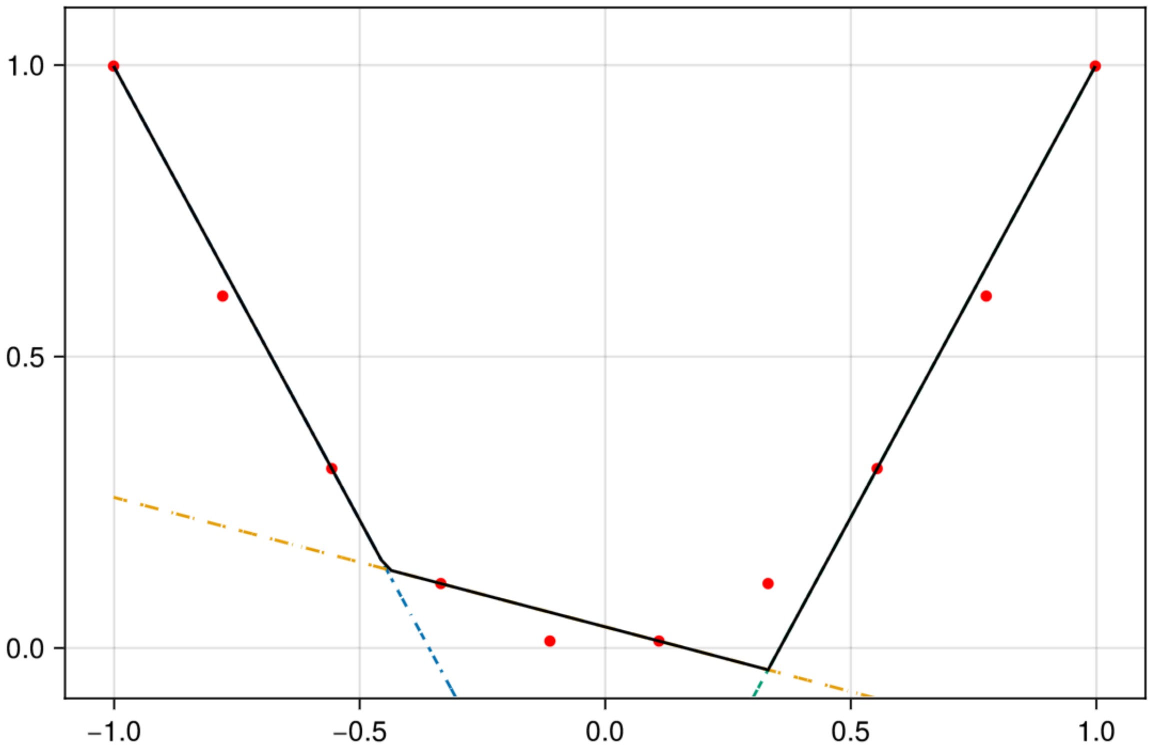
Present

PiecewiseAffineApprox.jl

Lars Hellemo, Truls Flatberg, Thiago Silva (SINTEF)



Agenda

- About Us
- Motivation
- Demo
- Energy Applications
- Future Improvements

About Us

- SINTEF is one of Europe's largest independent research organisations (~2200 employees).
- Department of Sustainable Energy Technology, Optimization group
- MILP optimization (mostly modelling) for different sectors:
    - Supply chain optimization
    - Energy systems
    - Health care
    - Sustainability
- Moving towards open models, more and more JuMP/Julia
    - EnergyModelsX
    - TimeStruct
    - SparseVariables

"

Motivation

- Often need to model nonlinearities by means of **linearization**
- Colleagues or customers often use **more detailed dynamical models** (not *yet* in Julia?)
- Spending computation time on good/optimal approximation may be a good trade-off
- Library with **robust methods**
    - Easy to update with **new data**
    - We hope it can be useful for others
    - We hope to improve the quality and robustness by having more users/contributors

