Pràctica 1: Implementació de l'algorisme Símplex

Optimització

Joan Saurina i Ricós, Sergi Tomàs Martínez

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

26 de març de 2023

Índex

1 El problema **2**
1.1 Conjunt de dades 2

2 Implementació **3**
2.1 Main.py 3
2.2 Fase_1.py 3
2.3 Fase_2.py 5
2.4 Auxiliar_functions.py 6
2.4.1 read_data() 6
2.4.2 costos_minims() 6
2.4.3 direccio() 7
2.4.4 longitud() 7
2.4.5 actualitzacio() 8
2.4.6 actualitzacio_inversa() 9
2.4.7 matriusn() 9

3 Output i instruccions d'ús **11**

4 Conclusió **12**

5 Outputs dades 40 i 42 **13**

1 El problema

A la primera pràctica de l'assignatura optimització (OPT) es demana implementar l'algorisme símplex vist a classe.

S'ha escollit utilitzar el llenguatge de programació *Python* perquè té una sintaxi simple i fàcil d'entendre, fet que el fa molt accessible per a aquells que busquen resoldre problemes de forma eficient i ràpida. És també, amb diferència, el llenguatge amb el qual els estudiants del grau d'Intel·ligència Artificial es mouen amb més soltesa i facilitat.

Pel que fa al càlcul de la solució bàsica factible inicial (SBF) amb la fase I del símplex, es proposen dues opcions. S'ha escollit la primera d'aquestes, que consisteix a fer un codi únic que integri la fase I al codi, de manera que formuli i resolgui el problema de **fase I** automàticament a partir dels paràmetres que defineixen el problema original **(c, b i A)**, i un cop identificada una SBF, si n'hi ha, que continuï amb la **fase II**.

El conjunt de problemes que s'ha assignat són les dades 40 i 42.

1.1 Conjunt de dades

Al conjunt de dades, la matriu A correspon a la forma estàndard d'un problema amb les restriccions d'1 a 4 de "=", les restriccions 5, 8 i 9 de "ï les restriccions 6, 7 i 10 de "". Observar com les darreres 6 variables corresponen a variables de folgança/excés. La matriu c conté els coeficients de la funció objectiu i la matriu b que conté els termes independents de les restriccions.

A més a més, al primer problema de cada conjunt de dades, es mostra la solució òptima (valor de les variables bàsiques a l'òptim **(vb*)** i valor de la funció objectiu **(z*)**) perquè es pugui comprovar l'algorisme.

Els problemes PL 2, 3 i 4 de cada conjunt de dades poden tenir solució òptima, ser infactibles, no acotats o degenerats. Es tracta de comprovar la implementació es comporta davant d'aquestes situacions.

2 Implementació

2.1 Main.py

La funció **main()** és la funció principal del programa i s'encarrega de llegir un fitxer de text que conté una sèrie de problemes de programació lineal en format estàndard, resolent-los utilitzant l'algorisme simplex.

En primer lloc, s'obre el fitxer de text i es llegeixen les dades del primer problema. A continuació, s'anomena la funció fase1() per obtenir una solució bàsica factible inicial. Si hi ha una solució, es construeixen les variables no bàsiques, el vector de solució i els vectors de costos i restriccions necessaris per realitzar la fase 2 de l'algorisme simplex. S'anomena llavors la funció fase2() per aconseguir la solució òptima del problema. Si hi ha una solució òptima, s'imprimeix per pantalla la solució trobada.

El programa continua llegint i resolent els problemes següents del fitxer de text fins que es troba la instrucció *"fi"*, moment en què s'acaba l'execució del programa. Finalment, s'imprimeix "He acabat!!"per indicar que l'execució ha finalitzat.

2.2 Fase_1.py

L'arxiu Fase_1.py conté únicament la funció **fase1()**. Aquesta serà l'encarregada de trobar una Solució Bàsica Factible (**SBF**) per tal d'iniciar la minimització del problema. És a dir, és aquí on es crea el problema artificial per a trobar una base sobre la qual es pot començar a iterar a la fase 2.

