PiecewiseLinearOpt.jl

Solving optimization problems containing piecewise linear functions
Linear optimization (LP)

\[ \min_x c^T x \]
\[ \text{s.t. } Ax \leq b \]

- Surprisingly powerful modeling capabilities
- Can solve large instances, quickly
Piecewise linear (PWL) optimization

\[
\min_{x} \quad f(x) \\
\text{s.t.} \quad Ax \leq b
\]

Where

\[
f(x) = \begin{cases} 
  c_1^T x + b_1 & x \in P^1 \\
  c_2^T x + b_2 & x \in P^2 \\
  \vdots \\
  c_d^T x + b_d & x \in P^d 
\end{cases}
\]

- Appears in economics, operations, engineering, ...
- Also, “simple” LP approximation for nonlinear optimization
Piecewise linear (PWL) optimization

\[
\min_x f(x) \\
\text{s.t. } Ax \leq b
\]

Where

\[
f(x) = \begin{cases} 
  c_1^T x + b_1 & x \in P^1 \\
  c_2^T x + b_2 & x \in P^2 \\
  \vdots & \\
  c_d^T x + b_d & x \in P^d 
\end{cases}
\]

- When \( f \) is convex, exists canonical transformation to LP
- What about when \( f \) is nonconvex?
Mixed-integer optimization (MIP)

- Discrete variables let you to model discrete decisions
- Discrete decision for PWL function: which piece you are on

\[(x, y) \in \{ (x, c_1^T x + b_1) : x \in P^1 \} \cup \{ (x, c_2^T x + b_2) : x \in P^2 \} \cup \cdots \cup \{ (x, c_d^T x + b_d) : x \in P^d \} \]

- Standard techniques for modeling logic with MIP → d auxiliary binary variables
  - Almost never what you want!
- 10+ MIP formulations in literature for univariate functions (!)
- Even more complex for higher dimensions
What MIP formulations are there?
Logarithmic formulation (J.P. formulation)

- For univariate PWL functions \( f : \mathbb{R} \to \mathbb{R} \)
- One particularly small, strong formulation
- Built around Gray codes: sequence of distinct vectors \( \{v^i\}_{i=1}^{N-1} \subseteq \{0, 1\}^{\lfloor \log_2(N-1) \rfloor} \)
- with \(|v^i - v^{i+1}| = 1\) and \(v^0 = v^1, v^N = v^{N-1}\)
- Take \( L^j = \{ i \in [N] : v_{j-1}^i + v_j^i \geq 1 \}, R^j = \{ i \in [N] : v_{j-1}^i + v_j^i \leq 1 \} \)
  for each \( j \in [\lfloor \log_2(N-1) \rfloor] \)
- Take univariate PWL function given by breakpoints \( \{(x^i, y^i)\}_{i=1}^{N} \)
Logarithmic formulation (J.P. formulation)

\[
x = \sum_{i=1}^{N} x^i \lambda_i
\]

\[
y = \sum_{i=1}^{N} y^i \lambda_i
\]

\[
\sum_{i \in [N] \setminus L^j} \lambda_i \leq z_j \quad \forall j \in \lfloor \log_2(N - 1) \rfloor
\]

\[
\sum_{i \in [N] \setminus R^j} \lambda_i \leq 1 - z_j \quad \forall j \in \lfloor \log_2(N - 1) \rfloor
\]

\[
\sum_{i=1}^{N} \lambda_i = 1
\]

\[
\lambda \geq 0
\]

\[
z \in \{0, 1\}^{\lfloor \log_2(N - 1) \rfloor}, y \in H
\]
Logarithmic formulation (J.P. formulation)

- What we need:
  1. Understand the paper
  2. Construct a Gray code
  3. Construct the sets \( \{(A^j, B^j)\}_{j=1}^{\log_2(N-1)} \)
  4. Find the breakpoints (if you have a functional form)

- Not impossible, but non-trivial
- Anecdotally, a real barrier for practitioners
- And this is just univariate functions!
2-dimensional PWL functions

- How do we partition the domain?
- Apply gridding along each dimension, triangulate each subrectangle
- Much richer combinatorial structure
- Even harder to model with MIP
- Some formulations work for “regular” triangulations only
PiecewiseLinearOpt.jl

