



Università della Calabria

---

Dipartimento DI Economia, Statistica E Finanza “*Giovanni Anania*”

Corso di Laurea in Statistica per l'azienda

## *Elaborato finale*

Titolo: Programmazione lineare in Python

*Relatore*

Prof. Alfredo Garro

*Candidato*

Nicola Lavecchia  
Matricola 212169

---

Anno Accademico 2021/2022

# Indice:

Introduzione.....	3
1.La Programmazione Lineare.....	4
1.1 Storia della Programmazione Lineare.....	4
1.2 Problemi di Programmazione Lineare.....	5
1.3 Esempi tipici di Programmazione Lineare.....	9
1.4 Metodo grafico per la risoluzione di problemi di Programmazione Lineare.....	13
2. Il metodo del simplesso.....	16
2.1 La forma standard.....	16
2.2 Soluzioni di base.....	18
2.3 L'algoritmo del simplesso.....	19
3. Il risolutore Excel.....	24
4. Programmazione Lineare con Python.....	29
Conclusioni .....	45
Bibliografia e sitografia .....	46

# Introduzione

Nel seguente elaborato si analizza la Programmazione Lineare, parte fondamentale della Ricerca Operativa. Essa si sviluppa come tecnica di risoluzione di particolari problemi in cui si presenta il bisogno di scegliere soluzioni ottimali. Esistono diverse tecniche adatte alla risoluzione di tali problemi, ad esempio tecniche grafiche e tecniche algebriche. Tra queste ultime si colloca il metodo del simplesso che, grazie alla sua efficienza, riuscì a far emergere la Programmazione Lineare. Ad oggi esistono alcuni software che, sia tramite l'algoritmo del simplesso, sia tramite altri metodi, riescono in tempi molto brevi a risolvere problemi, che fino a prima erano lunghi da sviluppare e difficili da svolgere, come il Risolutore di Microsoft Excel e Python. Si approfondirà proprio quest'ultimo facendo notare come è possibile ottenere una soluzione ottimale precisa e dettagliata con poche righe di codice.

La motivazione che mi ha spinto a trattare questa tematica è il duplice interesse in discipline come la Ricerca Operativa data la sua caratteristica adattabilità a molti scenari economici e statistici, e come la programmazione informatica che ha da subito attirato il mio interesse permettendo di semplificare e automatizzare molti dei metodi matematici e statistici analizzati nel corso di questi tre anni.

## **1. La programmazione lineare**

La Ricerca operativa si configura come un approccio scientifico in grado di orientare le decisioni di qualunque soggetto, chiamato a risolvere un problema di scelta, verso la linea più razionale e conveniente.

La Programmazione Lineare è indubbiamente l'argomento centrale fra i vari modelli della Ricerca Operativa, tra tutte, infatti, è la tecnica che viene più ampiamente utilizzata e, non solo si applica a numerosi problemi reali che hanno di per sé una struttura lineare, ma è anche un importante strumento di supporto nell'analisi e nella risoluzione di problemi di programmazione matematica più complessi.

Problema di riferimento: occorre svolgere un certo numero di attività e si dispone di una quantità limitata di risorse, oppure si devono rispettare dei vincoli nelle assegnazioni delle risorse disponibili alle singole attività. Le risorse possedute (siano esse scarse o meno) vanno distribuite tra le varie attività in modo da ottimizzare l'efficienza complessiva.

### **1.1 Storia della programmazione lineare**

Il padre della PL viene comunemente, e giustamente, indicato in George Dantzig che per primo ne ideò, nel 1947, un algoritmo risolutivo (il metodo del Simplex). Tuttavia, alcuni dei concetti fondamentali della programmazione lineare possono essere fatti risalire molto più indietro nel tempo. Già Fourier, nel 1827, aveva studiato come trovare soluzioni ammissibili di un sistema di disuguaglianze lineari; un metodo di calcolo destinato a minimizzare gli errori d'osservazione e dovuto a Vallée Poussin (1910) presenta lati simili al metodo del simplex; infine lavori di von Neumann degli anni venti e trenta, sulla teoria dei giochi e su alcuni modelli economici, sono antecedenti diretti del lavoro di Dantzig. Nel 1939, poi, il matematico sovietico Kantorovich aveva pubblicato (in russo) una monografia sulla programmazione della produzione che anticipa, sotto molti aspetti importanti, i temi trattati da Dantzig. Purtroppo questo lavoro fu a lungo ignorato in Occidente essendo riscoperto solo venti anni dopo, quando la PL aveva già avuto un grande sviluppo. Ancora prima della pubblicazione

dello studio di Kantorovich, Leontief (1932) aveva presentato il suo lavoro fondamentale, in cui si proponeva una struttura matriciale (Interindustry Input-Output model) per lo studio dell'economia americana. Per questo modello Leontief vinse il Premio Nobel per l'Economia nel 1976. La caratteristica di tutti i lavori antecedenti a quelli di Dantzig riguardava uno scarso interesse verso l'applicabilità pratica, dovuta principalmente all'impossibilità di effettuare i calcoli necessari. Il metodo del Simplex, che fu il primo algoritmo pratico per la risoluzione di problemi di programmazione lineare, proposto da Dantzig, si rivelò invece efficiente in pratica e questo, unitamente al simultaneo avvento dei calcolatori elettronici, decretò il successo della Programmazione Lineare e, con esso, l'inizio dello sviluppo rigoglioso della Ricerca Operativa.

## **1.2 Problemi di Programmazione Lineare**

Un problema generale di Programmazione Lineare (PL) ha come scopo l'allocazione ottimale di risorse soggette a vincoli, al fine di soddisfare una data funzione obiettivo

In particolare la metodologia per risolvere un problema di PL si divide in 4 fasi:

1. Individuazione degli obiettivi che si vogliono perseguire e dei fattori che possono influenzare il raggiungimento degli obiettivi prefissati.
2. Costruzione del modello mediante l'uso di variabili decisionali, di una funzione obiettivo e di funzioni di vincolo.
  - a. Le variabili decisionali quantificano le grandezze su cui il decisore può direttamente intervenire.
  - b. La funzione obiettivo esprime una misura dell'efficacia delle decisioni prese e deve essere massimizzata (rispettivamente

minimizzata) se rappresenta un profitto (rispettivamente un costo).

$$f(x) = c_1x_1 + \dots + c_nx_n = \sum_{j=1}^n c_jx_j.$$

- c. Un numero finito di vincoli lineari che, supponendo siano  $m$ , possono essere scritti nella forma:  
vincoli:

$$\begin{array}{ccccccc} a_{11}x_1 + & \dots & + a_{1n}x_n & \geq & b_1 \\ a_{21}x_1 + & \dots & + a_{2n}x_n & \geq & b_2 \\ \vdots & \dots & \vdots & & \vdots \\ a_{m1}x_1 + & \dots & + a_{mn}x_n & \geq & b_m. \end{array}$$

### 3. Analisi del modello:

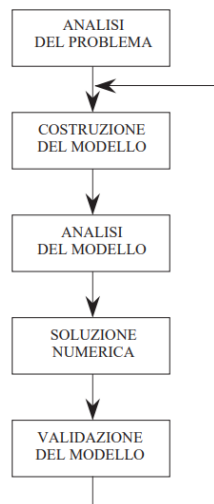
Qualunque sia la forma in cui è presentato un problema di PL questo si fonda sulle seguenti ipotesi:

- Proporzionalità
- Additività
- Continuità

dove per proporzionalità si intende il contributo di una variabile di decisione alla funzione obiettivo e ai vincoli che è proporzionale, secondo una costante moltiplicativa, alla quantità rappresentata dalla variabile stessa; per additività

si intende che il contributo delle variabili di decisione alla funzione obiettivo e ai vincoli è dato dalla somma dei contributi di ogni singola variabile; infine, per continuità si intende che ogni variabile di decisione può assumere tutti i valori reali nell'intervallo di ammissibilità, e quindi le variabili possono assumere valori frazionari.

4. Nella quarta fase vengono definiti opportuni algoritmi di risoluzione. Un algoritmo è una sequenza finita di operazioni elementari tale da fornire la soluzione corretta a ogni istanza del problema in esame (per istanza si intende ogni specifico caso del problema); si cercano algoritmi che siano il più efficienti possibili, infatti i modelli reali hanno dimensioni molto elevate ed è quindi necessario l'uso del calcolatore che con opportuni programmi di calcolo, possa rapidamente fornire una soluzione numerica.
5. Infine si verifica se il modello costruito rappresenta accuratamente la realtà. In caso contrario, si esegue una fase di revisione.



La particolare attenzione dedicata ai modelli di Programmazione Lineare deriva dai numerosi vantaggi che essa presenta e che possono essere così sintetizzati:

1. Generalità e flessibilità.

I modelli di Programmazione Lineare possono descrivere moltissime situazioni reali anche assai diverse tra loro e quindi hanno un carattere di universalità e di adattabilità alle diverse realtà applicative e, anche quando l'ipotesi di linearità non è accettabile, il modello lineare costituisce una buona base di partenza per successive generalizzazioni.

