Tema 9: Aproximación lineal de problemas no lineales

Nota importante:

El tema 9 forma parte del programa oficial y su contenido puede ser de mucha utilidad para modelar comportamientos no lineales que aparecen con frecuencia en el mundo industrial. Por ello es muy recomendable que los alumnos estudien su contenido y se inicien en el uso de técnicas de programación entera y restricciones especiales para aproximar comportamientos no-lineales. Sin embargo **en este tema no habrá que entregar ningún ejercicio.**

Objetivos del tema:

- ❑ Introducir el concepto de convexidad aplicados a la función objetivo y a las restricciones de un problema de optimización.
- ❑ Introducir el concepto de programación separable.
- ❑ Definir e implementar las restricciones sobre conjuntos ordenados de variables SOS1 y SOS2.
- ❑ Aproximar una función no-lineal de una variable por una lineal a tramos utilizando SOS2 y/o variables binarias.
- ❑ Aproximar una función no-lineal de dos variables utilizando SOS2.

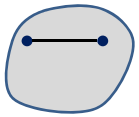
Programación matemática no-lineal

Los problemas de programación lineal son muy comunes y cubren un amplio rango de aplicaciones. Sin embargo, en la vida real aparecen frecuentemente problemas que no tienen comportamientos lineales. Cuando alguna de las restricciones, la función objetivo, o ambos, son no lineales, se dice que se trata de un *problema de programación no lineal.* Los modelos de programación no-lineal son más complejos de resolver que sus análogos lineales. Sin embargo muchos de ellos es posible resolverlos mediante aproximaciones lineales.

Región convexa

Se dice que una región del espacio es convexa si el segmento que une dos puntos cualesquiera de la región está íntegramente en la región.

región convexa

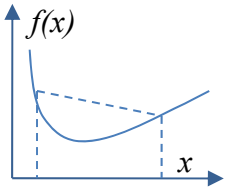


región no-convexa

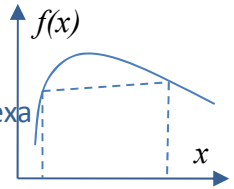
Función convexa

Se dice que una función es convexa si el conjunto de los puntos (x, y) donde $y \geq f(x)$ forma un región convexa.

función convexa



función no-convexa



Modelo de programación matemática convexo

Se dice que un modelo de Programación Matemática es convexo si implica la minimización de una función convexa sobre una región convexa.

modelo de programación matemática convexo

$$\begin{aligned} &Minimizar && x_1^2 - 4x_1 - 2x_2 \\ &sujetoa: && x_1 + x_2 \leq 4 \\ & && 2x_1 + x_2 \leq 5 \\ & && -x_1 + 4x_2 \geq 2 \\ & && x_1, x_2 \geq 0 \end{aligned}$$

modelo de programación matemática no-convexo

$$\begin{aligned} &Minimizar && -4x_1^3 + 3x_1 - 6x_2 \\ &sujetoa: && x_1 + x_2 \leq 4 \\ & && 2x_1 + x_2 \leq 5 \\ & && -x_1 + 4x_2 \geq 2 \\ & && x_1, x_2 \geq 0 \end{aligned}$$

En el segundo ejemplo la región de factibilidad es convexa, aunque la función objetivo no lo es, por lo que el modelo es no-convexo.

Si el problema es convexo cualquier óptimo que se encuentre es óptimo global. Sin embargo encontrar un óptimo global en un problema no-convexo requiere algoritmos mucho más sofisticados.

Función separable

Una función separable es aquella que puede expresarse como la suma de funciones de una única variables. Estas funciones tienen la ventaja de que los términos no lineales pueden ser aproximados por términos lineales a tramos (*piecewise*). Utilizando esta técnica se puede escribir un programa lineal entero, e incluso lineal continuo para estas funciones. El término separable puede aparecer en la función objetivo o en las restricciones de un problema de optimización.

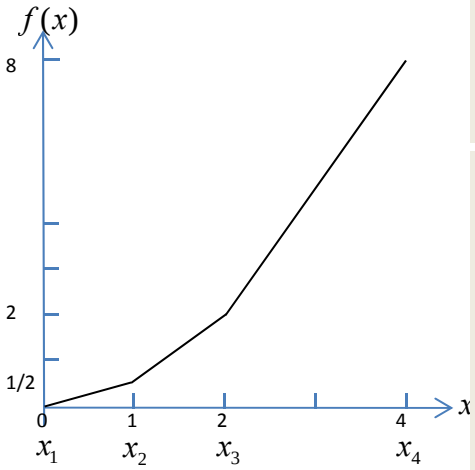
Algunos ejemplos de funciones separables son:

$$x_2 + 5x_1 = g_1(x_1) + g_2(x_2)$$
$$x_2 + 1/x_2 - 2x_3 = f_1(x_1) + f_2(x_2) + f_3(x_3)$$

