Элементы анализа чувствительности с использованием GLPK

Пономарев А.В.

В данном документе изложены некоторые приемы исследования чувствительности решения задачи линейного программирования, полученного решения с использованием пакета GLPK. Рекомендуется перед чтением этого документа ознакомиться с документом «Элементы анализа чувствительности с использованием GNU Octave», поскольку там содержится обсуждение основных характеристик чувствительности – теневой цены (shadow cost) и приведенной цены (reduced cost).

Для работы с GLPK под Windows рекомендуется загрузить среду GUSEK по ссылке на сайте, распаковать архив (установка не требуется) и найти в нем программу `glpsol`. Доступ к большинству функций солвера можно получить и из среды GUSEK, но в данном документе мы будем использовать только интерфейс командной строки (программу `glpsol`).

Постановка задачи

Фабрика производит три вида продукции: П1, П2 и П3. Известна цена на продукцию для распространителей и приблизительный спрос на каждый из видов продукции в неделю (см. Таблицу 1). Процессы производства продукции разных видов имеют отличия. На фабрике есть три цеха: Ц1, Ц2 и Ц3. Для производства продукции П1 необходимы только технологические операции, производимые цехом Ц1, для П2 – Ц1 и Ц3, для производства П3 – необходима полная технологическая цепочка, включающая обработку во всех трех цехах. Причем, если в цехах Ц1 и Ц2 продукция разных видов обрабатывается одинаково, и известна общая производительность этих цехов в единицах обработанной продукции в неделю, то в цехе Ц3 предполагается ручная обработка (см. Таблицу 2). Из всех видов материалов, используемых при производстве продукции, ограниченным является только один, поставки его в неделю и потребности для каждого из видов продукции приведены в таблице 3.

Необходимо составить производственный план на неделю, максимизирующий выручку от реализации продукции.

Таблица 1 – Характеристики продукции

Вид продукции	Цена, руб.	Спрос, шт. в неделю
П1	1200	35
П2	2500	25
П3	1400	30

Таблица 2 – Производительность цехов

Ц1, шт. в неделю	Ц2, шт. в неделю	Ц3, часов (П2/П3/Общий фонд)
40	20	8/2/80

Таблица 3 – Материалы

Поставки в неделю, кг	Потребление на ед. продукта П1, кг	Потребление на ед. продукта П2, кг	Потребление на ед. продукта П3, кг
50	0,8	0,6	0,7

Формальная постановка

Пусть x_i – количество единиц продукции i-того вида, которое необходимо произвести за неделю ($i \in \{1,..,3\}$).

Тогда условие задачи можно формально записать следующим образом:

$$1200x_1 + 2500x_2 + 1400x_3 \rightarrow max$$
$$x_1 + x_2 + x_3 \leq 40 \quad (1)$$
$$x_3 \leq 20 \quad (2)$$
$$8x_2 + 2x_3 \leq 80 \quad (3)$$
$$x_1 \leq 35 \quad (4)$$
$$x_2 \leq 25 \quad (5)$$
$$x_3 \leq 30 \quad (6)$$
$$0{,}8x_1 + 0{,}6x_2 + 0{,}7x_3 \leq 50 \quad (7)$$
$$x_{1,2,3} \geq 0$$

Ограничения (1)-(3) диктуются производительностью цехов, ограничения (4)-(6) обусловлены спросом на продукцию, а (7) выражает ограничение на использование материала.

Решение задачи линейного программирования с помощью GLPK

Для решения задачи с помощью GLPK составим описание модели данной задачи на языке GNU MathProg Language.

```
param p{j in 1..3};
param d{i in 1..3};
param c{i in 1..3};

var x{i in 1..3} >= 0;

maximize z : sum{i in 1..3} c[i]*x[i];

s.t. Prod1: x[1] + x[2] + x[3] <= p[1];
s.t. Prod2: x[3] <= p[2];
s.t. Prod3: 8*x[2] + 2*x[3] <= p[3];
s.t. Demand{i in 1..3}: x[i] <= d[i];
s.t. Mat: 0.8*x[1] + 0.6*x[2] + 0.7*x[3] <= 50;

data;

param p:= 1 40 2 20 3 80;
param d:= 1 35 2 25 3 30;
param c:= 1 1200 2 2500 3 1400;

end;
```