2. Semplicità.

I modelli di Programmazione Lineare sono espressi attraverso il linguaggio dell'algebra lineare e quindi sono facilmente comprensibili anche in assenza di conoscenze matematiche più elevate.

3. Efficienza degli algoritmi risolutivi.

Come accennato in precedenza, i modelli reali hanno dimensioni molto elevate ed è quindi indispensabile l'uso del calcolatore che, con opportuni programmi di calcolo, possa rapidamente fornire una soluzione numerica. Relativamente ai modelli di Programmazione Lineare esistono programmi molto efficienti e largamente diffusi che sono in grado di risolvere rapidamente problemi con migliaia di vincoli e centinaia di migliaia di variabili.

4. Possibilità di analisi qualitative.

I modelli di Programmazione Lineare permettono di ottenere, oltre la soluzione numerica del problema, anche ulteriori informazioni relative alla dipendenza della soluzione da eventuali parametri presenti, che possono avere significative interpretazioni economiche.



### 1.3 Esempi tipici di PL

La necessità dell'uso dei metodi della PL all'interno di molteplici situazioni del mondo reale è stata col passare degli anni sempre più riconosciuta con una sempre maggiore e rapida espansione delle aree di possibile applicazione.

Alcuni esempi di problemi di PL:

- problemi di allocazione ottima di risorse

Si tratta di modelli che considerano il problema di come dividere (allocare) risorse limitate tra varie esigenze in competizione fra di loro. Il generico termine "risorse" può rappresentare, ad esempio, disponibilità di macchinari, materie prime, mano d'opera, energia, tempi macchina, capitali, etc.

- problemi di miscelazione

Nei modelli di allocazione ottima le risorse devono essere ripartite, mentre nei modelli di miscelazione le risorse devono essere combinate tra di loro. I modelli di miscelazione decidono come combinare (miscelare) tali risorse in maniera da soddisfare al meglio determinati obiettivi rispettando opportune richieste.

- problemi di trasporto

Si tratta di problemi in cui si hanno un certo numero di località (origini), ciascuna delle quali ha una quantità fissata di merce disponibile e un certo numero di clienti residenti in altre località (destinazioni) i quali richiedono quantitativi precisi di merce. Quindi, conoscendo il costo unitario del trasporto della merce da ciascuna località origine a ciascuna località destinazione, è necessario pianificare i trasporti, cioè la quantità di merce che deve essere trasportata da ciascuna località origine a ciascuna località destinazione in modo da soddisfare l'ordine dei clienti minimizzando il costo complessivo derivante dai trasporti.

Si riporta ora un esempio di formulazione di uno dei tre problemi sopracitati:

Un colorificio produce due tipi di coloranti **C1** e **C2** utilizzando 3 preparati base in polvere **P1**, **P2**, **P3** che vengono sciolti in acqua. La differente concentrazione dei preparati base dà origine ai due diversi tipi di coloranti. Le quantità (in ettogrammi) di preparati base necessarie per produrre un litro di colorante di ciascuno dei due tipi è riportato nella seguente tabella:

	<b>C1</b>	<b>C2</b>
<b>P1</b>	<i>1</i>	<i>1</i>
<b>P2</b>	<i>1</i>	<i>2</i>
<b>P3</b>	<i>-</i>	<i>1</i>

Ogni giorno la quantità di ciascuno dei preparati base (in ettogrammi) della quale il colorificio può disporre è la seguente:

<b>P1</b>	<b>P2</b>	<b>P3</b>
<i>750</i>	<i>1000</i>	<i>400</i>

Il prezzo di vendita del colorante **C1** è di 7 Euro al litro, mentre il colorante **C2** viene venduto a 10 Euro al litro.

L'obiettivo è determinare la strategia ottimale di produzione giornaliera in modo da massimizzare i ricavi ottenuti dalla vendita dei due coloranti.

### **Formulazione.**

Si vuole costruire il modello di Programmazione Lineare che rappresenti il problema in analisi, considerando le limitazioni date dalle produzioni effettivamente realizzabili. È immediato associare le variabili di decisione ai

quantitativi di coloranti prodotti. Siano, quindi, rispettivamente  $x_1$  e  $x_2$  i quantitativi (in litri) da produrre giornalmente dei due coloranti. Nel formulare il modello di Programmazione Lineare si deve verificare che siano soddisfatte le ipotesi fondamentali:

- proporzionalità.

I consumi dei preparati base e i ricavi ottenibili sono proporzionali ai quantitativi di coloranti prodotti. Ad esempio, per produrre una quantità  $x_2$  di colorante **C2** si consumano  $2x_2$  ettogrammi di **P2** e dalla vendita di  $x_2$  litri di **C2** si ricavano  $10x_2$  Euro indipendentemente dalla quantità prodotta e venduta dell'altro tipo di colorante.

- additività.

I consumi dei preparati base e i ricavi rispettivamente associati alla produzione dei due coloranti sono additivi, nel senso che per produrre  $x_1$  litri di colorante **C1** e  $x_2$  di **C2** si consumano  $x_1 + 2x_2$  ettogrammi di preparato di base **P2** e si ricavano  $7x_1 + 10x_2$  Euro.

- continuità.

Ogni variabile introdotta nel modello può assumere tutti i valori reali nell'intervallo di ammissibilità.

## Variabili

Come già detto, si prendono come variabili di decisione  $x_1$  e  $x_2$ , rispettivamente i quantitativi (in litri) di colorante **C1** e **C2** da produrre giornalmente.

➤ Funzione obiettivo

È rappresentata dal profitto totale che, per le ipotesi fatte, è dato (in Euro) da  
 $Z = 7x_1 + 10x_2$ .

➤ Vincoli

Poiché il consumo di preparati base non può essere superiore alla disponibilità si deve avere:

$$\begin{aligned}x_1 + x_2 &\leq 750 \\x_1 + 2x_2 &\leq 1000 \\x_2 &\leq 400.\end{aligned}$$

Inoltre si deve esplicitare il vincolo di non negatività sulle variabili:

$$x_1 \geq 0, x_2 \geq 0.$$

Quindi la formulazione finale è:

$$\begin{cases} \max (7x_1 + 10x_2) \\ x_1 + x_2 \leq 750 \\ x_1 + 2x_2 \leq 1000 \\ x_2 \leq 400 \\ x_1 \geq 0, x_2 \geq 0. \end{cases}$$

Si espone ora uno dei principali metodi per la risoluzione di problemi di programmazione lineare:

## 1.4 Metodo grafico per la risoluzione di problemi di programmazione lineare

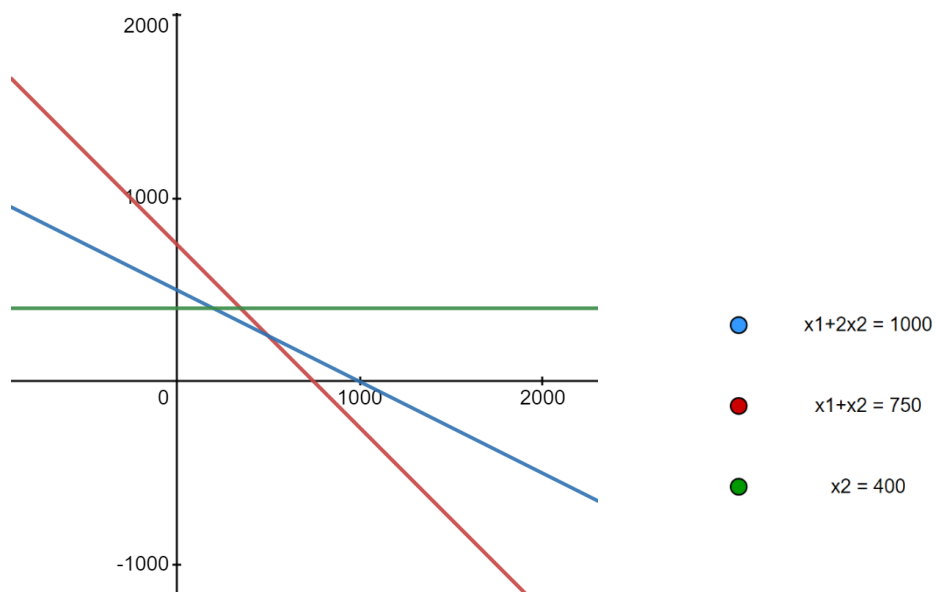
È possibile risolvere i problemi di PL attraverso il metodo grafico, vale a dire attraverso l'iscrizione dei vincoli in forma di rette su un piano cartesiano, l'individuazione della regione ammissibile e la ricerca delle soluzioni ottimali sui vertici della figura che ne esce. Questo metodo risulta molto efficace se usato in problemi che presentano due variabili, all'aumentare delle quali la sua efficacia diminuisce.