En cambio, no son separables las siguientes:

$$x_1 x_2 + 3x_2 = g_1(x_1, x_2) + g_2(x_2)$$
$$1/(x_1 + x_2) + x_3 = f_1(x_1, x_2) + f_2(x_3)$$

Consideremos un ejemplo simple con sólo un término no-lineal para ser aproximado: f(x) = 1/2x². La siguiente figura muestra la curva dividida en tres tramos que son aproximados por líneas rectas. Los puntos donde cambia la pendiente de la función lineal por tramos se conocen como puntos de ruptura. Esta aproximación se puede expresar matemáticamente de varias formas. Veremos el método conocido como formulación-λ



Sean x_1, x_2, x_3 y x_4 los cuatro puntos de ruptura del eje x y sean $f(x_1)$, $f(x_2)$, $f(x_3)$ y $f(x_4)$ sus correspondientes valores. Cualquier punto entre dos de ruptura es una suma ponderada de los dos puntos de ruptura extremos. Por ejemplo, si los puntos de ruptura son 0, 1, 2 y 4, y los correspondientes valores son 0, 1/2 , 2 y 8, el valor de $x = 3$ se puede expresar: $x = 1/2 \cdot 2 + 1/2 \cdot 4 = 3$. El correspondiente valor de la función aproximada será: $f^a(3) = 1/2 \cdot 2 + 1/2 \cdot 5 = 5$

Sean λ_1, λ_2, λ_3, λ_4 cuatro pesos tal que su suma valga *1.* La aproximación lineal de *f(x) puede escribirse como:*

$$f^a(x) = \lambda_1 f(x_1) + \lambda_2 f(x_2) + \lambda_3 f(x_3) + \lambda_4 f(x_4)$$
$$x = \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3 + \lambda_4 x_4$$
$$\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 = 1$$
$$SOS2(\lambda_1, \lambda_2, \lambda_3, \lambda_4)$$

La restricción SOS2 impone el requisito añadido de que como mucho dos λ_i adyacentes sean mayores que 0.

Ejemplo de modelo con función separable:

$$\begin{aligned} &\textit{Minimizar} && x_1^2 - 4x_1 - 2x_2 \\ &\textit{sujetoa}: && x_1 + x_2 \le 4 \\ & && 2x_1 + x_2 \le 5 \\ & && -x_1 + 4x_2 \ge 2 \\ & && x_1, x_2 \ge 0 \end{aligned}$$

Solo es necesario convertir la función x_1^2

De la segunda restricción es evidente que no es posible que x_1 sea mayor que 2.5, por tanto, la función: $y=x_1^2$ $\quad x_1 \in [0, 2.5]$ puede aproximarse por:

$$y = \begin{cases} x & x \in [0,1] \\ 3x - 2 & x \in [0,2] \\ \dfrac{2.25}{0.5}x - 5 & x \in [2, 2.5] \end{cases}$$

Esta expresión es equivalente a:

$$\begin{aligned} x_1 &= 0\lambda_1 + 1\lambda_2 + 2\lambda_3 + 2.5\lambda_4 \\ y &= 0\lambda_1 + 1\lambda_2 + 4\lambda_3 + 6.25\lambda_4 \\ \lambda_1 &+ \lambda_2 + \lambda_3 + \lambda_4 = 1 \end{aligned}$$

Por tanto el modelo del ejemplo será el siguiente:

$$\begin{aligned} &\textit{Minimizar} && \mathrm{y} - 4x_1 - 2x_2 \\ &\textit{sujeto a}: && x_1 + x_2 \le 4 \\ & && 2x_1 + x_2 \le 5 \\ & && -x_1 + 4x_2 \ge 2 \\ & && -x_1 + 1\lambda_2 + 2\lambda_3 + 2.5\lambda_4 = 0 \\ & && -y + 1\lambda_2 + 4\lambda_3 + 6.25\lambda_4 = 0 \\ & && \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 = 1 \\ & && \mathrm{y}, x_1, x_2, \lambda_1, \lambda_2, \lambda_3, \lambda_4 \ge 0 \end{aligned}$$

Dado que la función es convexa no es necesario añadir ninguna restricción más. Si la función fuera no-convexa sería necesario añadir una restricción de tipo SOS2 a las variables λ_i

La restricción SOS2 suele aparecer con frecuencia en problemas reales y por ello suele venir implementadas en los paquetes comerciales junto a la restricción SOS1.