Поместим этот текст в файл `factory_plan.mod`. Для получения оптимального плана производства необходимо запустить программу `glpsol`, представляющую собой интерфейс командной строки к возможностям решателя GLPK:

```
glpsol -m factory_plan.mod -o solution.txt
```

Флаг «`-o`» означает, что сведения о решении должны быть помещены в соответствующий файл (в данном случае, `solution.txt`). В результате выполнения такой команды в терминал (в поток стандартного вывода) будет выведен отчет о выполнении:

```
GLPSOL: GLPK LP/MIP Solver, v4.55
Parameter(s) specified in the command line:
 -m factory_plan.mod -o solution.txt
```

```
Reading model section from factory_plan.mod...
Reading data section from factory_plan.mod...
21 lines were read
Generating z...
Generating Prod1...
Generating Prod2...
Generating Prod3...
Generating Demand...
Generating Mat...
Model has been successfully generated
GLPK Simplex Optimizer, v4.55
8 rows, 3 columns, 15 non-zeros
Preprocessing...
3 rows, 3 columns, 8 non-zeros
Scaling...
 A: min|aij| = 6.000e-001  max|aij| = 8.000e+000  ratio = 1.333e+001
Problem data seem to be well scaled
Constructing initial basis...
Size of triangular part is 3
*     0: obj =  0.000000000e+000  infeas = 0.000e+000 (0)
*     2: obj =  6.100000000e+004  infeas = 0.000e+000 (0)
OPTIMAL LP SOLUTION FOUND
Time used:   0.0 secs
Memory used: 0.1 Mb (112677 bytes)
Writing basic solution to 'solution.txt'...
```

Главная информация, которую мы пока что можем получить из отчета, это то что: а) `glpsol` не обнаружил ошибок в файле `factory_plan.mod`, б) было найдено оптимальное решение, в) решение было записано в файл `solution.txt` (как мы и хотели). Содержимое файла `solution.txt` приведено ниже:

```
Problem:    factory_plan
Rows:       8
Columns:    3
Non-zeros:  15
Status:     OPTIMAL
Objective:  z = 61000 (MAXimum)

   No.   Row name   St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------ ------------- ------------- -------------
     1 z            B          61000
     2 Prod1        NU            40                         40          1200
     3 Prod2        B              0                         20
     4 Prod3        NU            80                         80         162.5
     5 Demand[1]    B             30                         35
     6 Demand[2]    B             10                         25
     7 Demand[3]    B              0                         30
     8 Mat          B             30                         50

   No. Column name  St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------ ------------- ------------- -------------
     1 x[1]         B             30             0
     2 x[2]         B             10             0
     3 x[3]         NL             0             0                        -125

Karush-Kuhn-Tucker optimality conditions:

KKT.PE: max.abs.err = 0.00e+000 on row 0
        max.rel.err = 0.00e+000 on row 0
        High quality
```

```
KKT.PB: max.abs.err = 0.00e+000 on row 0
        max.rel.err = 0.00e+000 on row 0
        High quality

KKT.DE: max.abs.err = 0.00e+000 on column 0
        max.rel.err = 0.00e+000 on column 0
        High quality

KKT.DB: max.abs.err = 0.00e+000 on row 0
        max.rel.err = 0.00e+000 on row 0
        High quality

End of output
```

В этом файле можно выделить три основных блока (сверху вниз): 1) общая информация о решении, 2) информация о значениях переменных и ограничений в оптимальном решении, включая теневые и приведенные цены, 3) условия оптимальности Каруша-Куна-Таккера.

В первую очередь, нас интересует второй блок, рассмотрим его более подробно. Этот блок, в свою очередь включает две таблицы: в первой содержится информация о состоянии ограничений, во второй – о состоянии переменных. Перечень столбцов для ограничений и переменных приблизительно одинаков, назначение их следующее:

- No. – порядковый номер ограничения (переменной);
- Row name – название ограничения, заданное в файле модели;
- Column name – название переменной, заданное в файле модели;
- St – статус ограничения или переменной:
    - - BS – неактивное ограничение, базисная переменная;
    - - NL – неравенство с активным ограничением снизу (>=), небазисная переменная, значение которой находится на нижней границе;
    - - NU – неравенство с активным ограничением сверху (<=), небазисная переменная, значение которой находится на верхней границе;
    - - NS – активное ограничение равенства, фиксированная небазисная переменная;
    - - NF – активное свободное ограничение, небазисная свободная переменная.
- Activity – значение левой части ограничения или значение переменной;
- Lower bound и upper bound – нижняя и верхние границы для ограничений и переменных;
- Marginal – теневая цена (shadow price) для ограничений и приведенная цена (reduced cost) для переменных.