Applicazione del metodo grafico all'esempio precedente:

Si riporta la formulazione finale già scritta in precedenza:

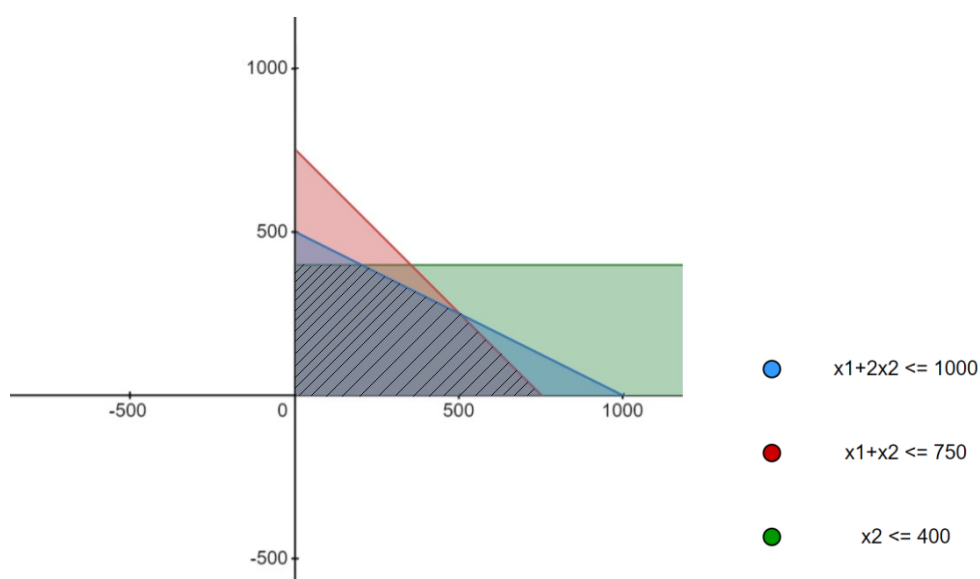
$$\begin{cases} \max (7x_1 + 10x_2) \\ x_1 + x_2 \leq 750 \\ x_1 + 2x_2 \leq 1000 \\ x_2 \leq 400 \\ x_1 \geq 0, x_2 \geq 0. \end{cases}$$

È possibile riscrivere i vincoli sotto forma di equazioni implicite di rette del tipo  $ax_1 + bx_2 = c$  e tracciarle su un piano cartesiano:

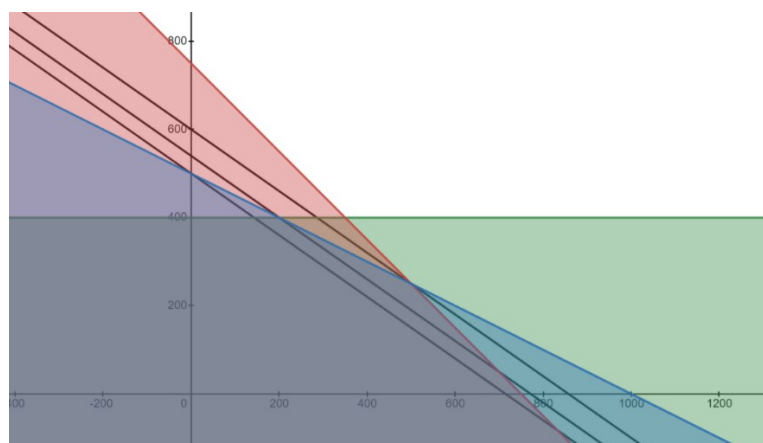


Avendo posto i vincoli di non negatività, si considerano le aree sottostanti alle rette circoscritte solamente al primo quadrante

In tal modo è possibile individuare la regione ammissibile (tratteggiata), ovvero la parte di figura che contiene tutte le soluzioni ammissibili, tra cui la soluzione ottima, avente come lati le rette e come vertici le loro intersezioni.



Infine si tracciano le rette parallele alla retta passante per l'origine ed avente pendenza  $-7/10$  (che si ricavano dall'equazione della funzione obiettivo) e passanti per ogni vertice della regione ammissibile.



Per il Teorema Fondamentale della PL (di cui si inserisce la formulazione di seguito), almeno una soluzione ottima del problema coincide con un vertice del poliedro. Si può allora concludere che tale soluzione coincide con il vertice nel quale passa la retta con termine noto più elevato tra quelle parallele alla retta passante per l'origine e derivante dalla funzione obiettivo.

**Teorema 5.2.1** – TEOREMA FONDAMENTALE DELLA PROGRAMMAZIONE LINEARE

*Si consideri il problema di Programmazione Lineare*

$$\begin{cases} \min c^T x \\ Ax \geq b. \end{cases} \quad (\text{PL})$$

*Supponiamo che il poliedro  $P = \{x \in \mathbb{R}^n \mid Ax \geq b\}$  non contenga rette. Allora una e una sola delle seguenti tre affermazioni è vera:*

- 1. Il problema (PL) è inammissibile, ovvero il poliedro  $P$  è vuoto;*
- 2. Il problema (PL) è illimitato inferiormente;*
- 3. Il problema (PL) ammette soluzioni ottime e almeno una di queste soluzioni è un vertice del poliedro  $P$ .*

Nel caso dell'esempio del colorificio le rette passanti per i vertici del poliedro sono le seguenti:

$$y = -\frac{7}{10}x + 540$$

$$y = -\frac{7}{10}x + 500$$

$$y = -\frac{7}{10}x + 600$$

Scegliendo la retta con termine noto maggiore si nota che questa interseca il poliedro nel punto (500 ; 250), perciò il colorificio massimizzerà il profitto producendo 500 litri di colorante C1 e 250 litri di colorante C2.

## 2. Il metodo del simplesso

Come detto in precedenza, il metodo grafico riportato nel capitolo 1 è un metodo molto utile quando si analizzano problemi che prendono in considerazione due variabili decisionali ma, quando il numero di incognite cresce, diventa difficile da applicare ed è preferibile utilizzare il metodo del simplesso.

Come già accennato, il Metodo del Semplesso è stato il primo algoritmo pratico per la risoluzione di problemi di Programmazione Lineare ed è tuttora il più usato e uno dei più efficienti.

### 2.1 La Forma Standard

Per poter risolvere un problema di PL con questo metodo è necessario che esso sia presentato in forma standard, nella quale:

- la funzione obiettivo deve essere presentata come un problema di minimo

Se si richiede la massimizzazione di un obiettivo lineare, ci si può ricondurre a un problema di minimizzazione semplicemente cambiando segno ai coefficienti della funzione obiettivo.

$$\begin{array}{ll} -\text{Min} & -C^T x \\ \text{s.v.} & \\ & Ax=b \\ & x \geq 0_n \end{array}$$

Il segno “-” prima dell’operatore “Min” indica che, al termine, occorre cambiare segno al valore dell’ottimo trovato (solo al valore di funzione obiettivo corrispondente alla soluzione trovata, non alla soluzione stessa).



- i vincoli strutturali sono espressi in forma di equazioni grazie all'aggiunta di variabili slack

Un vincolo di questo tipo:

$$\sum_{j=1}^n a_{ij} x_j \leq b_i$$

può essere trasformato in un vincolo di uguaglianza introducendo una nuova variabile non negativa detta variabile di slack o di scarto.

Per esempio la variabile  $s(i)$  all'interno del seguente vincolo trasformato da quello precedente:

$$\sum_{j=1}^n a_{ij} x_j + s_i = b_i, \quad s_i \geq 0.$$

Se nel problema compaiono diversi vincoli di disuguaglianza del tipo “ $\leq$ ”, si potrà operare una simile trasformazione per ciascuno di essi, ricordandosi di utilizzare, per ciascun vincolo, una diversa variabile di slack.

- le variabili sono soggette a vincoli di non negatività

Talvolta una variabile decisionale è vincolata ad assumere valori non positivi:  $x(j) \leq 0$ . In questo caso, sostituendo ogni occorrenza della variabile  $x(j)$  con il suo opposto  $y(j) = -x(j)$ , si può scrivere il vincolo di non negatività:  $y(j) \geq 0$ .

Grazie alle trasformazioni appena viste, un problema generico di PL può sempre essere ricondotto a un problema equivalente in forma standard.

## 2.2 Soluzioni di base

È possibile formulare un problema di PL in forma matriciale considerando i vettori  $c, x \in \mathbb{R}^n$ , il vettore  $b \in \mathbb{R}^m$  e la matrice  $A(m, n)$  con  $m < n$  e rango  $r(A) = m$  così da poter scrivere:

$$\begin{cases} \text{Max } Cx \\ Ax = b \\ x \geq 0 \end{cases}$$

Sia  $a_j$  la generica  $j$ -ma colonna di  $A$ ; se, riordinando le colonne, i vettori  $(a_1, a_2, \dots, a_m)$  sono linearmente indipendenti, essi costituiscono una base di  $\mathbb{R}^m$  e il sistema  $Ax = b$  si può riscrivere:

$$x_h = d_h + \sum_{k=1}^{n-m} d_{hk} + x_{m+k} \quad h = 1, 2, \dots, m.$$

Ponendo:

$$x_{m+1} = x_{m+2} = \dots = x_{m+n} = 0$$

Il sistema ha una unica soluzione:

$$(d_1, d_2, \dots, d_m, 0, 0, \dots, 0)$$

la quale, se  $d_h \geq 0 \ \forall h$ , è una soluzione ammissibile ed è detta soluzione di base ammissibile (SBA), ovvero quella soluzione di base avente almeno  $n - m$  elementi nulli e al più  $m$  elementi positivi, dove le prime  $m$  componenti non negative sono dette variabili di base, e le successive  $n - m$  componenti nulle sono dette variabili fuori base.

## 2.3 L'algoritmo del simplesso

Il metodo del Simplexso è composto da 2 fasi:

- Nella Fase I viene controllata l'ammissibilità del problema da risolvere; vengono individuati ed eliminati i vincoli di uguaglianza linearmente dipendenti dagli altri fino a ottenere un sistema di vincoli di uguaglianza descritto da una matrice a rango massimo e viene identificata una soluzione di base ammissibile (SBA).
- Nella Fase II viene risolto il problema di programmazione lineare.

Tale risultato è ottenuto partendo dalla SBA calcolata nella Fase I ed eseguendo i seguenti passaggi:

- si controlla se la SBA soddisfa il criterio di ottimalità;
- si controlla se il problema soddisfa il criterio di illimitatezza;
- se nessuno dei due criteri è soddisfatto, viene determinata una nuova SBA.

Perciò il metodo del simplesso considera un problema in forma standard e necessita di una SBA di partenza, infatti tramite questo metodo si passa iterativamente da quest'ultima ad una soluzione adiacente che permette di migliorare il valore corrente della funzione obiettivo fino al raggiungimento dell'ottimo, oppure fino a determinare il problema come illimitato.

Si analizza ora il metodo del simplesso in forma tabellare tramite il seguente esempio:

$$\begin{array}{rcll} \max & 130x_1 & + & 100x_2 \\ & 1,5x_1 & + & x_2 \leq 27 \\ & x_1 & + & x_2 \leq 21 \\ & 0,3x_1 & + & 0,5x_2 \leq 9 \\ & x_1 & , & x_2 \geq 0 \end{array}$$

Si trasforma successivamente il problema in forma standard:

$$\begin{array}{rcll} \min z = & -130x_1 & - & 100x_2 \\ & 1,5x_1 & + & x_2 + s_1 = 27 \\ & x_1 & + & x_2 + s_2 = 21 \\ & 0,3x_1 & + & 0,5x_2 + s_3 = 9 \\ & x_1 & , & x_2 , s_1 , s_2 , s_3 \geq 0 \end{array}$$

Da variabili, coefficienti e termini noti dell'esempio sopra si può costruire il tableau contenenti i dati del problema:

	TN	X1	X2	S1	S2	S3
-Z	0	-130	-100	0	0	0
S1	27	1.5	1	1	0	0
S2	21	1	1	0	1	0
S3	9	0.3	0.5	0	0	1

Si noti che:

- nella prima colonna sono presenti le variabili che vengono considerate in base cioè  $s_1, s_2, s_3$ ; non compaiono le variabili  $x_1, x_2$  che vengono considerate fuori base e alle quali si assegna il valore 0

- la seconda colonna contiene i termini noti
- la funzione obiettivo viene trascritta invertita di segno, perciò compariranno termini con segno negativo; si arriverà alla conclusione del metodo tabellare quando nella riga Z non compariranno coefficienti negativi.

Nel tableau sono state già inserite le variabili slack e quindi per la prima iterazione la SBA è (0, 0, 27, 21, 9) e la funzione obiettivo dà come risultato 0, non essendo presente in essa nessuna variabile di base.

In questo caso sia  $x_1$  che  $x_2$  hanno segno negativo; si dovrà, dunque, cercare una SBA adiacente e verificarne l'ottimalità.

Algoritmo:

Si rende necessario, quindi, inserire come variabile entrante una variabile in  $x$  nella tabella e far uscire una variabile di base. Per far ciò si deve individuare la variabile uscente con il test del minimo rapporto attraverso tre passaggi:

1. si individua la colonna pivot, ovvero la colonna della variabile entrante;
2. si calcola il rapporto tra i termini noti e i coefficienti positivi presenti nella colonna pivot;
3. si individua la variabile uscente, ovvero quella a cui corrisponde il rapporto più basso. La riga che corrisponde alla variabile uscente è detta riga pivot e l'elemento che appartiene contemporaneamente alla riga e alla colonna pivot è detto coefficiente pivot.

	TN	X1	X2	S1	S2	S3	
-Z	0	-130	-100	0	0	0	
S1	27	1.5	1	1	0	0	$27 / 1.5 = 18$
S2	21	1	1	0	1	0	$21 / 1 = 21$
S3	9	0.3	0.5	0	0	1	$9 / 0.3 = 30$

Si è perciò evidenziata la colonna di pivot, scelta poiché corrisponde al coefficiente minore della funzione obiettivo.

Si sono poi calcolati i rapporti tra TN e coefficienti della colonna di pivot

Si è evidenziato in rosso il termine di pivot, in questo caso pari a 1.5, che corrisponde all'incrocio tra colonna di pivot e riga con il rapporto minore tra i precedentemente calcolati, cioè la riga  $s_1$ .

Dopo aver trovato il pivot è necessario annullare tutti gli elementi della colonna di pivot ad eccezione del pivot stesso, per fare ciò innanzitutto si divide la riga di pivot per il termine di pivot tralasciando gli zeri:

$$27/1.5 = 18$$

$$1.5/1.5 = 1$$

$$1/1.5 = 2/3$$

$$1/1.5 = 2/3$$

Inoltre, dopo le operazioni appena descritte, la variabile  $x_1$  entra in base al posto di  $s_1$ .

Perciò la tabella diventa:

	TN	X1	X2	S1	S2	S3
-Z	0	-130	-100	0	0	0
X1	18	1	2/3	2/3	0	0
S2	21	1	1	0	1	0
S3	9	0.3	0.5	0	0	1

Per continuare ad annullare gli elementi nella colonna di pivot si usa l'eliminazione di Gauss:

$$\text{Riga}(Z) = 130 \text{ R}(X1) + \text{R}(Z)$$

$$\text{Riga}(S2) = -\text{R}(X1) + \text{R}(S2)$$

$$\text{Riga}(S3) = -0.3\text{R}(X1) + \text{R}(S3)$$

Dopo queste operazioni il tableau è così formato:

	TN	X1	X2	S1	S2	S3
-Z	2340	0	-40/3	260/3	0	0
X1	18	1	2/3	2/3	0	0
S2	3	0	1/3	-2/3	1	0
S3	3.6	0	0.3	-0.2	0	1

Al termine della prima iterazione dell'algoritmo si trova un solo termine negativo nella riga Z della funzione obiettivo ed è necessaria una nuova iterazione:

Questa volta il pivot è dato dall'incrocio della riga S2(variabile uscente) con la colonna X2(Variabile entrante), si procede nuovamente ad annullare gli altri valori contenuti sulla colonna di pivot e infine si giunge al tableau finale:

	TN	X1	X2	S1	S2	S3
-Z	2460	0	0	60	40	0
X1	12	1	0	2	-2	0
X2	9	0	1	-2	3	0
S3	0.9	0	0	-0.4	-0.9	1

Ora tutti i valori della riga Z della funzione obiettivo sono positivi perciò si conclude l'algoritmo e la soluzione ottima è data da:

$$X1 = 12 \quad X2 = 9 \quad S1, S2 = 0 \quad S3 = 0.9$$

Con  $Z = 2460$  che è soluzione ottima.

### 3. Il risolutore Excel

Il Risolutore è un componente aggiuntivo di Excel che può essere usato per eseguire analisi di simulazione. Si usa il Risolutore per trovare un valore ottimale (massimo o minimo) per una formula contenuta in una cella, denominata cella obiettivo, soggetta ai vincoli, o limiti, dei valori di altre celle di formule di un foglio di lavoro. Il Risolutore usa un gruppo di celle denominate variabili di decisione o semplicemente celle variabili, che contribuiscono al calcolo delle formule nelle celle obiettivo e nelle celle vincolo. Il Risolutore modifica inoltre i valori nelle celle variabili di decisione in modo da soddisfare i limiti nelle celle vincolo e produrre i risultati desiderati per la cella obiettivo.

Tra i vantaggi del Risolutore Excel si distinguono:

- facile integrazione con l'uso dei fogli di lavoro Excel;
- precisione nel calcolo delle formule.

Tra i limiti del risolutore vi sono:

- problemi di programmazione lineare con dati di media o grande dimensione, per via del limite di al più di 200 numero di celle variabili;
- difficoltà nella convergenza all'ottimo.

Si svolge ora il seguente esempio tramite il risolutore Excel:

Abbiamo un negozio fittizio che vende tre tipi di droni (drone 1, drone 2, drone 3). Questi droni hanno un profitto unitario differente (100€, 400€, 50€). Questi droni, però, hanno anche un costo diverso, inoltre occupano spazio in magazzino (in maniera differente). Quindi per poterli comprare abbiamo due **vincoli** (1) il costo e (2) lo spazio. Mentre per rivenderli ci interessa il profitto che diventa l'obiettivo da massimizzare, perciò:



Funzione obiettivo:  $\max 100x_1 + 400x_2 + 50x_3$

Vincoli: 
$$\begin{cases} 320x_1 + 1300x_2 + 120x_3 \leq 100.000 \\ 0,5x_1 + x_2 + 0,4x_3 \leq 256 \end{cases}$$

Si riportano i dati su un foglio Excel:


	A	B	C	D	E	F	G	H	
1									
2				Drone 1	Drone 2	Drone 3			
3			Profitto	€ 100	€ 400	€ 50	Usati	Disponibili	
4			Costo	€ 320	€ 1.300	€ 120	0	€ 100.000	
5			Spazio	0,5	1	0,4	0	256	
6								Profitto	
7			Unità	0	0	0		0	

All'interno delle celle relative all'unità è possibile lasciare gli zeri in quanto sarà il risolutore a dirci quali sono le quantità che massimizzano il profitto.

Per quanto riguarda le celle di denaro usato (costi), spazio usato e costi è necessario impostare una formula:


La formula che si utilizza è la somma tra i prodotti di matrici, per calcolare il profitto si sommano i prodotti tra vettore del profitto e vettore delle unità, per il costo si sostituisce il vettore dei costi a quello del profitto e la stessa cosa avviene per lo spazio utilizzato.

Profitto:

 =MATR.SOMMA.PRODOTTO(D3:F3;D7:F7)


	C	D	E	F	G	H
		Drone 1	Drone 2	Drone 3		
Profitto		€ 100	€ 400	€ 50	Usati	Disponibili
Costo		€ 320	€ 1.300	€ 120	0	€ 100.000
Spazio		0,5	1	0,4	0	256
Unità		0	0	0		Profitto F7)

Costi (denaro usato):

 =MATR.SOMMA.PRODOTTO(D4:F4;D7:F7)

	C	D	E	F	G	H	I
		Drone 1	Drone 2	Drone 3			
Profitto		€ 100	€ 400	€ 50	Usati	Disponibili	
Costo		€ 320	€ 1.300	€ 120	D7:F7)	€ 100.000	
Spazio		0,5	1	0,4	0	256	
Unità		0	0	0		Profitto 0	

Spazio usato:

 =MATR.SOMMA.PRODOTTO(D5:F5;D7:F7)

	C	D	E	F	G	H
		Drone 1	Drone 2	Drone 3		
Profitto		€ 100	€ 400	€ 50	Usati	Disponibili
Costo		€ 320	€ 1.300	€ 120	0	€ 100.000
Spazio		0,5	1	0,4	D7:F7)	256
Unità		0	0	0		Profitto 0

Si ricorre ora al risolutore per verificare quali siano le quantità che massimizzano la funzione obiettivo dati i vincoli.

Per farlo si seleziona la cella di cui si desidera il risultato, in questo caso il profitto, e nella scheda “Dati” di Excel si accede al risolutore.

Nella schermata successiva si dichiara cosa si intende fare con la funzione obiettivo, in questo caso si sceglie di massimizzarla, si impostano le celle che si devono modificare al fine del risultato cioè le unità e si inseriscono i vincoli:

The screenshot shows the Excel Solver Parameters dialog box. The objective cell is set to \$H\$7, and the 'Max' option is selected. The variable cells are \$D\$7:\$F\$7. The constraints are \$G\$4 <= \$H\$4 and \$G\$5 <= \$H\$5. The 'Make non-negative variables without constraints' checkbox is checked. The 'Simplex LP' method is selected. The 'Solve' button is highlighted.

	Drone 1	Drone 2	Drone 3	Usati	Disponibili
Profitto	€ 100	€ 400	€ 50	0	€ 100.000
Costo	€ 320	€ 1.300	€ 120	0	€ 100.000
Spazio	0,5	1	0,4	0	256
Unità	0	0	0	0	0

Nella precedente finestra si sbarra l’opzione che rende non negative le variabili senza vincoli e si sceglie un metodo di risoluzione, in questo caso si sceglie “Simplex LP”, ovvero il metodo del simplesso tramite il quale Excel individua una soluzione ottima globale; ultimo passo si clicca sul pulsante Risolvi e si visionano i risultati:

Formula bar:  $\text{=MATR.SOMMA.PRODOTTO(D3:F3;D7:F7)}$

	Drone 1	Drone 2	Drone 3	Usati	Disponibili
Profitto	€ 100	€ 400	€ 50	100000	€ 100.000
Costo	€ 320	€ 1.300	€ 120	256	256
Spazio	0,5	1	0,4		
Unità	0	23,2	582		Profitto 38380

**Risultati Risolutore**

È stata trovata una soluzione. Tutti i vincoli e le condizioni di ottimizzazione sono stati soddisfatti.

☒ Mantieni soluzione del Risolutore

☐ Ripristina valori originali

☐ Torna alla finestra di dialogo parametri Risolutore

☐ Rapporti struttura

OK Annulla Salva scenario...

È stata trovata una soluzione. Tutti i vincoli e le condizioni di ottimizzazione sono stati soddisfatti.  
Se si utilizza il motore GRG, è stata individuata almeno una soluzione ottimale locale. Se si utilizza Simplex LP, è stata trovata una soluzione ottimale globale.

Dopo aver trovato la soluzione, cioè le quantità che massimizzano il profitto dati i vincoli, è possibile creare un rapporto sbarrando la casella “Rapporti struttura” e selezionando il rapporto di cui si ha bisogno.

Esempio rapporto valori:

A

B

C

D

E

F

G

H

I

J

1

Microsoft Excel 16.0 Rapporto valori

2

Foglio di lavoro: [Cartel1]Foglio1

3

Data creazione rapporto: 01/12/2022 00:59:17

4

Risultato: È stata trovata una soluzione. Tutti i vincoli e le condizioni di ottimizzazione sono stati soddisfatti.

5

Motore Risolutore

9

Opzioni Risolutore

12

13

14

Cella obiettivo (Max)

15

Cella	Nome	Valore originale	Valore finale
\$H\$7	Unità Profitto	0	38380

16

17

18

19

Celle variabili

20

Cella	Nome	Valore originale	Valore finale	Intere
\$D\$7:\$F\$7				

21

25

26

27

28

Vincoli

29

Cella	Nome	Valore della cella	Formula	Stato	Tolleranza
\$G\$4	Costo Usati	100000	\$G\$4<=\$H\$4	Vincolante	0
\$G\$5	Spazio Usati	256	\$G\$5<=\$H\$5	Vincolante	0

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

Rapporto valori 1

Foglio1

+

#### 4. Programmazione Lineare con Python

Python è un linguaggio di programmazione potente e facile da imparare. Ha efficienti strutture dati di alto livello e un approccio semplice ma efficace alla programmazione orientata agli oggetti. L'elegante sintassi e la tipizzazione dinamica di Python, insieme alla sua natura interpretata, ne fanno un linguaggio ideale per lo scripting e lo sviluppo rapido di applicazioni in molte aree sulla maggior parte delle piattaforme data la presenza di numerose librerie.

Grazie alle suddette librerie, Python può essere usato come supporto in ogni disciplina. In particolare, con una di queste librerie è in grado di risolvere problemi di Programmazione lineare.

La libreria che si usa in questi casi è PuLP che è un modellatore PL scritto in Python. PuLP può generare file MPS o LP e chiamare GLPK, COIN-OR CLP/ CBC, CPLEX, GUROBI, MOSEK, XPRESS, CHOCO, MIPCL, HiGHS, SCIP / FSCIP per risolvere problemi lineari.

Si espongono alcune classi usate dalla libreria PuLP e di seguito alcuni suoi utilizzi al fine di creare un problema di PL:

- LpProblem : classe contenitore per un problema di PL
- LpVariable: variabili che vengono aggiunte ai vincoli nella PL
- LpConstraint: un vincolo dalla forma generale:  $a_1x_1 + a_2x_2 \dots a_nx_n (<=, =, >=) b$

Utilizzare LpVariable() per creare nuove variabili:

Per creare una variabile  $0 \leq x \leq 3$  oppure  $0 \leq y \leq 1$  :

```
x = pulp.LpVariable("x",0,3)
```

```
y = pulp.LpVariable("y",0,1)
```

Su LpVariable è possibile impostare quattro parametri, il primo è il nome arbitrario di ciò che questa variabile rappresenta, il secondo è il limite inferiore su questa variabile, il terzo è il limite superiore e il quarto è essenzialmente il tipo di dati (discreto o continuo). I limiti possono essere immessi direttamente come numero o in alternativa si può impostare

“None” per fare in modo che la variabile non abbia un limite superiore o inferiore. Le opzioni per il quarto parametro sono LpInteger oppure LpContinuous, quest’ultimo è il valore predefinito.

Si noti che mettendo lo zero come limite inferiore, si impostano implicitamente i vincoli di non negatività.

