

PROGRAMAÇÃO LINEAR

MC558 - Projeto e Análise de
Algoritmos II

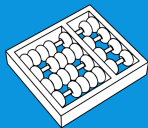
Santiago Valdés Ravelo
[https://ic.unicamp.br/~santiago/
ravelo@unicamp.br](https://ic.unicamp.br/~santiago/ravelo@unicamp.br)

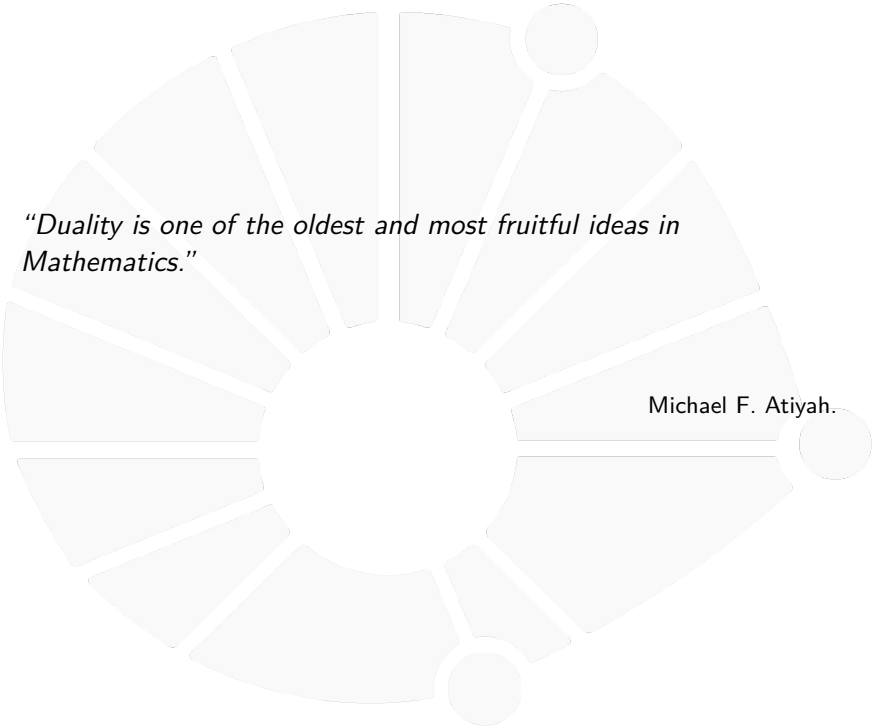
01/24

10



UNICAMP



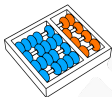


“Duality is one of the oldest and most fruitful ideas in Mathematics.”

Michael F. Atiyah.



PROBLEMA DUAL



Primal

Considere um programa linear **PRIMAL** de minimização (**PLP**(c, A, b)) composto por dois vetores $c \in \mathbb{Q}^n$ e $b \in \mathbb{Q}^m$, e uma matriz $A \in \mathbb{Q}^{m \times n}$ ($m, n \in \mathbb{N}$), formulado como segue:

$$\min \quad c^t x$$

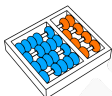
s.a:

$$Ax \geq b$$

$$x_i \geq 0 \quad \forall 1 \leq i \leq n$$

$$x \in \mathbb{Q}^n$$

* Restrições da forma $a_i^t x = b_i$ podem ser escritas como $a_i^t x \geq b_i$ e $a_i^t x \leq b_i$, enquanto as da forma $a_i^t x \geq b_i$ podem ser escritas como $-a_i^t x \leq -b_i$.



Dual

Dado um programa linear primal de minimização **PLP**(c, A, b), o programa linear **DUAL** associado **PLD**(c, A, b) é formulado como segue:

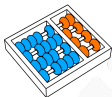
$$\max \quad b^t y$$

s.a:

$$A^t y \leq c$$



$$y_i \geq 0 \quad \forall 1 \leq i \leq m$$

$$y \in \mathbb{Q}^m$$



Exemplo. Almoço

Minimize o total de gorduras em um almoço que consiste em salda e sopa, considerando a seguinte informação nutricional:



	Vitamina A	Vitamina B	Gorduras
Salada	80mcg/100g	0.4mcg/100g	4mg/100g
Sopa	60mcg/100g	0.2mcg/100g	6mg/100g

Os requerimentos nutricionais são: pelo menos **450mcg** de vitamina **A** e **2mcg** de vitamina **B**, além de evitar consumir mais de **700g**.



Exemple. Almoço

Formulação (aula anterior)

$$\min \quad 4x_{\text{salada}} + 6x_{\text{sopa}}$$

s.a :

$$80x_{\text{salada}} + 60x_{\text{sopa}} \geq 450$$

$$0.4x_{\text{salada}} + 0.2x_{\text{sopa}} \geq 2$$

$$x_{\text{salada}} + x_{\text{sopa}} \leq 7$$

$$x_{\text{salada}}, x_{\text{sopa}} \geq 0$$



Exemplo. Almoço

Formulação como *PLP*

$$\min \quad 4x_{\text{salada}} + 6x_{\text{sopa}}$$

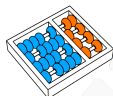
s.a :

$$80x_{\text{salada}} + 60x_{\text{sopa}} \geq 450$$

$$0.4x_{\text{salada}} + 0.2x_{\text{sopa}} \geq 2$$

$$x_{\text{salada}} + x_{\text{sopa}} \leq 7$$

$$x_{\text{salada}}, x_{\text{sopa}} \geq 0$$



Exemplo. Almoço

Formulação como *PLP*

$$\min \quad 4x_{\text{salada}} + 6x_{\text{sopa}}$$

s.a :

$$80x_{\text{salada}} + 60x_{\text{sopa}} \geq 450$$

$$0.4x_{\text{salada}} + 0.2x_{\text{sopa}} \geq 2$$

$$x_{\text{salada}} + x_{\text{sopa}} \leq 7$$

$$x_{\text{salada}}, x_{\text{sopa}} \geq 0$$

$$\min \quad 4x_{\text{salada}} + 6x_{\text{sopa}}$$

s.a :

$$80x_{\text{salada}} + 60x_{\text{sopa}} \geq 450$$

$$0.4x_{\text{salada}} + 0.2x_{\text{sopa}} \geq 2$$

$$-x_{\text{salada}} - x_{\text{sopa}} \geq -7$$

$$x_{\text{salada}}, x_{\text{sopa}} \geq 0$$



Exemplo. Almoço

Primal e dual

$$\min \quad 4x_{\text{salada}} + 6x_{\text{sopa}}$$

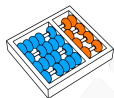
s.a :

$$80x_{\text{salada}} + 60x_{\text{sopa}} \geq 450$$

$$0.4x_{\text{salada}} + 0.2x_{\text{sopa}} \geq 2$$

$$-x_{\text{salada}} - x_{\text{sopa}} \geq -7$$

$$x_{\text{salada}}, x_{\text{sopa}} \geq 0$$



Exemplo. Almoço

Primal e dual

$$\min \quad 4x_{\text{salada}} + 6x_{\text{sopa}}$$

s.a :

$$80x_{\text{salada}} + 60x_{\text{sopa}} \geq 450$$

$$0.4x_{\text{salada}} + 0.2x_{\text{sopa}} \geq 2$$

$$-x_{\text{salada}} - x_{\text{sopa}} \geq -7$$

$$x_{\text{salada}}, x_{\text{sopa}} \geq 0$$

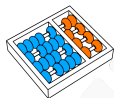
$$\max \quad 450y_A + 6y_B - 7y_{\text{peso}}$$

s.a :

$$80y_A + 0.4y_B - y_{\text{peso}} \leq 4$$

$$60y_A + 0.2y_B - y_{\text{peso}} \leq 6$$

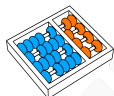
$$y_A, y_B, y_{\text{peso}} \geq 0$$



Exemplo. Almoço

Interpretação dual

Cada variável é relacionada a um nutriente ou a uma medida e podem ser interpretadas como a quantidade de miligramas de gordura por cada unidade do nutriente/medida no almoço:

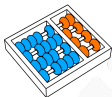


Exemplo. Almoço

Interpretação dual

Cada variável é relacionada a um nutriente ou a uma medida e podem ser interpretadas como a quantidade de miligramas de gordura por cada unidade do nutriente/medida no almoço:

y_A , quantidade de miligramas de gordura por unidade de vitamina **A**.



Exemplo. Almoço

Interpretação dual

Cada variável é relacionada a um nutriente ou a uma medida e podem ser interpretadas como a quantidade de miligramas de gordura por cada unidade do nutriente/medida no almoço:

y_A , quantidade de miligramas de gordura por unidade de vitamina **A**.

y_B , quantidade de miligramas de gordura por unidade de vitamina **B**.



Exemplo. Almoço

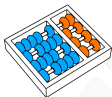
Interpretação dual

Cada variável é relacionada a um nutriente ou a uma medida e podem ser interpretadas como a quantidade de miligramas de gordura por cada unidade do nutriente/medida no almoço:

y_A , quantidade de miligramas de gordura por unidade de vitamina **A**.

y_B , quantidade de miligramas de gordura por unidade de vitamina **B**.

y_{peso} , quantidade de miligramas de gordura por cada **100g** de comida.



Exemplo. Almoço

Interpretação dual

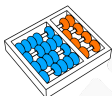
Cada variável é relacionada a um nutriente ou a uma medida e podem ser interpretadas como a quantidade de miligramas de gordura por cada unidade do nutriente/medida no almoço:

y_A , quantidade de miligramas de gordura por unidade de vitamina **A**.

y_B , quantidade de miligramas de gordura por unidade de vitamina **B**.

y_{peso} , quantidade de miligramas de gordura por cada **100g** de comida.

O dual procura uma solução onde a quantidade de gordura por nutriente/medida não seja maior que a quantidade de gordura por prato:



Exemplo. Almoço

Interpretação dual

Cada variável é relacionada a um nutriente ou a uma medida e podem ser interpretadas como a quantidade de miligramas de gordura por cada unidade do nutriente/medida no almoço:

y_A , quantidade de miligramas de gordura por unidade de vitamina **A**.

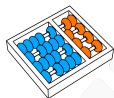
y_B , quantidade de miligramas de gordura por unidade de vitamina **B**.

y_{peso} , quantidade de miligramas de gordura por cada **100g** de comida.