La funció rep els següents paràmetres:

1. **A :** La matriu en forma estàndard de les restriccions problema.
2. **b :** El vector dels termes independents d'aquestes restriccions
3. **c :** El vector dels coeficients de la funció objectiu. Aquest ens servirà per a determinar si la Base trobada és una SBF o no.
4. **n :** El nombre de variables
5. **m :** El nombre de restriccions

La fase 1 del símplex crea una variable nova a cada una de les restriccions. Aquestes variables artificials són les úniques que tindran coeficient no nul (1, per a totes elles) a la funció objectiu. Amb totes aquestes variables a la base, i degut al fet que la base és tan gran com el nombre de restriccions i s'ha afegit una variable a cada una, hi ha **m** variables artificials. La base, per tant, seran tots els índexos des de 0 fins a **n-1** i les variables no bàsiques seran tots els

indexos des de **n** fins a **n+m-1**. Cal remarcar que les variables al programa s'indexen des del 0.

Les variables tenen coeficient 1 a la seva restricció, és a dir, la matriu B inicial serà una matriu identitat d'ordre **n**; i la nova matriu **A** serà una concatenació de la matriu **A** anterior i la identitat d'ordre m a l'eix horitzontal. Aquesta estratègia permet no haver de calcular la inversa de B, ja que $I^{-1} = I$.

D'aquesta manera, durant les iteracions tant de Fase 1 com de Fase 2 s'anirà actualitzant la inversa de B, i s'evitarà realitzar cap càlcul d'inversa que algorísmicament és molt costós.

El vector c tindrà **n** 0 seguits de **m** 1. I el vector **cb** inicial serà de **m** 1.

Les matrius **An** i **cn** es generen amb la funció **matriusn**. Detalls d'aquesta es troben a l'apartat **2.4.7 matriusn()**.

El vector Xb es calcula multiplicant la inversa de **B**, el vector **b**. El vector X (que realment només és útil per la interfície d'usuari) es construeix concatenant **n** 0 i el vector **Xb**.

Una vegada totes les matrius han estat creades es crida a **fase_2()** amb el problema artificial. La funció la base final del problema, la inversa final de **B**, el valor final de la **z** del problema artificial i el vector final de **X**.

El programa comprovarà si $z > 0$. Si es compleix, vol dir que a la millor solució possible encara hi ha variables artificials i, a més a més, aquestes tenen coordenada no nul·la, és a dir, el problema no és factible. **fase1()** retorna None i el programa passa al següent problema.

En el cas que $\mathbf{z} = 0$, s'ha de contemplar dues opcions:

1. La base final no conté cap variable artificial.
2. La base final té alguna variable artificial.

La primera opció implica que la base trobada forma una SBF i, per tant, es pot retornar al problema original i tornar a cridar fase 2 amb aquesta.

El segon cas implica que es tracta d'un problema degenerat, és a dir, hi ha diverses bases que donen la mateixa solució. Es troba una variable artificial dins de la base, però aquesta té coordenada = 0. En aquest cas, es procedeix a retirar-la i afegir-ne d'altres de no artificials arbitràriament a la seva posició fins a trobar-ne una tal que $det(B) \neq 0$.

Finalment, es retorna $B^{-1}, base, Xb$ per a poder iniciar la fase 2 amb la base trobada i no haver de tornar a realitzar cap càlcul ja realitzat anteriorment.

2.3 Fase_2.py

L'arxiu Fase_2.py conté la funció **fase2()**, dins de la qual es troba el bucle de les iteracions, així com les crides a les funcions auxiliars que realitzen els càlculs.

Abans d'entrar dins del bucle com a tal, el programa calcula els costos reduïts del problema inicial per a comprovar si el problema es troba ja a l'òptim. A més a més, també es calcula el valor de z inicial el qual s'anirà actualitzant amb el valor de θ a mesura que passin les iteracions.