Utilizzare LpProblem() per creare nuovi problemi:

```
problema = pulp.LpProblem (“problema1”, LpMinimize)
```

il secondo parametro indica il tipo di problema che si vuole risolvere e potrebbe essere LpMinimize o LpMaximize, rispettivamente se è necessario minimizzare o massimizzare

Aggiungere un’espressione al problema per farla diventare funzione obiettivo:

```
problema = problema + 4*x + y
```

Aggiungere un vincolo al problema:

```
problema = problema + x + y <= 2
```

Perciò aggiungendo un’espressione al problema senza un operatore di confronto ( <=, >=, ==) questa diventa funzione obiettivo, se l’operatore è presente essa diventa vincolo.

Risolvere il problema:

```
problema.solve()
```

Se le parentesi dopo il solve vengono lasciate vuote allora il problema sarà risolto utilizzando il risolutore scelto da PuLP, altrimenti è possibile specificare in parentesi uno specifico risolutore da usare, per esempio: `problema.solve(CPLEX())`

Si veda ora un esempio pratico:

Il cibo per gatti Whiskas, mostrato sopra, è prodotto da Uncle Ben's. L'azienda desidera produrre i propri prodotti alimentari per gatti nel modo più economico possibile, garantendo al contempo stesso che soddisfino i requisiti di analisi nutrizionale indicati sulle lattine. Pertanto vuole variare le quantità di ciascun ingrediente utilizzato (gli ingredienti principali sono pollo, manzo, montone, riso, grano e gel) pur rispettando i loro standard nutrizionali. I costi di pollo, manzo e montone sono rispettivamente di € 0,013, € 0,008 e € 0,010, mentre i costi di riso, grano e gel sono rispettivamente di € 0,002, € 0,005 e € 0,001. (Tutti i costi sono per grammo.) Ogni ingrediente contribuisce al peso totale di proteine, grassi, fibre e sale nel prodotto finale. Gli apporti (in grammi) per grammo di ingrediente sono riportati nella tabella sottostante.

Ingredienti	Proteina	Grasso	Fibra	Sale
Pollo	0,100	0,080	0,001	0,002
Manzo	0,200	0,100	0,005	0,005
Montone	0,150	0,110	0,003	0,007
Riso	0,000	0,010	0,100	0,002
Grano	0,040	0,010	0,150	0,008
Gel	0,000	0,000	0,000	0,000

Si seguiranno alcuni passi per l'impostazione del problema:

1. Identificare le variabili decisionali.

Nel problema in esempio, le variabili decisionali sono le percentuali dei diversi ingredienti che includiamo nel barattolo. Essendo il barattolo da 100g, queste percentuali rappresentano anche la quantità in grammi di ogni ingrediente contenuto. Dobbiamo definire formalmente le nostre variabili decisionali (si noti che essendo percentuali devono essere comprese tra 0 e 100):

$x_1$  = percentuale di carne di pollo in un barattolo di cibo per gatti

$x_2$  = percentuale di carne di manzo in un barattolo di cibo per gatti

$x_3$  = percentuale di carne di montone in un barattolo di cibo per gatti

$x_4$  = percentuale di riso in un barattolo di cibo per gatti

$x_5$  = percentuale di grano in un barattolo di cibo per gatti

$x_6$  = percentuale di gel in un barattolo di cibo per gatti

2. Formulare la funzione obiettivo.

L'obiettivo è minimizzare il costo totale degli ingredienti per barattolo di cibo per gatti. Conoscendo il costo per grammo di ogni ingrediente è possibile formulare la funzione obiettivo al fine di trovare le quantità ( $x_1, x_2, \dots, x_6$ ) che rendono minimo il costo totale.

$$\text{Min } 0.013x_1 + 0.008x_2 + 0.010x_3 + 0.002x_4 + 0.005x_5 + 0.001x_6$$

3. Formulare i vincoli.

Le percentuali devono sommare ovviamente a 100 perciò:

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 = 100$$

Per soddisfare i requisiti dell'analisi nutrizionale, dobbiamo avere in 100 grammi almeno 8 g di proteine , 6 g di grassi, ma non più di 2 g di fibre e 0,4 g di sale. Per formulare questi vincoli ci avvaliamo della precedente tabella dei contributi di ciascun ingrediente. Questo ci permette di formulare i seguenti vincoli

Vincolo proteine:

$$0.100x_1 + 0.200x_2 + 0.150x_3 + 0.000x_4 + 0.040x_5 + 0.000x_6 \geq 8.0$$

Vincolo grassi:

$$0.080x_1 + 0.100x_2 + 0.110x_3 + 0.010x_4 + 0.010x_5 + 0.000x_6 \geq 6.0$$

Vincolo fibre:

$$0.001x_1 + 0.005x_2 + 0.003x_3 + 0.100x_4 + 0.150x_5 + 0.000x_6 \leq 2.0$$

Vincolo sale:

$$0.002x_1 + 0.005x_2 + 0.007x_3 + 0.002x_4 + 0.008x_5 + 0.000x_6 \leq 0.4$$



La risoluzione in Python può avvenire tramite due diversi metodi.

- Il primo più intuitivo prevede semplicemente l'inserimento del problema e dei vincoli tramite i costrutti forniti da PuLP per poi usare il risolutore scelto dal modellatore stesso:

Si inizia importando tutte le funzioni della libreria PuLP e successivamente si definisce il problema che in questo caso bisogna minimizzare

```
"""
Tesi di laurea triennale Nicola Lavecchia
Problema di programmazione lineare in Python
Esempio Cibo per gatti Whiskas tramite il modellatore PuLP
"""

from pulp import *

problema = LpProblem("Problema", LpMinimize)
```

Si definiscono le variabili con limite inferiore impostato a 0 e senza limite superiore, specificando che si tratta di variabili discrete(numero di barattoli da produrre) tramite l'argomento "LpInteger":

```
x1 = LpVariable("Percentuale Pollo", 0, None, LpInteger)
x2 = LpVariable("Percentuale Manzo", 0, None, LpInteger)
x3 = LpVariable("Percentuale Montone", 0, None, LpInteger)
x4 = LpVariable("Percentuale Riso", 0, None, LpInteger)
x5 = LpVariable("Percentuale Grano", 0, None, LpInteger)
x6 = LpVariable("Percentuale Gel", 0, None, LpInteger)
```

Dopo aver definito le variabili, si vengono aggiunti al problema creato inizialmente la funzione obiettivo e i vincoli:

```
problema += 0.013 * x1 + 0.008 * x2 + 0.010 * x3 + 0.002 * x4 + 0.005 * x5 + 0.001 * x6, "Costo totale ingredienti per porzione"

problema += x1 + x2 + x3 + x4 + x5 + x6 == 100, "Percentuale somma a 100"
problema += 0.100 * x1 + 0.200 * x2 + 0.150 * x3 + 0.000 * x4 + 0.040 * x5 + 0.000 * x6 >= 8.0, "Richiesta proteine"
problema += 0.080 * x1 + 0.100 * x2 + 0.110 * x3 + 0.010 * x4 + 0.010 * x5 + 0.000 * x6 >= 6.0, "Richiesta Grasso"
problema += 0.001 * x1 + 0.005 * x2 + 0.003 * x3 + 0.100 * x4 + 0.150 * x5 + 0.000 * x6 <= 2.0, "Richiesta fibre"
problema += 0.002 * x1 + 0.005 * x2 + 0.007 * x3 + 0.002 * x4 + 0.008 * x5 + 0.000 * x6 <= 0.4, "Richiesta sale"
```

Tramite il comando `solve()` si chiede al modellatore PuLP la soluzione del problema, successivamente con l'aiuto di un ciclo `for` si stampano le percentuali di ogni ingrediente che garantiscono l'ottimalità e rispettano i vincoli, infine si stampa il valore ottimo della funzione obiettivo:

```
problema.solve()

print("Stato del problema: ", LpStatus[problema.status])
print()

for variabile in problema.variables():
    print(variabile.name, " = ", variabile.varValue)

print()

print("Costo totale degli ingredienti per porzione: ", value(problema.objective))
```

Risultati:

```
Result - Optimal solution found

Objective value:           0.52000000
Enumerated nodes:         0
Total iterations:         0
Time (CPU seconds):       0.02
Time (Wallclock seconds): 0.02

Option for printingOptions changed from normal to all
Total time (CPU seconds):  0.02   (Wallclock seconds):  0.02

Stato del problema:  Optimal

Percentuale_Gel  = 40.0
Percentuale_Grano = 0.0
Percentuale_Manzo = 60.0
Percentuale_Montone = 0.0
Percentuale_Pollo = 0.0
Percentuale_Riso = 0.0

Costo totale degli ingredienti per porzione: 0.52
```

Il modellatore PuLP ci comunica che è stata trovata una soluzione ottima:

L'azienda Uncle Ben's riesce a produrre il cibo per gatti Whiskas nel modo più economico possibile garantendo al contempo i requisiti nutrizionali componendo un barattolo con il 60% di carne di manzo e con il 40% di gel, il costo di produzione totale è di 52 centesimi.