O dual procura uma solução onde a quantidade de gordura por nutriente/medida não seja maior que a quantidade de gordura por prato:

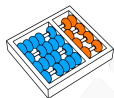
$$80y_A + 0.4y_B - y_{\text{peso}} \leq 4$$

$$60x_A + 0.2y_B - y_{\text{peso}} \leq 6$$



Primal e dual

O dual de **PLP**(c, A, b) é:



Primal e dual

O dual de $\text{PLP}(c, A, b)$ é:

$$\text{PLD}(b, A^t, c) = \text{PLP}(-b, -A^t, -c).$$

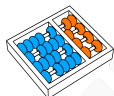


Primal e dual

O dual de $\text{PLP}(c, A, b)$ é:

$$\text{PLD}(b, A^t, c) = \text{PLP}(-b, -A^t, -c).$$

O dual de $\text{PLP}(-b, -A^t, -c)$ é:



Primal e dual

O dual de $\text{PLP}(c, A, b)$ é:

$$\text{PLD}(b, A^t, c) = \text{PLP}(-b, -A^t, -c).$$

O dual de $\text{PLP}(-b, -A^t, -c)$ é:

$$\text{PLD}(-c, (-A^t)^t, -b) = \text{PLD}(-c, -A, -b)$$



Primal e dual

O dual de $\text{PLP}(c, A, b)$ é:

$$\text{PLD}(b, A^t, c) = \text{PLP}(-b, -A^t, -c).$$

O dual de $\text{PLP}(-b, -A^t, -c)$ é:

$$\begin{aligned}\text{PLD}(-c, (-A^t)^t, -b) &= \text{PLD}(-c, -A, -b) \\ &= \text{PLP}(-(-c), -(-A), -(-b))\end{aligned}$$



Primal e dual

O dual de $\text{PLP}(c, A, b)$ é:

$$\text{PLD}(b, A^t, c) = \text{PLP}(-b, -A^t, -c).$$

O dual de $\text{PLP}(-b, -A^t, -c)$ é:

$$\begin{aligned}\text{PLD}(-c, (-A^t)^t, -b) &= \text{PLD}(-c, -A, -b) \\ &= \text{PLP}(-(-c), -(-A), -(-b)) \\ &= \text{PLP}(c, A, b).\end{aligned}$$



Primal e dual

O dual de $\text{PLP}(c, A, b)$ é:

$$\text{PLD}(b, A^t, c) = \text{PLP}(-b, -A^t, -c).$$

O dual de $\text{PLP}(-b, -A^t, -c)$ é:

$$\begin{aligned}\text{PLD}(-c, (-A^t)^t, -b) &= \text{PLD}(-c, -A, -b) \\ &= \text{PLP}(-(-c), -(-A), -(-b)) \\ &= \text{PLP}(c, A, b).\end{aligned}$$

Portanto, o dual do dual é o primal.



TEOREMAS DE DUALIDADE



Dualidade fraca

Teorema (dualidade fraca)

Se x é uma solução viável de $PLP(c, A, b)$ e y é uma solução viável de $DLP(c, A, b)$, então:

$$b^t y \leq c^t x.$$



Prova

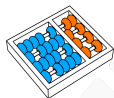
Como y é uma solução viável de $DLP(c, A, b)$ e x é uma solução viável de $PLP(c, A, b)$, temos que:



Prova

Como y é uma solução viável de $DLP(c, A, b)$ e x é uma solução viável de $PLP(c, A, b)$, temos que:

$$A^t y \leq c$$



Prova

Como y é uma solução viável de $DLP(c, A, b)$ e x é uma solução viável de $PLP(c, A, b)$, temos que:

$$A^t y \leq c \quad (x \geq 0)$$



Prova

Como \mathbf{y} é uma solução viável de $\text{DLP}(c, A, b)$ e \mathbf{x} é uma solução viável de $\text{PLP}(c, A, b)$, temos que:

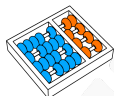
$$\begin{aligned} A^t \mathbf{y} &\leq c & (\mathbf{x} \geq 0) \\ \Rightarrow (A^t \mathbf{y})^t \mathbf{x} &\leq c^t \mathbf{x} \end{aligned}$$



Prova

Como y é uma solução viável de $DLP(c, A, b)$ e x é uma solução viável de $PLP(c, A, b)$, temos que:

$$\begin{aligned} & A^t y \leq c \quad (x \geq 0) \\ \Rightarrow & (A^t y)^t x \leq c^t x \\ \Rightarrow & y A x \leq c^t x \end{aligned}$$



Prova

Como y é uma solução viável de $DLP(c, A, b)$ e x é uma solução viável de $PLP(c, A, b)$, temos que:

$$\begin{aligned} & A^t y \leq c \quad (x \geq 0) \\ \Rightarrow & (A^t y)^t x \leq c^t x \\ \Rightarrow & y A x \leq c^t x \quad (A x \geq b) \end{aligned}$$



Prova

Como \mathbf{y} é uma solução viável de $\text{DLP}(c, A, b)$ e \mathbf{x} é uma solução viável de $\text{PLP}(c, A, b)$, temos que:

$$A^t \mathbf{y} \leq c \quad (\mathbf{x} \geq 0)$$

$$\Rightarrow (A^t \mathbf{y})^t \mathbf{x} \leq c^t \mathbf{x}$$

$$\Rightarrow \mathbf{y} A \mathbf{x} \leq c^t \mathbf{x} \quad (A \mathbf{x} \geq b)$$

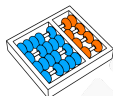
$$\Rightarrow \mathbf{y}^t b \leq c^t \mathbf{x}$$



Prova

Como \mathbf{y} é uma solução viável de $\text{DLP}(c, A, b)$ e \mathbf{x} é uma solução viável de $\text{PLP}(c, A, b)$, temos que:

$$\begin{aligned} & A^t \mathbf{y} \leq c && (\mathbf{x} \geq 0) \\ \Rightarrow & (A^t \mathbf{y})^t \mathbf{x} \leq c^t \mathbf{x} \\ \Rightarrow & \mathbf{y} A \mathbf{x} \leq c^t \mathbf{x} && (A \mathbf{x} \geq b) \\ \Rightarrow & \mathbf{y}^t b \leq c^t \mathbf{x} \\ \Rightarrow & b^t \mathbf{y} \leq c^t \mathbf{x}. \end{aligned}$$



Implicações da dualidade fraca

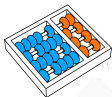
Se **PLP**(c, A, b) e **DPL**(c, A, b) possuem soluções ótimas, então:



Implicações da dualidade fraca

Se $PLP(c, A, b)$ e $DPL(c, A, b)$ possuem soluções ótimas, então:

- ▶ Dada qualquer solução viável y do dual, o valor $b^t y$ é um **LIMITANTE INFERIOR** para o valor ótimo do primal.



Implicações da dualidade fraca

Se $PLP(c, A, b)$ e $DPL(c, A, b)$ possuem soluções ótimas, então:

- ▶ Dada qualquer solução viável y do dual, o valor $b^t y$ é um **LIMITANTE INFERIOR** para o valor ótimo do primal.
- ▶ Dada qualquer solução viável x do primal, o valor $c^t x$ é um **LIMITANTE SUPERIOR** para o valor ótimo do dual.



Dualidade forte

Teorema (dualidade forte)

x^* é uma solução ótima de $PLP(c, A, b)$ se e somente se y^* é uma solução ótima de $DLP(c, A, b)$, onde:

$$c^t x^* = b^t y^*.$$



Ideia da prova

- ▶ Provar que se existe uma solução ótima x^* para $\text{PLP}(c, A, b)$, então existe uma solução viável y para $\text{DLP}(c, A, b)$, tal que:
 $c^t x^* = b^t y$.¹

¹A prova pode ser feita através do método Simplex, também uma alternativa pode ser consultando o artigo “A short note on strong duality: without Simplex and without theorems of alternatives” de Somdeb Lahiri (2017).



Ideia da prova

- ▶ Provar que se existe uma solução ótima x^* para $\text{PLP}(c, A, b)$, então existe uma solução viável y para $\text{DLP}(c, A, b)$, tal que:
 $c^t x^* = b^t y$.¹
- ▶ A dualidade fraca implica que tal y é ótimo de $\text{DLP}(c, A, b)$.

¹A prova pode ser feita através do método Simplex, também uma alternativa pode ser consultando o artigo “A short note on strong duality: without Simplex and without theorems of alternatives” de Somdeb Lahiri (2017).



Ideia da prova

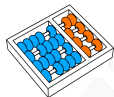
- ▶ Provar que se existe uma solução ótima x^* para **PLP**(c, A, b), então existe uma solução viável y para **DLP**(c, A, b), tal que: $c^t x^* = b^t y$.¹
- ▶ A dualidade fraca implica que tal y é ótimo de **DLP**(c, A, b).
- ▶ Como o dual do dual é o primal, então a outra direção é também válida.

¹A prova pode ser feita através do método Simplex, também uma alternativa pode ser consultando o artigo “A short note on strong duality: without Simplex and without theorems of alternatives” de Somdeb Lahiri (2017).



Implicações da dualidade forte

Dados um programa linear primal e seu dual, existem quatro possibilidades:



Implicações da dualidade forte

Dados um programa linear primal e seu dual, existem quatro possibilidades:

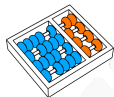
- ▶ O primal e o dual primal são inviáveis.



Implicações da dualidade forte

Dados um programa linear primal e seu dual, existem quatro possibilidades:

- ▶ O primal e o dual primal são inviáveis.
- ▶ O primal é inviável e o dual ilimitado.



Implicações da dualidade forte

Dados um programa linear primal e seu dual, existem quatro possibilidades:

- ▶ O primal e o dual primal são inviáveis.
- ▶ O primal é inviável e o dual ilimitado.
- ▶ O primal é ilimitado e o dual inviável.



Implicações da dualidade forte

Dados um programa linear primal e seu dual, existem quatro possibilidades:

- ▶ O primal e o dual primal são inviáveis.
- ▶ O primal é inviável e o dual ilimitado.
- ▶ O primal é ilimitado e o dual inviável.
- ▶ O primal e o dual são viáveis e o valor de uma solução ótima é o mesmo para os dois.

