Script per Ottimizzare Composti Chimici

Foskya

Contents

1 Introduzione **3**

2 Variabili **3**

3 Il problema **5**
3.1 Obiettivo 5
3.1.1 Vincoli 5
3.2 Soluzione 6

4 Limitazioni ed Implementazioni **6**
4.1 Limitazioni attuali 6

5 Code Reference **6**

1 Introduzione

Siccome stiamo affrontanto un problema di ottimizzazione lineare scrivo la documentazione per tenere traccia sia della teoria matematica che del codice python.

2 Variabili

Abbiamo:

- n rappresenta il numero dei composti chimici (es: $CaCO_3$, CH_4N_2O, *etc*).
  Nel codice è espresso con:

```
num_compounds = len(compounds)
```

- m rappresenta il numero degli elementi (es: Na, Cl, Ca, K, *etc*).
  Nel codice è espresso con:

```
elements=['Na','Cl','Ca','K','N','Fe','S','Mg','P','C']
```

- x_i rappresenta la percentuale del composto i nella soluzione finale.
- x è il vettore (array) delle percentuali dei vari composti. in quanto tale è di dimensioni $n \cdot 1$.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_n \end{bmatrix}$$

  Nota che, essendo percentuali, la somma di tutti gli x_i deve fare 1.

$$\sum_{i=1}^{n} x_i = 1$$

- c è il vettore con i coeficenti di costo. In parole semplici indica il "peso" (o meglio, il costo) di ogni composto permettendo di cercare la soluzione che minimizzi il costo. Nota che ha le stessi dimensioni di x (quindi $n \cdot 1$)
  Io ho avuto tutti i composti aggratis (grazie Pippo) quindi sono valorizzati a zero dato che non devo minimizzare nessun costo.

$$c = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \dots \\ 0 \end{bmatrix}$$

  Dal punto di vista del codice questo è fatto con:

```
num_compounds = len(compounds)
c = np.zeros(num_compounds)
```

- A è la matrice dei coeficenti, questa è una matrice di dimensioni $m \cdot n$ (quindi dove le righe sono gli elmenti, mentre le colonne sono i composti chimici).

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.57 & 0.39 & 0 \\ 0 & 0 & 0.35 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.61 & 0 \\ 0.40 & 0.24 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.45 & 0 & 0 & 0 & 0 \\ 0 & 0.17 & 0 & 0.82 & 0 & 0.47 & 0.21 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.27 & 0 & 0 & 0.18 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.12 & 0 & 0.20 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.23 & 0 & 0 & 0 & 0 & 0 \\ 0.12 & 0 & 0 & 0 & 0 & 0.20 & 0 & 0 & 0.40 & 0 & 0 & 0.38 \end{bmatrix}$$

Nota che invece nel codice i coeficenti sono stati riportati in maniera trasposta, (in altre parole gli elementi sulle colonne mentre i composti sulle righe). Questo perchè l'ho ereditata da un foglio G.Sheet

```
compounds = { # element espressed in %
   #                Na      Cl      Ca      K       N       Fe      S       Mg      P       C
   'CaCO3':         [0     , 0     , 0.400, 0     , 0     , 0     , 0     , 0     , 0     , 0.120],
   'Ca(NO3)2':      [0     , 0     , 0.244, 0     , 0.171, 0     , 0     , 0     , 0     , 0    ],
   'MgCl2x6H2O':    [0     , 0.349, 0     , 0     , 0     , 0     , 0     , 0.120, 0     , 0    ],
   'NH3':           [0     , 0     , 0     , 0     , 0.822, 0     , 0     , 0     , 0     , 0    ],
   'MgSO4':         [0     , 0     , 0     , 0     , 0     , 0     , 0.266, 0.202, 0     , 0    ],
   'CH4N2O':        [0     , 0     , 0     , 0     , 0.466, 0     , 0     , 0     , 0     , 0.200],
   '(NH4)2HPO4':    [0     , 0     , 0     , 0     , 0.212, 0     , 0     , 0     , 0.235, 0    ],
   'K2SO4':         [0     , 0     , 0     , 0.449, 0     , 0     , 0.184, 0     , 0     , 0    ],
   'CH3COOH':       [0     , 0     , 0     , 0     , 0     , 0     , 0     , 0     , 0     , 0.400],
   'NaOH':          [0.575, 0     , 0     , 0     , 0     , 0     , 0     , 0     , 0     , 0    ],
   'NaCl':          [0.393, 0.607, 0     , 0     , 0     , 0     , 0     , 0     , 0     , 0    ],
   'CitricoC6H8O':  [0     , 0     , 0     , 0     , 0     , 0     , 0     , 0     , 0     , 0.375]
   }
```

