ORF 522: Lecture 17

Nonlinear Optimization: Models

Robert J. Vanderbei

November 19, 2013

Slides last edited at 4:20pm on Tuesday 3rd December, 2013

Operations Research and Financial Engineering, Princeton University
http://www.princeton.edu/∼rvdb

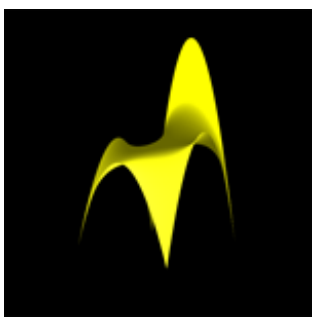
Examples: Convex Optimization Models

Minimal Surfaces

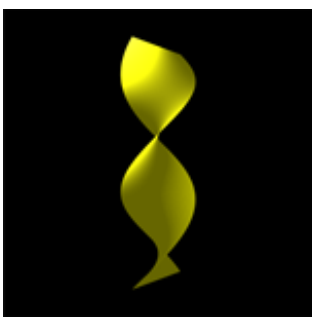
- Given: a domain D in R^2 and an embedding $\mathbf{p} = (x, y, z)$ of its boundary ∂D in R^3;
- Find: an embedding of the entire domain into R^3 that is consistent with the boundary embedding and has minimal surface area:

$$\begin{aligned} \text{minimize} \quad & \iint_D \left\| \frac{\partial \mathbf{p}}{\partial s} \times \frac{\partial \mathbf{p}}{\partial t} \right\| ds dt \\ \text{subject to} \quad & \mathbf{p}(s,t) \text{ fixed for } (s,t) \in \partial D \\ & x(s,t) \text{ fixed for } (s,t) \in D \\ & y(s,t) \text{ fixed for } (s,t) \in D \end{aligned}$$

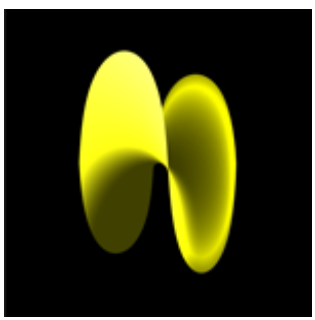
The specific problems coded below take D to be either a square or an annulus.



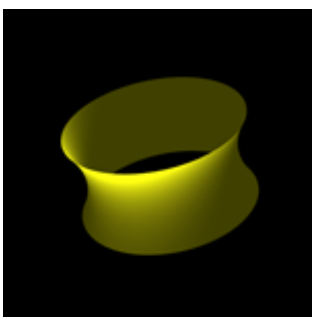
chair



helicoid



scherk



catenoid

twist

Specific Example: Chair

minsurf.mod with D discretized into a 92×92 grid gives the following results:

constraints	0
variables	8464
time (secs)	1.597 sec

AMPL Model

```
param n0 := 92;
param n  := n0+1;

param x {j in 0..n} := 2*j/n;
param y {j in 0..n} := 2*j/n;

param dx := 2/n;
param dy := 2/n;

param gamma0 {j in 0..n} := 1.5*x[j]*(2-x[j]);
param gamma1 {j in 0..n} := 2*y[j]*(2-y[j]);
param gamma2 {j in 0..n} := 4*x[j]*(2-x[j]);
param gamma3 {j in 0..n} := 2*y[j]*(2-y[j]);

var z {0..n, 0..n};

minimize area:
   (dx*dy/2)*
   sum {i in 0..n0, j in 0..n0}
      (
      sqrt(1 + ((z[i+1,j] - z[i,j])/dx)^2 + ((z[i,j+1] - z[i,j])/dy)^2)
      +
      sqrt(1 + ((z[i+1,j+1] - z[i,j+1])/dx)^2 + ((z[i+1,j+1] - z[i+1,j])/dy)^2)
      );

subject to bndcnd0 {i in 0..n}: z[i,0] = gamma0[i];
subject to bndcnd1 {j in 0..n}: z[n,j] = gamma1[j];
subject to bndcnd2 {i in 0..n}: z[i,n] = gamma2[i];
subject to bndcnd3 {j in 0..n}: z[0,j] = gamma3[j];

printf {i in 0..n, j in 0..n}:
  "\t \t x[%2d][%2d] = %6.3f; y[%2d][%2d] = %6.3f; z[%2d][%2d] = %6.3f; \n",
  i, j, x[i] - 1, i, j, y[j] - 1, i, j, z[i, j]-2 > "before.dat";

solve;
```