1. Representations for arbitrary, multidimensional piecewise linear functions
   ○ Helper constructors to go from functional form → “breakpoint” representation
2. JuMP extension to build MIP models for 1D and 2D PWL functions
   ○ 10 formulations for 1D
   ○ 9 formulations for 2D
julia> UnivariantePWLFUNCTION(1:5, [1,2,4,7,11])
# => PWLFUNCTION{1}([(1.0,),(2.0,),(3.0,),(4.0,),(5.0,)],[1.0,2.0,4.0,7.0,11.0],[[1,2],[2,3],[3,4],[4,5]],Dict())
julia> UnivariatePWLFunction(1:5, sin)
# => PWLFunction{1}([(1.0,), (2.0,), (3.0,), (4.0,), (5.0,)],
    [0.8414, 0.9092, 0.141, -0.7568, -0.9589],
    [[1, 2], [2, 3], [3, 4], [4, 5]],
Dict())
BivariatePWLFuncti

BivariatePWLFuntion(1:5, 1:5, (x,y)->(x-1/3)^2+3(y-4/7)^4) # => PWLFuncti\n


Dict(:structure=>:BestFit)
julia> BivariatePWLFunction(1:5, 1:5, (x,y)->(x-1/3)^2+3(y-4/7)^4)
# => PWLFunction{2}([[(1.0,1.0),(2.0,1.0),..., (4.0,5.0),(5.0,5.0)]],
                      
                      [0.545652,2.87899, ...,1167.36,1175.7],
                      [[1,6,2],[2,6,7],...,[19,24,20],[20,24,25]],
                      Dict(:structure=>:BestFit))
MIP formulations for grid triangulations

- State-of-the-art, circa 2014:
  - “Universal” MIP formulation approaches (# of binaries = # of triangles)
  - Small formulations for highly structured triangulations (# of binaries = log(# of triangles)+1)
- In 2015:
  - J.P. shows a small formulation for looser structure (# of binaries = log(# of triangles)+2)
- In 2016:
  - Small formulation for any triangulation (# of binaries = log(# of triangles)+9)
- In 2017:
  - Whittled down to log(# of triangles)+6
- Constants have appreciable affect on performance
Piecewise linear functions in JuMP

using JuMP, PiecewiseLinearOpt, Gurobi
model = Model(solver=GurobiSolver())
@variable(model, x)
d = 0:(π/4):2π
z = piecewiselinear(model, x, d, sin)
# = piecewiselinear(model, x, UnivariatePWLFunction(d,sin))
@objective(model, Max, z)
using JuMP, PiecewiseLinearOpt, Gurobi
model = Model(solver=GurobiSolver())
@variable(model, x)
d = 0:(pi/4):2pi
z = piecewiselinear(model, x, d, sin, method=:Incremental)
# = piecewiselinear(model, x, UnivariantePWLFunction(d,sin))
@objective(model, Max, z)
New research directions (i.e. open PRs)
“Optimal” MIP formulations

- The “constant” term for grid triangulations can make a difference of up to 3-5x
- Can find the “optimal” formulation (smallest # of binaries) by solving a MIP!
- MIP is not very scaleable (yet), but...
- Optimal MIP formulations is best performer, once computed
- Somewhat surprising: loses grid structure along each dimension (x and y)
- Trivial to implement in JuMP:
  - Solve a MIP in JuMP during model generation
Moment curve formulations

- Relaxed notion of “MIP formulation”
  a. Combines:
    i. Algebraic relaxation for (convex hull of) disjunctive set, with auxiliary integer variables
    ii. Constraint programming -like treatment of control variables ($y \in H$)

- E.g. embed disjunctive sets along moment curve
  a. Have inequality description for algebraic description
  b. Treat control variables with custom branching scheme

- Implement in JuMP (and CPLEX) using:
  a. Branching callbacks
  b. Incumbent callbacks
Future directions

- JuMP models for...
  - Higher dimensional PWL functions
  - More complex partitions of domains (not just triangulations)
- Connections with similar problems
  - Convex envelopes for bilinear (and multilinear) functions
  - Approximations for structured nonlinear functions
    - Signomial programming
    - Difference of convex programming