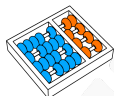


Folgas complementares

Teorema (folgas complementares)

Dadas uma solução ótima x^* de $PLP(c, A, b)$ e uma solução ótima y^* de $DLP(c, A, b)$, temos:

$$(c - A^t y^*)^t x^* = 0 \text{ e } (b - A x^*)^t y^* = 0.$$



Prova

Qualquer par (x, y) de soluções viáveis do primal e o dual satisfazem:



Prova

Qualquer par (x, y) de soluções viáveis do primal e o dual satisfazem:

$$A^t y \leq c$$



Prova

Qualquer par (x, y) de soluções viáveis do primal e o dual satisfazem:

$$A^t y \leq c \quad (x \geq 0)$$



Prova

Qualquer par (x, y) de soluções viáveis do primal e o dual satisfazem:

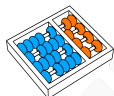
$$\Rightarrow \begin{array}{l} A^t y \leq c \quad (x \geq 0) \\ (A^t y)^t x \leq c^t x \end{array}$$



Prova

Qualquer par (x, y) de soluções viáveis do primal e o dual satisfazem:

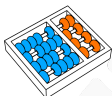
$$\begin{aligned} & A^t y \leq c && (x \geq 0) \\ \Rightarrow & (A^t y)^t x \leq c^t x \\ \Rightarrow & y^t A x \leq c^t x \end{aligned}$$



Prova

Qualquer par (x, y) de soluções viáveis do primal e o dual satisfazem:

$$\begin{aligned} & A^t y \leq c && (x \geq 0) \\ \Rightarrow & (A^t y)^t x \leq c^t x \\ \Rightarrow & y^t A x \leq c^t x && (A x \geq b) \end{aligned}$$



Prova

Qualquer par (x, y) de soluções viáveis do primal e o dual satisfazem:

$$\begin{aligned} & A^t y \leq c && (x \geq 0) \\ \Rightarrow & (A^t y)^t x \leq c^t x \\ \Rightarrow & y^t A x \leq c^t x && (A x \geq b) \\ \Rightarrow & y^t b \leq y^t A x \leq c^t x \end{aligned}$$

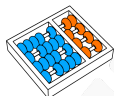


Prova

Qualquer par (x, y) de soluções viáveis do primal e o dual satisfazem:

$$\begin{aligned} & A^t y \leq c \quad (x \geq 0) \\ \Rightarrow & (A^t y)^t x \leq c^t x \\ \Rightarrow & y^t A x \leq c^t x \quad (A x \geq b) \\ \Rightarrow & y^t b \leq y^t A x \leq c^t x \end{aligned}$$

Como (x^*, y^*) são ótimas (pela dualidade forte):



Prova

Qualquer par (\mathbf{x}, \mathbf{y}) de soluções viáveis do primal e o dual satisfazem:

$$\begin{aligned} & A^t \mathbf{y} \leq \mathbf{c} && (\mathbf{x} \geq 0) \\ \Rightarrow & (A^t \mathbf{y})^t \mathbf{x} \leq \mathbf{c}^t \mathbf{x} \\ \Rightarrow & \mathbf{y}^t A \mathbf{x} \leq \mathbf{c}^t \mathbf{x} && (A \mathbf{x} \geq \mathbf{b}) \\ \Rightarrow & \mathbf{y}^t \mathbf{b} \leq \mathbf{y}^t A \mathbf{x} \leq \mathbf{c}^t \mathbf{x} \end{aligned}$$

Como $(\mathbf{x}^*, \mathbf{y}^*)$ são ótimas (pela dualidade forte):

$$\mathbf{y}^{*t} \mathbf{b} = \mathbf{y}^{*t} A \mathbf{x}^* = \mathbf{c}^t \mathbf{x}^*$$



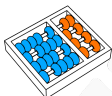
Prova

Qualquer par (x, y) de soluções viáveis do primal e o dual satisfazem:

$$\begin{aligned} & A^t y \leq c && (x \geq 0) \\ \Rightarrow & (A^t y)^t x \leq c^t x \\ \Rightarrow & y^t A x \leq c^t x && (A x \geq b) \\ \Rightarrow & y^t b \leq y^t A x \leq c^t x \end{aligned}$$

Como (x^*, y^*) são ótimas (pela dualidade forte):

$$\begin{aligned} & y^{*t} b = y^{*t} A x^* = c^t x^* \\ y^{*t} b - y^{*t} A x^* &= 0 && 0 = c^t x^* - y^{*t} A x^* \end{aligned}$$



Prova

Qualquer par (\mathbf{x}, \mathbf{y}) de soluções viáveis do primal e o dual satisfazem:

$$\begin{aligned} & A^t \mathbf{y} \leq \mathbf{c} && (\mathbf{x} \geq 0) \\ \Rightarrow & (A^t \mathbf{y})^t \mathbf{x} \leq \mathbf{c}^t \mathbf{x} \\ \Rightarrow & \mathbf{y}^t A \mathbf{x} \leq \mathbf{c}^t \mathbf{x} && (A \mathbf{x} \geq \mathbf{b}) \\ \Rightarrow & \mathbf{y}^t \mathbf{b} \leq \mathbf{y}^t A \mathbf{x} \leq \mathbf{c}^t \mathbf{x} \end{aligned}$$

Como $(\mathbf{x}^*, \mathbf{y}^*)$ são ótimas (pela dualidade forte):

$$\mathbf{y}^{*t} \mathbf{b} = \mathbf{y}^{*t} A \mathbf{x}^* = \mathbf{c}^t \mathbf{x}^*$$

$$\mathbf{y}^{*t} \mathbf{b} - \mathbf{y}^{*t} A \mathbf{x}^* = 0$$

$$(\mathbf{b} - A \mathbf{x}^*)^t \mathbf{y}^* = 0$$

$$0 = \mathbf{c}^t \mathbf{x}^* - \mathbf{y}^{*t} A \mathbf{x}^*$$

$$0 = (\mathbf{c} - A^t \mathbf{y}^*)^t \mathbf{x}^*.$$



Teste de otimalidade via folgas complementares

A solução ($x_{\text{salada}} = 3$, $x_{\text{sopa}} = 4$) é ótima para o problema do almoço?

$$\min \quad 4x_{\text{salada}} + 6x_{\text{sopa}}$$

s.a :

$$80x_{\text{salada}} + 60x_{\text{sopa}} \geq 450$$

$$0.4x_{\text{salada}} + 0.2x_{\text{sopa}} \geq 2$$

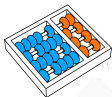
$$-x_{\text{salada}} - x_{\text{sopa}} \geq -7$$

$$x_{\text{salada}}, x_{\text{sopa}} \geq 0$$



Teste de otimalidade via folgas complementares

Dada a solução ($x_{\text{salada}} = 3$, $x_{\text{sopa}} = 4$), para testar sua otimalidade, analisamos as restrições e as variáveis duais:



Teste de otimalidade via folgas complementares

Dada a solução ($x_{\text{salada}} = 3$, $x_{\text{sopa}} = 4$), para testar sua otimalidade, analisamos as restrições e as variáveis duais:

$$80x_{\text{salada}} + 60x_{\text{sopa}} \geq 450 \quad (y_A)$$

$$0.4x_{\text{salada}} + 0.2x_{\text{sopa}} \geq 2 \quad (y_B)$$

$$-x_{\text{salada}} - x_{\text{sopa}} \geq -7 \quad (y_{\text{peso}})$$



Teste de otimalidade via folgas complementares

Dada a solução ($x_{\text{salada}} = 3$, $x_{\text{sopa}} = 4$), para testar sua otimalidade, analisamos as restrições e as variáveis duais:

$$80x_{\text{salada}} + 60x_{\text{sopa}} \geq 450 \quad (y_A)$$

$$0.4x_{\text{salada}} + 0.2x_{\text{sopa}} \geq 2 \quad (y_B)$$

$$-x_{\text{salada}} - x_{\text{sopa}} \geq -7 \quad (y_{\text{peso}})$$

Substituímos os valores ($x_{\text{salada}} = 3$, $x_{\text{sopa}} = 4$) nas restrições:



Teste de otimalidade via folgas complementares

Dada a solução ($x_{\text{salada}} = 3$, $x_{\text{sopa}} = 4$), para testar sua otimalidade, analisamos as restrições e as variáveis duais:

$$80x_{\text{salada}} + 60x_{\text{sopa}} \geq 450 \quad (y_A)$$

$$0.4x_{\text{salada}} + 0.2x_{\text{sopa}} \geq 2 \quad (y_B)$$

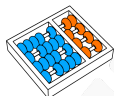
$$-x_{\text{salada}} - x_{\text{sopa}} \geq -7 \quad (y_{\text{peso}})$$

Substituímos os valores ($x_{\text{salada}} = 3$, $x_{\text{sopa}} = 4$) nas restrições:

$$80(3) + 60(4) = 480 > 450 \quad (y_A = 0)$$

$$0.4(3) + 0.2(4) = 2 \quad (y_B \geq 0)$$

$$-3 - 4 = -7 \quad (y_{\text{peso}} \geq 0)$$

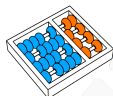


Teste de otimalidade via folgas complementares

$$80(\mathbf{3}) + 60(\mathbf{4}) = 480 > 450 \quad (y_A = 0)$$

$$0.4(\mathbf{3}) + 0.2(\mathbf{4}) = 2 \quad (y_B \geq 0)$$

$$-\mathbf{3} - \mathbf{4} = -7 \quad (y_{\text{peso}} \geq 0)$$



Teste de otimalidade via folgas complementares

$$80(3) + 60(4) = 480 > 450 \quad (y_A = 0)$$

$$0.4(3) + 0.2(4) = 2 \quad (y_B \geq 0)$$

$$-3 - 4 = -7 \quad (y_{\text{peso}} \geq 0)$$

Analisamos as restrições duais para $(y_A = 0, y_B, y_{\text{peso}})$:



Teste de otimalidade via folgas complementares

$$80(\mathbf{3}) + 60(\mathbf{4}) = 480 > 450 \quad (y_A = 0)$$

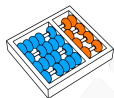
$$0.4(\mathbf{3}) + 0.2(\mathbf{4}) = 2 \quad (y_B \geq 0)$$

$$-\mathbf{3} - \mathbf{4} = -7 \quad (y_{\text{peso}} \geq 0)$$

Analisamos as restrições duais para $(y_A = 0, y_B, y_{\text{peso}})$:

$$80(\mathbf{0}) + 0.4y_B - y_{\text{peso}} \leq 4$$

$$60(\mathbf{0}) + 0.2y_B - y_{\text{peso}} \leq 6$$



Teste de otimalidade via folgas complementares

$$80(\mathbf{3}) + 60(\mathbf{4}) = 480 > 450 \quad (y_A = 0)$$

$$0.4(\mathbf{3}) + 0.2(\mathbf{4}) = 2 \quad (y_B \geq 0)$$

$$-\mathbf{3} - \mathbf{4} = -7 \quad (y_{\text{peso}} \geq 0)$$

Analisamos as restrições duais para $(y_A = 0, y_B, y_{\text{peso}})$:

$$80(\mathbf{0}) + 0.4y_B - y_{\text{peso}} \leq 4$$

$$60(\mathbf{0}) + 0.2y_B - y_{\text{peso}} \leq 6$$

As restrições duais associadas com variáveis primais que não são nulas devem ser satisfeitas na igualdade:



Teste de otimalidade via folgas complementares

$$80(3) + 60(4) = 480 > 450 \quad (y_A = 0)$$

$$0.4(3) + 0.2(4) = 2 \quad (y_B \geq 0)$$

$$-3 - 4 = -7 \quad (y_{\text{peso}} \geq 0)$$

Analisamos as restrições duais para $(y_A = 0, y_B, y_{\text{peso}})$:

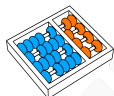
$$80(0) + 0.4y_B - y_{\text{peso}} \leq 4$$

$$60(0) + 0.2y_B - y_{\text{peso}} \leq 6$$

As restrições duais associadas com variáveis primais que não são nulas devem ser satisfeitas na igualdade:

$$80(0) + 0.4y_B - y_{\text{peso}} = 4 \quad (x_{\text{salada}} = 3 \neq 0)$$

$$60(0) + 0.2y_B - y_{\text{peso}} = 6 \quad (x_{\text{sopa}} = 4 \neq 0)$$



Teste de otimalidade via folgas complementares

$$80(\mathbf{0}) + 0.4\mathbf{y}_B - \mathbf{y}_{\text{peso}} = 4 \quad (\mathbf{x}_{\text{salada}} = 3 \neq 0)$$

$$60(\mathbf{0}) + 0.2\mathbf{y}_B - \mathbf{y}_{\text{peso}} = 6 \quad (\mathbf{x}_{\text{sopa}} = 4 \neq 0)$$



Teste de otimalidade via folgas complementares

$$80(\mathbf{0}) + 0.4\mathbf{y}_B - \mathbf{y}_{\text{peso}} = 4 \quad (\mathbf{x}_{\text{salada}} = 3 \neq 0)$$

$$60(\mathbf{0}) + 0.2\mathbf{y}_B - \mathbf{y}_{\text{peso}} = 6 \quad (\mathbf{x}_{\text{sopa}} = 4 \neq 0)$$

A solução do sistema é $(\mathbf{y}_A = 0, \mathbf{y}_B = -10, \mathbf{y}_{\text{peso}} = -8)$ que não é viável.