Specific Example: Scherk

Scherk.mod with D discretized into a 92×92 grid gives the following results:

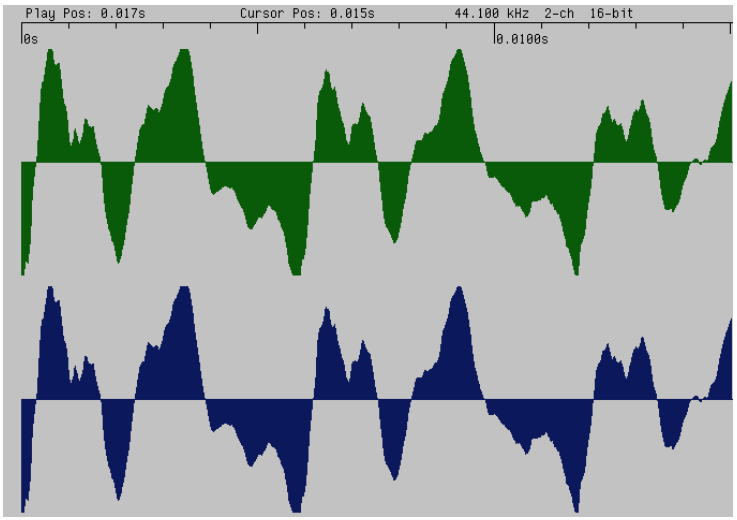
constraints	0
variables	8464
time (secs)	0.47 sec

AMPL Model

```
param pi := 6;

param n := 93;

set S := {0..n}; # endpoints
set T := {0..n}; # endpoints

set S0 := {0..n-1};   # edges
set T0 := {0..n-1};   # edges

param x {s in S, t in T} := s*pi/(2*n)-pi/4;
param y {s in S, t in T} := t*pi/(2*n)-pi/4;
var z {s in S, t in T} := log(cos(y[s,t])/cos(x[s,t]));

param dxds {s in S0, t in T0} := (x[s+1,t]-x[s,t]+x[s+1,t+1]-x[s,t+1])/2;
param dxdt {s in S0, t in T0} := (x[s,t+1]-x[s,t]+x[s+1,t+1]-x[s+1,t])/2;
param dyds {s in S0, t in T0} := (y[s+1,t]-y[s,t]+y[s+1,t+1]-y[s,t+1])/2;
param dydt {s in S0, t in T0} := (y[s,t+1]-y[s,t]+y[s+1,t+1]-y[s+1,t])/2;

var   dzds {s in S0, t in T0}  = (z[s+1,t]-z[s,t]+z[s+1,t+1]-z[s,t+1])/2;
var   dzdt {s in S0, t in T0}  = (z[s,t+1]-z[s,t]+z[s+1,t+1]-z[s+1,t])/2;

minimize area:
   sum {s in S0, t in T0}
        sqrt(
          ( dyds[s,t]*dzdt[s,t] - dzds[s,t]*dydt[s,t] )^2 +
          ( dzds[s,t]*dxdt[s,t] - dxds[s,t]*dzdt[s,t] )^2 +
          ( dxds[s,t]*dydt[s,t] - dyds[s,t]*dxdt[s,t] )^2
        )
      ;

fix {s in S} z[s,0];
fix {t in T} z[n,t];
fix {s in S} z[s,n];
fix {t in T} z[0,t];

solve;
```

Finite Impulse Response (FIR) Filter Design

- Audio is stored digitally in a computer as a stream of short (8-bit) integers: u_k, $k \in \mathbb{Z}$.
- When the music is played, these integers are used to drive the displacement of the speaker from its resting position.
- For CD quality sound, 44100 short integers get played per second per channel.