Overview

- Generate piecewise affine approximations from point estimates
    - convenience function to sample functions
- Convex/Concave functions
    - method to convexify estimates with numerical errors/noise
- Currently support 3 methods:
    1. `MILP` to fit a set of points, partially based on Toriello & Vielma, 2012.
    2. `Cluster` uses a heuristic to fit the set of points, based on Magnani & Boyd, 2009.
    3. `Progressive` uses a heuristic to add planes until a certain accuracy is met, based on Kazda & Li, 2024

Demo

```julia
using JuMP, HiGHS, PlutoUI
```

```julia
using PiecewiseAffineApprox  # Now registered in General
```

```
optimizer =  OptimizerWithAttributes(HiGHS.Optimizer, [Silent() ⇒ true])
```

```julia
optimizer = optimizer_with_attributes(HiGHS.Optimizer, MOI.Silent()=>true)
```

```
x =
  [-1.0, -0.777778, -0.555556, -0.333333, -0.111111, 0.111111, 0.333333, 0.555556, 0.777778,
```

```julia
x = collect(range(-1, 1; length = 10))
```

```
z =
  [1.0, 0.604938, 0.308642, 0.111111, 0.0123457, 0.0123457, 0.111111, 0.308642, 0.604938, 1.0
```

```julia
z = x .^ 2
```

```
pwa = PiecewiseAffineApprox.PWAFunc{PiecewiseAffineApprox.Convex, 1} with 3 planes:
       z ≥ -1.555555555557575 x₁ +  -0.555555555557575
       z ≥ -0.22222222222149085 x₁ +  0.037037037037273006
       z ≥ 1.5555555555555556 x₁ +  -0.5555555555555556
```

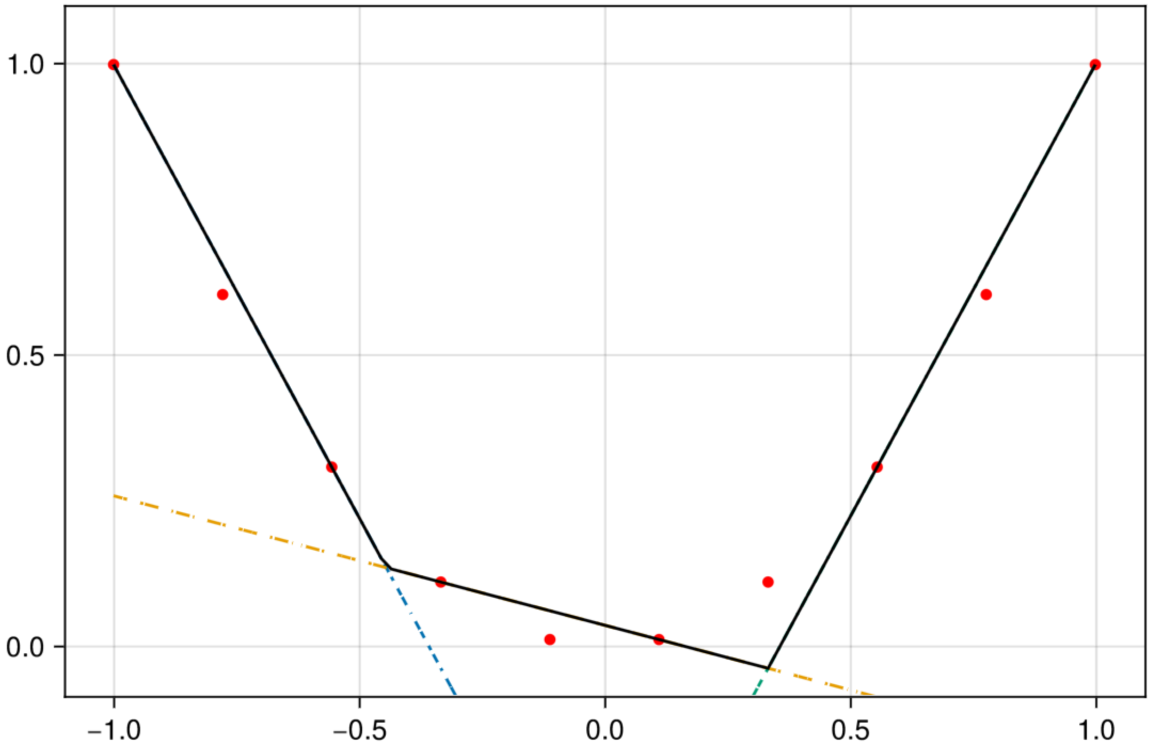
```julia
pwa = approx(
        FunctionEvaluations(Tuple.(x), z),
        Convex(),
        MILP(;optimizer, metric = metric, planes = nplanes, strict=strict),
    )
```

```
PWAFunc{Convex, 1}
  planes: Array{Plane{1}}((3,))
    1: Plane{1}
      α: Tuple{Float64}
        1: Float64 -1.555555555557575
      β: Float64 -0.555555555557575
    2: Plane{1}
      α: Tuple{Float64}
        1: Float64 -0.22222222222149085
      β: Float64 0.037037037037273006
    3: Plane{1}
      α: Tuple{Float64}
        1: Float64 1.5555555555555556
      β: Float64 -0.5555555555555556
```

```julia
Dump(pwa)
```