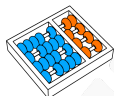
Teste de otimalidade via folgas complementares

$$80(\mathbf{0}) + 0.4\mathbf{y}_B - \mathbf{y}_{\text{peso}} = 4 \quad (\mathbf{x}_{\text{salada}} = 3 \neq 0)$$

$$60(\mathbf{0}) + 0.2\mathbf{y}_B - \mathbf{y}_{\text{peso}} = 6 \quad (\mathbf{x}_{\text{sopa}} = 4 \neq 0)$$

A solução do sistema é $(\mathbf{y}_A = 0, \mathbf{y}_B = -10, \mathbf{y}_{\text{peso}} = -8)$ que não é viável.

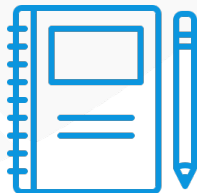
Como não há solução viável \mathbf{y} do dual associada com $(\mathbf{x}_{\text{salada}} = 3, \mathbf{x}_{\text{sopa}} = 4)$, temos que essa solução do primal **NÃO PODE SER ÓTIMA**.



Sobre dualidade



Vamos fazer alguns exercícios?





Exercício 1.

Responda as seguintes questões para cada um dos exercícios da aula anterior: transporte aéreo, barco de carga e comunicação entre servidores:

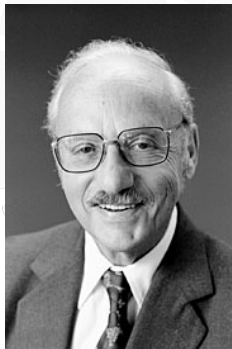
- a) Apresente uma formulação dual e explique o significado das variáveis.
- b) Selecione uma solução viável do primal e teste otimalidade via folgas complementares.



ALGORITMOS PARA PL



Simplex. George Dantzig



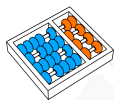
George Bernard Dantzig (8/11/1914–13/5/2005, Estados Unidos)

Pesquisa:

- ▶ Programação linear.
- ▶ Pesquisa operacional.
- ▶ Engenharia industrial.

Prêmios:

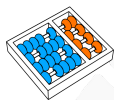
- ▶ **John von Neumann Theory Prize** do *Institute for Operations Research and the Management Sciences* (1975).
- ▶ **National Medal of Science** em *Mathematical, Statistical, and Computational Sciences* (1975).
- ▶ **Harvey Prize** (1985).
- ▶ **Harold Pender Award** (1995).



Simplex. Propriedades

Como as restrições de um programa linear definem um polítopo P como região viável, se existe uma solução ótima para o programa linear, então um dos extremos do polítopo é ótimo.

Ademais, os extremos do polítopo são pontos em que as variáveis não nulas correspondem a colunas linearmente independentes da matriz de restrições. Portanto, se existe um ótimo, então uma solução ótima pode ser obtida considerando uma base da matriz de restrições.



Algoritmo Simplex

Ideia. Começa com uma base B da matriz de restrições A e computa a solução associada (um ponto extremo). A cada iteração, se movimenta a um ponto extremo adjacente somente se o valor objetivo pode melhorar. Se existir uma melhora, então esta é obtida aumentando o valor de uma variável x_i cuja coluna i está fora de B ($x_i = 0$). Nesse caso, substituir uma coluna de B por i .

Complexidade. Para garantir eficiência, o algoritmo mantém uma tabela (*tableau*) que atualiza a cada iteração. Essa tabela permite encontrar a variável x_i e calcular o novo extremo muito rápido. Contudo, no pior caso, o algoritmo precisará enumerar cada extremo, sendo o número de pontos extremos exponencial no número de variáveis do programa linear.



Elipsoides. Leonid Khachiyan



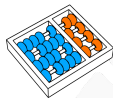
Leonid Genrikhovich Khachiyan (3/5/1952 – 29/4/2005, Rússia)

Pesquisa:

- ▶ Programação linear.
- ▶ Programação matemática.
- ▶ Teoria poliedral.
- ▶ Teoria da complexidade e dos grafos.

Prêmios:

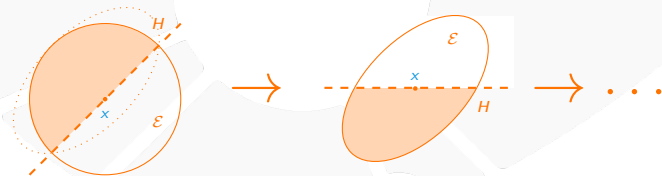
- ▶ **Fulkerson Prize** da *American Mathematical Society* & *Mathematical Programming Society* (1982).



Método de elipsoides para problemas convexos

Considere um **ORÁCULO SEPARADOR**, ou seja, uma função que recebe um conjunto convexo S e um ponto x , indicando se $x \in S$ ou retornando um hiperplano que separa x de S .

Para determinar se um conjunto convexo S é viável, começamos com um elipsoide \mathcal{E} (grande o suficiente) que contenha S e, a cada iteração, perguntamos ao oráculo se o centro do elipsoide x está em S . Se não estiver, considere que H é um hiperplano que separa x de S e atualize \mathcal{E} para ser o elipsoide de menor volume que contém $\mathcal{E} \cap H^+$. Para se $x \in S$ ou se o volume de \mathcal{E} for menor que certo limite.

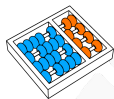




Método dos elipsoides para PL

Foi provado que o método dos elipsoides aplicado a programação linear requer tempo polinomial para solucionar o problema.

Complexidade. $O(n^6 \times \langle I \rangle)$, onde n é o número de variáveis e $\langle I \rangle$ é o tamanho (em bits) da instância.



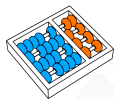
Projeções. Narendra Karmarkar



- ▶ Programação linear.
- ▶ Problemas não-lineares.
- ▶ Arquitetura de computadores.

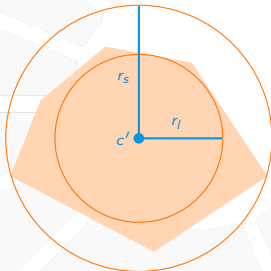
Prêmios:

- ▶ **Frederick W. Lanchester Prize** da *Operations Research Society of America* (1984).
- ▶ **Fulkerson Prize** da *American Mathematical Society & Mathematical Programming Society* (1988).
- ▶ **Paris Kanellakis Award** da *Association for Computing Machinery* (2000).

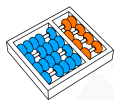


Propriedade

Dados um politopo P e um ponto interior $c \in P$, existe uma transformação para um politopo P' com ponto interior c' , tal que o fator entre o raio (r_s) da menor esfera que contém P' com centro em c' e o raio (r_l) da maior esfera contida em P' com centro em c' é $O(n)$.



$$\frac{r_s}{r_l} = O(n)$$



Algoritmo

Ideia. Aplicar repetidamente transformações, cada uma seguida por uma otimização da esfera inscrita, para obter uma sequência de pontos que converge a uma solução ótima em tempo polinomial.

Complexidade. $O(n^{3.5} \times \langle I \rangle)$, onde n é o número de variáveis e $\langle I \rangle$ o tamanho (em bits) da instância.



Resultados mais rápidos

- ▶ $O((m+n)^{1.5} \times n \times L)$. Pravin M. Vaidya. "Speeding-up linear programming using fast matrix optimization". **FOCS**, 1989.
- ▶ $O((nnz(A) + n^2) \times \sqrt{n} \times L)$. Yin Tat Lee and Aaron Sidford. "Efficient inverse maintenance and faster algorithms for linear programming". **FOCS**, 2015.
- ▶ $O(n^{2.166} \times L)$. Michael B. Cohen, Yin Tat Lee and Zhao Song. "Solving linear programs in the current matrix multiplication time". **STOC**, 2019.
- ▶ $O(n^{2.055} \times L)$. Shunhua Jiang, Zhao Song, Omri Weinstein and Hengjie Zhang. "Faster dynamic matrix inverse for faster LPs". **ArXiv**, 2020.

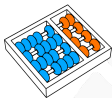


Famílias de algoritmos para PL

- ▶ Troca de base: Simplex, Geração de colunas, Algoritmos criss-cross.
- ▶ Métodos de ponto interior: Elipsoides, Projetivos, Multiplicação de matrizes.



PROGRAMAÇÃO LINEAR
INTEIRA MISTA



Limpendo o planeta

A indústria química **Sem Canudos** desenvolveu uma substância capaz de eliminar o plástico desperdiçado. A quantidade de moléculas que a substância consegue reduzir é três vezes o volume da substância. Contudo, a reação entre o plástico e a substância aumenta a distância entre as moléculas de plástico sete vezes, que implica em um aumento de volume proporcional. Como a reação ocorre em tanques de 20000gal , o total do volume que a reação pode ocupar deve ser limitado por esse espaço. Ademais, a reação é somente bem sucedida se o volume de plástico não for maior que 2000gal a mais do que o volume da substância. Também, por razões de segurança, a diferença entre 16 vezes o volume de plástico menos 10 vezes o da substância deve ser de pelo menos 19000gal .

A indústria não produz menos que 500gal de substância por cada reação e para cada 1000gal de plástico eliminado o lucro é \$50.00, enquanto o custo de produzir 1000gal da substância é \$20.00.

Qual deve ser o volume de plástico e de substância por cada reação para maximizar o lucro?



Limpendo o planeta

x_p , volume (em 1000gal) de plástico a ser usado na reação.



Limpendo o planeta

x_p , volume (em 1000gal) de plástico a ser usado na reação.

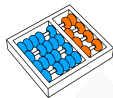
x_s , volume (em 1000gal) de substancia a ser usada na reação.



Limpendo o planeta

x_p , volume (em 1000gal) de plástico a ser usado na reação.

x_s , volume (em 1000gal) de substancia a ser usada na reação.



