Entendendo Primal, Dual e Sensibilidade no JuMP

📥 Forma Primal vs. Dual

Em otimização, todo problema (o **Primal**) tem um problema "espelho" associado (o **Dual**).

No JuMP, a regra geral é: **Você sempre escreve o problema Primal.**

O Problema Primal (O que você escreve)

- É o problema original que você quer resolver.
- **Exemplo:** *Maximizar* o lucro (função objetivo) sujeito a limitações de recursos como madeira, aço e mão de obra (restrições).
- As variáveis são suas decisões: `x`, `y`.
- A solução é `value(x)` e `value(y)`.

O Problema Dual (O "espelho" calculado)

- É um problema matemático derivado do primal.
- **Exemplo:** Se o primal *maximiza* lucro, o dual *minimiza* o "custo" ou "valor" dos recursos.
- Cada **restrição** no Primal (ex: `c_madeira <= 100`) corresponde a uma **variável dual** no Dual.
- O JuMP permite acessar a *solução* desse problema dual (o valor dessas variáveis duais) sem que você precise escrevê-lo.

Diferença Prática no Código

```
# 1. DEFININDO O MODELO PRIMAL
model = Model(HiGHS.Optimizer)

@variable(model, x >= 0)
@variable(model, y >= 0)

# 'c_madeira' é uma restrição PRIMAL
@constraint(model, c_madeira, 2x + y <= 100)

@objective(model, Max, 40x + 30y)

optimize!(model)

# 2. ACESSANDO AS SOLUÇÕES

# Solução PRIMAL (Quanto produzir de x e y)
primal_x = value(x)
primal_y = value(y)

# Solução DUAL (O valor da restrição 'c_madeira')
dual_madeira = shadow_price(c_madeira)
```

> **Resumo:** Você **define o Primal** (`@variable` , `@constraint`) para obter as **soluções primais** (`value()`). Depois, você pode consultar as **soluções duais** (`shadow_price()`) que foram calculadas pelo solver.

🔬 O que é Análise de Sensibilidade?

A Análise de Sensibilidade estuda como a **solução ótima** (lucro, valores de `x`, `y`) muda quando os **parâmetros** do modelo (limites de recursos, custos, lucros) mudam.

No JuMP, isso é feito principalmente através de duas funções:

1. Preço Sombra (Shadow Price)

- **O que é:** Informa o quanto a sua **função objetivo** (ex: lucro) irá melhorar se você **relaxar uma restrição** em uma unidade.
- **Função no JuMP:** `shadow_price(sua_restricao)`

- **Exemplo Prático:**
  - Você tem a restrição: `@constraint(model, c_madeira, ... <= 100)`
  - Você roda `optimize!(model)` e descobre que:
  - `shadow_price(c_madeira)` é **$30**.
- **Significado:**
  - Isso quer dizer que, se você conseguir +1 unidade de madeira (mudando a restrição para `<= 101`), seu lucro total **aumentará em $30**.
  - Se o preço sombra for **0**, significa que essa restrição não está te limitando (você já tem madeira de sobra).

2. Custo Reduzido (Reduced Cost)

- **O que é:** Focado nas **variáveis** (ex: produtos). Se uma variável é **zero** na solução ótima (ex: o modelo decidiu "não produzir o Produto Z"), o custo reduzido informa quanto o coeficiente dela na função objetivo (ex: seu lucro) precisa **melhorar** para que valha a pena produzi-la.
- **Função no JuMP:** `reduced_cost(sua_variavel)`
- **Exemplo Prático:**
  - Você tem a variável `@variable(model, z >= 0)` e o objetivo é `Max, ... + 40z`.
  - Você roda `optimize!(model)` e descobre que `value(z)` é **0** (o modelo não quis produzir `z`).
  - Você consulta `reduced_cost(z)` e o valor é **-20**.

- **Significado:**
  - O lucro do produto `z` (que era $40) precisa aumentar em $20 (ou seja, ir para $60) antes que o modelo considere produzir `z` (antes que `value(z)` se torne positivo).