0	-32768	8	-23681	16	12111
1	-32768	9	-18449	17	17311
2	-32768	10	-11025	18	21311
3	-30753	11	-6913	19	23055
4	-28865	12	-4337	20	23519
5	-29105	13	-1329	21	25247
6	-29201	14	1743	22	27535
7	-26513	15	6223	23	29471

FIR Filter Design—Continued

- A *finite impulse response (FIR) filter* takes as input a digital signal and convolves this signal with a finite set of fixed numbers $h_{-n}, \ldots, h_n$ to produce a filtered output signal:

$$y_k = \sum_{i=-n}^{n} h_i u_{k-i}.$$

- Sparing the details, the output power at frequency ν is given by

$$|H(\nu)|$$

where

$$H(\nu) = \sum_{k=-n}^{n} h_k e^{2\pi i k \nu},$$

- Similarly, the mean squared deviation from a flat frequency response over a frequency range, say $\mathcal{L} \subset [0, 1]$, is given by

$$\frac{1}{|\mathcal{L}|} \int_{\mathcal{L}} |H(\nu) - 1|^2 \, d\nu$$

Filter Design: Woofer, Midrange, Tweeter

minimize ρ

subject to $\int_0^1 \left(H_w(\nu) + H_m(\nu) + H_t(\nu) - 1\right)^2 d\nu \leq \epsilon$

$$\left(\frac{1}{|W|}\int_W H_w^2(\nu)d\nu\right)^{1/2} \leq \rho \qquad W = [.2, .8]$$

$$\left(\frac{1}{|M|}\int_M H_m^2(\nu)d\nu\right)^{1/2} \leq \rho \qquad M = [.4, .6] \cup [.9, .1]$$

$$\left(\frac{1}{|T|}\int_T H_t^2(\nu)d\nu\right)^{1/2} \leq \rho \qquad T = [.7, .3]$$

where

$$H_i(\nu) = h_i(0) + 2\sum_{k=1}^{n-1} h_i(k)\cos(2\pi k\nu), \quad i = W, M, T$$

$h_i(k)$ = filter coefficients, i.e., **decision variables**

Specific Example

filter length: $n = 14$

integral discretization: $N = 1000$

constraints	4	
variables	43	
time (secs)		
LOQO	79	($\Longrightarrow$ 0.931)
MINOS	164	
LANCELOT	3401	
SNOPT	35	

Click *here* for demo

AMPL Model

```
param n := 14;
param N := 1000;
param pi := 4*atan(1);
param eps := 1.0e-4;

set NU := {0..1 by 1/N};

param Sw {nu in NU} := (if 0.2<nu && nu<0.8 then 1 else 0);
param Sm {nu in NU} :=
    (if nu<0.1 || (0.4<nu && nu<0.6) || nu>0.9 then 1 else 0);
param St {nu in NU} := (if nu<0.3 || nu>0.7 then 1 else 0);

var rho >= 0;
var hw {0..n-1};
var hm {0..n-1};
var ht {0..n-1};

var Hw {nu in NU} =
    hw[0] + 2* sum {k in 1..n-1} (hw[k]*cos(2*pi*k*nu));

var Hm {nu in NU} =
    hm[0] + 2* sum {k in 1..n-1} (hm[k]*cos(2*pi*k*nu));

var Ht {nu in NU} =
    ht[0] + 2* sum {k in 1..n-1} (ht[k]*cos(2*pi*k*nu));
```

```
minimize power_bnd: rho;

subject to passband:
    sqrt(sum {nu in NU} (Hw[nu]+Hm[nu]+Ht[nu]-1)^2)
    <= sqrt(N)*sqrt(eps);

subject to wooferband:
    sqrt(sum {nu in NU} Hw[nu]^2 * Sw[nu])
    <= sqrt(0.6*N)*rho;

subject to midrangeband:
    sqrt(sum {nu in NU} Hm[nu]^2 * Sm[nu])
    <= sqrt(0.4*N)*rho;

subject to tweeterband:
    sqrt(sum {nu in NU} Ht[nu]^2 * St[nu])
    <= sqrt(0.6*N)*rho;

let hw[0] := 2;
let hm[0] := 2;
let ht[0] := 2;

solve;
```