Cheat sheet GLPK
(separación de los datos y el modelo)

En este apunte vamos a ver como separar un mismo problema en dos partes, el modelo y los datos. Esto tiene múltiples beneficios en especial para problemas combinatorios y además aporta claridad a lo que estamos haciendo.

¿Qué va en el modelo?

- El modelo en sí, restricciones, el funcional, declaración de variables.
- Declaración de conjuntos (set), constantes que dependen de esos conjuntos (param).

¿Qué va en los datos?

- Definición de los conjuntos.
- Definición de las constantes relacionadas con esos conjuntos.

Ejemplo

Supongamos que en un determinado problema tenemos 2 recursos usados para producir productos. Supongamos también que esos 2 productos pueden venir de 2 proveedores diferentes (cada uno con una cantidad máxima que puede ofrecer).

Definiendo C_a como la cantidad de materia prima A que compro, y lo mismo para B y sabiendo que de A se puede obtener 50 del proveedor1 y 50 del 2 y de B 35 y 70.

En el modelo quedaría así:

$C_a <= 50 + 50$

$C_b <= 35 + 70$

Pasemos eso a GLPK:

```
var CA >= 0;
var CB >= 0;
s.t. MaxA: CB <= 50 + 50
s.t. MaxB: CA <= 35 + 70
```

Ahora, ¿Qué pasaría si quisiéramos agregar una nueva materia prima? Debería agregar la declaración de la variable, y la restricción correspondiente a la cantidad máxima que se puede comprar. Bien, ¿Y si quisiéramos agregar un proveedor? Tendríamos que sumar para cada materia prima lo que aporta este nuevo proveedor de cada una. Suena tedioso ¿no?

Alguien entrenado en modelización dirá: “Bueno, la cantidad máxima que puedo adquirir de cada materia prima es la sumatoria de lo que ofrece cada proveedor de ella”, ¡Genial! Entonces el modelo quedaría así:

$$C_i \leqslant \sum_{j=1}^{n} P_{ij} \qquad \forall i$$

Siendo i las materias primas (A, B), P_{ij} lo que ofrece el proveedor j de la materia prima i (ejemplo $P_{A2} = 50$) y n la cantidad de proveedores.

Todo muy bonito, hemos resuelto, a nivel modelo, el problema de manera genérica. La pregunta

del millón es entonces ¿Cómo hacemos eso en GLPK? Aprendamos cosas nuevas...

Set:

Se usa para definir conjuntos, por ejemplo si se tienen 2 tipos de materia prima A y B.

En los datos:

```
set mp := A B;
```

En el modelo:

```
set mp;
```

De esta forma si quisiéramos agregar una materia prima, sólo cambiamos en los datos el set mp:

```
set mp := A B C;
```

Param:

Se usa para definir constantes que tienen relación con los conjuntos. Por ejemplo si tenemos el set mp de antes y queremos representar los distintos proveedores de la materia prima.

En los datos:

```
set providers := P1 P2;
param prov :  A     B     C :=
        P1    50    35    0
        P2    50    75    20;
```

Es decir, el proveedor 1 puede proveer 50 de A, 35 de B y 0 de C (ídem para el resto de los proveedores).

Con esto, si queremos agregar un proveedor sólo tenemos que agregar la siguiente línea en param prov:

```
        P3    0     0     100;
```

(el punto y coma tiene que estar al final de la declaración, como si fuera lenguaje C, por lo que hay que sacar el ';' después del 20)

Y agregar P3 al set providers:

```
set providers := P1 P2 P3;
```

En el modelo declaramos las constantes “Pij”:

```
param prov{j in mp, i in providers};
```

(notar que las i de Pij son las columnas [j] de la matriz y las j de Pij son las filas [i], también se puede declarar la matriz traspuesta a como se expresó acá y poner

param prov{i in mp, j in providers})

Declaramos las variables que indican la cantidad que compramos de cada materia prima:

```
var C{i in mp} >=0;
```

Luego tenemos la restricción de cantidad máxima:

```
s.t. MaxMP{i in mp}: C[i] <= sum {j in providers}  prov[i,j];
```

En limpio quedaría:

Modelo:

```
set mp;
set providers;

var C{i in mp} >=0;
param prov{i in providers, j in mp};
#Agregamos el param de los costos
param costo{i in mp};

#Agregamos un funcional que maximiza la cantidad a comprar
maximize z: sum{i in mp} C[i];


s.t. MaxMP{i in mp}: C[i] <= sum {j in providers}  prov[j,i];
#Agregamos una restricción de límite de dinero
s.t. LimPesos: sum{i in mp} costo[i]*C[i] <= 500;

#DATOS:
data;

set mp := A B C;
set providers := P1 P2 P3;

param prov : A     B     C :=
            P1    50    35    0
            P2    50    75    20
            P3    0     0     100;

param costo :=
      A 4
      B 5
      C 7;

end;
```

Ejemplo del viajante

Veremos a continuación un modelo del viajante utilizando estos conceptos.