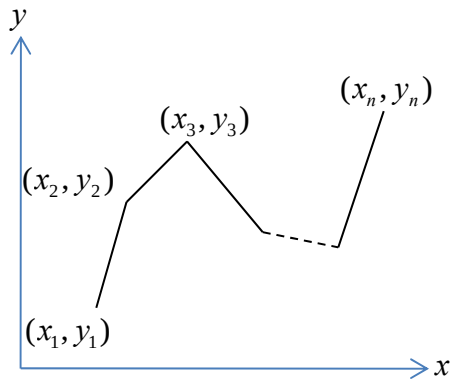
Restricción SOS1
Impuesta a un conjunto de variables continuas obliga a que sólo una pueda adoptar un valor no nulo.

Restricción SOS2
Impuesta a un conjunto de variables continuas ordenadas obliga a que sólo dos variables pueden tener valor no-nulo y además deben ser consecutivas. La restricción SOS2 queda definida especificando un conjunto de variable { t_1, t_2, … ,t_n } y es equivalente a las siguientes tres restricciones:

- $t_1, t_2, ..., t_n \geq 0$
- $t_1 + t_2 + ... + t_n = 1$
- Sólo dos variables adyacentes, t_i y t_{i+1} pueden ser distintas de cero.

Como ya hemos comentado, la restricción SOS2 se utiliza para aproximar términos no-lineales con funciones lineales por tramos (*piecewise*) en programación separable convexa y no-convexa.

Se trata de modelar la función continua lineal por tramos $y = f(x)$ especificada por sus n puntos (x_1, y_1), (x_2, y_2), ... ,(x_n, y_n) como se muestra en la figura. La descripción estándar utilizando la restricción SOS2 es la siguiente:



- $x = x_1 t_1 + x_2 t_2 + ... + x_n t_n$
- $y = y_1 t_1 + y_2 t_2 + ... + y_n t_n$
- $SOS2\{t_1, t_2, ..., t_n\}$

Donde las variables $t_1, t_2, ..., t_n$ de $SOS2$ juegan el papel de parámetros de interpolación.

Modelado de la función lineal por tramos (*picewise*)

Las funciones lineales a tramos expresadas con SOS2 se pueden utilizar para modelar no solo objetivos no-lineales, sino también restricciones de igualdad y desigualdad no-lineales. Por tanto, problemas de programación no-lineal pueden reformularse como problemas de programación entera-mixta.

Si no se dispone de la restricción SOS2 implementada en el Simplex, la aproximación lineal por tramos (*picewise*) se puede modelar utilizando variables binarias.

$z_i \in \{0,1\}, \quad s_i \in \Re$

- $z_1 + z_2 + ... + z_{n-1} = 1$
- $\forall i = 1,2,...,n\text{-}1 : 0 \le s_i \le z_i$
- $x =$

  $x_1 z_1 + (x_2 - x_1) s_1 +$

  $x_2 z_2 + (x_3 - x_2) s_2 +$

  ...

  $x_i z_i + (x_{i+1} - x_i) s_i +$

  ...

  $x_{n-1} z_{n-1} + (x_n - x_{n-1}) s_{n-1}$
- $y =$

  $y_1 z_1 + (y_2 - y_1) s_1 +$

  $y_2 z_2 + (y_3 - y_2) s_2 +$

  ...

  $y_i z_i + (y_{i+1} - y_i) s_i +$

  ...

  $y_{n-1} z_{n-1} + (y_n - y_{n-1}) s_{n-1}$

```
int n = 9;

float X[1..n] = [1,  2,3,4,5,6,7,8,9];
float Y[1..n] = [2,2.5,3,4,5,6,7,8,9];

dvar int z[1..n-1] in 0..1;
dvar float+ s[1..n-1];
dvar float x;
dvar float y;

subject to
{
  sum(i in 1..n-1)z[i] == 1;
  forall(i in 1..n-1)
  {
      s[i] <= z[i];
  }
  x == sum(i in 1..n-1)(X[i]*z[i] + (X[i+1]-X[i])*s[i]);
  y == sum(i in 1..n-1)(Y[i]*z[i] + (Y[i+1]-Y[i])*s[i]);
}
```

Modelado de la restricción SOS2 con variables binarias

La restricción SOS2 se puede implementar con la ayuda de variables binarias en caso de no disponer de ella directamente en el paquete optimización o en lenguaje de modelado .

