Piecewise Linear Programming with PULP

Tom van der Hoeven

December 20, 2016

Abstract

Convex piecewise linear programming is a small and useful extension to linear programming. However not all linear programming packages have this extension. By introducing extra variables in an organized way an LP package can be used for PLP purposes. This article describes a set of Python modules wich allow easy formulation of a PLP problem with variables and equations. The CBC solver is accessed via the PULP package.

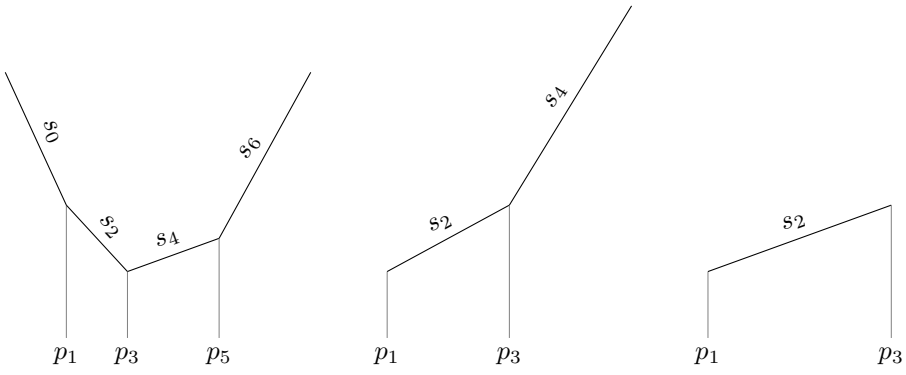
1 Introduction

The convex piecewise linear prgramming problem is

$$\begin{cases} \text{minimize} & C(x) \qquad x \in \mathbb{R}^n \\ \text{where} & C(x) = \sum_{i=1}^{n} \text{plf}_i(x_i) \\ \text{subject to} & Ax = b \qquad b \in \mathbb{R}^m \end{cases}$$

Here follow a few examples of piecewise linear functions (plf). A plf can be defined over $\mathbb{R}$ or over an interval. In case a plf has only one slope, we just have a linear problem. A plf consist of breakpoints and slopes.

$$\text{plf} = (s, p, s, p)$$

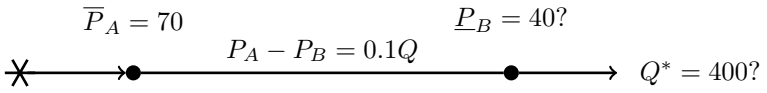


2 Soft boundaries

If an LP problem is infeasable it is often hard to find what is wrong. Instead of using hard boundaries one can use plf's with the original boundaries as points and left and right a penalty. We call this soft boundaries and if the solution is outside the soft boundaries we have a soft violation. The values of the penalties must be chosen such that the interpretation of a soft violation is easy. Eventualy some boundaries can be applied as hard bondaries. In that case one must make sure that there is a feasible solution within the hard boundaries.

3 Example

Consider a pipe between node A and B. The maximum pressure is 70 bar and the minimum contractual pressure is 40 bar. Pressure cannot go below zero bar. The contractual flow $Q = 400$. The pipe equation $P_A - P_B = 0.1Q$. With $Q = 400$ it can be calculated that $P_A - P_B = 40$, while the maximum pressure drop is 30 bar. So this problem is infeasible, when we apply the constraints and equate the flow to 400.

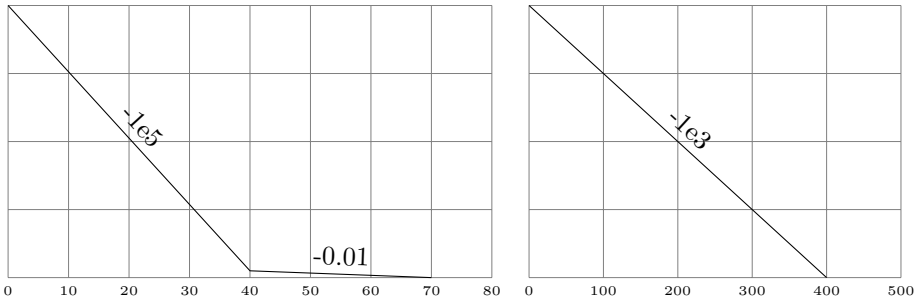


In this example there is always a solution if the pressures are between 0 and 70 bar, and the flow is between 0 and 400. So these values can be used as hard boundaries. A soft boundary is on 40 bar, so there is a penalty of 1e5 below 40 bar. If the pressures are between 40 and 70 we like the result to be as hight as possible, so below 70 bar we put a small penalty of 0.01. Al together for the pressures

$$(p, s, p, s, p) = (0 \quad , \quad -1e5 \quad , \quad 40 \quad , \quad -0.01 \quad , \quad 70)$$

