A brute-force solution to problem "`beam`"

As noted in the "Five example optimization problems" handout, problem `beam` is intrinsically infinite-dimensional.

The set of possible inputs to the functional $I[h]$ is

$$\mathcal{X} = \left\{ f \,:\, f'' \text{ is square-integrable on } [0,\pi] \text{ and also } f(0) = f(\pi) = 0 \right\}.$$

This is a real vector space of functions.[1] Why is $\mathcal{X}$ infinite-dimensional, you say? The answer is that you cannot specify each element using a fixed, finite number of coefficients.[2]

For example, and as a hint about the solution method adopted below, the following infinite list of functions live in $\mathcal{X}$:

$$S = \{\sin x, \sin 2x, \sin 3x, \sin 4x, \dots\} \subset \mathcal{X}.$$

This set is linearly-independent; that is, on cannot write one element of S, say $\sin kx$, exactly as a finite linear combination of other elements of the set S. In the appropriate senses, S is a basis for $\mathcal{X}$ and it is an orthogonal set. For example—this is an instance of Fourier series—the function $f(x) = x(\pi - x)$ is in $\mathcal{X}$ and, on the other hand, there are coefficients a_k so that[3]

$$f(x) = x(\pi - x) = \sum_{k=1}^{\infty} a_k \sin kx \in \mathcal{X}.$$

But one cannot (exactly) write this $f(x)$ without an infinite sum.

Despite being infinite-dimensional, this kind of beam problem is completely standard in the engineering and physics literature; it is a problem in the *calculus of variations*. Because of the infinite-dimensionality, only an approximation of the solution can be computed in a finite amount of solution time.[4]

The problem is constrained, however, so the solution cannot be just anywhere in the infinite-dimensional vector space $\mathcal{X}$. The solution lives in an infinite-dimensional convex[5] subset of $\mathcal{X}$:

$$\mathcal{K} = \{f \in \mathcal{X} \,:\, 0.9 \le f(1) \le 1.1 \text{ and } 1.2 \le f(2) \le 1.4 \text{ and } 0.4 \le f(3) \le 0.6\}.$$

This is the *feasible set*. It is like a polyhedron; it is a *polytope* in $\mathcal{X}$.

---

[1] So that, for example, given any pair of functions $f_1, f_2 \in \mathcal{X}$, and any real numbers a_1, a_2, the linear combination $a_1 f_1 + a_2 f_2$ is also in $\mathcal{X}$.

[2] If it were possible, the number of such coefficients would be the dimension of $\mathcal{X}$.

[3] A few points extra credit will be given to anyone who computes the coefficients a_k exactly, and then plots a finite Fourier sum $f_N(x) = \sum_{k=1}^{N} a_k \sin kx$ showing that the computed coefficients are likely to be correct.

[4] Actually, this approximate-only status already holds for problem `calcone`, which is in one-dimension. In theory the exact solutions of `fit`, `salmon`, and `tsp` are all possible in finite time. Do you see why for `fit`?

[5] A few points extra credit for proving it is convex.

For now we just want a method that finds an acceptable approximate solution, even if by brute-force. So we restrict our functions to a five-dimensional (5D)[6] space of truncated Fourier sine series:

$$\mathcal{X}_5 = \mathbb{R}^5 = \{[c_1\, c_2\, c_3\, c_4\, c_5]\},$$

where $c \in \mathcal{X}_5$ represents the function $h(x) = \sum_{k=1}^{5} c_k \sin kx \in \mathcal{X}$. Using trigonometry we can calculate $I[h]$ for a function $h(x)$ from $\mathcal{X}_5$ *exactly*:

$$\begin{aligned} I[h] &= \frac{1}{2}\int_0^\pi |h''(x)|^2\,dx = \frac{1}{2}\int_0^\pi \left(-\sum_{k=1}^{5} k^2 c_k \sin kx\right)^2 dx = \frac{1}{2}\sum_{j=1}^{5}\sum_{k=1}^{5} c_j c_k j^2 k^2 \int_0^\pi \sin jx \sin kx\,dx \\ &= \frac{1}{4}\sum_{j=1}^{5}\sum_{k=1}^{5} c_j c_k j^2 k^2 \int_0^\pi \cos((j-k)x) - \cos((j+k)x)\,dx = \frac{1}{4}\sum_{j=1}^{5}\sum_{k=1}^{5} c_j c_k j^2 k^2\,(\pi\delta_{jk} - 0) = \frac{\pi}{4}\sum_{k=1}^{5} k^4 c_k^2. \end{aligned}$$

(This calculation, which uses the identity $\sin\alpha\sin\beta = \frac{1}{2}\cos(\alpha-\beta) - \frac{1}{2}\cos(\alpha+\beta)$, will not surprise those who have used Fourier series before.) The constraint "$0.9 \le h(1) \le 1.1$" can be enforced in $\mathcal{X}_5$ by using the formula $h(x) = \sum_{k=1}^{5} c_k \sin kx$ and evaluating at $x = 1$, and similarly for $x = 2, 3$.