$z_i \in \{0,1\}, \quad s_i \in \Re$

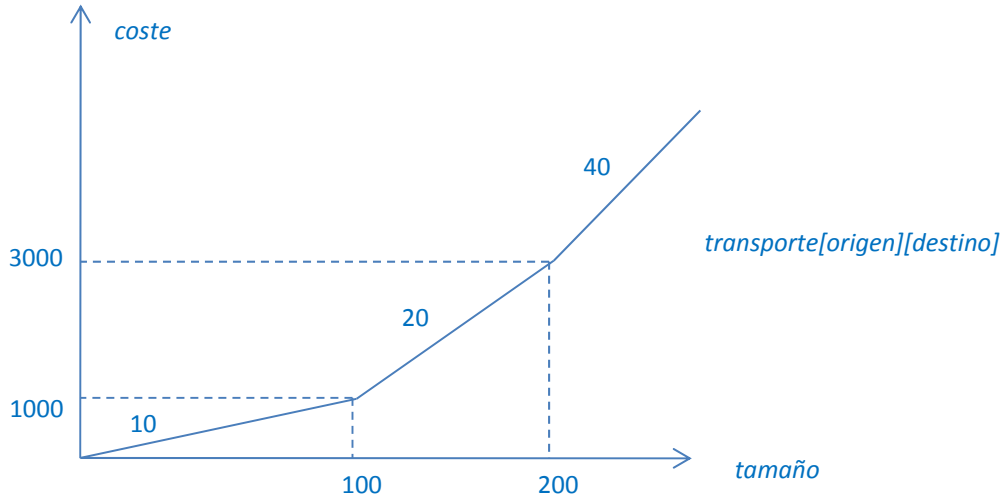
- $z_1 + z_2 + ... + z_{n-1} = 1$
- $\forall i = 1,2,...,n\text{-}1 : 0 \le s_i \le z_i$
- $\{t_1 =$
- $\begin{cases} t_1 = z_1 - s_1 \\ t_2 = z_2 - s_2 + s_1 \\ ... \\ t_i = z_i - s_i + s_{i-1} \\ ... \\ t_{n-1} = z_{n-1} - s_{n-1} + s_{n-2} \\ t_n = s_{n-1} \end{cases}$

```
int n = 9;

dvar int z[1..n-1] in 0..1;
dvar float+ s[1..n-1];

dvar float t[1..n];


subject to
{
  sum(i in 1..n-1)z[i] == 1;
  forall(i in 1..n-1)
  {
      s[i] <= z[i];
  }
  t[1] == z[1]-s[1];
  forall(i in 2..n-1)
  {
      t[i] == z[i]-s[i]+s[i-1];
  }
  t[n] == s[n-1];
}
```

Funciones lineales a tramos en OPL

Aunque CPLEX dispone de las restricciones SOS1 y SOS2 implementadas directamente en sus algoritmos de optimización lineal entera-mixta, el lenguaje de modelado OPL no las utiliza directamente sino a través de una construcción sintáctica específica para las funciones *piecewise*.

Para ver esta sintaxis consideremos un problema de transporte en el que el coste del transporte entre dos localidades *origen* y *destino* depende del tamaño del transporte *transporte[origen][destino]* de acuerdo con la siguiente gráfica:



Esta gráfica define una función que tiene pendientes 10, 20 y 40, con putos de ruptura 100 y 200, y con valor 0 en el punto 0.

En OPL esta función se expresa de la siguiente forma:

```
Piecewise{10 -> 100;20 ->200;40}(0,0) transporte[origen][destino]
```

Esta función es equivalente a la siguiente expresión:

```
10*transporte[origen][destino]                cuando transporte[origen,destino] <= 100
10*100+20*(trasnporte[origen][destino]-100)   cuando 100 <= transporte[origen][destino] <= 200
10*100+20*200+40*(ship[origen][destino]-200)  caso contrario.
```

Funciones lineales a tramos en OPL: generalización

La expresión anterior se puede generalizar de la siguiente forma:

```
piecewise(i in 1..n) { pendiente[i] -> ruptura[i]; pendiente[n+1];}
transporte[origen][destino];
```

Expresión general equivalente a la siguientes expresiones simples:

```
pendiente[1]*transporte[origen][destino]
                cuando transporte[origen][destino] <= breakpoint[1]
```

$$\texttt{pendiente[1]*ruptura[1]+}\sum_{i=2}^{k-1}\texttt{pendiente[i]*(ruptura[i]-ruptura[i-1])+}$$

```
pendiente[k]*(transporte[origen][destino]-ruptura[k-1])
                cuando ruptura[k-1] < transporte[origen][destino] <= ruptura[k] (1 < k <= n )
```

$$\texttt{pendiente[1]*ruptura[1]+}\sum_{i=1}^{n}\texttt{pendiente[i]*(ruptura[i]-ruptura[i-1])+}$$

```
pendiente[n+1]*(transporte[origen][destino]-ruptura[n])
                caso contrario
```

