) dx \\
&= \int_{x_0}^{x_1} \frac{\partial f}{\partial y}\phi_i + \frac{\partial f}{\partial y'}\phi_i'\, dx \\
&= \int_{x_0}^{x_1} \left[\frac{\partial f}{\partial y} - \frac{d}{dx}\frac{\partial f}{\partial y'}\right] \phi_i\, dx, \\
&= 0, \quad \forall i = 1, 2, \ldots, n.
\end{aligned}$$

using the chain rule and integration by parts (noting that $\phi_i(x_i) = 0$ for $i = 1, 2, \ldots, n$). Obviously the above would be satisfied if the Euler-Lagrange equations were satisfied. However, the underlying assumption of numerical methods is that the Euler-Lagrange equations are difficult.

When we substitute y_n into the above formula we get

$$\begin{aligned}
\frac{\partial}{\partial c_i} F(c_1, c_2, \ldots, c_n) &= \int_{x_0}^{x_1} \left[\frac{\partial f}{\partial y} - \frac{d}{dx}\frac{\partial f}{\partial y'}\right] \phi_i\, dx, \\
&= \int_{x_0}^{x_1} g(x, \phi_1, \ldots, \phi_n, \phi_1', \ldots, \phi_n')\phi_i(x)\, dx, \\
&= 0, \quad \forall i = 1, 2, \ldots, n.
\end{aligned}$$

and if this can be calculated, we can then determine values of c_i to approximate the extremal.

However, Galerkin noted that an alternative

8. **Rayleigh-Ritz:** Find approximate solutions to the following DE

$$\frac{d^2u}{dt^2} = 36t^2 + 12t - 4,$$

subject to the boundary conditions $u(1) = u(-1) = 0$, using Rayleigh-Ritz by constructing an appropriate functional, and approximating u using

$$u_n = \sum_{i=1}^{n} c_i t^{i-1}(t^2 - 1),$$

for $n = 1, 2$ and 3, and comment on the accuracy of these solutions.

Solution: The true solution to this is

$$u(t) = 3t^4 + 2t^3 - 2t^2 - 2t - 1.$$

We can construct a functional for this problem

$$F\{u\} = \int_{-1}^{1} \frac{1}{2}\left(\frac{du}{dt}\right)^2 + (36t^2 + 12t - 4)u\,dx.$$

The Euler-Lagrange DE is the equation to be solved.

Given the approximation:

$$\begin{aligned} u_n &= \sum_{i=1}^{n} c_i t^{i-1}(t^2 - 1), \\ u_n' &= \sum_{i=2}^{n} (i-1)c_i t^{i-2}(t^2 - 1) + \sum_{i=1}^{n} 2c_i t^i. \end{aligned}$$

Use Matlab's symbolic manipulation toolkit, we construct the functional for these cases, and then differentiate, to obtain a set of equations for the c_i.

CODE

The results are:

- $n = 1$:
- $n = 2$:
- $n = 3$: