Non Linear Programming: Homework 6

vishvAs vAsuki

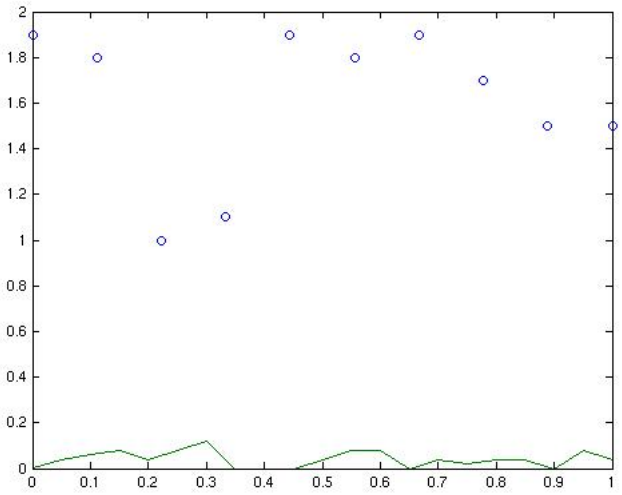
March 2, 2010

1 The illumination problem

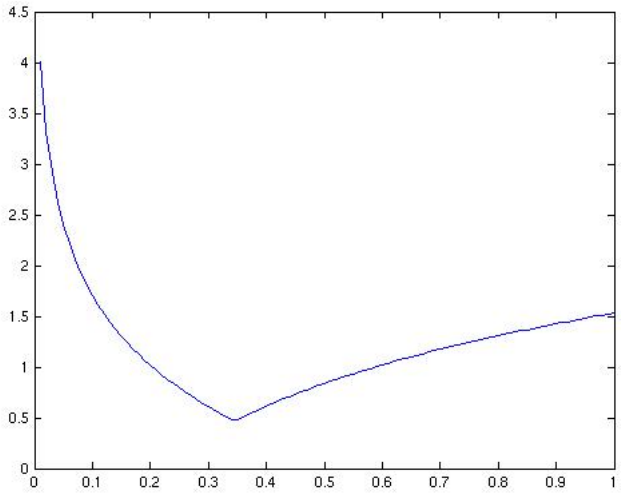
1.1 The problem

$\min_p g_0(p) = max_{k\in[1,n]} h(a_k^T p) : \forall j p_j \in [0,1]$.
$f_0(p) = \log g_0(p)$.

1.2 a Problem instance



1.3 b Equal lamp powers



Graphically, the optimal power associated with each lamp seems to be 0.35.

1.4 c Least squares with saturation

$\min \|Ap - 1\|$, set values of $p_i > 1$ to 1, and values $p_i < 0$ to 0.

Least squares with saturattion: 8.627836e-01.

1.5 d Regularized least squares

$\min \|Ap - 1\|^2 + l \|p - (1/2)1\|^2$. We want to find l which satisfies that atleast one i has $p_i \in [0, 0.05]$ or $p_i \in [0.95, 1]$.

The problems is solved by solving the equation: $(A^T A + lI)p = A^T 1 + (l/2)1$.

Least squares with regularization: 4.463576e-01

1.6 e Chebyshev approximation

$\min \|Ap - 1\|_\infty : 0 \leq p \leq 1$.

Constrained Infinity norm approximation: 4.198242e-01

1.7 f Exact solution

Exact solution: 3.574743e-01

1.8 Code used

```
function cvxCheck
illumdata;
```

```
%      equalLampPowers(A);
%      leastSquaresWithSaturation(A);
%      leastSquaresWithRegularization(A);
%      inftyNormApprox(A);
%      exactSolution(A);
constraintTest
display 'All done,ready for review!';
%      keyboard;
end

function f_0 = illuminationObjective(A,p)
f_0 = max(abs(log(A*p)));
end

function equalLampPowers(A)
[numRows,numParams] = size(A);
g = [0:0.01:1];
objFn = @(x)illuminationObjective(A,x*ones(numParams,1));
f_0 = VectorFunctions.doElementwise(objFn,g);
plot(g,f_0);
end

function leastSquaresWithSaturation(A)
%  %  $\min \norm{Ap -1}$,set values of $p_i >1$ to 1,and values $p_i<0$
to 0.
[numRows,numParams] = size(A);
p = A\ones(numRows,1);
p(find(p>1)) = 1;
p(find(p<0)) = 0;
f_0 = illuminationObjective(A,p);
fprintf(1,'Least squares with saturattion: %d\n',f_0);
end

function leastSquaresWithRegularization(A)
%  $\min \norm{Ap -1}^{2} + l \norm{p -(1/2)1}^{2}$. We want to find
l which satisfies that atleast one i has $p_i \in [0,0.05]$ or $p_i
\in [0.95,1]$.
[numRows,numParams] = size(A);
getRegP = @(l)(A'*A + l*eye(numParams))\(A'*ones(numRows,1) + (l/2)*ones(numParams,1));
bestL=0;
for l = [0:0.1:1]
fprintf(1,'l: %d\n',l);
p = getRegP(l);
p'
if(size(find(p<0.05 & p>=0))>0)
bestL = l;
```

```
break;
end
end
bestL
p'
f_0 = illuminationObjective(A,p);
fprintf(1,'Least squares with regularization: %d\n',f_0);
end

function inftyNormApprox(A)
%  %  $\min \norm{Ap -1}_\infty : 0 \leq p \leq 1$.
[numRows,numParams] = size(A);
cvx_begin
variable p(numParams);
minimize norm(A*p -ones(numRows,1),Inf);
subject to
p >= 0;
p <= 1;
cvx_end
f_0 = illuminationObjective(A,p);
fprintf(1,'Constrained Infinity norm approximation: %d\n',f_0);
end

function exactSolution(A)
[numRows,numParams] = size(A);
cvx_begin
variable p(numParams);
minimize max(max(A*p,inv_pos(A*p)));
subject to
p >= 0;
p <= 1;
cvx_end
f_0 = illuminationObjective(A,p);
fprintf(1,'Exact solution: %d\n',f_0);
end

function constraintTest
cvx_begin
%  variable x(2);
variable x;
variable y;
variable z;
minimize 5
subject to
%  norm( [ x + 2*y ,x -y ] ) == 0;
%  [x + 2*y ,x -y] == [0,0]
```

```
%  square_pos( square( x + y ) ) <= x -y
%  1/x + 1/y <= 1; x >= 0; y >= 0;
%  inv_pos(x) + inv_pos(y) <=1; x >= 0; y >= 0;
%  norm([ max( x ,1 ) ,max( y ,2 ) ]) <= 3*x + y;
variable x1;
variable y2;

max( x ,1 ) -x1 <= 0;
max( y ,2 ) -y2 <= 0;
norm([x1,y2]) <= 3*x + y;

%  x*y >= 1; x >= 0; y >= 0;
%  x >= inv_pos(y); x >= 0; y >= 0;
%  ( x + y )^2 / sqrt( y ) <= x -y + 5;
%  x^3 + y^3 <= 1; x>=0; y>=0;
%  pow_pos(x,3) + pow_pos(y,3) <= 1; x>=0; y>=0;
%  x+z <= 1+sqrt(x*y-z^2); x>=0; y>=0;
%      A = [x z; z y];
%      x+z -det_rootn(A)<= 1; x>=0; y>=0;
cvx_end
end
```

2 Reformulating constraints in cvx

Each of the following cvx code fragments describes a convex constraint on the scalar variables x, y, and z, but violates the cvx rule set, and so is invalid. Briefly explain why each fragment is invalid. Then, rewrite each one in an equivalent form that conforms to the cvx rule set. In your reformulations, you can use linear equality and inequality constraints, and inequalities constructed using cvx functions.

You can also introduce additional variables, or use LMIs. Be sure to explain (briefly) why your reformulation is equivalent to the original constraint, if it is not obvious.

Check your reformulations by creating a small problem that includes these constraints, and solving it using cvx. Your test problem doesnt have to be feasible; its enough to verify that cvx processes your constraints without error.

Hint. You will definitely want to have Appendix B from the CVX Users Guide handy while you do this problem. In fact, it will help to read it before you begin. In particular, learn about the functions that end in _pos, such as square_pos. Remark. This looks like a problem about how to use cvx software, or tricks for using cvx. But it really checks whether you understand the various composition rules, convex analysis, and constraint reformulation rules.

2.1

```
(a) norm( [ x + 2*y ,x -y ] ) == 0
Invalid constraint: {convex} == {constant}
Fix: [x + 2*y ,x -y] == [0,0]
```

2.2

```
(b) square( square( x + y ) ) <= x -y
Illegal operation: square( {convex} ).
Square is not monotonic.
Fix:
square_pos( square( x + y ) ) <= x -y
```

2.3

```
(c) 1/x + 1/y <= 1; x >= 0; y >= 0;
Cannot perform the operation:
{positive constant} ./ {real affine}
Fix:
inv_pos(x) + inv_pos(y) <=1; x >= 0; y >= 0;
```

2.4

```
(d) norm([ max( x ,1 ) ,max( y ,2 ) ]) <= 3*x + y
Cannot perform the operation norm( {convex},2 )
Fix:
variable x1;
variable y2;

max( x ,1 ) <= x1;
max( y ,2 ) <= y2;
norm([x1,y2]) <= 3*x + y;
```

Above, we assume that the objective is altered to minimize x1 and y2 too.

2.5

```
(e) x*y >= 1; x >= 0; y >= 0;
Invalid quadratic form(s): not a square.
Fix:
x >= inv_pos(y); x >= 0; y >= 0;
```

2.6

```
(f) ( x + y )^2 / sqrt( y ) <= x -y + 5
Cannot perform the operation: {convex} ./ {concave}
```

```
Fix:
```

2.7

```
(g) x^3 + y^3 <= 1; x>=0; y>=0;
Illegal operation: {real affine} .^ {3}
(Consider POW_P,POW_POS,or POW_ABS instead.)
Fix:
pow_pos(x,3) + pow_pos(y,3) <= 1; x>=0; y>=0;
```

2.8

```
(h) x+z <= 1+sqrt(x*y-z^2); x>=0; y>=0;
Invalid quadratic form(s): not a square.
Fix:
A = [x z; z y];
x+z -det_rootn(A)<= 1; x>=0; y>=0;
```