Visualizing the results

```julia
using WGLMakie
```



```
plot(x, z, pwa)
```

```
@bind nplanes PlutoUI.Slider(1:5; default=3)
```

none

```
@bind strict PlutoUI.Select([:none,:inner,:outer]; default=:none)
```

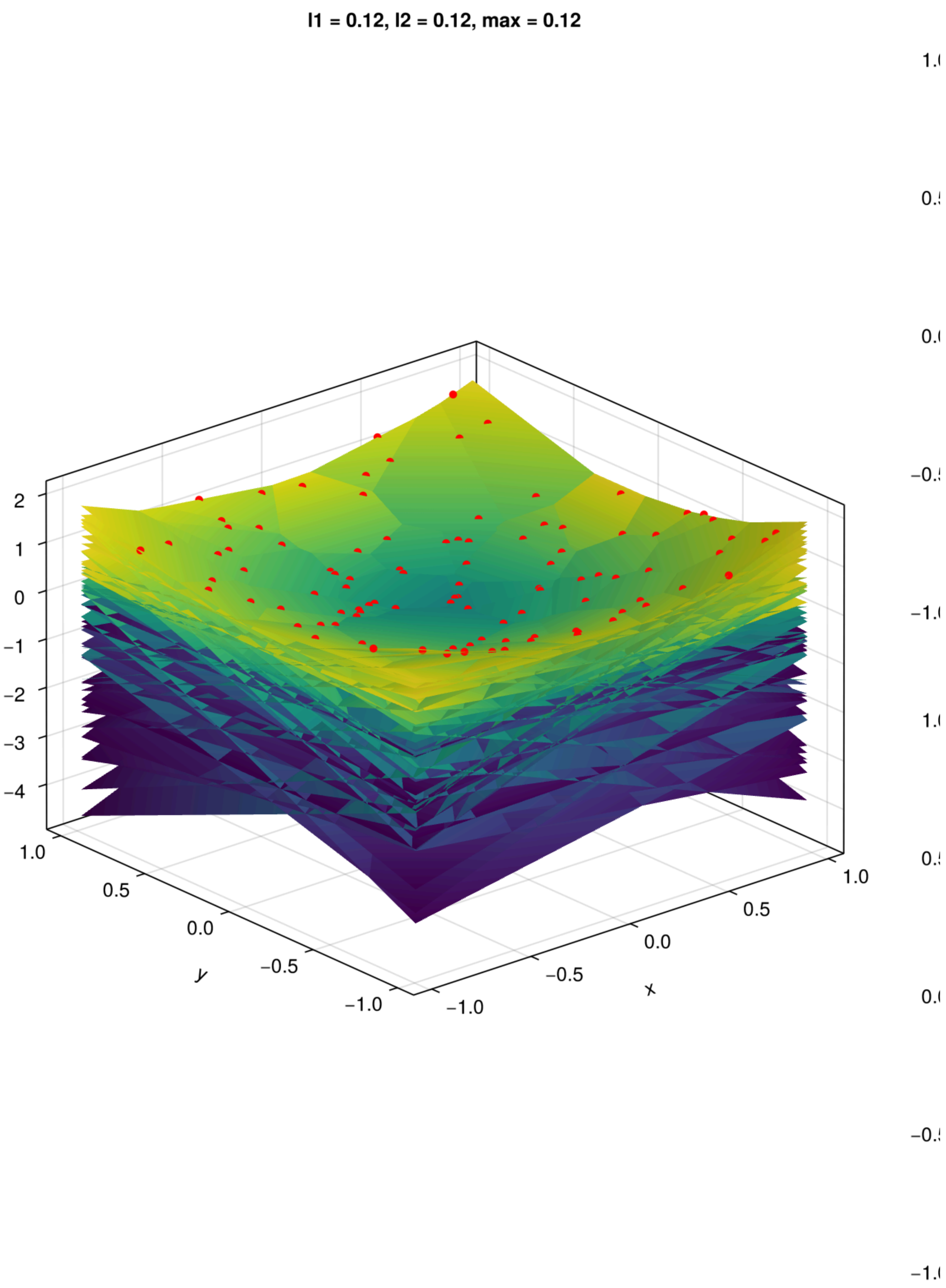
l1

```
@bind metric PlutoUI.Select([:l1,:max,:l2]; default=:l1)
```