Per "rigirarla" utilizziamo:

```
   element_matrix = np.array(list(compounds.values())).T
```

- T è il vettore obiettivo (Target), contenente la percentuale di ogni elmento che si desidera avere nella composizione finale.

$$T = \begin{bmatrix} 0.257 \\ 0.028 \\ 0.000 \\ 0.030 \\ 0.019 \\ 0.000 \\ 0.010 \\ 0.001 \\ 0.004 \\ 0.109 \end{bmatrix}$$

nel codice è ottenuta semplicemente con:

```
target_composition =[0.257,0.028,0.000,0.030,0.019,0.000,0.010,0.001,0.004,0.109]
```

- ε è la variabile che rappresenta la tolleranza, esprime quindi di quanto, nella composizone finale, ogni elmento può discostare rispetto all'obiettivo.
  Nel codice è semplicemente espresso con:

```
tolerance = 0.1
```

3 Il problema

3.1 Obiettivo

Il programma trova l'insieme delle soluzioni fattibili, nello specifico quelle che soddisfano:

$$T - \varepsilon \leq Ax \leq T + \varepsilon$$

L'equazione può essere suddivisa nelle due sue parti:

- Il limite superiore (upper bound constraint) espresso come:

$$Ax \leq T + \epsilon$$

- Il imite inferiore (lower bound constraint) espresso come:

$$Ax \geq T - \epsilon$$

  che può anche essere scritto come:

$$-Ax \leq -(T - \epsilon)$$

  questa forma permette di impilare i due limiti l'uno sull'altro

Quindi possiamo riscrivere l'obiettivo come:

$$\begin{bmatrix} Ax \\ -Ax \end{bmatrix} \leq \begin{bmatrix} T - \epsilon \\ -(T - \epsilon) \end{bmatrix}$$

Siccome x è un vettore, possiamo portalro fuori ed esprimere la disequazione come:

$$\begin{bmatrix} A \\ -A \end{bmatrix} x \leq \begin{bmatrix} T + \epsilon \\ -(T - \epsilon) \end{bmatrix}$$

Facciamo ciò perchè nel codice la funzione *linprog* di *scipy.optimize* **accetta solo vincoli riferiti al limite superiore**. quindi nella forma:

$$A_{ub}x \leq b_{ub}$$

dove A_{ub} è dato da $\begin{bmatrix} A \\ -A \end{bmatrix}$, nel codice quindi:

```
A_ub = np.vstack([element_matrix, -element_matrix])
```

e similmente b_{ub} è dato da $\begin{bmatrix} T + \epsilon \\ -(T - \epsilon) \end{bmatrix}$, nel codice quindi:

```
b_ub = np.hstack([np.array(target_composition) + tolerance, -(np.array(target_composition) - tolerance)])
```

3.1.1 Vincoli

È possibile inserire dei vincoli per quanto riguarda i valori x_i, nel nostro caso abbiamo il limite di non negatività come limite inferiore e nessun limite come limite superiore:

```
bounds = [(0, None) for _ in range(num_compounds)]
```

3.2 Soluzione

una volta ottenuto il set della soluzione fattibile trova la soluzione che risolve:

$$minimizza \qquad c^T x$$

dove c^T è il vettore delle coeficenti di costo trasposto (per permettere la moltiplicazione).
In forma esplicita:

$$Minimizza \qquad c^T x = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_n \end{bmatrix}$$

$$Minimizza \qquad 0x_1 + 0x_2 + \ldots + 0x_n = 0$$

Come detto prima nel mio caso non devo ottimizzare nulla, quindi sono tutti valorizzati a zero (i.e. questo passaggio non serve).