- Il secondo metodo prevede l'utilizzo dei dizionari, tramite questo costrutto python è possibile definire scrivere la tabella in traccia in Python dividendola per colonna, si possono poi combinare i dizionari delle caratteristiche nutrizionali con la lista degli ingredienti in modo da definire precisamente il problema e arrivare ad una soluzione ottima.

```
from pulp import *

# Si importano i costi e la tabella in Python tramite l'uso dei dizionari

# Lista degli ingredienti

ingredienti = ["Pollo", "Manzo", "Montone", "Riso", "Grano", "Gel"]

# Dizionario dei costi di ogni ingrediente

costi = {
    "Pollo": 0.013,
    "Manzo": 0.008,
    "Montone": 0.010,
    "Riso": 0.002,
    "Grano": 0.005,
    "Gel": 0.001}

# Dizionario della percentuale di proteine in ogni ingrediente

proteine = {
    "Pollo": 0.100,
    "Manzo": 0.200,
    "Montone": 0.150,
    "Riso": 0.000,
    "Grano": 0.040,
    "Gel": 0.000}

# Dizionario della percentuale di Grasso in ogni ingrediente

grasso = {
    "Pollo": 0.080,
    "Manzo": 0.100,
    "Montone": 0.110,
    "Riso": 0.010,
    "Grano": 0.010,
    "Gel": 0.000}
```

```

# Dizionario della percentuale di fibre in ogni ingrediente

fibre = {
    "Pollo": 0.001,
    "Manzo": 0.005,
    "Montone": 0.003,
    "Riso": 0.100,
    "Grano": 0.150,
    "Gel": 0.000}

# Dizionario della percentuale di sale in ogni ingrediente

sale = {
    "Pollo": 0.002,
    "Manzo": 0.005,
    "Montone": 0.007,
    "Riso": 0.002,
    "Grano": 0.008,
    "Gel": 0.000}

# Creazione della variabile problema
problema = LpProblem("Cibo per gatti Whiskas", LpMinimize)

# Si creano le variabili ingredienti con limite inferiore pari a 0 partendo dalla lista inizialmente creata
# il tutto si converte in dizionario tramite la funzione LpVariable.dicts
variabili_ingredienti = LpVariable.dicts("Ingredienti", ingredienti, 0)

# Si aggiunge al problema la funzione obiettivo tramite l'associazione dei dizionari "costi" e "variabili_ingredienti":

problema += (
    lpSum([costi[i] * variabili_ingredienti[i] for i in ingredienti]),
    "Costo totale degli ingredienti per barattolo",
)

# Si definiscono i vincoli tramite l'associazione del dizionario "variabili_ingredienti" con i dizionari dei singoli ingredienti

problema += (lpSum([variabili_ingredienti[i] for i in ingredienti]) == 100, "Vincolo somma percentuale")

problema += (lpSum([proteine[i] * variabili_ingredienti[i] for i in ingredienti]) >= 8.0, "Requisiti proteine")

problema += (lpSum([grasso[i] * variabili_ingredienti[i] for i in ingredienti]) >= 6.0, "Requisiti grasso")

problema += (lpSum([fibre[i] * variabili_ingredienti[i] for i in ingredienti]) <= 2.0, "Requisiti fibre")

problema += (lpSum([sale[i] * variabili_ingredienti[i] for i in ingredienti]) <= 0.4, "Requisiti sale")

# Si risolve il problema tramite il risolutore scelto da PuLP
problema.solve()

# Si stampa lo stato del problema
print("Status:", LpStatus[problema.status])

# Ogni variabile viene stampata con il suo valore ottimo
for v in problema.variables():
    print(v.name, " = ", round(v.varValue, "%"))
|
# Viene stampato il valore ottimo della funzione obiettivo
print("Costo totale degli ingredienti per barattolo = ", value(problema.objective), "Euro")

```

Risultati:

```
Status: Optimal
Ingredienti_Gel  = 40 %
Ingredienti_Grano = 0 %
Ingredienti_Manzo = 60 %
Ingredienti_Montone = 0 %
Ingredienti_Pollo = 0 %
Ingredienti_Riso = 0 %
Costo totale degli ingredienti per barattolo = 0.52 Euro
```

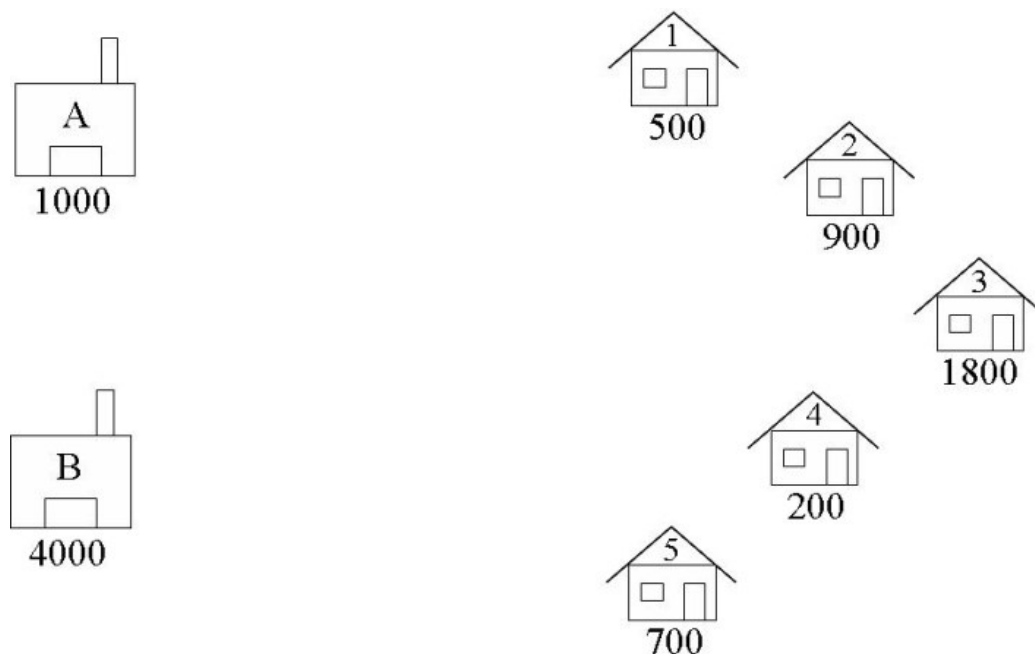
Come si può notare, i risultati dei due metodi coincidono, si può affermare che i due siano equivalenti e che si prestino entrambi bene a trovare la soluzione di un problema di Programmazione Lineare.

Si analizza un secondo esempio riguardante un problema di trasporto risolto su Python tramite PuLP con il metodo dei dizionari:

Un birrificio ha due magazzini da cui distribuisce la birra a cinque bar scelti con cura. All'inizio di ogni settimana, ogni bar invia alla sede del birrificio un ordine per tante casse di birra, che viene poi spedito dal magazzino appropriato al bar. Il birrificio vorrebbe avere un programma per computer interattivo che può eseguire settimana per settimana per dire loro quale magazzino deve rifornire quale barra in modo da ridurre al minimo i costi dell'intera operazione. Ad esempio, supponiamo che all'inizio di una data settimana il birrificio abbia 1000 casse nel magazzino A e 4000 casse nel magazzino B e che i bar richiedano rispettivamente 500, 900, 1800, 200 e 700 casse. Quale magazzino dovrebbe fornire quale bar?

Nei problemi di trasporto si decide come trasportare le merci dai loro nodi di offerta ai loro nodi di domanda.

Per questi tipi di problema ci si può aiutare con una rappresentazione grafica:



Come nello scorso esempio, si seguono i passi principali per la formulazione del problema:

1. Identificare le variabili decisionali:

A1 = numero di casse di birra da spedire dal Magazzino A al Bar 1

A2 = numero di casse di birra da spedire dal Magazzino A al Bar 2

A3 = numero di casse di birra da spedire dal Magazzino A al Bar 3

A4 = numero di casse di birra da spedire dal Magazzino A al Bar 4

A5 = numero di casse di birra da spedire dal Magazzino A al Bar 5

B1 = numero di casse di birra da spedire dal Magazzino B al Bar 1

B2 = numero di casse di birra da spedire dal Magazzino B al Bar 2

B3 = numero di casse di birra da spedire dal Magazzino B al Bar 3

B4 = numero di casse di birra da spedire dal Magazzino B al Bar 4

B5 = numero di casse di birra da spedire dal Magazzino B al Bar 5

Si scrivono le variabili in una forma diversa in modo da facilitare la costruzione del modello.

In particolare si userà l'insieme  $W$  per indicare i magazzini (warehouse) e l'insieme  $B$  per indicare i Bar:

$$\begin{aligned}W &= \{A, B\} \\B &= \{1, 2, 3, 4, 5\} \\x_{(w,b)} &\geq 0 \dots \forall w \in W, b \in B \\x_{(w,b)} &\in \mathbb{Z}^+ \dots \forall w \in W, b \in B\end{aligned}$$

Il limite inferiore delle variabili è zero e i valori devono essere tutti numeri interi (poiché il numero di casse non può essere negativo o frazionario). Non esiste un limite superiore.

## 2. Formulare la funzione obiettivo

La funzione obiettivo è stata genericamente definita come costo. Il problema può essere formulato come un programma lineare solo se il costo del trasporto dal magazzino al pub è una funzione lineare delle quantità di casse trasportate. Si noti che questo a volte non è il caso. Ciò può essere dovuto a fattori quali economie di scala o costi fissi.