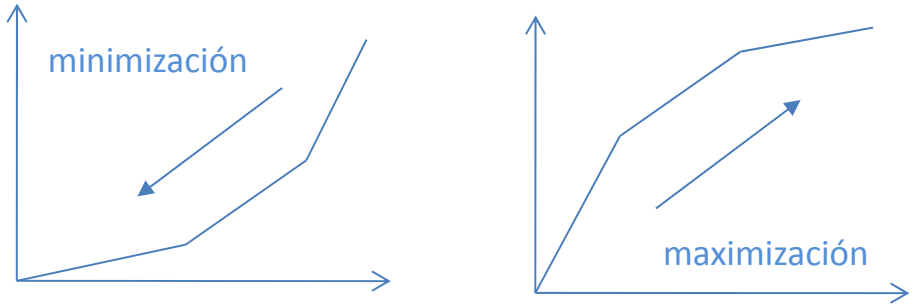
Ejemplo:

```
int n=2;
float objectivoparaxigual0 = 300;
float ruptura[1..n] = [100,200];
float pendiente[1..n+1] = [1,2,-3];
dvar int x;

maximize piecewise(i in 1..n)
{pendiente[i] -> ruptura[i];
pendiente[n+1]}(objectivoparaxigual0) x;

subject to
{
 true;
}
```

Funciones lineales a tramos en OPL

Por motivos de complejidad computacional es importante entender cuando una función objetivo lineal a tramos se traduce a un programa lineal y cuando es necesario utilizar un programa entero-mixto.

Las siguientes figuras describen las formas de las funciones que producen programas lineales puros: funciones lineales a tramos convexas en problemas de minimización y funciones lineales a tramos no-convexas (cóncavas)en problemas de maximización.



Para las restricciones operan consideraciones análogas.

Una función lineal a tramos convexa puede aparecer en el lado izquierdo de una desigualdad menor-o-igual-que, y sobre el lado derecho de una desigualdad mayor-o-igual-que.

Una función lineal a tramos cóncava puede aparecer en el lado derecho de una desigualdad menor-o-igual-que o sobre el lado izquierdo de una desigualdad mayor-o-igual-que.

En cualquier otro caso el programa lineal a tramos es transformado en un programa lineal entero-mixto.

Función lineal de dos o más variables

La restricción *SOS2* también se puede utilizar para aproximar una función no lineal de 2 variables, por ejemplo $z = g(x, y)$. Para ello se define una malla de valores (x, y) no necesariamente equidistantes y asociamos pesos no negativos λ_{ij} con cada punto de la malla. Si los valores de (x, y) en cada punto de la malla los designamos por (Xs, Yk), y es $g(Xs, Yk)$ el valor de la función $z = g(x, y)$ en dicho punto, podemos aproximarla por medio de las siguientes restricciones:

$$x = \sum_s \sum_k X_s \lambda_{sk}$$

$$y = \sum_s \sum_k Y_k \lambda_{sk}$$

$$z = \sum_s \sum_k g(X_s, Y_k) \lambda_{sk}$$

$$\sum_s \sum_k \lambda_{sk} = 1$$

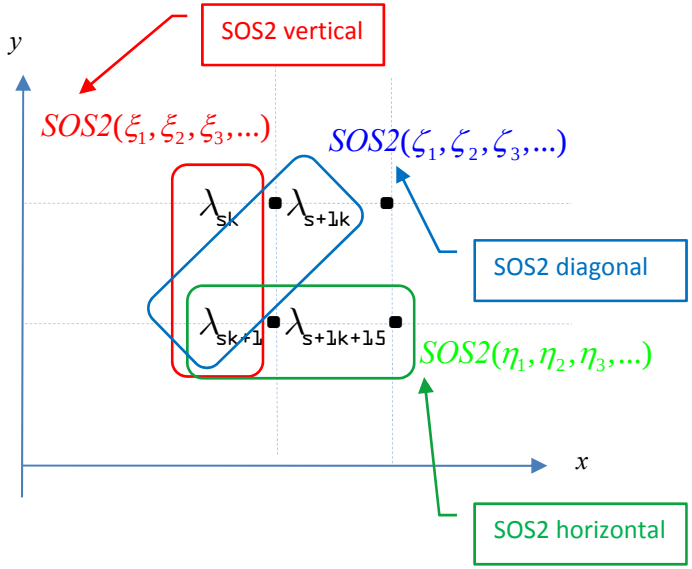
A lo más 4 λ_{sk} vecinas pueden ser distintas de cero

$$\xi_s = \sum_k \lambda_{sk} \qquad SOS2(\xi_1, \xi_2, \xi_3, \ldots)$$

$$\eta_k = \sum_s \lambda_{sk} \qquad SOS2(\eta_1, \eta_2, \eta_3, \ldots)$$

Restringe los λ_{sk} distintas de cero a 3 vecinos, evitando el problema de la falta de unicidad que surge si sólo imponemos las 2 SOS2 anteriores

$$\zeta_t = \sum_s \lambda_{s,t+s} \qquad SOS2(\zeta_1, \zeta_2, \zeta_3, \ldots)$$