3D example

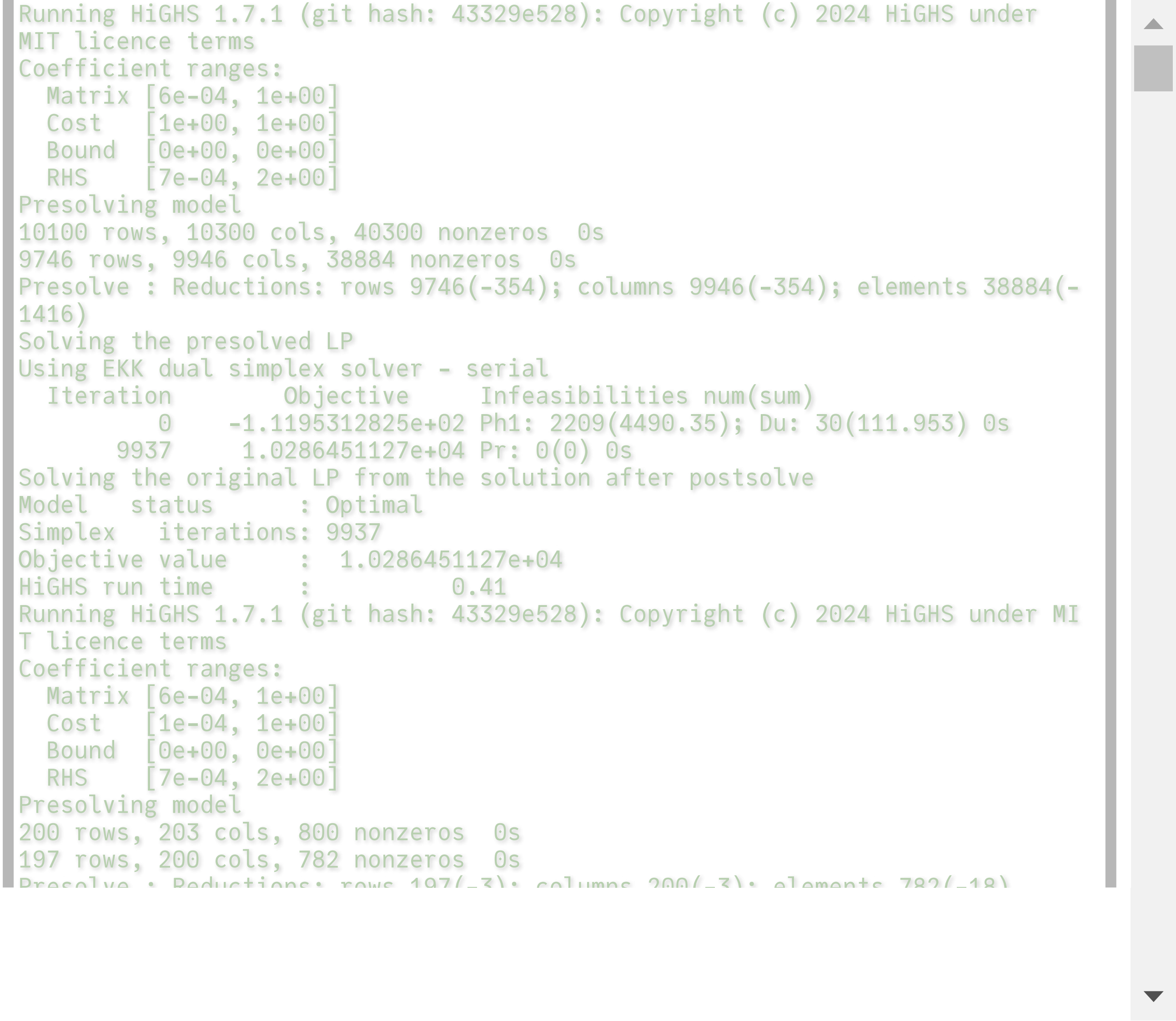
```
demo (generic function with 1 method)
```

```
function demo()
    optimizer = optimizer_with_attributes(HiGHS.Optimizer, MOI.Silent()=>true)

    I = 100
    xmat = 2 * rand(2, I) .- 1
    x = [Tuple(xmat[:, i]) for i = 1:size(xmat, 2)]
    z = [p[1]^2 + p[2]^2 for p in x]
    vals = FunctionEvaluations(x, z)

    pwa = approx(
        vals,
        Convex(),
        Progressive(; optimizer = HiGHS.Optimizer, ),
    )
    p = plot(vals, pwa)
end
```



```
demo()
```

```
Fitting finished, error = 0.12, p = 100
```



