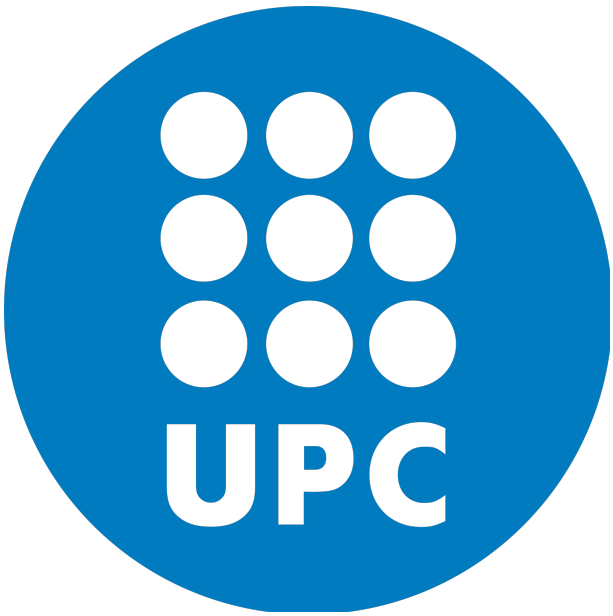
Implementació de símplex a python



Grau en Intel·ligència Artificial

Optimització

Autors:
Eduard Barnadas Conangla, **Pau Hidalgo**

29 de març de 2024

Índex

1 Introducció

L'objectiu d'aquesta pràctica era aconseguir implementar l'algorisme de Símplex amb el nostre pròpi codi, i aconseguir que aquest solucionés uns problemes que teniem assignats per a cadascún dels alumnes. En el nostre cas, hem decidit programar-lo en Pyhton3, ja que és el llengüatge en el que els dos integrants del grup tenim més experiència.

2 Lectura del document d'input

El primer pas per a poder aplicar Símplex era obtenir la informació del document de input, per a fer això vam dissenyar la funció de read_dades(), que té com a paràmetres el número de l'alumne i el nombre de la prova d'aquest. Una vegada trobat el començament del text del que volem extreure les dades, podem començar a buscar els caràcters que ens indicaràn les dades que llegirem, que en aquest cas son ç=", "A=", "b=", i en alguns casos "z*=ï "vb*=", una vegada trobats aquests, emmagatmememem cada nombre en una llista que posteriorment serà convertida en una array de numpy, ja que d'aquesta forma podrem treballar amb aquestes llistes com si fossin matrius (fer multiplicacions, sumes inverses...). Per a la lectura d'aquests elements hem utilitzar expressions regulars ja que així ens facilitava la feina de processar les irregularitats del text.

3 Script de Símplex

Un cop llegides les dades, es pot ja executar directament l'algorisme de Símplex. En el nostre cas, hem optat per dissenyar una sola funció encarregada de realitzar-ho tot, ja que vam considerar que així era més senzill.

El primer pas que fa és observar quantes variables i quantes restriccions hi ha al problema. Si hi ha menys variables que restriccions, retorna que el sistema és incorrecte.

A continuació, observa si li toca fer fase 1. Per defecte, sempre la farà, però realment se li pot especificar que no la faci (tot i que llavors cal tenir en compte que les variables bàsiques hauran de ser a les m columnes finals).

Per realitzar-la, afegim a la matriu A una matriu identitat a la dreta usant la funció hstack de numpy, i definim els nous costos (ja que la z de la fase I és diferent). Aleshores, simplement executem la mateixa funció indicant-li que no realitzi fase I, i obtenim el resultat.

Aquest resultat ens dóna informació sobre el nostre problema, ja que sí no troba cap SBF o la z és major a 0, el problema no tindrà sol·lució factible i per tant no cal continuar amb l'execució. En cas contrari, realitzem la comprovació que no hi hagi cap variable artificial degenerada. Sí n'hi ha, aquesta s'ha de tractar.

Com que no disposem del dual per iterar-hi, hem optat per provar de realitzar la inversa amb totes les no bàsiques (i no artificials). Si en troba alguna, substituirà la variable degenerada per aquella i continuarà amb l'execució. Si no, retorna directament la SBF degenerada trobada per la

fase I, ja que no podrà iterar per la fase II per trobar una solució millor.

Mencionar que en la fase I també calculem la inversa (partim de que és la identitat i anem realitzant acutalitzacions) i li passem a la fase II. D'aquesta manera, a no ser que sigui el cas excepcional de trobar una variable artificial degenerada, no hem de recalcular cap inversa ni usar la funció de numpy.

L'última comprovació que realitzem abans de procedir normalment amb la fase II és que no hi hagi el mateix nombre de variables que restriccions, ja que en aquest cas no tindrem cap no bàsica i per tant, serà aquell l'òptim (com a solució única possible).