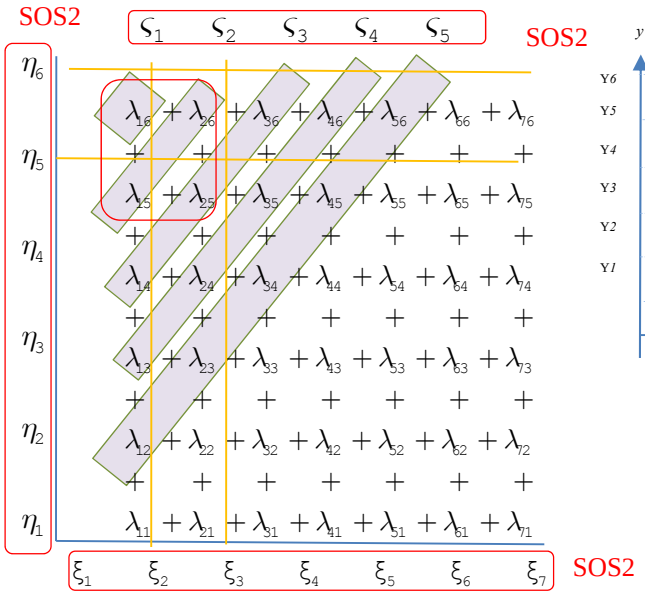


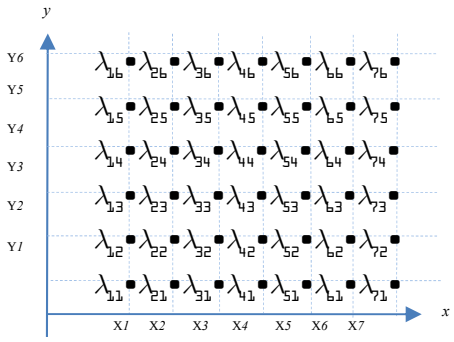
Restricciones para una malla de 7 x 6

$$
\begin{aligned}
x = {} & X_1\lambda_{16} + X_2\lambda_{26} + X_3\lambda_{36} + X_4\lambda_{46} + X_5\lambda_{56} + X_6\lambda_{66} + X_7\lambda_{76} + \\
& X_1\lambda_{15} + X_2\lambda_{25} + X_3\lambda_{35} + X_4\lambda_{45} + X_5\lambda_{55} + X_6\lambda_{65} + X_7\lambda_{75} + \\
& X_1\lambda_{14} + X_2\lambda_{24} + X_3\lambda_{34} + X_4\lambda_{44} + X_5\lambda_{54} + X_6\lambda_{64} + X_7\lambda_{74} + \\
& X_1\lambda_{13} + X_2\lambda_{23} + X_3\lambda_{33} + X_4\lambda_{43} + X_5\lambda_{53} + X_6\lambda_{63} + X_7\lambda_{73} + \\
& X_1\lambda_{12} + X_2\lambda_{22} + X_3\lambda_{32} + X_4\lambda_{42} + X_5\lambda_{52} + X_6\lambda_{62} + X_7\lambda_{72} + \\
& X_1\lambda_{11} + X_2\lambda_{21} + X_3\lambda_{31} + X_4\lambda_{41} + X_5\lambda_{51} + X_6\lambda_{61} + X_7\lambda_{71} +
\end{aligned}
$$

$$
\begin{aligned}
y = {} & Y_6\lambda_{16} + Y_6\lambda_{26} + Y_6\lambda_{36} + Y_6\lambda_{46} + Y_6\lambda_{56} + Y_6\lambda_{66} + Y_6\lambda_{76} + \\
& Y_5\lambda_{15} + Y_5\lambda_{25} + Y_5\lambda_{35} + Y_5\lambda_{45} + Y_5\lambda_{55} + Y_5\lambda_{65} + Y_5\lambda_{75} + \\
& Y_4\lambda_{14} + Y_4\lambda_{24} + Y_4\lambda_{34} + Y_4\lambda_{44} + Y_4\lambda_{54} + Y_4\lambda_{64} + Y_4\lambda_{74} + \\
& Y_3\lambda_{13} + Y_3\lambda_{23} + Y_3\lambda_{33} + Y_3\lambda_{43} + Y_3\lambda_{53} + Y_3\lambda_{63} + Y_3\lambda_{73} + \\
& Y_2\lambda_{12} + Y_2\lambda_{22} + Y_2\lambda_{32} + Y_2\lambda_{42} + Y_2\lambda_{52} + Y_2\lambda_{62} + Y_2\lambda_{72} + \\
& Y_1\lambda_{11} + Y_1\lambda_{21} + Y_1\lambda_{31} + Y_1\lambda_{41} + Y_1\lambda_{51} + Y_1\lambda_{61} + Y_1\lambda_{71} +
\end{aligned}
$$