Limpendo o planeta

x_p , volume (em 1000gal) de plástico a ser usado na reação.

x_s , volume (em 1000gal) de substancia a ser usada na reação.

$$\max \quad 50x_p - 20x_s$$



Limpendo o planeta

x_p , volume (em 1000gal) de plástico a ser usado na reação.

x_s , volume (em 1000gal) de substancia a ser usada na reação.

$$\max \quad 50x_p - 20x_s$$

s.a :



Limpendo o planeta

x_p , volume (em 1000gal) de plástico a ser usado na reação.

x_s , volume (em 1000gal) de substancia a ser usada na reação.

$$\max \quad 50x_p - 20x_s$$

s.a :

$$7x_p - 3x_s \leq 20$$



Limpendo o planeta

x_p , volume (em 1000gal) de plástico a ser usado na reação.

x_s , volume (em 1000gal) de substancia a ser usada na reação.

$$\max \quad 50x_p - 20x_s$$

s.a :

$$7x_p - 3x_s \leq 20$$

$$x_p - x_s \leq 2$$



Limpendo o planeta

x_p , volume (em 1000gal) de plástico a ser usado na reação.

x_s , volume (em 1000gal) de substancia a ser usada na reação.

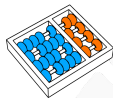
$$\max \quad 50x_p - 20x_s$$

s.a :

$$7x_p - 3x_s \leq 20$$

$$x_p - x_s \leq 2$$

$$16x_p - 10x_s \geq 19$$



Limpendo o planeta

x_p , volume (em 1000gal) de plástico a ser usado na reação.

x_s , volume (em 1000gal) de substancia a ser usada na reação.

$$\max \quad 50x_p - 20x_s$$

s.a :

$$7x_p - 3x_s \leq 20$$

$$x_p - x_s \leq 2$$

$$16x_p - 10x_s \geq 19$$

$$x_s \geq 0.5$$



Limpendo o planeta

x_p , volume (em 1000gal) de plástico a ser usado na reação.

x_s , volume (em 1000gal) de substancia a ser usada na reação.

$$\max \quad 50x_p - 20x_s$$

s.a :

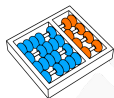
$$7x_p - 3x_s \leq 20$$

$$x_p - x_s \leq 2$$

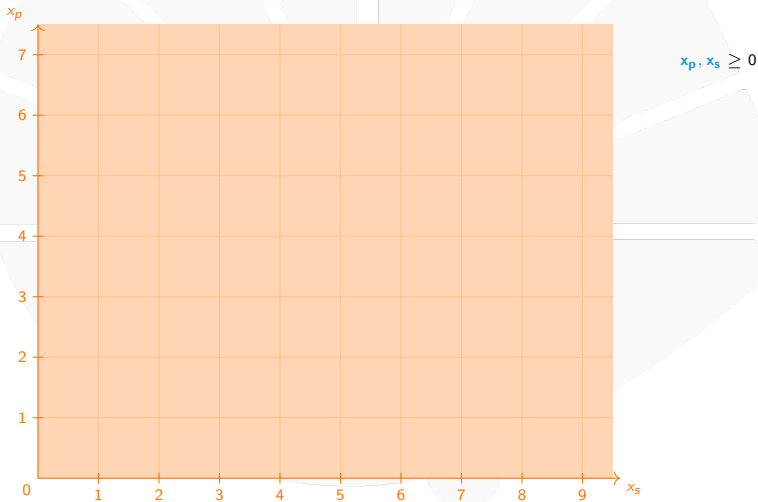
$$16x_p - 10x_s \geq 19$$

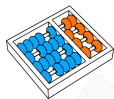
$$x_s \geq 0.5$$

$$x_s, x_p \geq 0$$

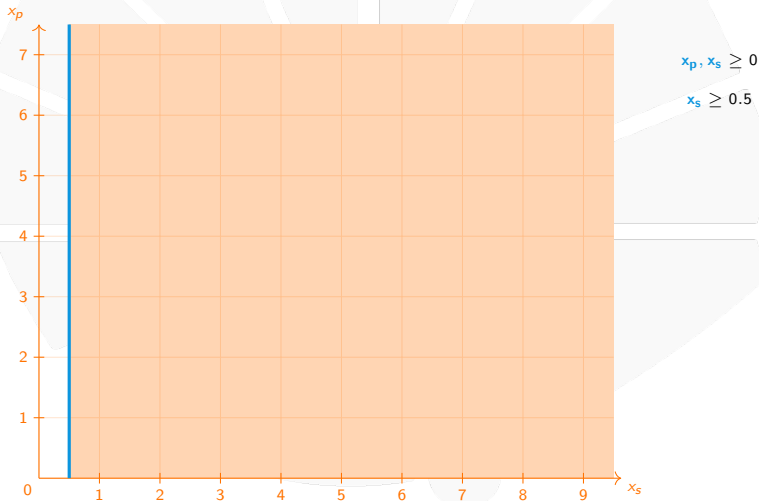


Limpando o planeta. Interpretação gráfica



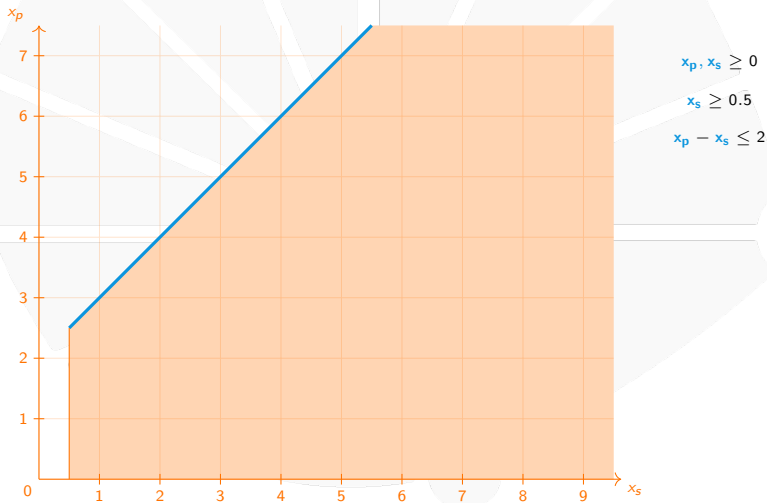


Limpando o planeta. Interpretação gráfica



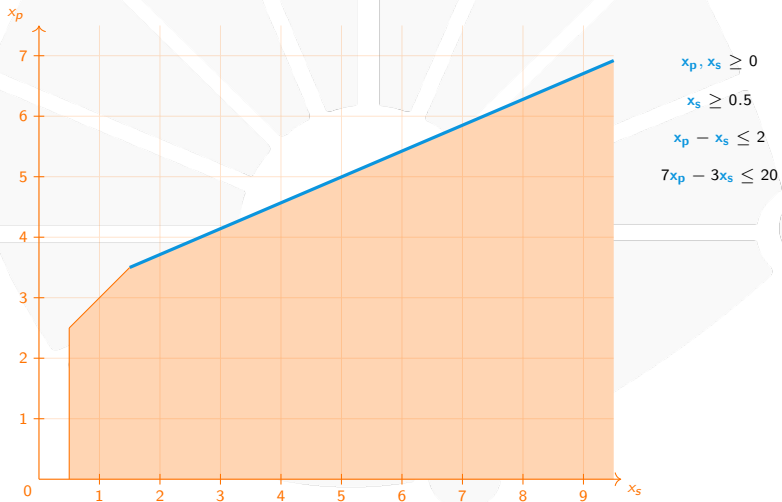


Limpando o planeta. Interpretação gráfica



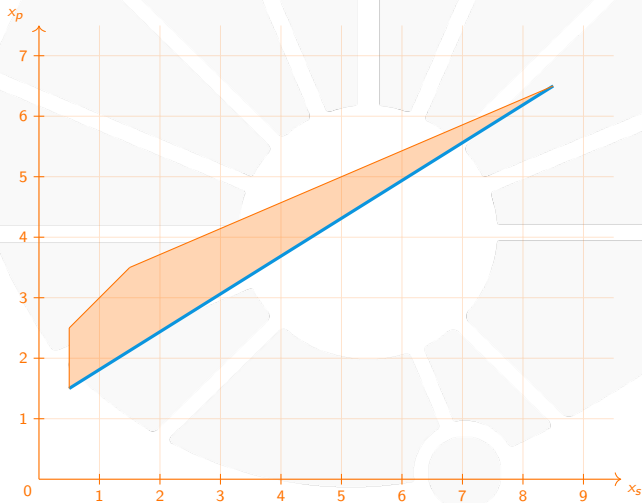


Limpando o planeta. Interpretação gráfica





Limpando o planeta. Interpretação gráfica



$$x_p, x_s \geq 0$$

$$x_s \geq 0.5$$

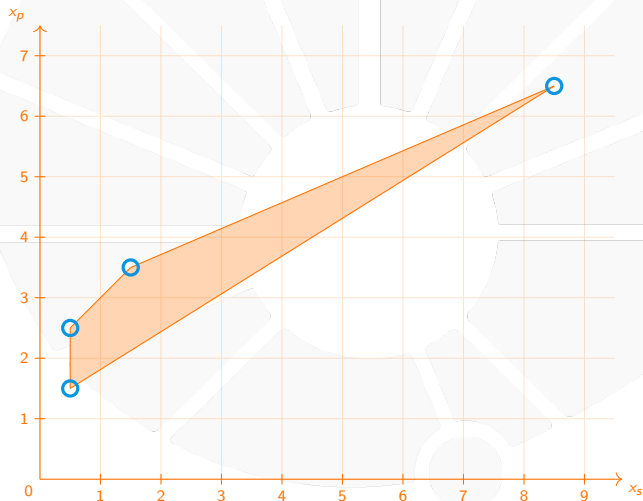
$$x_p - x_s \leq 2$$

$$7x_p - 3x_s \leq 20$$

$$16x_p - 10x_s \geq 19$$



Limpando o planeta. Interpretação gráfica



$$x_p, x_s \geq 0$$

$$x_s \geq 0.5$$

$$x_p - x_s \leq 2$$

$$7x_p - 3x_s \leq 20$$

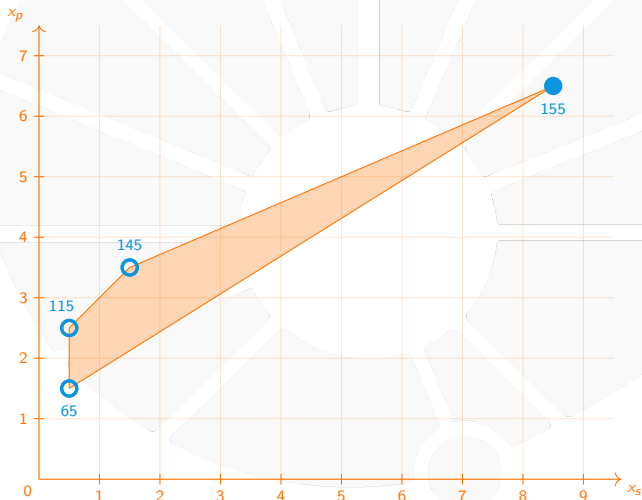
$$16x_p - 10x_s \geq 19$$

Função objetivo:

$$\max \quad 50x_p - 20x_s$$



Limpando o planeta. Interpretação gráfica



$$x_p, x_s \geq 0$$

$$x_s \geq 0.5$$

$$x_p - x_s \leq 2$$

$$7x_p - 3x_s \leq 20$$

$$16x_p - 10x_s \geq 19$$

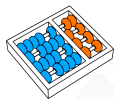
Função objetivo:

$$\max 50x_p - 20x_s$$

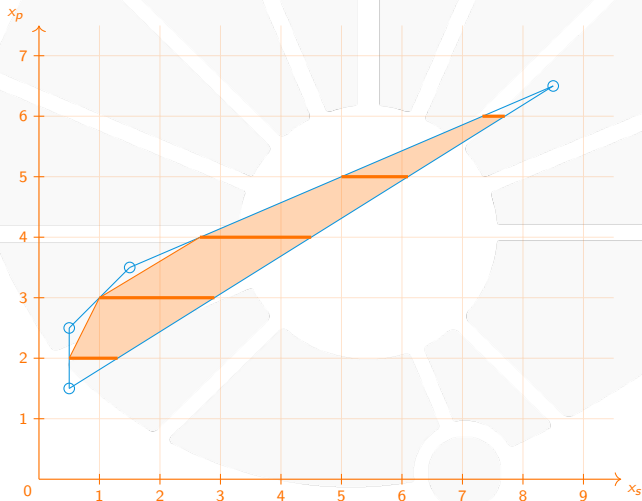


Limpendo o planeta. Restrições de integralidade

Se o plástico foi previamente processado em blocos de volume $1000gal$ cada, que não podem ser divididos, então o que acontece com a região viável do problema?