Исследование задачи на диапазоне исходных данных

Определенные сведения, касающиеся анализа чувствительности, можно получить, передав параметр –ranges <filename> при вызове glpsol. В файл, переданный в качестве параметра, будет помещен отчет об анализе чувствительности, в частности, содержащий диапазоны осуществимости для всех ограничений. Для более «масштабного» исследования можно применять комплексные методики, включающие написание вспомогательных скриптов.

В данном подразделе мы произведем исследование задачи при различных значениях производительности цехов (параметр p модели).

В первую очередь, полностью отделим модель от данных. В файле модели будет только описание параметров модели, ее переменных и ограничений, а в файле данных – значения параметров.

Поместим следующий текст в файл sens_lp.mod:

```
param p{j in 1..3};
```

```
param d{i in 1..3};
param c{i in 1..3};

var x{i in 1..3} >= 0;

maximize z : sum{i in 1..3} c[i]*x[i];

s.t. Prod1: x[1] + x[2] + x[3] <= p[1];
s.t. Prod2: x[3] <= p[2];
s.t. Prod3: 8*x[2] + 2*x[3] <= p[3];
s.t. Demand{i in 1..3}: x[i] <= d[i];
s.t. Mat: 0.8*x[1] + 0.6*x[2] + 0.7*x[3] <= 50;

solve;

printf ">>>>>>>TOKEN>>>>>>:";
for {i in 1..3}: printf "%f\t", x[i];
printf "Obj: %f\n", z;

#printfdisplay x[1], x[2], x[3];

end;
```

Обратите внимание, что в файл модели мы включили еще строки, осуществляющие вывод специальной строки-маркера `">>>>>>>TOKEN>>>>>>:"` и значений переменных и целевой функции в оптимальном решении.

Далее создадим два файла с данными: `sens_lp.dat` и `sens_lp_var.dat`. В первый поместим неизменные параметры *c* и *d*, а во второй – варьируемый параметр *p*.

Файл sens_lp.dat:

```
param d:= 1 35 2 25 3 30;
param c:= 1 1200 2 2500 3 1400;
end;
```

Файл sens_lp_var.dat:

```
param p:= 1 60 2 20 3 80;
end;
```

Вызовем решатель, передав ему в качестве параметров файл модели и оба файла с данными:

```
glpsol.exe -m sens_lp.mod -d sens_lp.dat -d sens_lp_var.dat
```

В результате будет сформирован отчет приблизительно следующего вида:

```
GLPSOL: GLPK LP/MIP Solver, v4.55
Parameter(s) specified in the command line:
 -m sens_lp.mod -d sens_lp.dat -d sens_lp_var.dat
Reading model section from sens_lp.mod...
25 lines were read
Reading data section from sens_lp.dat...
5 lines were read
Reading data section from sens_lp_var.dat...
2 lines were read
Generating z...
Generating Prod1...
Generating Prod2...
Generating Prod3...
Generating Demand...
```

```
Generating Mat...
Model has been successfully generated
GLPK Simplex Optimizer, v4.55
8 rows, 3 columns, 15 non-zeros
Preprocessing...
3 rows, 3 columns, 8 non-zeros
Scaling...
 A: min|aij| = 6.000e-001  max|aij| = 8.000e+000  ratio = 1.333e+001
Problem data seem to be well scaled
Constructing initial basis...
Size of triangular part is 3
*     0: obj =  0.000000000e+000  infeas = 0.000e+000 (0)
*     3: obj =  8.250000000e+004  infeas = 0.000e+000 (0)
OPTIMAL LP SOLUTION FOUND
Time used:   0.0 secs
Memory used: 0.1 Mb (112672 bytes)
>>>>>>>TOKEN>>>>>>:35.000000    5.000000     20.000000    Obj: 82500.000000
Model has been successfully processed
```

Мы видим, что, во-первых, было найдено оптимальное решение (строка OPTIMAL LP SOLUTION FOUND), во-вторых, была сформирована строчка со значениями переменных и целевой функции.

