Constrained Optimziation: KKT & LICQ

- Mathe 3 (CES)
- WS24
- Lambert Theisen (`theisen@acom.rwth-aachen.de`)

Use `SymPy` symbolic library

- Is a wrapper to Python's `SymPy`
- Using Python directly would be (probably) better
- But we don't want to loose the luxury of `Pluto.jl`

```
using PlutoUI, SymPy, PyPlot, LinearAlgebra
```

Define Variables and Lagrange Multipliers

x = $[x_1, x_2]$

```
x = [
    symbols("x1", real=true),
    symbols("x2", real=true),
]
```

Define Objective and Constraints

```
A = 2×2 Matrix{Int64}:
  1  5
  5  1
```

```
[-4.0, 6.0]
```

```
eigen(A).values
```

```
f1 (generic function with 1 method)
```

```
f1(x) = (x[1]-1)^2 + (x[2]-2)^2
# f1(x) = x' * A * x
```

$$(x_1 - 1)^2 + (x_2 - 2)^2$$

```
f1(x)
```

lambdas = (λ_0, λ_1)

```
lambdas =
    symbols("lambda:$(length(g))", real=true, nonnegative=true)
    # for i=1:length(g)
# ]
```

mus = $[\mu_1, \mu_2]$

Define Lagrangian

$$\mathcal{L}(x, \lambda, \mu) = f(x) - \sum_{i=1}^{m} \lambda_i g_i(x) - \sum_{j=1}^{q} \mu_j h_j(x)$$

lagrangian (generic function with 1 method)

```
function lagrangian(x, f, g, h, λs, μs, Ig)
    return (
        f(x)
        - sum([g[i](x) * λs[i] for i=1:size(g)[1]]; init=0)
        - sum([h[i](x) * μs[i] for i=1:size(h)[1]]; init=0)
        # - reduce(+, [g[i](x) * λs[i] for i=1:size(g)[1]], init=0)
        # - reduce(+, [h[i](x) * μs[i] for i=1:size(h)[1]], init=0)  # sum
    )
end
```

$$-\lambda_0 (x_2 + 1) - \lambda_1 (x_1 + x_2) - \mu_1 \left(-5x_2 + (x_1 - 1)^2\right) - \mu_2 \left(-10x_2 - (x_1 - 1)^2 + 2\right) + ($$

```
lagrangian(x, f1, g, h, lambdas, mus, [])
```

KKT Points

KKT points (x^*, λ^*, μ^*) fulfill:

1. $$\nabla_x \mathcal{L}(x, \lambda, \mu) = 0$$
2. $$h_j(x) = 0 \quad \forall j = 1, \dots, q$$

3. $$g_i(x) \geq 0 \quad \forall i = 1, \ldots, m$$

4. $$\lambda_i \geq 0 \quad \forall i = 1, \ldots, m$$

5. $$g_i(x)\lambda_i = 0 \quad \forall i = 1, \ldots, m$$

kktpoints (generic function with 1 method)

```
function kktpoints(x, f, g, h, λs, μs, Ig)
    lag = lagrangian(x, f, g, h, λs, μs, Ig)
    eqs = [
        diff(lag, x[1]),
        diff(lag, x[2]),
        [diff(lag, mus[i]) for i=1:length(h)]...,  # <=> h_i(x)==0
        [diff(lag, lambdas[i])*lambdas[i] for i=1:length(g)]...,  # use active g's
        [g[i](x)*lambdas[i] for i=1:length(g)]...,
    ]
    sols = solve(eqs, [x...,mus...,lambdas...])
    # filter for "gi > 0" solutions since sympy cannot really solve ineqs...
    return filter(sol->all([gi(sol[1:2])>=0 for gi in g]),sols)
end
```

Test KKT Points

Inequality Constraints: $g_i(x) \geq 0$

$[x_2 + 1, \; x_1 + x_2]$

```
g = [
    x -> x[2] + 1,
    x -> x[2] + x[1] - 0,
]; [gi(x) for gi in g]
```

Equality Constraints: $h_j(x) = 0$

$[-5x_2 + (x_1 - 1)^2, \; -10x_2 - (x_1 - 1)^2 + 2]$

```
h = [
    x -> (x[1]-1)^2 - 5*x[2],
    x -> 2-(x[1]-1)^2 - 10*x[2],
    # x -> x' * x - 1
]; [hi(x) for hi in h]
```