Limpando o planeta. Interpretação gráfica



$$x_p \in \mathbb{Z}, x_s \in \mathbb{Q}$$

$$x_p, x_s \geq 0$$

$$x_s \geq 0.5$$

$$x_p - x_s \leq 2$$

$$7x_p - 3x_s \leq 20$$

$$16x_p - 10x_s \geq 19$$

Função objetivo:

$$\max 50x_p - 20x_s$$



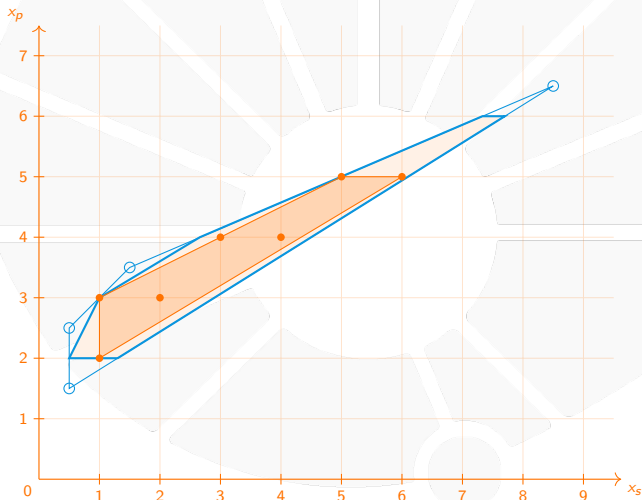
Limpendo o planeta. Restrições de integralidade

Se o plástico foi previamente processado em blocos de volume $1000gal$ cada, que não podem ser divididos, então o que acontece com a região viável do problema?

Que acontece se a substancia é também produzida em contêineres de $1000gal$ e tampouco pode ser dividida?



Limpando o planeta. Interpretação gráfica



$$x_p \in \mathbb{Z}, x_s \in \mathbb{Q}$$

$$x_p, x_s \geq 0$$

$$x_s \geq 0.5$$

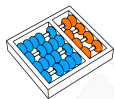
$$x_p - x_s \leq 2$$

$$7x_p - 3x_s \leq 20$$

$$16x_p - 10x_s \geq 19$$

Função objetivo:

$$\max 50x_p - 20x_s$$



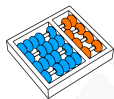
Programação linear inteira mista

$$\min \quad c^t x + d^t y$$

s.a:

$$Ax + By \geq b$$

$$x \in \mathbb{Q}_+^{n_1}, y \in \mathbb{Z}_+^{n_2}$$



Programação linear inteira pura

$$\min \quad c^t x$$

s.a:

$$Ax \geq b$$

$$x \in \mathbb{Z}_+^n$$



Programação linear binária (ou 0-1)

$$\min \quad c^t x$$

s.a:

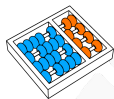
$$Ax \geq b$$

$$x \in \{0, 1\}_+^n$$



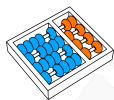
Algumas equivalências

$$\min c^t x \equiv \max -c^t x$$



Algumas equivalências

$$\begin{aligned} \min c^t x &\equiv \max -c^t x \\ Ax \geq b &\equiv -Ax \leq -b \end{aligned}$$

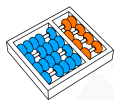


Algumas equivalências

$$\min c^t x \equiv \max -c^t x$$

$$Ax \geq b \equiv -Ax \leq -b$$

$$Ax = b \equiv \begin{cases} Ax \leq b \\ Ax \geq b \end{cases}$$



Algumas equivalências

$$\min c^t x \equiv \max -c^t x$$

$$Ax \geq b \equiv -Ax \leq -b$$

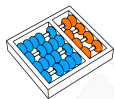
$$Ax = b \equiv \begin{cases} Ax \leq b \\ Ax \geq b \end{cases}$$

$$x_i \in \mathbb{Z} \equiv \begin{cases} x'_i, x''_i \in \mathbb{Z}_+ \\ x_i = x'_i - x''_i \end{cases}$$



Vantagens

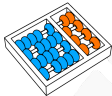
Melhora consideravelmente a capacidade de modelar:



Vantagens

Melhora consideravelmente a capacidade de modelar:

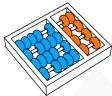
- ▶ Maior flexibilidade na modelagem.



Vantagens

Melhora consideravelmente a capacidade de modelar:

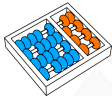
- ▶ Maior flexibilidade na modelagem.
- ▶ Modelos mais realistas.



Vantagens

Melhora consideravelmente a capacidade de modelar:

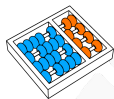
- ▶ Maior flexibilidade na modelagem.
- ▶ Modelos mais realistas.
- ▶ Permite modelar restrições lógicas.



Vantagens

Melhora consideravelmente a capacidade de modelar:

- ▶ Maior flexibilidade na modelagem.
- ▶ Modelos mais realistas.
- ▶ Permite modelar restrições lógicas.
- ▶ Capaz de modelar funções não lineares.



Desvantagens

- ▶ Maior dificuldade para modelar.



Desvantagens

- ▶ Maior dificuldade para modelar.
- ▶ Pode ser muito mais difícil de solucionar.



USO DE RESOLVEDORES EM PYTHON



Programas comerciais

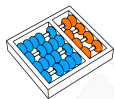
IBM

CPLEX

FICOTM



GUROBI
OPTIMIZATION



Programas livres



[Home](#) / [Browse](#) / [Development](#) / [Algorithms](#) / [Ipsolve](#)



Ipsolve

Mixed Integer Linear Programming (MILP) solver

Brought to you by: [keikland](#), [peno64](#)

GLPK (GNU Linear Programming Kit)





Por que Python?

- ▶ Linguagem de programação muito intuitiva e fácil para programar.



Por que Python?

- ▶ Linguagem de programação muito intuitiva e fácil para programar.
- ▶ Popularidade e constante crescimento na comunidade:



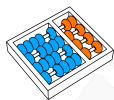
Por que Python?

- ▶ Linguagem de programação muito intuitiva e fácil para programar.
- ▶ Popularidade e constante crescimento na comunidade:
 - ▶ 1^{ra} no índice TIOBE [↗](#).



Por que Python?

- ▶ Linguagem de programação muito intuitiva e fácil para programar.
- ▶ Popularidade e constante crescimento na comunidade:
 - ▶ 1^{ra} no índice TIOBE [↗](#).
 - ▶ 2^{da} no GitHub [↗](#).



Por que Python?

- ▶ Linguagem de programação muito intuitiva e fácil para programar.
- ▶ Popularidade e constante crescimento na comunidade:
 - ▶ 1^{ra} no índice TIOBE [↗](#).
 - ▶ 2^{da} no GitHub [↗](#).
- ▶ Interface comum para diferentes resolvedores de programação linear inteira e mista(e.g., **PuLP/DipPy**):



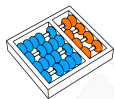
Por que Python?

- ▶ Linguagem de programação muito intuitiva e fácil para programar.
- ▶ Popularidade e constante crescimento na comunidade:
 - ▶ 1^{ra} no índice TIOBE [↗](#).
 - ▶ 2^{da} no GitHub [↗](#).
- ▶ Interface comum para diferentes resolvedores de programação linear inteira e mista(e.g., **PuLP/DipPy**):
 - ▶ Simplifica a codificação de um modelo.



Por que Python?

- ▶ Linguagem de programação muito intuitiva e fácil para programar.
- ▶ Popularidade e constante crescimento na comunidade:
 - ▶ 1^{ra} no índice TIOBE [↗](#).
 - ▶ 2^{da} no GitHub [↗](#).
- ▶ Interface comum para diferentes resolvedores de programação linear inteira e mista (e.g., **PuLP/DipPy**):
 - ▶ Simplifica a codificação de um modelo.
 - ▶ Facilita chamados aos resolvedores durante a execução de algoritmos ou aplicações.



Pulp. Descrição

PuLP é um modelador de programas lineares escrito em Python e permite integrar diferentes resolvedores.



Pulp. Descrição

PuLP é um modelador de programas lineares escrito em Python e permite integrar diferentes resolvedores.

As principais classes para codificar formulações são **LpProblem**, **LpVariable**, **LpConstraint** e **LpConstraintVar**.

As interfaces para os resolvedores são dadas pelas classes **LpSolver** e **LpSolver_CMD**.