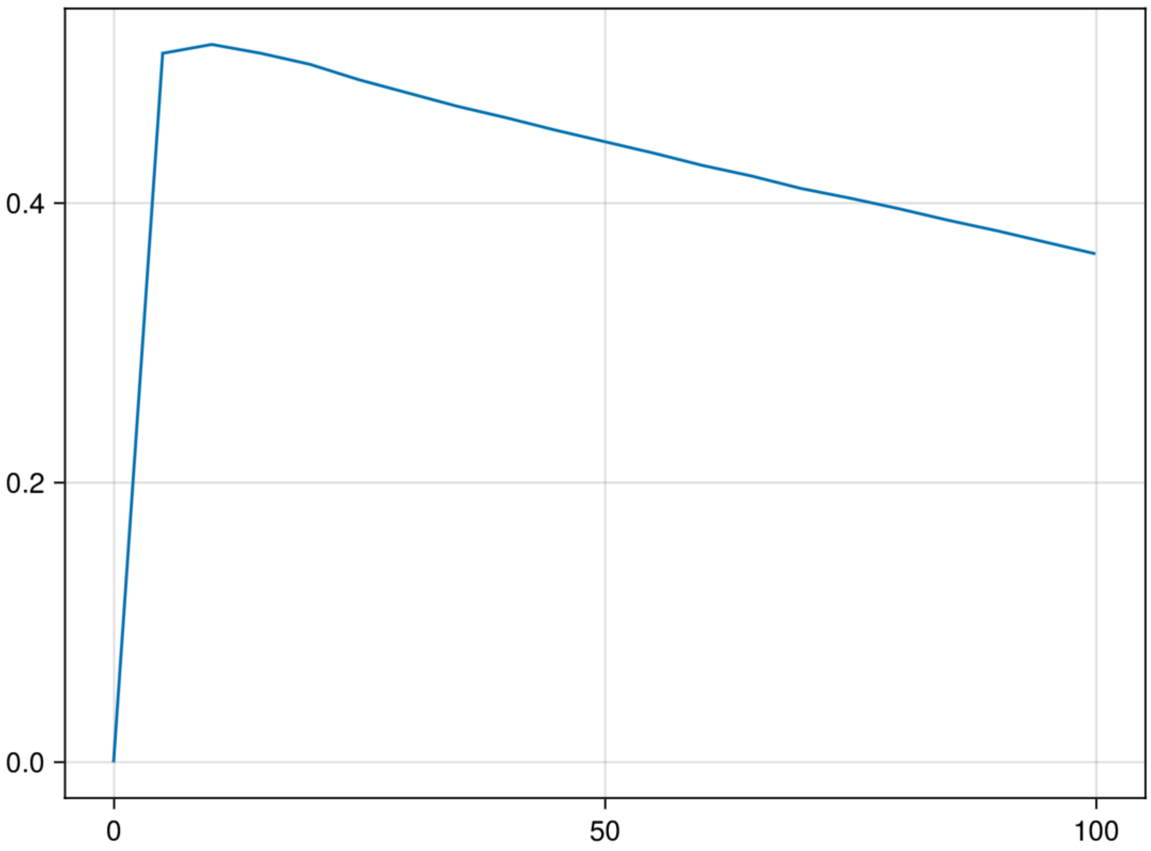
```
Running HiGHS 1.7.1 (git hash: 43329e528): Copyright (c) 2024 HiGHS under
MIT licence terms
Coefficient ranges:
  Matrix [6e-04, 1e+00]
  Cost   [1e+00, 1e+00]
  Bound  [0e+00, 0e+00]
  RHS    [7e-04, 2e+00]
Presolving model
10100 rows, 10300 cols, 40300 nonzeros  0s
9746 rows, 9946 cols, 38884 nonzeros  0s
Presolve : Reductions: rows 9746(-354); columns 9946(-354); elements 38884(-
1416)
Solving the presolved LP
Using EKK dual simplex solver - serial
  Iteration        Objective     Infeasibilities num(sum)
          0    -1.1195312825e+02 Ph1: 2209(4490.35); Du: 30(111.953) 0s
       9937     1.0286451127e+04 Pr: 0(0) 0s
Solving the original LP from the solution after postsolve
Model   status      : Optimal
Simplex   iterations: 9937
Objective value     :  1.0286451127e+04
HiGHS run time      :          0.41
Running HiGHS 1.7.1 (git hash: 43329e528): Copyright (c) 2024 HiGHS under MI
T licence terms
Coefficient ranges:
  Matrix [6e-04, 1e+00]
  Cost   [1e-04, 1e+00]
  Bound  [0e+00, 0e+00]
  RHS    [7e-04, 2e+00]
Presolving model
200 rows, 203 cols, 800 nonzeros  0s
197 rows, 200 cols, 782 nonzeros  0s
Presolve : Reductions: rows 197(-3); columns 200(-3); elements 782(-18)
```