En cas que haguem aconseguit una SBF amb la fase I i després de contemplar totes les excepcions, procedirem amb la fase II. En aquesta iterarem de manera indefinida (while True) fins que trobem que no és factible, un raig o bé que un òptim. Alhora, en cada iteració comprovem que no hi hagi cap x més petita que zero, ja que significaria que hi ha hagut algun error i estariem violant alguna restricció. Els passos són els mateixos que els del símplex normal: calculem els costos reduïts, observem si tots són majors a zero (el mínim és major a zero) o iguals. En aquests casos, retornem que hem trobat la sol·lució, amb els valors de les x, la z, les bàsiques i la inversa.

En cas de no estar en un òptim, continuarem amb l'algorisme escollint el cost reduït negatiu d'índex més baix. Com que tenim les no bàsiques ordenades (per com està creada la llista ho serà sempre) simplement cal iterar pels costos reduïts fins trobar-ne un de negatiu, on aturem la iteració. Aquest, serà el que tingui l'índex més petit, aplicant així la Regla de Bland.

Llavors és el torn de calcular les direccions bàsiques factibles de les variables bàsiques. Si no n'hi ha cap de descens, aleshores es tracta d'un raig (no acotació), no serà possible trobar cap theta. En aquest cas, aturem l'execució retornant les variables bàsiques en aquest punt i no acotat. Sinó, cal calcular les longituds de pas i escollir l'índex de la mínima. Per fer-ho, aprofitem l'operador de Python enumerate, que ens permet recòrrer els elements d'una llista així com els seus índexs alhora, i busquem el mínim d'una llista de tuples. Les tuples contindran les longituds de pas i els índexs. Aquest fet ens permet no haver de realitzar comprovacions extres per aplicar la regla de Bland, ja que la funció min() de Python, en una llista de tuples, observa primer en els primers elements, i en cas d'empat l'escull en funció dels segons, escollint així la theta d'índex més petit [1].

Arribats a aquest punt, tenim seleccionades les variables d'entrada i de sortida. L'únic que falta per passar a la següent iteració són les actualitzacions. Aquestes també les hem implementat, així com l'actualització de la inversa.

4 Resultats

Els resultats de l'execució del Simplex es guarden en un fitxer output.txt. Aquest, conté per cada problema i conjunt de dades, la següent informació: - D'entrada, escriu que comença realitzant la Fase I - Per cada iteració, indica: índex de la variable d'entrada (q), índex de la variable de sortida (B[p]), índex dins les bàsiques de la variable de sortida (p), la longitud de pas i el valor de z. - Fi de la Fase I. Mostra les variables bàsiques i el nombre d'iteracions. En cas que sigui SBF, comença la fase II. En cas contrari, indica que no hi ha solució factible. - En la fase II, torna a indicar el mateix per cada iteració. - En cas que trobi un raig, escriu Optim no acotat, i retorna igualment les bàsiques i el nombre total d'iteracions. En cas que trobi solució òptima, escriu els valors de x en aquella SBF, el valor de z, r i les bàsiques.

Addicionalment, cada cop que troba una solució degenerada ho escriu.

Mencionar que, al estar programat en Python, tots els índexs de les variables són contant des de 0.

Observant els resultats, vam veure que en gairebé tots els casos, el problema 1 era factible, el problema 2 també, el 3 era infactible i el 4 no acotat.

Els que ens tocaven a nosaltres es poden observar en el fitxer output_nostre.txt, eren els problemes 15 i 42.

5 Test addicional

A part de provar amb tots els conjunts de dades que se'ns oferien, vam decidir realitzar una petita col·lecció de problemes addicionals per observar el comportament de la nostra implementació. Alguns d'aquests nous problemes de test van ser obtinguts dels recursos trobats a internet [3] i [2] Més concretament, vam decidir buscar específicament problemes amb degeneració o infactibilitat, ja que són els casos que poden comportar més problemes.

Aquests es poden trobar al fitxer Datos_práctica_1_test.txt, i la seva sortida es troba també al fitxer d'output amb els alumnes 67-70.

Referències

[1] *How does tuple comparison work in Python?* Stack Overflow, març de 2011. URL: https://stackoverflow.com/a/5292332 (cons. 28-03-2024).

[2] K.K. Kansal. *Degeneracy in Simplex Method*. URL: https://www.nascollege.org/e%20cotent%2010-4-20/DR%20K%20K%20KANSAL/L%2010%20M%20COM%2020-4-E.pdf.

[3] Taha. *Operations Research: An Introduction, 8/E*. Pearson Education India, set. de 2008. URL: https://dstm.ntou.edu.tw/var/file/71/1071/attach/65/pta_9086_6330280_65743.pdf.