$$
\begin{aligned}
z = {} & g(X_1, Y_6)\lambda_{16} + g(X_2, Y_6)\lambda_{26} + g(X_3, Y_6)\lambda_{36} + g(X_4, Y_6)\lambda_{46} + g(X_5, Y_6)\lambda_{56} + g(X_6, Y_6)\lambda_{66} + g(X_7, Y_6)\lambda_{76} + \\
& g(X_1, Y_5)\lambda_{15} + g(X_2, Y_5)\lambda_{25} + g(X_3, Y_5)\lambda_{35} + g(X_4, Y_5)\lambda_{45} + g(X_5, Y_5)\lambda_{55} + g(X_6, Y_5)\lambda_{65} + g(X_7, Y_5)\lambda_{75} + \\
& g(X_1, Y_4)\lambda_{14} + g(X_2, Y_4)\lambda_{24} + g(X_3, Y_4)\lambda_{34} + g(X_4, Y_4)\lambda_{44} + g(X_5, Y_4)\lambda_{54} + g(X_6, Y_4)\lambda_{64} + g(X_7, Y_4)\lambda_{74} + \\
& g(X_1, Y_3)\lambda_{13} + g(X_2, Y_3)\lambda_{23} + g(X_3, Y_3)\lambda_{33} + g(X_4, Y_3)\lambda_{43} + g(X_5, Y_3)\lambda_{53} + g(X_6, Y_3)\lambda_{63} + g(X_7, Y_3)\lambda_{73} + \\
& g(X_1, Y_2)\lambda_{12} + g(X_2, Y_2)\lambda_{22} + g(X_3, Y_2)\lambda_{32} + g(X_4, Y_2)\lambda_{42} + g(X_5, Y_2)\lambda_{52} + g(X_6, Y_2)\lambda_{62} + g(X_7, Y_2)\lambda_{72} + \\
& g(X_1, Y_1)\lambda_{11} + g(X_2, Y_1)\lambda_{21} + g(X_3, Y_1)\lambda_{31} + g(X_4, Y_1)\lambda_{41} + g(X_5, Y_1)\lambda_{51} + g(X_6, Y_1)\lambda_{61} + g(X_7, Y_1)\lambda_{71}
\end{aligned}
$$





$$
\begin{aligned}
& \lambda_{16} + \lambda_{26} + \lambda_{36} + \lambda_{46} + \lambda_{56} + \lambda_{66} + \lambda_{76} + \\
& \lambda_{15} + \lambda_{25} + \lambda_{35} + \lambda_{45} + \lambda_{55} + \lambda_{65} + \lambda_{75} + \\
& \lambda_{14} + \lambda_{24} + \lambda_{34} + \lambda_{44} + \lambda_{54} + \lambda_{64} + \lambda_{74} + \\
& \lambda_{13} + \lambda_{23} + \lambda_{33} + \lambda_{43} + \lambda_{53} + \lambda_{63} + \lambda_{73} + \\
& \lambda_{12} + \lambda_{22} + \lambda_{32} + \lambda_{42} + \lambda_{52} + \lambda_{62} + \lambda_{72} + \\
& \lambda_{11} + \lambda_{21} + \lambda_{31} + \lambda_{41} + \lambda_{51} + \lambda_{61} + \lambda_{71} = 1
\end{aligned}
$$

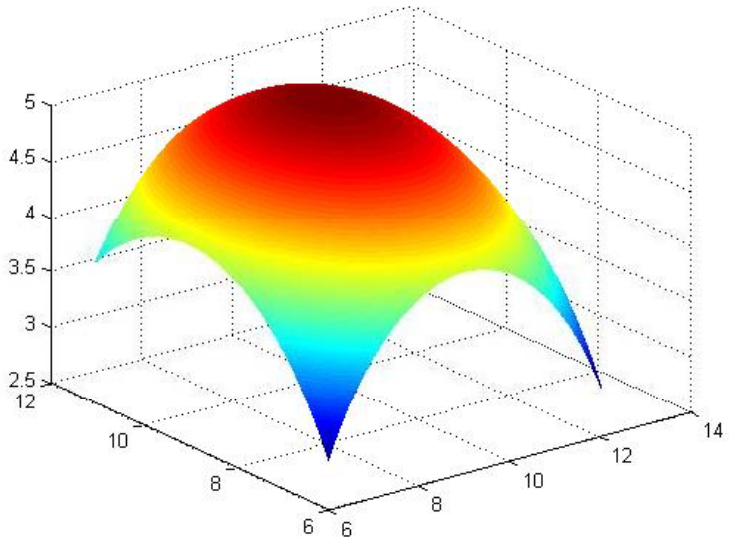
$$SOS2(\xi_1, \xi_2, \xi_3, \xi_4, \xi_5, \xi_6, \xi_7)$$