Il risultato a livello di codice è ottenuto con:

```
result = linprog(c=c, A_ub=A_ub, b_ub=b_ub, A_eq=A_eq, b_eq=b_eq, bounds=bounds, method='highs')
```

4 Limitazioni ed Implementazioni

4.1 Limitazioni attuali

- Assume mischiaggio ideale senza interazioni chimiche tra i componenti
- viene estratta una soluzione nell'insieme delle soluzioni fattibili, non necessariamente la migliore

5 Code Reference

```
import numpy as np
from scipy.optimize import linprog


tolerance = 0.1


compounds = { # element espressed in %
   #                Na      Cl      Ca      K       N       Fe      S       Mg      P       C
   'CaCO3':         [0    , 0    , 0.400, 0    , 0    , 0    , 0    , 0    , 0    , 0.120],
   'Ca(NO3)2':      [0    , 0    , 0.244, 0    , 0.171, 0    , 0    , 0    , 0    , 0    ],
   'MgCl2x6H2O':    [0    , 0.349, 0    , 0    , 0    , 0    , 0    , 0.120, 0    , 0    ],
   'NH3':           [0    , 0    , 0    , 0    , 0.822, 0    , 0    , 0    , 0    , 0    ],
   'MgSO4':         [0    , 0    , 0    , 0    , 0    , 0    , 0.266, 0.202, 0    , 0    ],
   'CH4N2O':        [0    , 0    , 0    , 0    , 0.466, 0    , 0    , 0    , 0    , 0.200],
   '(NH4)2HPO4':    [0    , 0    , 0    , 0    , 0.212, 0    , 0    , 0    , 0.235, 0    ],
   'K2SO4':         [0    , 0    , 0    , 0.449, 0    , 0    , 0.184, 0    , 0    , 0    ],
   'CH3COOH':       [0    , 0    , 0    , 0    , 0    , 0    , 0    , 0    , 0    , 0.400],
   'NaOH':          [0.575, 0    , 0    , 0    , 0    , 0    , 0    , 0    , 0    , 0    ],
   'NaCl':          [0.393, 0.607, 0    , 0    , 0    , 0    , 0    , 0    , 0    , 0    ],
   'CitricoC6H8O':  [0    , 0    , 0    , 0    , 0    , 0    , 0    , 0    , 0    , 0.375]
   }
elements =          ['Na',  'Cl',  'Ca',  'K',   'N',   'Fe',  'S',   'Mg',  'P',   'C']
target_composition =[0.257, 0.028, 0.000, 0.030, 0.019, 0.000, 0.010, 0.001, 0.004, 0.109]


num_compounds = len(compounds)
c = np.zeros(num_compounds)
```

```
# inequality constraints (A_ub * x <= b_ub) ---
element_matrix = np.array(list(compounds.values())).T
A_ub = np.vstack([element_matrix, -element_matrix])
b_ub = np.hstack([np.array(target_composition) + tolerance, -(np.array(target_composition) - tolerance)])


A_eq = np.ones((1, num_compounds))
b_eq = np.array([1])


bounds = [(0, None) for _ in range(num_compounds)] # constraint: 0 imply non negative bounds and None
    implies no upper bound


result = linprog(c=c, A_ub=A_ub, b_ub=b_ub, A_eq=A_eq, b_eq=b_eq, bounds=bounds, method='highs')


if result.success:
   print("A feasible mixture was found within the specified tolerance:")
   proportions = result.x
   for i, name in enumerate(compounds.keys()):
       print(f"- {name}: {proportions[i] * 100:.2f}%")
   for i, name in enumerate(compounds.keys()):
       print(f"{proportions[i] * 100:.2f}%")

   # Verify the composition of the resulting mixture
   final_composition = element_matrix @ proportions
   print("\nResulting elemental composition:")
   for i, el in enumerate(elements):
       print(f"- {el}: {final_composition[i] * 100:.2f}% (Target: {target_composition[i] * 100:.2f}%)")
else:
   print("No mixture could be found, even with the specified tolerance.")
   print(f"Message: {result.message}")
   print("\nSuggestions:")
   print("1. Double-check your input percentages for all compounds.")
   print("2. Verify that your target composition is chemically possible with the given inputs.")
   print("3. Consider increasing the tolerance.")
```