Energy Applications

- Process **efficiency**
  - Electrolyser
  - **Fuel Cell**
  - Compression
  - Etc
- When solving repeatedly or with high temporal resolution, **reusing a piecewise convex approximation** pays off in solution time
- ISMP TB870 Example application in the Arctic

Fuel Cell example

Get system efficency estimates from dynamic model (Dymola/Modelica):



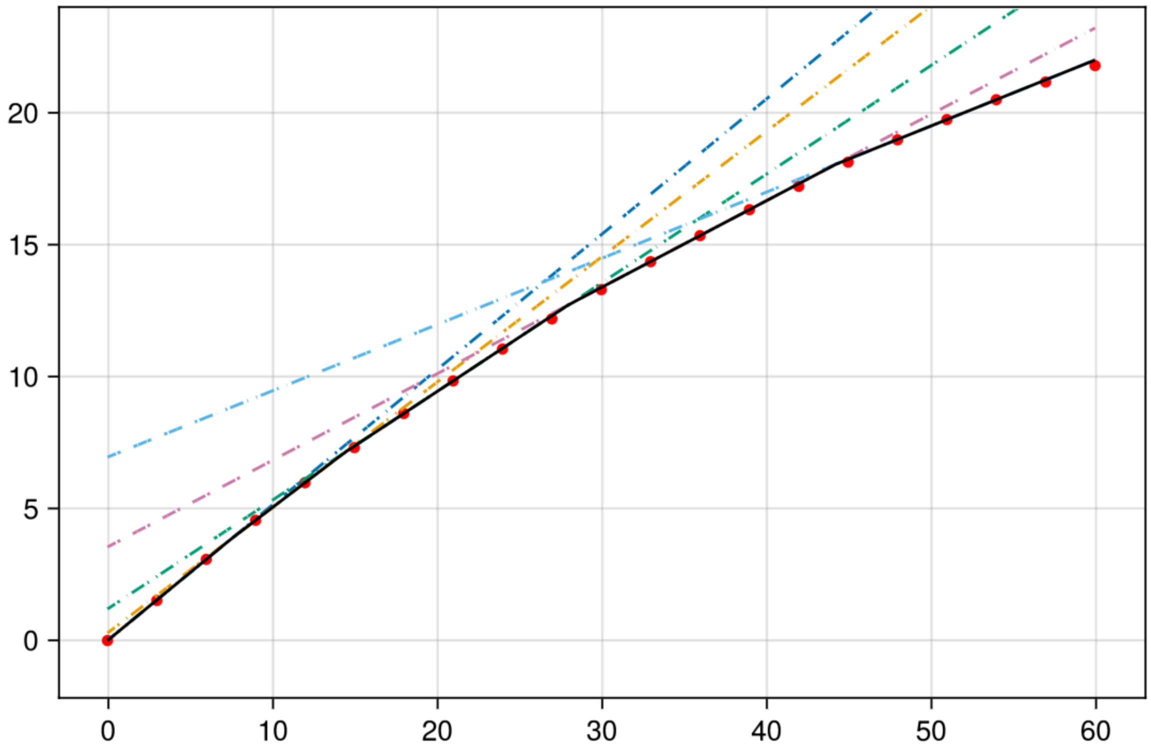
```
    lines(fc_data[:,1],fc_data[:,2])
```

Transform to input->output and linearize:

```
    fc_io = transform_io(fc_data, 60);
```

```
pwa_fc = PiecewiseAffineApprox.PWAFunc{PiecewiseAffineApprox.Concave, 1} with 5 planes:
         z ≤ -0.5135509999999998 x₁ +  -0.0
         z ≤ -0.47562699999999997 x₁ +  -0.28443600000000036
         z ≤ -0.4124460000000064 x₁ +  -1.1946059999998677
         z ≤ -0.32839100000000104 x₁ +  -3.549311999999961
         z ≤ -0.25128300000000264 x₁ +  -6.962111999999864
```

```
    pwa_fc = approx(
            FunctionEvaluations(Tuple.(fc_io[:,1]), fc_io[:,2]),
            Concave(),
            MILP(;optimizer, metric = :l1, planes = 5, strict=:outer),
        )
```



```
    plot(fc_io[:,1], fc_io[:,2], pwa_fc)
```