Осталось автоматизировать эту процедуру. Одним из удобных языков для автоматизации подобного рода является Python. Простой скрипт, осуществляющий перебор производительности первого цеха и вывод оптимальных значений плана для каждого варианта производительности может быть таким:

```
001. # -*- coding: utf-8 -*-
002.
003. import subprocess
004. import os
005.
006. CMD = "glpsol -m sens_lp.mod -d sens_lp.dat -d sens_lp_var.dat -o o > l 2> e"
007.
008. def get_results():
009.     """
010.     Возвращает кортеж из содержимого трех файлов (l, o, e). Предполагается, что
011.     именно в эти файлы были перенаправлены стандартный поток вывода, печать
012.     результата, и стандартный поток ошибок при вызове glpsol.
013.     Если какого-то из файлов не окажется, будет выброшено исключение.
014.     """
015.     with open('o') as f:
016.         out = f.readlines()
017.     with open('e') as f:
018.         err = f.readlines()
019.     with open('l') as f:
020.         log = f.readlines()
021.     return (log, out, err)
022.
023. def clean_results():
024.     """
025.     Производит ненавязчивые попытки удалить все файлы, порожденные в результате
026.     выполнения glpsol.
027.     """
028.     for name in ['o', 'e', 'l']:
029.         try:
030.             os.remove(name)
031.         except Exception, e:
032.             print e
033.
034. def optimal_solution_found(log, out, err):
```

```
035.     """
036.     На основе анализа выходных файлов glpsol определяет, было ли найдено
037.     оптимальное решение.
038.     """
039.     return "OPTIMAL LP SOLUTION FOUND\n" in log
040.
041. def get_marked_line(log):
042.     """
043.     Извлекает из выходных данных glpsol строчку, выведенную со
044.     специальным префиксом
045.     """
046.     token = ">>>>>>>TOKEN>>>>>>:"
047.     for line in log:
048.         if line.startswith(token):
049.             return line.strip()[len(token):]
050.
051. def run_glpsol():
052.     retcode = subprocess.call(CMD, shell=True)
053.     try:
054.         if retcode == 0:
055.             log, out, err = get_results()
056.             if optimal_solution_found(log, out, err):
057.                 return get_marked_line(log)
058.             else:
059.                 print "Couldn't find optimal solution or log parsing error"
060.     finally:
061.         clean_results()
062.
063. if __name__ == "__main__":
064.     for i in range(40, 61):
065.         with open("sens_lp_var.dat", "w") as f:
066.             f.write("param p:= 1 %d 2 20 3 80;\nend;\n" % (i, ))
067.         print run_glpsol()
```

Основное тело скрипта находится в строках 064-067, здесь осуществляется перебор значений в диапазоне от 40 до 60, для каждого из значений открывается на запись файл с варьируемыми параметрами `sens_lp_var.dat`, в него записывается строка, сформированная в соответствии с синтаксисом GMPL, в которую подставляется тестируемое значение параметра. Видно также, что подстановка происходит в первый элемент параметра *p*, а значит, варьируем мы производительность первого цеха. Далее происходит вызов функции `run_glpsol()` и печать возвращенного этой функцией значения.

Функция `run_glpsol()` является простой «оберткой» для вызова `glpsol.exe` с использованием встроенной библиотеки `subprocess`. Важной информацией, которая используется этой функцией, но не передается ей в качестве параметров, является строка, посредством которой происходит вызов `glpsol.exe` – эта строка сформирована в глобальной переменной `CMD` (строка 006).

Функция `get_results()` создает три списка строк, соответствующих строкам журнала (выводимого `glpsol.exe` в стандартный поток вывода), выходного файла и потока ошибок. На основе анализа содержимого этих списков делается вывод о том, было ли найдено оптимальное решение.

При запуске скрипта предполагается, что `glpsol.exe` находится в папке GUSEK, которая, в свою очередь, находится в папке скрипта.