Modelo matemático:

Yij = Bivalente que indica si el viajante va desde la ciudad i a la ciudad j.
Ui = Número de secuencia en el que la ciudad i es visitada. (i >0)
n = cantidad de nodos a visitar, es constante.

$$Z_{min} = \sum_{\substack{i=0 \\ i \neq j}}^{n} \sum_{j=0}^{n} C_{ij} * Y_{ij}$$

$$Salgo\ del\ nodo\ i\ hasta\ un\ sólo\ lugar: \sum_{j=0}^{n} Y_{ij} = 1 \qquad \forall\, i = 0\ ..\ n$$

$$Llego\ al\ nodo\ \ j\ desde\ un\ sólo\ lugar: \sum_{i=0}^{n} Y_{ij} = 1 \qquad \forall\, j = 0\ ..\ n$$

$$Secuencia: U_i - U_j + n * Y_{ij} \leq n-1 \qquad \forall\, i, \forall\, j,\ i \neq j \quad i, j = 1\ ..\ n$$

Modelo GLPK:

```
#Conjuntos:
#Las ciudades del viajante
set CIUDADES;

#Parámetros (o constantes):
# Los costos de ir de i a j, i tiene que ser distinto de j
param COSTO{i in CIUDADES, j in CIUDADES : i<>j};

#Variables:
#Yij, bivalente que vale 1 si va desde la ciudad i hasta la j (i != j)
var Y{i in CIUDADES, j in CIUDADES: i<>j} >= 0, binary;

#Ui orden de secuencia en que la ciudad i es visitada (excluyendo el punto de partida que
en este caso lo tomamos como A). Declarada entera sólo a efectos de mostrar la
declaración.
var U{i in CIUDADES: i<>'A'} >=0, integer;

#Funcional:
minimize z: sum{i in CIUDADES, j in CIUDADES : i<>j} COSTO[i,j]*Y[i,j];

#Llego desde un solo lugar hasta la ciudad j
s.t. llegoJ{j in CIUDADES}: sum{i in CIUDADES: i<>j} Y[i,j] = 1;
#Voy hacia un sólo lugar desde i
s.t. voyI{i in CIUDADES}: sum{j in CIUDADES: i<>j} Y[i,j] = 1;
#Secuencia para evitar subtours
s.t. orden{i in CIUDADES, j in CIUDADES: i<>j and i<>'A' and j<>'A'}: U[i] - U[j] +
card(CIUDADES)*Y[i,j] <= card(CIUDADES) - 1;

solve;

#Data segment
data;

set CIUDADES:= A B C D E F;

param COSTO :  A B C D E F :=
            A  . 2 4 1 5 8
            B  5 . 5 7 1 3
            C  2 4 . 3 7 1
            D  2 9 2 . 8 4
            E  3 6 7 1 . 4
            F  5 7 2 6 3 .;

end;
```

Algunos detalles:

- Recordar que la declaración de variables necesita el >= 0 (para cumplir la condición de no negatividad).
- card(nombre del conjunto) es una función que devuelve la cantidad de elementos del conjunto.
- binary es para declarar variables bivalentes.
- integer es para declarar variables enteras (por defecto las variables son continuas)
- El punto en el param COSTO indica que ahí no hay datos (si no estuviesen tomaría por ejemplo el 2 de la primer fila como el costo de ir desde A hasta A).
- Sentencia data, que sirve para indicar al software que a partir de ahí vienen los datos.
- El input de datos puede hacerse en un archivo aparte (y ahí sí tenemos una verdadera separación de los datos). Para ellos se crea un archivo .dat cuyo nombre sea igual al

nombre del archivo .mod (¡¡¡ No olvidar la sentencia data; !!!).

- Si se usa un archivo de datos separado:
    - En Gusek: Ir a “Tools” y marcar la opción “Use External .dat”
    - En consola: glpsol -m [nombre].mod -d [nombre].dat -o [nombre].sol
- Para que Gusek muestre la salida luego de correr el modelo ir a Tools->Generate Output File on Go.