Lower and upper flow bound are 0 and 400, but the value 400 is very much wanted. Below 400 we put a penalty of 1e3. The resulting plf

$$(p, s, p) = (0 \quad , \quad -1e3 \quad , \quad 400)$$



```
from plpcom import plpinit, plpvar, plpeq, plpexit, plpresults

plpinit()

PA = plpvar('pa',[None, 0, -1e5, 40, -0.01 , 70] )
PB = plpvar('pb',[None, 0, -1e5, 40, -0.01 , 70] )
Q  = plpvar('q' ,[None, 0, -1e3, 400] )

Epipe = plpeq('epi', [(1,PA), (-1,PB), (-0.1,Q)] )

plpexit(1)

print '               PA          PB           Q'
print 'x     {:11.2f} {:11.2f} {:11.2f} '.format(PA.x, PB.x, Q.x)
print 'force {:11.2f} {:11.2f} {:11.2f} '.format(PA.force, PB.force, Q.force)
```

The result is

```
               PA          PB           Q
x           70.00       40.00      300.00
force    10000.00   -10000.00    -1000.00
```

The flow is not on target, but as hight as possible. There is a positive force (10000) on pa (70) and a negative force (-10000) on pb (40). This means that a larger maximum pressure or a lower minimum pressure will give rise to a larger flow. In this example the soft violation is on flow. If the flow panalty is changed from 1e3 to 1e6 the violation will be on the soft pressure boundary.

```
Q.plf = [None, 0, -1e6, 400]
```

The result is PA = 70, PB = 30 and Q = 400.
Another way to reach the same goal is to lessen the pressure penalty on PB. The penalty must be lower than the force on PB from the first example, $9900 < 10000$.

```
PB.plf = [None, 0, -9900, 40, -0.01 , 70]
```

Suppose that you want a flow of 800

```
Q.plf = [None, 0, -1e6, 800]
```

Only a flow of 700 is reached because the pressure cannnot go below 0.
In this example the soft violations can easy be interpreted. This holds for larger complicated networks as well.

4 Reference

4.1 class plpvar

The class variable plpvar.varis is a list of plpvar objects. For instance plpvar.varis = [PA, PB, Q] An object from the class plpvar has the following attributes:

attribute	type	in/res	description
name	string	input	name
plf	list of scalars	input	plf
x	scalar	result	x value
force	scalar	result	indirect derivative
ix	integer	result	index in plf
plfcost	scalar	result	plf(x)

A variables is instantiated in one out of three forms
A = plpvar('presa')
A = plpvar('presa', 3)
A = plpvar('presa', plf)
The name 'presa' is used by the LP solver and can be found in the results. Without a second argument there is only one slope with $s = 0$. If the second argument is a scalar we have one slope with $s = 3$. If the second argument is a piecewise linear function we have plfcost(x) = plf(x) A plf can always be changed after instantiatian of the variable, for instance A.plf = [None, 0, -1e6, 400]

4.1.1 Piecewise linear function

A piecewise linear function is described by points (x-values) and slopes. Examples:

(p) , (s) , (p, s) , (s, p) , (p_1, s, p_2) , (s_1, p, s_2) , (p_1, s_1, p_2, s_2) , etcetera

In the list representation an even index refers to a slope and an odd index refers to a point. If a plf starts with a point the first member is None. So

$[None, p]$, $[s]$, $[None, p, s]$, $[s, p]$, $[None, p_1, s, p_2]$, $[s_1, p, s_2]$, etcetera

The list of points must be monotone increasing. The list of slopes must be monotone increasing. The lenght of the list is not limited.
Apart from a constant a plf is well defined. A value x_o can be choosen for which plf(x_o) = 0. If the plf has no points $x_o = 0$ else one can choose

- if $0 \in D$(plf) then $x_o = 0$ else $x_o = x_k$ for the lowest value of $|x_k|$
- $x_o = x_k$ where x_k minimize plf

4.2 class plpeq

The class variable plpeq.eqs is a list of plpeq objects. For instance plpeq.eqs = [Epipe]. An object from the class plpeq has the following attributes:

attribute	type	in/res	description
naam	string	input	name
coefvar	list of tuples and scalars	input	plf
x	scalar	result 0	x value
force	scalar	result	indirect derivative
ix	integer	result 1	

An equation is instantiated by E = plpeq('eqa',coefvar)
E can be used for later reference and the name of the equation is 'eqa'
coefvar is a list of tuples and one or more scalars. So if A = plpvar('x') and B = plpvar('y') and the equation is 6x - 3y = 7 ,
we get coefvar = [(6,A), (-3,B), -7]

4.3 procedure plpinit

The procedure plpinit set plpvar.varis = [ ] and plpeq.eqs = [ ]

4.4 procedure plpexit

The procedure plpexit transforms the plp variables to lp variables in PULP. It executes the CBC solver and calculates the plp results.