The approximating 5D optimization problem is now in a form close to (1.1) from the textbook:

$$\min_{c\in\mathbb{R}^5} I_5(c) \qquad \text{subject to} \qquad \begin{array}{c} 0.9 \le \sum_{k=1}^{5} c_k \sin k \le 1.1 \\ 1.2 \le \sum_{k=1}^{5} c_k \sin 2k \le 1.4 \\ 0.4 \le \sum_{k=1}^{5} c_k \sin 3k \le 0.6 \end{array}$$

where

$$I_5(c) = \frac{\pi}{4}\sum_{k=1}^{5} k^4 c_k^2.$$

This is called a *quadratic programming problem*; see Chapter 16 of the textbook. Let $\mathcal{K}_5 \subset \mathbb{R}^5$ be the (convex) set where the constraints are satisfied; this is the *feasible* set for the approximate optimization problem. Our problem is now a constrained, 5D version of the unconstrained 3D problem `fit`.

We do not know, however, which of the inequality constraints is active (i.e. the inequality is equality) when c is the solution of the problem. Taking partial derivatives of I_5 and setting them to zero just yields $c = 0$, which does not satisfy the constraints, so $c = 0$ is not the solution. We do not know the shape, in five dimensions, of the subset of $\mathbb{R}^5$ which satisfies the constraints; it is a 5D polytope. Thus we will solve the five-dimensional problem by brute force, searching on a grid of points inside a box in $\mathbb{R}^5$, in the hope that we cover enough of the constrained set to get near the minimum. Only trial-and-error can make this possible.

The proposed box, based on trial-and-error, is

$$\mathcal{B}_5 = \{0 \le c_1 \le 2,\ -1 \le c_2 \le 1,\ -0.5 \le c_3 \le 0.5,\ -0.2 \le c_4 \le 0.2,\ -0.2 \le c_5 \le 0.2\}.$$

---

[6]It can be N dimensional for any N ... but the number of (search) grid points goes up exponentially with N! The value $N = 5$ represents a trial-and-error-determined compromise between answer quality and execution time. Strategies in Chapter 16 will overcome this difficulty.

The code below puts a grid with a given spacing on this box and then evaluates $I_5(c)$ at every point on the grid. I have also posted a code `plotbeam.m` which plots the tent pole corresponding to a given $c \in \mathbb{R}^5$. Running the code with a grid of sufficient coarseness to give a reasonable execution time, namely a few minutes, looks like

```
>> [z h] = beam(0.05)
...
z =  5.8316
h =
  1.40000  -0.30000  0.15000  -0.05000  0.05000
>> plotbeam(h)
```

Thus $c_1 = 1.4$, $c_2 = -0.3$, $c_3 = 0.15$, $c_4 = -0.05$, and $c_5 = 0.05$ from this brute-force search, giving minimum value $I_5(c) = 5.8316$. (During the run the code shows characters `./o/*` for progress made so far, including the best solution so far; this is not shown.) The last "`plotbeam`" command plots the figure at the end, which I believe is pretty close to the solution!

beam.m

```
function [z h] = beam(cspace)
% BEAM Solve tent pole optimization problem by approximation and brute force.
% The height of the pole is represented by a list of N=5 coefficients in a
% Fourier sine series,
%   h(x) = c1 sin(x) + c2 sin(2 x) + c3 sin(3 x) + c4 sin(4 x) + c5 sin(5 x)
% Grid of coefficients c_j, with spacing cspace, in five dimensions, is
% searched.  Does integral exactly.
%
% Example run:
%   >> [z h] = beam(0.05)
%   >> plotbeam(h)
% WARNING: Several minutes run time!  This case checks 2.9 million
% = 41*41*21*9*9 points.  It runs at least 5 times faster if 0.05 --> 0.1.

N = 5;
bounds = [0.9 1.1;
          1.2 1.4;
          0.4 0.6];

z = 1.0e100;
h = zeros(1,N);
for c1 = 0.0:cspace:2.0
    for c2 = -1.0:cspace:1.0
        fprintf('.')
        for c3 = -0.5:cspace:0.5
            for c4 = -0.2:cspace:0.2
                for c5 = -0.2:cspace:0.2
                    htest = [c1 c2 c3 c4 c5];
                    h1 = evalh(htest,1);
                    p1 = (h1 >= bounds(1,1)) & (h1 <= bounds(1,2));
                    if p1
                        h2 = evalh(htest,2);
                        p2 = (h2 >= bounds(2,1)) & (h2 <= bounds(2,2));
                        if p2
```

```
                              h3 = evalh(htest,3);
                              p3 = (h3 >= bounds(3,1)) & (h3 <= bounds(3,2));
                              if p3
                                  fval = f(htest);
                                  if fval < z
                                      z = fval;
                                      h = htest;
                                      fprintf('*')
                                  else
                                      fprintf('o')
                                  end
                              end
                          end
                      end
                  end
              end
          end
      end
  end
  fprintf('\n')

      function z = evalh(h,x)
          k = 1:N;
          z = h * sin(k * x)';
      end

      function z = f(h)
          k = 1:N;
          z = (pi/4.0) * k.^4 * (h.^2)';
      end
  end
```