Si qualsevol valor dels costos reduïts és negatiu, la funció entra dins del bucle i comença a buscar una variable d'entrada. Amb un petit bucle s'aplica la **regla de Bland**, i s'escull la variable amb índex menor de les quals tenen r_i negatiu. Una opció contemplada per a dur a terme aquest procés va ser mantenir la llista de variables no bàsiques ordenades, és a dir, que en haver de recórrer la llista serà suficient amb quedar-se amb la primera variable amb cost reduït negatiu. El cost d'ordenar la llista cada vegada hauria estat de $O(nlog(n))$, mentre que el de recórrer la llista en busca de l'índex menor és de $O(n)$, fet que ho fa algorísmicament més barat.

Una vegada amb **q** definida, es passa a avaluar la direcció de descens amb la funció **direccio()**. A part del vector db, aquesta funció també retorna el valor *flag*, el qual marca si ha trobat que el problema no és acotat. En aquest cas, la funció fase2() retornaria *None* i el programa passaria al següent problema.

Seguidament, es crida a la funció **longitud()**, que retorna tant θ com el valor de **p**, la posició a la base de la variable de sortida.

Per a preparar la iteració següent es passa a actualitzar les matrius amb la funció **actualitzacio()**, segons les variables d'entrada i sortida decidides.

Finalment es calculen de nou els costos negatius amb les matrius actualitzades per a poder comprovar amb el bucle *while np,any(r¿0)* si la següent iteració és l'òptim.

Quan se surt de l'òptim es retornen les matrius i valors convenients o bé per a quan Fase 2 és cridada per Fase 1, o bé per al *print* de final de problema.

2.4 Auxiliar_functions.py

El fitxer conté una col·lecció de funcions que s'utilitzen com a funcions auxiliars. Aquestes funcions realitzen tasques com operacions amb matrius, manipulacions vectorials i altres càlculs matemàtics necessaris per resoldre el problema d'optimització.

En organitzar aquestes funcions en un fitxer separat, permet una codificació modular, un manteniment més fàcil i una millor llegibilitat del codi de l'algoritme d'optimització principal.

2.4.1 read_data()

Aquesta funció llegeix un sistema d'equacions lineals en forma d'Ax=b des d'un fitxer. La funció pren un objecte de fitxer com a entrada i llegeix les dades del fitxer.

Les primeres línies de la funció llegeixen els valors de la variable **c** del fitxer. El bucle while llegeix les línies fins que es troba la línia que comença amb *'c='*, que marca l'inici del vector **c**. Els valors de **c** es llegeixen a la línia següent després d'aquesta i s'emmagatzemen en una matriu *numpy*.

La següent secció de la funció llegeix la matriu de coeficients **A**. El bucle *while* llegeix les línies fins que es troba una línia en blanc, que marca el final de la matriu A. Les línies s'emmagatzemen com una llista de llistes, on cada subllista correspon a una fila de la matriu. A continuació, aquesta llista de llistes es converteix en una matriu *numpy*.

Finalment, la funció llegeix els valors del vector constant **b** del fitxer. Els valors s'emmagatzemen com a vector fila en una matriu *numpy*.

La funció retorna la matriu de coeficients **A**, el vector constant **b** i el vector constant **c** com a matrius *numpy*.

2.4.2 costos_minims()

Aquesta funció calcula els costos mínims d'un problema de programació lineal tenint en compte els paràmetres d'entrada:

1. **cn :** La matriu que conté els coeficients de la funció objectiu a minimitzar.
2. **cb :** La matriu que conté els coeficients de les variables bàsiques de la solució factible actual.
3. **B_1 :** La inversa de la matriu base B.

4. **An :** La matriu que conté els coeficients de les variables no bàsiques de la solució factible actual.