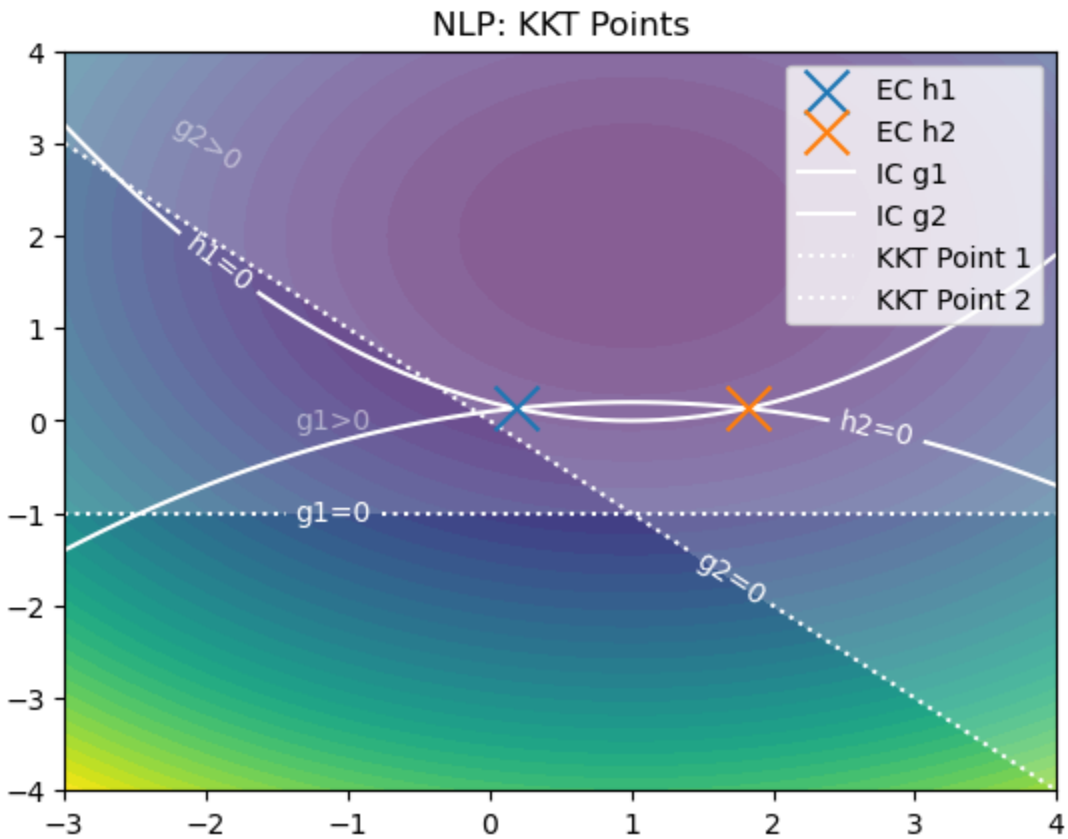
$$[(1-\frac{\sqrt{6}}{3}, \frac{2}{15}, \frac{206}{225}, -\frac{19}{225}, 0, 0), (\frac{\sqrt{6}}{3}+1, \frac{2}{15}, \frac{206}{225}, -\frac{19}{225}, 0, 0)]$$

```
kktpoints(x,f1,g,h,lambdas,mus,[])
```

```
2×2 Matrix{Float64}:
 -0.707107  0.707107
  0.707107  0.707107
```

```
eigen(A).vectors
```

Visualize KKT Points



Linear Independence Constraint Quality (LICQ)

Point $x \in \chi$ satisfies LICQ if:

$$\{\nabla h_j(x)\}_{j=1}^{q}, \{\nabla g_i(x)\}_{i \in I_g(x)}$$

are linearly independent. The set of active inequality constraints at point x is labelled with $I_g(x)$.

Index Set of Active Constraints:

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

```julia
function Ig(x,g)
    return [i for i=1:size(g)[1] if g[i](x)==0]
end
```

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

```julia
function LICQ(ξ, g, Ig, h)
    set = sympy.Matrix([
        Matrix([diff(g[i](x), x).subs(x[1], ξ[1]).subs(x[2], ξ[2]) for i ∈ Ig(ξ,g)]')...,
        Matrix([diff(h[i](x), x).subs(x[1], ξ[1]).subs(x[2], ξ[2]) for i ∈ 1:size(h)[1]]')...
    ])'
    return set
end
```

Test LICQ in potential KKT Point

set =

$$\begin{bmatrix} -\frac{2\sqrt{6}}{3} & \frac{2\sqrt{6}}{3} \\ -5 & -10 \end{bmatrix}$$

```julia
set = LICQ(kktpts[1], g, Ig, h)  # check first pt
```

2

```julia
set.rank()  # full rank <=> linearly independent
```

[true, true]

```julia
[LICQ([kktpts[i][1], kktpts[i][2]], g, Ig, h).rank() == findmin(size(LICQ([kktpts[i][1], kktpts[i][2]], g, Ig, h)))[1] for i=1:length(kktpts)]
```

See you next week ✌️

- Questions?