Acesse a documentação aqui [🔗](#)



Codificando formulações

Importando **PuLP** com suas classes e funções: `from pulp import *`



Codificando formulações

Importando **PuLP** com suas classes e funções: `from pulp import *`

Definindo as variáveis:

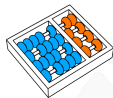


Codificando formulações

Importando **PuLP** com suas classes e funções: `from pulp import *`

Definindo as variáveis:

- ▶ `minhaVariavel = LpVariable(nome, lowBound = None, upBound = None, cat = 'Continuous', e = None)`



Codificando formulações

Importando **PuLP** com suas classes e funções: `from pulp import *`

Definindo as variáveis:

- ▶ `minhaVariavel = LpVariable(nome, lowBound = None, upBound = None, cat = 'Continuous', e = None)`
- ▶ `minhaVariavel = LpVariable.dicts(nome, índices, lowBound = None, upBound = None, cat = 0)`



Codificando formulações

Importando **PuLP** com suas classes e funções: `from pulp import *`

Definindo as variáveis:

- ▶ `minhaVariavel = LpVariable(nome, lowBound = None, upBound = None, cat = 'Continuous', e = None)`
- ▶ `minhaVariavel = LpVariable.dicts(nome, índices, lowBound = None, upBound = None, cat = 0)`

Definindo o problema:



Codificando formulações

Importando **PuLP** com suas classes e funções: `from pulp import *`

Definindo as variáveis:

- ▶ `minhaVariavel = LpVariable(nome, lowBound = None, upBound = None, cat = 'Continuous', e = None)`
- ▶ `minhaVariavel = LpVariable.dicts(nome, índices, lowBound = None, upBound = None, cat = 0)`

Definindo o problema:

- ▶ `meuProblema = LpProblem(nome, LpMinimize)`



Codificando formulações

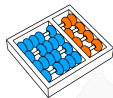
Importando **PuLP** com suas classes e funções: `from pulp import *`

Definindo as variáveis:

- ▶ `minhaVariavel = LpVariable(nome, lowBound = None, upBound = None, cat = 'Continuous', e = None)`
- ▶ `minhaVariavel = LpVariable.dicts(nome, índices, lowBound = None, upBound = None, cat = 0)`

Definindo o problema:

- ▶ `meuProblema = LpProblem(nome, LpMinimize)`
- ▶ `meuProblema = LpProblem(nome, LpMaximize)`



Codificando formulações

Importando **PuLP** com suas classes e funções: `from pulp import *`

Definindo as variáveis:

- ▶ `minhaVariavel = LpVariable(nome, lowBound = None, upBound = None, cat = 'Continuous', e = None)`
- ▶ `minhaVariavel = LpVariable.dicts(nome, índices, lowBound = None, upBound = None, cat = 0)`

Definindo o problema:

- ▶ `meuProblema = LpProblem(nome, LpMinimize)`
- ▶ `meuProblema = LpProblem(nome, LpMaximize)`

Definindo a função objetivo: `meuProblema += expressão, nome`



Codificando formulações

Importando **PuLP** com suas classes e funções: `from pulp import *`

Definindo as variáveis:

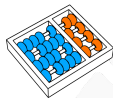
- ▶ `minhaVariavel = LpVariable(nome, lowBound = None, upBound = None, cat = 'Continuous', e = None)`
- ▶ `minhaVariavel = LpVariable.dicts(nome, índices, lowBound = None, upBound = None, cat = 0)`

Definindo o problema:

- ▶ `meuProblema = LpProblem(nome, LpMinimize)`
- ▶ `meuProblema = LpProblem(nome, LpMaximize)`

Definindo a função objetivo: `meuProblema += expressão, nome`

Definindo as restrições:



Codificando formulações

Importando **PuLP** com suas classes e funções: `from pulp import *`

Definindo as variáveis:

- ▶ `minhaVariavel = LpVariable(nome, lowBound = None, upBound = None, cat = 'Continuous', e = None)`
- ▶ `minhaVariavel = LpVariable.dicts(nome, índices, lowBound = None, upBound = None, cat = 0)`

Definindo o problema:

- ▶ `meuProblema = LpProblem(nome, LpMinimize)`
- ▶ `meuProblema = LpProblem(nome, LpMaximize)`

Definindo a função objetivo: `meuProblema += expressão, nome`

Definindo as restrições:

- ▶ `meuProblema += expressão <= valor`



Codificando formulações

Importando **PuLP** com suas classes e funções: `from pulp import *`

Definindo as variáveis:

- ▶ `minhaVariavel = LpVariable(nome, lowBound = None, upBound = None, cat = 'Continuous', e = None)`
- ▶ `minhaVariavel = LpVariable.dicts(nome, índices, lowBound = None, upBound = None, cat = 0)`

Definindo o problema:

- ▶ `meuProblema = LpProblem(nome, LpMinimize)`
- ▶ `meuProblema = LpProblem(nome, LpMaximize)`

Definindo a função objetivo: `meuProblema += expressão, nome`

Definindo as restrições:

- ▶ `meuProblema += expressão <= valor`
- ▶ `meuProblema += expressão == valor`



Codificando formulações

Importando **PuLP** com suas classes e funções: `from pulp import *`

Definindo as variáveis:

- ▶ `minhaVariavel = LpVariable(nome, lowBound = None, upBound = None, cat = 'Continuous', e = None)`
- ▶ `minhaVariavel = LpVariable.dicts(nome, índices, lowBound = None, upBound = None, cat = 0)`

Definindo o problema:

- ▶ `meuProblema = LpProblem(nome, LpMinimize)`
- ▶ `meuProblema = LpProblem(nome, LpMaximize)`

Definindo a função objetivo: `meuProblema += expressão, nome`

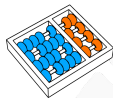
Definindo as restrições:

- ▶ `meuProblema += expressão <= valor`
- ▶ `meuProblema += expressão == valor`
- ▶ `meuProblema += expressão >= valor`



Um exemplo simples

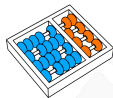
Considere o seguinte problema:



Um exemplo simples

Considere o seguinte problema:

$$\max \quad 3x + 2y$$



Um exemplo simples

Considere o seguinte problema:

$$\max \quad 3x + 2y$$

s.a :



Um exemplo simples

Considere o seguinte problema:

$$\max \quad 3x + 2y$$

s.a :

$$x - y + z = 1$$



Um exemplo simples

Considere o seguinte problema:

$$\max \quad 3x + 2y$$

s.a :

$$x - y + z = 1$$

$$x + 2y \leq 14$$



Um exemplo simples

Considere o seguinte problema:

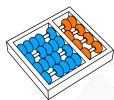
$$\max \quad 3x + 2y$$

s.a :

$$x - y + z = 1$$

$$x + 2y \leq 14$$

$$4x + y \leq 20$$



Um exemplo simples

Considere o seguinte problema:

$$\max \quad 3x + 2y$$

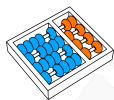
s.a :

$$x - y + z = 1$$

$$x + 2y \leq 14$$

$$4x + y \leq 20$$

$$x, y \in \mathbb{Z}_+, z \in \mathbb{Q}_+$$



Um exemplo simples

$\max \quad 3x + 2y$
 $s.a :$
 $x - y + z = 1$
 $x + 2y \leq 14$
 $4x + y \leq 20$
 $x, y \in \mathbb{Z}_+, z \in \mathbb{Q}_+$

```

1 from pulp import *
2
3 x = LpVariable("x", lowBound = 0, cat = 'Integer')
4 y = LpVariable("y", lowBound = 0, cat = 'Integer')
5 z = LpVariable("z", lowBound = 0)
6
7 problema = LpProblem("ProblemaSimples",
8 ↪ LpMaximize)
9
10 problema += 3 * x + 2 * y, "objetivo"
11
12 problema += x - y + z == 1
13 problema += x + 2 * y <= 14
14 problema += 4 * x + y <= 20
15

```



Exemplo. Mochila binária

Instância: conjunto de objetos O , função de lucro $\nu : O \rightarrow \mathbb{Q}_+$,
função de pesos $\omega : O \rightarrow \mathbb{Q}_+$, peso máximo $W \in \mathbb{Q}_+$.



Exemplo. Mochila binária

Instância: conjunto de objetos O , função de lucro $\nu : O \rightarrow \mathbb{Q}_+$,
função de pesos $\omega : O \rightarrow \mathbb{Q}_+$, peso máximo $W \in \mathbb{Q}_+$.

Formulação:



Exemplo. Mochila binária

Instância: conjunto de objetos O , função de lucro $\nu : O \rightarrow \mathbb{Q}_+$,
função de pesos $\omega : O \rightarrow \mathbb{Q}_+$, peso máximo $W \in \mathbb{Q}_+$.

Formulação:

$$\max \sum_{o \in O} \nu(o) x_o$$



Exemplo. Mochila binária

Instância: conjunto de objetos O , função de lucro $\nu : O \rightarrow \mathbb{Q}_+$,
função de pesos $\omega : O \rightarrow \mathbb{Q}_+$, peso máximo $W \in \mathbb{Q}_+$.

Formulação:

$$\begin{aligned} \max \quad & \sum_{o \in O} \nu(o) x_o \\ \text{s.a :} \quad & \end{aligned}$$



Exemplo. Mochila binária

Instância: conjunto de objetos O , função de lucro $\nu : O \rightarrow \mathbb{Q}_+$, função de pesos $\omega : O \rightarrow \mathbb{Q}_+$, peso máximo $W \in \mathbb{Q}_+$.

Formulação:

$$\max \sum_{o \in O} \nu(o) x_o$$

s.a :

$$\sum_{o \in O} \omega(o) x_o \leq W$$



Exemplo. Mochila binária

Instância: conjunto de objetos O , função de lucro $\nu : O \rightarrow \mathbb{Q}_+$, função de pesos $\omega : O \rightarrow \mathbb{Q}_+$, peso máximo $W \in \mathbb{Q}_+$.

Formulação:

$$\begin{aligned} \max \quad & \sum_{o \in O} \nu(o) x_o \\ \text{s.a :} \quad & \sum_{o \in O} \omega(o) x_o \leq W \\ & x_o \in \{0, 1\}, \forall o \in O \end{aligned}$$



Exemplo. Mochila binária

```
1 from pulp import *
2
3 def mochila(lucros, pesos, W):
4     x = LpVariable.dicts("x", [o for o in range(len(lucros))], lowBound = 0, upBound = 1,
5         ↪ cat='Integer')
6     problema = LpProblem("MochilaBinaria", LpMaximize)
7
8     problema += lpSum(lucros[o] * x[o] for o in range(len(lucros))), "lucro"
9
10    problema += lpSum(pesos[o] * x[o] for o in range(len(lucros))) <= W
11
12
```



Exemplo. Mochila múltipla

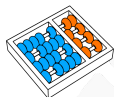
Instância: conjunto de objetos O , função de lucro $\nu : O \rightarrow \mathbb{Q}_+$,
função de pesos $\omega : O \rightarrow \mathbb{Q}_+$, vetor de pesos máximos $W \in \mathbb{Q}_+^m$.



Exemplo. Mochila múltipla

Instância: conjunto de objetos O , função de lucro $\nu : O \rightarrow \mathbb{Q}_+$,
função de pesos $\omega : O \rightarrow \mathbb{Q}_+$, vetor de pesos máximos $W \in \mathbb{Q}_+^m$.

Formulação:



Exemplo. Mochila múltipla

Instância: conjunto de objetos O , função de lucro $\nu : O \rightarrow \mathbb{Q}_+$, função de pesos $\omega : O \rightarrow \mathbb{Q}_+$, vetor de pesos máximos $W \in \mathbb{Q}_+^m$.

Formulação:

$$\max \sum_{o \in O} \sum_{i=1}^m \nu(o) x_{oi}$$



Exemplo. Mochila múltipla

Instância: conjunto de objetos O , função de lucro $\nu : O \rightarrow \mathbb{Q}_+$, função de pesos $\omega : O \rightarrow \mathbb{Q}_+$, vetor de pesos máximos $W \in \mathbb{Q}_+^m$.

Formulação:

$$\begin{aligned} \max \quad & \sum_{o \in O} \sum_{i=1}^m \nu(o) x_{oi} \\ \text{s.a :} \quad & \end{aligned}$$



Exemplo. Mochila múltipla

Instância: conjunto de objetos O , função de lucro $\nu : O \rightarrow \mathbb{Q}_+$, função de pesos $\omega : O \rightarrow \mathbb{Q}_+$, vetor de pesos máximos $W \in \mathbb{Q}_+^m$.