La funció calcula els costos mínims realitzant operacions de multiplicació de matrius entre les matrius d'entrada. Concretament, calcula el producte de **cb** i B^{-1} i després multiplica el resultat per **An**. Finalment, resta la matriu resultant de **cn**.

La sortida resultant és una matriu que conté els costos mínims del problema de programació lineal.

2.4.3 direccio()

Aquesta funció calcula el vector de direcció per al mètode simplex per avançar cap al cost mínim. Els paràmetres d'entrada són:

1. **B_1 :** La inversa de la matriu base B.
2. **A :** La matriu en forma estàndard de les restriccions problema.
3. **columna :** L'índex de la columna corresponent a la variable que està entrant a la base. (p)

La funció primer calcula el vector db multiplicant B_1 amb la columna d'A corresponent a la variable introduïda. El vector direcció és el negatiu d'aquest vector **db**.

Si tots els elements de **db** són majors o iguals a zero, el problema és no acotat (és a dir, no hi ha un cost mínim finit). En aquest cas, la funció retorna un valor de senyal d'1 per indicar a altres funcions aquesta característica del problema. En cas contrari, el valor del senyal és 0.

La funció retorna una tupla que conté el vector de direcció **db** i el senyal.

2.4.4 longitud()

Aquesta funció calcula la longitud del pas en el mètode símplex per a la programació lineal. Els paràmetres d'entrada són:

1. **Xb :** La matriu que representa les variables bàsiques de la base actual.
2. **db :** La matriu que representa la direcció de cada variable bàsica en la base actual.
3. **base :** Llista de nombres enters que representen els índexs de les variables bàsiques del problema original

La funció primer inicialitza una variable **p** buida i estableix la millor θ (mida del pas) a infinit. A continuació, itera sobre els índexs de les variables bàsiques amb valors negatius en db. Per a cada variable bàsica, calcula el valor θ corresponent com a relació negativa de l'element corresponent en **Xb** i **db**.

Si el valor θ és inferior al millor valor θ actual, la funció actualitza el millor θ i estableix p a l'índex de variable bàsic corresponent. Si el valor θ és igual al millor valor θ actual, la funció aplica la **regla de Bland**, que tria la variable amb l'índex més baix en cas d'empat.

Finalment, la funció retorna una tupla amb el millor valor θ i l'índex de la variable bàsica escollida.

2.4.5 actualitzacio()

Aquesta funció s'utilitza per actualitzar les variables i les matrius després d'una operació de pivot en l'algorisme simplex. Els paràmetres d'entrada són:

1. **p :** Índex de la variable bàsica que surt de la base.
2. **q :** Índex de la variable no bàsica que entra a la base.
3. **base :** Llista d'índexs de les variables bàsiques.
4. **no_base :** Llista d'índexs de les variables no bàsiques.
5. **Xb :** Vector columna de les variables bàsiques.
6. **theta :** Mida del pas.
7. **db :** La matriu que representa la direcció de cada variable bàsica en la base actual.
8. **cb :** Vector fila dels coeficients de les variables bàsiques de la funció objectiu.
9. **c :** Vector fila dels coeficients de les variables de la funció objectiu.
10. **A:** La matriu en forma estàndard de les restriccions problema.
11. **B_1 :** La matriu inversa de la base.

La funció primer actualitza les llistes base i no_base intercanviant els índexs de les variables d'entrada i de sortida.

A continuació, actualitza **Xb** multiplicant θ amb **db** i establint l'element **p** de **Xb** a θ. Després, es reconstrueix el vector **X** concatenant les variables bàsiques en **Xb** i les variables no bàsiques posades a zero.

Aleshores, la funció actualitza la matriu B^{-1} utilitzant la funció **actualitzacio_inversa()** i torna a calcular les matrius **An**, **cn** i **cb** utilitzant la funció **matriusn()**.

Finalment, actualitza el vector cb establint l'element p al coeficient de la variable que entra a la funció objectiu.