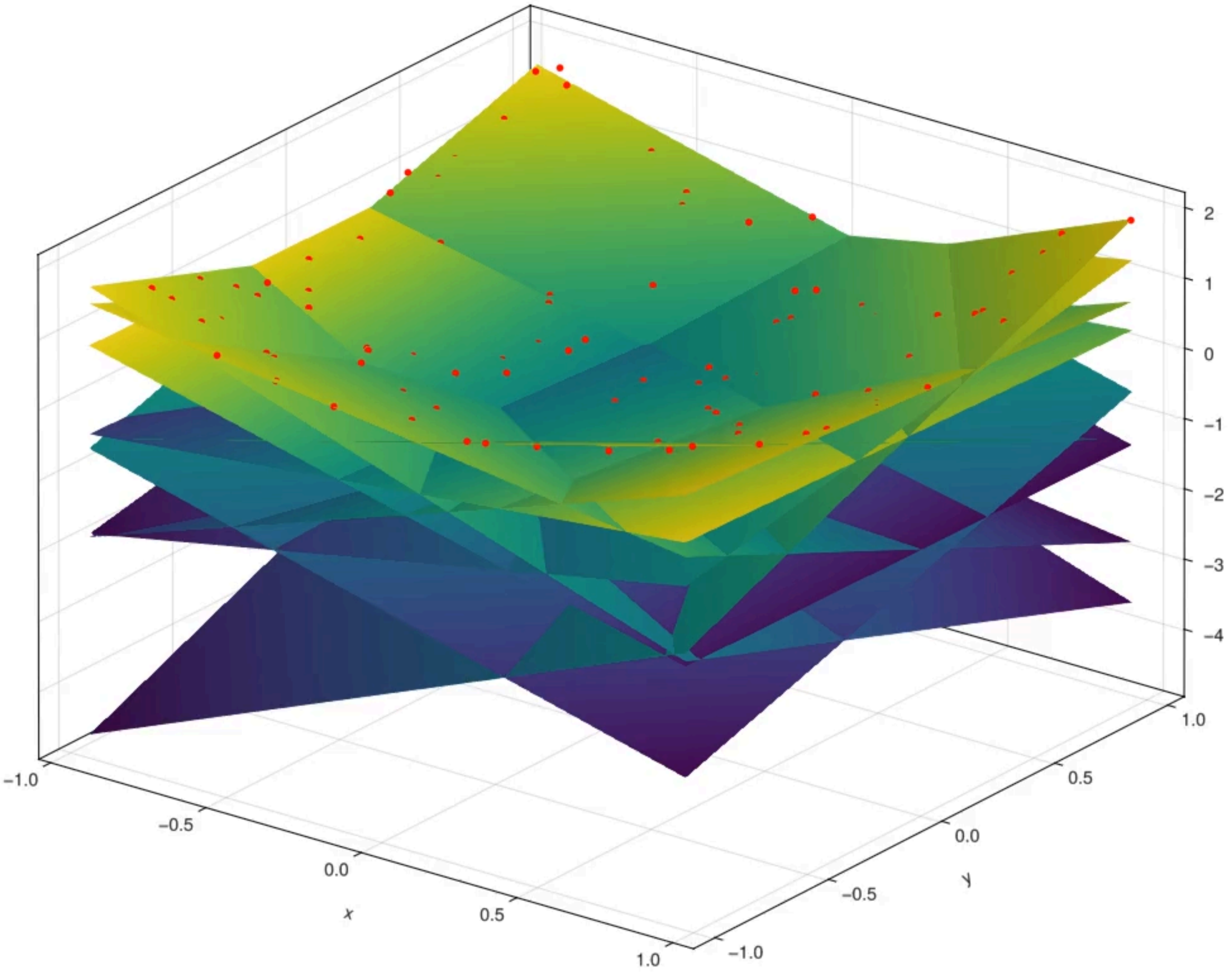
```
transform_io (generic function with 2 methods)
```

```
    function transform_io(d, cap=50)
        input = d[:,1] .* 0.01 .* cap
        output = d[:,2] .* input
        io = [input output]
    end
```

Future Improvements

- Interface improvements
- Improve robustness - better method to calculate bigM
- More methods
- Give it a try and contribute: https://github.com/sintefore/PiecewiseAffineApprox.jl

Technology for a better society



2
1
0
−1
−2
−3
−4
−1.0
−0.5
0.0
0.5
1.0
x
1.0
0.5
0.0
−0.5
−1.0
y