Formulação:

$$\begin{aligned} \max \quad & \sum_{o \in O} \sum_{i=1}^m \nu(o) x_{oi} \\ \text{s.a :} \quad & \sum_{o \in O} \omega(o) x_{oi} \leq W_i \quad \forall 1 \leq i \leq m \end{aligned}$$



Exemplo. Mochila múltipla

Instância: conjunto de objetos O , função de lucro $\nu : O \rightarrow \mathbb{Q}_+$, função de pesos $\omega : O \rightarrow \mathbb{Q}_+$, vetor de pesos máximos $W \in \mathbb{Q}_+^m$.

Formulação:

$$\begin{aligned} \max \quad & \sum_{o \in O} \sum_{i=1}^m \nu(o) x_{oi} \\ \text{s.a :} \quad & \sum_{o \in O} \omega(o) x_{oi} \leq W_i \quad \forall 1 \leq i \leq m \\ & \sum_{i=1}^m x_{oi} \leq 1 \quad \forall o \in O \end{aligned}$$



Exemplo. Mochila múltipla

Instância: conjunto de objetos O , função de lucro $\nu : O \rightarrow \mathbb{Q}_+$, função de pesos $\omega : O \rightarrow \mathbb{Q}_+$, vetor de pesos máximos $W \in \mathbb{Q}_+^m$.

Formulação:

$$\begin{aligned} \max \quad & \sum_{o \in O} \sum_{i=1}^m \nu(o) x_{oi} \\ \text{s.a :} \quad & \sum_{o \in O} \omega(o) x_{oi} \leq W_i \quad \forall 1 \leq i \leq m \\ & \sum_{i=1}^m x_{oi} \leq 1 \quad \forall o \in O \end{aligned}$$

$$x_{oi} \in \{0, 1\}, \forall o \in O, 1 \leq i \leq m$$



Exemplo. Mochila múltipla

```

1  from pulp import *
2
3  def mochilaMultipla(lucros, pesos, W):
4      x = LpVariable.dicts("x", [(o, i) for o in range(len(lucros)) for i in range(len(W))],
5          ↪ lowBound = 0, upBound = 1, cat='Integer')
6
7      problema = LpProblem("MochilaMultipla", LpMaximize)
8
9      problema += lpSum(lucros[o] * x[o, i] for o in range(len(lucros)) for i in
10         ↪ range(len(W)), "lucro")
11
12     for i in range(len(W)):
13         problema += lpSum(pesos[o] * x[o, i] for o in range(len(lucros))) <= W[i]
14
15     for o in range(len(W)):
16         problema += lpSum(x[o, i] for i in range(len(W))) <= 1

```



Solucionando formulações

A função `list_solvers()` retorna os resolvedores disponíveis:

- ▶ e.g., ['GLPK_CMD', 'PYGLPK', 'CPLEX_CMD', 'CPLEX_PY', 'GUROBI', 'GUROBI_CMD', 'XPRESS', 'COIN_CMD', 'SCIP_CMD']

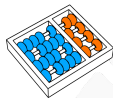


Solucionando formulações

A função `list_solvers()` retorna os resolvedores disponíveis:

- ▶ e.g., `['GLPK_CMD', 'PYGLPK', 'CPLEX_CMD', 'CPLEX_PY', 'GUROBI', 'GUROBI_CMD', 'XPRESS', 'COIN_CMD', 'SCIP_CMD']`

Definindo o resolvedor:



Solucionando formulações

A função `list_solvers()` retorna os resolvedores disponíveis:

- ▶ e.g., ['GLPK_CMD', 'PYGLPK', 'CPLEX_CMD', 'CPLEX_PY', 'GUROBI', 'GUROBI_CMD', 'XPRESS', 'COIN_CMD', 'SCIP_CMD']

Definindo o resolvedor:

- ▶ e.g., `solver = GUROBI(parameters)`.



Solucionando formulações

A função `list_solvers()` retorna os resolvedores disponíveis:

- ▶ e.g., ['GLPK_CMD', 'PYGLPK', 'CPLEX_CMD', 'CPLEX_PY', 'GUROBI', 'GUROBI_CMD', 'XPRESS', 'COIN_CMD', 'SCIP_CMD']

Definindo o resolvedor:

- ▶ e.g., `solver = GUROBI(parameters)`.
- ▶ Alguns parâmetros podem ser incluídos: `timeLimit` in seconds, `Cuts`, `Heuristics` e `Presolve` para indicar, respectivamente, se serão usados: geração de cortes padrão, heurísticas do resolvedor e pré-processamento.



Solucionando formulações

A função `list_solvers()` retorna os resolvedores disponíveis:

- ▶ e.g., `['GLPK_CMD', 'PYGLPK', 'CPLEX_CMD', 'CPLEX_PY', 'GUROBI', 'GUROBI_CMD', 'XPRESS', 'COIN_CMD', 'SCIP_CMD']`

Definindo o resolvedor:

- ▶ e.g., `solver = GUROBI(parameters)`.
- ▶ Alguns parâmetros podem ser incluídos: `timeLimit` in seconds, `Cuts`, `Heuristics` e `Presolve` para indicar, respectivamente, se serão usados: geração de cortes padrão, heurísticas do resolvedor e pré-processamento.

Solucionando o problema:



Solucionando formulações

A função `list_solvers()` retorna os resolvedores disponíveis:

- ▶ e.g., `['GLPK_CMD', 'PYGLPK', 'CPLEX_CMD', 'CPLEX_PY', 'GUROBI', 'GUROBI_CMD', 'XPRESS', 'COIN_CMD', 'SCIP_CMD']`

Definindo o resolvedor:

- ▶ e.g., `solver = GUROBI(parameters)`.
- ▶ Alguns parâmetros podem ser incluídos: `timeLimit` in seconds, `Cuts`, `Heuristics` e `Presolve` para indicar, respectivamente, se serão usados: geração de cortes padrão, heurísticas do resolvedor e pré-processamento.

Solucionando o problema:

- ▶ `meuProblema.solve(solver)`



Solucionando formulações

A função `list_solvers()` retorna os resolvedores disponíveis:

- ▶ e.g., ['GLPK_CMD', 'PYGLPK', 'CPLEX_CMD', 'CPLEX_PY', 'GUROBI', 'GUROBI_CMD', 'XPRESS', 'COIN_CMD', 'SCIP_CMD']

Definindo o resolvedor:

- ▶ e.g., `solver = GUROBI(parameters)`.
- ▶ Alguns parâmetros podem ser incluídos: `timeLimit` in seconds, `Cuts`, `Heuristics` e `Presolve` para indicar, respectivamente, se serão usados: geração de cortes padrão, heurísticas do resolvedor e pré-processamento.

Solucionando o problema:

- ▶ `meuProblema.solve(solver)`
- ▶ `solver.buildSolverModel(meuProblema)`, `solver.callSolver(meuProblema)` e `solver.findSolutionValues(meuProblema)`



Solucionando formulações

A função `list_solvers()` retorna os resolvedores disponíveis:

- ▶ e.g., `['GLPK_CMD', 'PYGLPK', 'CPLEX_CMD', 'CPLEX_PY', 'GUROBI', 'GUROBI_CMD', 'XPRESS', 'COIN_CMD', 'SCIP_CMD']`

Definindo o resolvedor:

- ▶ e.g., `solver = GUROBI(parameters)`.
- ▶ Alguns parâmetros podem ser incluídos: `timeLimit` in seconds, `Cuts`, `Heuristics` e `Presolve` para indicar, respectivamente, se serão usados: geração de cortes padrão, heurísticas do resolvedor e pré-processamento.

Solucionando o problema:

- ▶ `meuProblema.solve(solver)`
- ▶ `solver.buildSolverModel(meuProblema)`, `solver.callSolver(meuProblema)` e `solver.findSolutionValues(meuProblema)`

Obtendo o valor da solução: `value(meuProblema.objective)`.



Solucionando formulações

A função `list_solvers()` retorna os resolvedores disponíveis:

- ▶ e.g., `['GLPK_CMD', 'PYGLPK', 'CPLEX_CMD', 'CPLEX_PY', 'GUROBI', 'GUROBI_CMD', 'XPRESS', 'COIN_CMD', 'SCIP_CMD']`

Definindo o resolvedor:

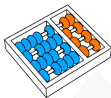
- ▶ e.g., `solver = GUROBI(parameters)`.
- ▶ Alguns parâmetros podem ser incluídos: `timeLimit` in seconds, `Cuts`, `Heuristics` e `Presolve` para indicar, respectivamente, se serão usados: geração de cortes padrão, heurísticas do resolvedor e pré-processamento.

Solucionando o problema:

- ▶ `meuProblema.solve(solver)`
- ▶ `solver.buildSolverModel(meuProblema)`, `solver.callSolver(meuProblema)` e `solver.findSolutionValues(meuProblema)`

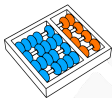
Obtendo o valor da solução: `value(meuProblema.objective)`.

As variáveis são elementos de `meuProblema.variables()` cada um com os atributos `name` e `varValue`.



Um exemplo simples

```
1 from pulp import *
2
3 x = LpVariable("x", lowBound = 0, cat = 'Integer')
4 y = LpVariable("y", lowBound = 0, cat = 'Integer')
5 z = LpVariable("z", lowBound = 0)
6
7 problema = LpProblem("Problema", objetivo)
8
9 problema += 3 * x + 2 * y, "objetivo"
10
11 problema += x - y + z == 1
12 problema += x + 2 * y <= 14
13 problema += 4 * x + y <= 20
14
15 problema.solve(GUROBI_CMD())
16
17 print('Valor otimo: ' + str(value(problema.objective)))
18 print('Solucao otima: ')
19 for variavel in problema.variables():
20     print('      ' + variavel.name + " = " + str(variavel.varValue))
21
```



Exemplo. Mochila binária

```
1 from pulp import *
2
3 def mochila(lucros, pesos, W):
4     x = LpVariable.dicts("x", [o for o in range(len(lucros))], lowBound = 0, upBound = 1,
5         ↪ cat='Integer')
6
7     problema = LpProblem("MochilaBinaria", LpMaximize)
8
9     problema += lpSum(lucros[o] * x[o] for o in range(len(lucros))), "lucro"
10
11    problema += lpSum(pesos[o] * x[o] for o in range(len(lucros))) <= W
12
13    solver = GUROBI(timeLimit = 3600)
14
15    solver.buildSolverModel(problema)
16    solver.callSolver(problema)
17    solver.findSolutionValues(problema)
18
19    print('Valor otimo: ' + str(value(problema.objective)))
20    print('Solucao otima: ')
21    for variavel in problema.variables():
22        print('      ' + variavel.name + " = " + str(variavel.varValue))
23
```

PROGRAMAÇÃO LINEAR

MC558 - Projeto e Análise de
Algoritmos II

Santiago Valdés Ravelo
[https://ic.unicamp.br/~santiago/
ravelo@unicamp.br](https://ic.unicamp.br/~santiago/ravelo@unicamp.br)

01/24

10



UNICAMP