La funció retorna els valors actualitzats de **base**, **no_base, Xb**, **B_1**, **An**, **cn**, **cb**, **c**, **X**.

2.4.6 actualitzacio_inversa()

Aquesta funció realitza l'actualització de la inversa de la matriu base. Els paràmetres d'entrada són:

1. **B_1 :** La matriu inversa actual de la matriu base.
2. **p :** L'índex de la fila que s'actualitzarà.
3. **db :** La matriu que representa la direcció de cada variable bàsica en la base actual.

La funció crea primer una matriu E com la matriu indeitat de dimensió **db**. L'element p d'E s'estableix en el recíproc de **db[p]**, i la resta dels elements de la columna **p** d'**E** s'estableix en **-db/db[p]**. Aleshores, B^{-1} s'actualitza multiplicant **E** per B^{-1} des de l'esquerra.

La funció retorna la inversa actualitzada de la matriu base B^{-1}.

2.4.7 matriusn()

Aquesta funció realitza el càlcul de les matrius associades a variables no bàsiques, així com els coeficients de funció objectiu d'aquestes variables. Els paràmetres d'entrada són:

1. **A:** La matriu en forma estàndard de les restriccions problema.
2. **c :** Vector fila dels coeficients de les variables de la funció objectiu.
3. **no_base :** Llista d'índexs de les variables no bàsiques.

La funció inicialitza primer una llista buida cn i una matriu **An** de zeros amb el mateix nombre de files que **A**. A continuació, recorre els índexs de les variables no bàsiques i afegeix el coeficient de funció objectiu corresponent a **cn** mentre concatena la columna corresponent de **A** a **An**.

Finalment, s'elimina la primera columna d'An (ja que inicialment són zeros) i les matrius An i cn resultants es tornen com una tupla. Aquestes matrius s'utilitzen en passos posteriors de l'algorisme simplex per actualitzar la solució bàsica factible actual.

3 Output i instruccions d'ús

El codi realitza una sèrie de prints a la consola d'un volum considerable. Aquests han sigut molt útils a l'hora d'analitzar l'execució i el desenvolupament dels problemes, veure l'evolució de z i dels valors de les variables a mesura que el símplex s'executava. Durant el programa se sol·liciten una sèrie de *inputs* de l'usuari, aquests no tenen rellevància i només serveixen perrquè el programa pari un instant i mostri diferents paràmetres rellevants.

Tot i la seva utilitat aquests poden ser una mica molestos si es vol, per exemple, imprimir tot l'output en un fitxer a part. En aquest cas, només caldria comentar les línies de **input()** i l'arxiu s'imprimiria normalment.

Si es vol canviar el fitxer d'input cal canviar la variable filename al fitxer *main.py*, a la línia **18**, per la direcció local o general de l'arxiu desitjat.

4 Conclusió

Durant la implementació de l'algorisme primal Simplex, ens hem enfrontat a diverses dificultats, especialment relacionades amb la selecció dels índexs correctes en algunes matrius. Va requerir molta atenció i una acurada consideració de les dimensions de cada matriu per evitar errors. A més, la depuració era un repte de vegades, ja que la complexitat del codi augmentava a mesura que afegíem noves funcions.

No obstant això, malgrat les dificultats, la implementació de l'algorisme ens va permetre entendre a fons com funciona l'algorisme Simplex i la importància de les matrius implicades. També ens va ajudar a desenvolupar les nostres habilitats de programació i en el camp de l'àlgebra lineal.

En el futur, podríem millorar encara més l'algorisme implementant l'algoritme Dual Simplex. És un algorisme alternatiu a Simplex, que de vegades pot ser més eficient, especialment en situacions en què el problema principal té moltes més restriccions que variables.

5 Outputs dades 40 i 42

Es troben adjuntades a l'arxiu .zip on es troba aquest document totes les traces i outputs dels problemes 40 i 42, per si no es desitja executar el programa completament.