Supponiamo quindi che vi sia un costo di trasporto fisso per cassa e che i costi siano i seguenti:

Consegna al bar	Dal magazzino A	Dal magazzino B
1	2	3
2	4	1
3	5	3
4	2	2
5	1	3

Ridurre al minimo i costi di trasporto = Costo per cassa per Consegna A1 \* A1 (numero di casse su ConsegnaA1) + ..... + Costo per cassa per RouteB5 \* B5 (numero di casse su RouteB5)

Perciò la funzione obiettivo, seguendo la notazione usata per definire le variabili, è:

$$\min \sum_{w \in W, b \in B} c_{(w,b)} x_{(w,b)}$$

Dove con c si intende il costo e con x la quantità delle casse



3. Formulare i vincoli:

I vincoli derivano da considerazioni di domanda e offerta. La fornitura di birra al magazzino A è di 1000 casse. La quantità totale di birra spedita dal magazzino A non può superare tale importo. Stesso discorso vale anche per B. Si ottengono così i primi vincoli:

$$A_1 + A_2 + A_3 + A_4 + A_5 \leq 1000$$

$$B_1 + B_2 + B_3 + B_4 + B_5 \leq 4000$$

Che seguendo la notazione precedente:

$$\sum_{b \in B} x_{(w,b)} \leq s_w \dots \forall w \in W$$

La domanda di birra al bar 1 è di 500 casse, quindi la quantità di birra consegnata lì deve essere almeno 500 per evitare vendite perse. Allo stesso modo, considerando che le quantità consegnate agli altri bar devono essere almeno pari alla domanda in quei bar.

$$A_1 + B_1 \geq 500$$

$$A_2 + B_2 \geq 900$$

$$A_3 + B_3 \geq 1800$$

$$A_4 + B_4 \geq 200$$

$$A_5 + B_5 \geq 700$$

$$\sum_{w \in W} x_{(w,b)} \geq d_b \dots \forall b \in B$$

È possibile ora risolvere il problema con Python.

Tramite il modellatore PuLP, per i problemi di trasporto, esiste un metodo più efficiente di quello usato in precedenza:

```
from pulp import *

# Creazione lista magazzini
magazzini = ["A", "B"]

# Creazione di un dizionario che associa ad ogni magazzino la propria disponibilit  di casse
fornitura = {"A": 1000, "B": 4000}

# Creazione lista bar
bar = ["1", "2", "3", "4", "5"]

# Creazione di un dizionario che associa ad ogni bar la quantit  ordinata
ordini = {
    "1": 500,
    "2": 900,
    "3": 1800,
    "4": 200,
    "5": 700,
}
```

L'inizio della formulazione   una semplice definizione dei nodi e dei loro limiti/capacit . I nomi dei nodi vengono inseriti in elenchi e le capacit  associate vengono inserite in dizionari con i nomi dei nodi come chiavi di riferimento:

```
# Creazione di una matrice dei costi
costi = [ # Bar
    # 1 2 3 4 5
    [2, 4, 5, 2, 1], # A   Magazzini
    [3, 1, 3, 2, 3], # B
]

# I dati dei costi vengono trasformati in un dizionario associato alle liste di magazzini e bar
costi = makeDict([magazzini, bar], costi, 0)
```

I dati di costo vengono quindi inseriti in una matrice, con due vettori: il primo contenente i costi di spedizione dal Magazzino A, e il secondo contenente i costi di spedizione dal Magazzino B. Si noti qui che il commento e la struttura del codice rende i dati visualizzabili come una tabella per una pi  semplice lettura

Gli elenchi Magazzini e Bar (nodi Domanda e Offerta) vengono aggiunti per creare una matrice e immessi nella funzione makeDict di PuLP per trasformarla in un dizionario. Una volta creato il dizionario dei costi, se viene invocato costi[A][1] , restituirà il costo del trasporto dal magazzino A al bar 1.

```
# Creazione della variabile problema
prob = LpProblem("Problema di trasporto della birra", LpMinimize)

# Creazione di una lista di tuple che contiene ogni possibile associazione di consegna magazzino-bar
consegne = [(m, b) for m in magazzini for b in bar]

# viene creato un dizionario per contenere le variabili decisionali riferite alle consegne
variabili = LpVariable.dicts("Consegna", (magazzini, bar), 0, None, LpInteger)
```

Viene creato un dizionario chiamato variabili che contiene le variabili LP. Viene impostato il limite inferiore di zero e illimitate superiormente, le variabili sono definite come numeri interi.

```
# Viene aggiunta la funzione obiettivo al problema
prob += (
    lpSum([variabili[m][b] * costi[m][b] for (m, b) in consegne]),
    "Somma totale dei costi di consegna",
)

# Vengono aggiunti al problema i vincoli di fornitura dei magazzini
for m in magazzini:
    prob += (
        lpSum([variabili[m][b] for b in bar]) <= fornitura[m],
        f"Somma dei prodotti inviati dal magazzino_{m}",
    )

# Vengono aggiunti al problema i vincoli di richiesta degli ordini da parte dei bar
for b in bar:
    prob += (
        lpSum([variabili[m][b] for m in magazzini]) >= ordini[b],
        f"Somma dei prodotti ricevuti dal bar{b}",
    )
```

La funzione obiettivo e i vincoli vengono aggiunti al problema utilizzando una comprensione di lista. Poiché variabili e costi sono ora dizionari (con ulteriori dizionari interni), possono essere utilizzati come se fossero tabelle.

```

# Il problema viene risolto utilizzando il risolutore scelto da PuLP
prob.solve()

# Lo stato del problema viene stampato su schermo
print("Status:", LpStatus[prob.status])

# Ogni variabile viene stampata con il suo valore ottimo
for v in prob.variables():
    print(v.name, "=", round(v.varValue))

# Il valore ottimo della funzione obiettivo viene stampato
print("Costo totale del trasporto = ", value(prob.objective))

```

Infine il problema viene risolto e vengono stampate le variabili e il valore ottimo della funzione obiettivo.

Risultati:

```

Status: Optimal
Consegna_A_1 = 300
Consegna_A_2 = 0
Consegna_A_3 = 0
Consegna_A_4 = 0
Consegna_A_5 = 700
Consegna_B_1 = 200
Consegna_B_2 = 900
Consegna_B_3 = 1800
Consegna_B_4 = 200
Consegna_B_5 = 0
Costo totale del trasporto = 8600.0

```

Il modellatore PuLP ci comunica che è stata trovata una soluzione ottima:

Il birrificio riesce a consegnare le casse soddisfacendo le richieste da parte dei bar spedendo le quantità viste nella soluzione al costo minimo possibile di 8.600€

# Conclusioni

A seguito delle tematiche analizzate nelle pagine precedenti sono possibili alcune considerazioni.

Si è constatata la possibilità di ridurre al minimo i calcoli necessari ad ottenere la soluzione di un problema di Programmazione Lineare: grazie ai software utilizzati si possono risolvere problemi di notevole complessità senza incorrere nei metodi classici come il dover analizzare ogni possibile vertice del poliedro o come il metodo del simplesso in forma tabellare che richiede maggiori tempistiche.

Da un lato il risolutore Excel permette tali soluzioni in maniera molto semplice e intuitiva pur mostrando alcuni limiti, dall'altra si è visto come con Python è possibile analizzare molto più velocemente grandi quantità di dati, consentendo la risoluzione a qualunque problema modificando poche righe di codice.

É emerso, tramite gli esempi svolti, come si possa far fronte a molteplici tipi di problema come la minimizzazione dei costi in un problema di trasporto, come la massimizzazione dei profitti per un dato prodotto aziendale.

Tutto ciò ci porta a sostenere che i metodi della Programmazione Lineare abbinati alle tecniche informatiche ci assicurano i massimi risultati tramite il minimo sforzo.

# Bibliografia

1) CARAMIA MASSIMILIANO- GIORDANI STEFANO – GUERRIERO FRANCESCA – MUSMANNO ROBERTO – PACCIARELLI DARIO. (2018) Ricerca Operativa. Isedi. NOVARA

2) MARIA LUISA DE CESARE – MARIA ROSARIA MADDALENA. (2001) Introduzione alla programmazione lineare. G. Giappichelli. TORINO

3) F. ARCHETTI -F. FAGIUOLI – A. SCIOMACHEN. (1989) Metodi della ricerca operativa. G. Giappichelli. TORINO

4) L. ERMINI. (1972) Programmazione lineare. Isedi. MILANO

# Sitografia

1) <https://www.diag.uniroma1.it/~roma/didattica/RONO/cap4-5.pdf>

2) <https://www.diag.uniroma1.it/~facchinei/cap3-01.pdf>

3) <http://www.diag.uniroma1.it/~roma/didattica/RO22-23/main.pdf>

4) <http://www.di.unito.it/~locatell/didattica/ro1/simplex-sl.pdf>

5) <https://it.know-base.net/7581611-linear-programming-in-excel>

6) <https://coin-or.github.io/pulp/>