4.5 procedure plpresults

The results can be listed by using plpresults(). For each variable plpresults() shows the attibutes x, force, ix and plfcost.

```
{'maxnumber': 1000000, 'dosort': False, 'equat': False, 'minvalue': 0.0}
name                          x        force ix         cost    low-point
pa                        70.00     10000.00  5         0.00
pb                        40.00    -10000.00  3         0.30
aq                       300.00     -1000.00  2    100000.00       400.00
```

If the solution is on a slope

- index ix is even
- the force is the slope value
- the low-point is the point with the lowest cost adjacent to the slope

If the solution is on a point

- index ix is odd
- force is the (negative) shadow price

- slope-left < force < slope right
- if force > 0 a higher value of the point is desirable
- if force < 0 a lower value of the point is desirable

In case of soft voilation the lower point shows the soft constraint. Normaly the results are obtained by referring to the attributes of the variable object like PA.x or PA.force. The procedure plpresults is used to get the variables with the largest cost or the largest force.

keyword	type	description
	'force'	sort on decreasing force
dosort	'cost'	sort on decreasing cost
	'name'	sort on name
maxnumber	integer	max number of results
minvalue	real	minimum value of force or cost
equat	boolean	show force on equations

If there are soft violations, there is tention in the system. Now plpresults(dosort = 'force', maxnumber = 30, minvalue = 1e4) gives you maximum 30 results ordered on force, where force greater than 1e4.

4.6 Adding plf's

Sometimes a variable has cost or is bounded from two perspectives. In that case it is usefull to be able to add two plf's. Roughly speaking the points of the sum is a superset and the slopes are added. sumplf = addplf(plf1, plf2)

5 Mathematics

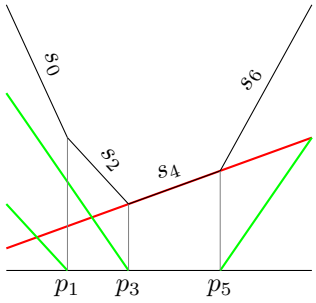
5.1 Slope variables

For each slope of a plf there has to be a lp variable. Let s be the index of a slope. Let c be the index of the central slope which has the minimal absolute value. For plp variable X Define the lp variable X_c such that

$$lb \leq X_c \leq ub \text{ and } \text{cost}_{X_c} = \text{plf}(c)X_k - \text{plf}(c)x_o$$

For each other slope s define a positive variable X_s with

$$\text{for } s < c : X_s \geq 0 \text{ and } X_s \geq -X_c + \text{plf}(s+1) \text{ and } \text{cost}_{Xs} = (\text{plf}(s+2) - \text{plf}(s)X_s$$
$$\text{for } s > c : X_s \geq 0 \text{ and } X_s \geq +X_c - \text{plf}(s-1) \text{ and } \text{cost}_{Xs} = (\text{plf}(s) - \text{plf}(i-2)X_s$$



In the figure the red line represents X_c and the green lines the other slope variables.

5.2 forces

Because there are more variables than constraints the set of variables can be devided in Free and Dependant variables. There is always an optimal solution where the free variables are at a breakpoint.
The left-hand derivative is

$$\left.\frac{\partial C}{\partial x_i}\right|_- = \left.\frac{df_i}{dx_i}\right|_- + \sum_{j \in D} \frac{\partial x_j}{\partial x_i} \frac{df j}{dx_j}$$

and the right-hand derivative is

$$\left.\frac{\partial C}{\partial x_i}\right|_+ = \left.\frac{df_i}{dx_i}\right|_+ + \sum_{j \in D} \frac{\partial x_j}{\partial x_i} \frac{df j}{dx_j}$$

An optimum is found if for all free variables

$$\left.\frac{\partial C}{\partial x_i}\right|_- \leq 0 \leq \left.\frac{\partial C}{\partial x_i}\right|_+$$

Refer to x_i as x, so drop the index. The left and right derivative both have an indirect component. It looks like a force that pushes the a breakpoint into a certain direction. Define the force at point x_p by

$$\text{force}(p) = -\sum_{j \in D} \frac{\partial x_j}{\partial x} \frac{df j}{dx_j}$$

Around a slope the derivatives can be written as

$$\left.\frac{\partial C}{\partial x}\right|_{-}^{xp} = \text{plf}(p-1) - \text{force}(p)$$

and

$$\left.\frac{\partial C}{\partial x}\right|_{+}^{xp} = \text{plf}(p+1) - \text{force}(p)$$

At optimality we have around a point p

$$\text{plf}(p-1) \leq \text{force}(p) \leq \text{plf}(p+1)$$

If $x_p$is the lowerbound of x, it is the lowerbound of x_c as well

$$\left.\frac{\partial C}{\partial x}\right|_{+}^{xp} = \text{Dual}(x_c) \implies \text{force}(p) = \text{plf}(p+1) - \text{Dual}(x_c)$$

If $x_p$is the upperbound of x, it is the upperbound of x_c as well

$$\left.\frac{\partial C}{\partial x}\right|_{-}^{xp} = \text{Dual}(x_c) \implies \text{force}(p) = \text{plf}(p-1) - \text{Dual}(x_c)$$

For breakpoint $p > c$

$$\left.\frac{\partial C}{\partial x}\right|_{+}^{xp} = \text{Dual}(x_{p+1}) \implies \text{force}(p) = \text{plf}(p+1) - \text{Dual}(x_{p+1})$$

For a breakpoint $p < c$

$$\left.\frac{\partial C}{\partial x}\right|_{=}^{xp} = \text{Dual}(x_{p-1}) \implies \text{force}(p) = \text{plf}(p-1) + \text{Dual}(x_{p-1})$$