$$SOS2(\eta_1, \eta_2, \eta_3, \eta_4, \eta_5, \eta_6)$$

Ejemplo de aproximación lineal de 2 variables con una malla de 7 x 6

Vamos aplicar el método a la siguiente función de dos variables:

$$z = g(x, y) = \sqrt{25 - (x - 10)^2 - (y - 10)^2}$$

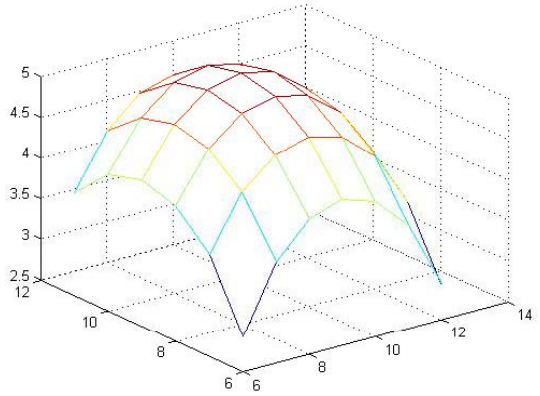


La malla la definiremos para los siguientes valores de X e Y:

X = [7, 8, 9, 10, 11, 12, 13]
Y = [7, 8, 9, 10, 11, 12]

Calculamos el valor de Z en cada punto de la malla:

Z=	2.6458	3.4641	3.8730	4.0000	3.8730	3.4641	2.6458
	3.4641	4.1231	4.4721	4.5826	4.4721	4.1231	3.4641
	3.8730	4.4721	4.7958	4.8990	4.7958	4.4721	3.8730
	4.0000	4.5826	4.8990	5.0000	4.8990	4.5826	4.0000
	3.8730	4.4721	4.7958	4.8990	4.7958	4.4721	3.8730
	3.4641	4.1231	4.4721	4.5826	4.4721	4.1231	3.4641



Código OPL para el ejemplo anterior (sin la restricción SOS2 diagonal)

```
int Nx = ...;
int Ny = ...;

range s = 1..Nx;
range k = 1..Ny;

float X[s] = ...;
float Y[k] = ...;
float g[k,s] = ...;

//SOS2 para e[i]
range eRR = 2..Ny-1;
range eR1 = 1..Ny-1;

//SOS2 para n[j]
range nRR = 2..Nx-1;
range nR1 = 1..Nx-1;

//SOS para e[i]
dvar int ed[eR1] in 0..1;

//SOS para n[j]
dvar int nd[nR1] in 0..1;

dvar float+ ld[k,s];
dvar float+ x;
dvar float+ y;
dvar float+ z;
dvar float+ e[k];
dvar float+ n[s];

minimize y-4*x-2*y;
```

```
subject to
{ // Begin Restriccion SOS2 para e[i]
    e[1]-ed[1] <=0;
    forall(i in eRR)
    {
        e[i] - ed[i-1]-ed[i] <= 0;
    }
    e[Ny] - ed[Ny-1] <= 0;
    sum(i in eR1)ed[i] == 1;
// End Variables SOS2 l[i]

// Begin Restriccion SOS2 para n[j]
    n[1]-nd[1] <=0;
    forall(j in nRR)
    {
        n[j] - nd[j-1]-nd[j] <= 0;
    }
    n[Nx] - nd[Nx-1] <= 0;
    sum(j in nR1)nd[j] == 1;
// End Variables SOS2 l[i]

//Aproximacion lineal
x == sum(i in k, j in s)X[j]*ld[i,j];
y == sum(i in k, j in s)Y[i]*ld[i,j];
z == sum(i in k, j in s)g[i,j]*ld[i,j];
sum(i in k, j in s)ld[i,j] == 1;

forall(i in k)
    e[i] == sum(j in s)ld[i,j];

forall(j in s)
    n[j] == sum(i in k)ld[i,j];

x==7.5;
y==7.5;
//z==3.4641 }
```

```
Nx = 7;
Ny = 6;
X   =   [7, 8, 9, 10, 11, 12, 13];
Y   =   [7, 8, 9, 10, 11, 12];
g = [
    [2.6458    3.4641    3.8730    4.0000    3.8730    3.4641    2.6458]
    [3.4641    4.1231    4.4721    4.5826    4.4721    4.1231    3.4641]
    [3.8730    4.4721    4.7958    4.8990    4.7958    4.4721    3.8730]
    [4.0000    4.5826    4.8990    5.0000    4.8990    4.5826    4.0000]
    [3.8730    4.4721    4.7958    4.8990    4.7958    4.4721    3.8730]
    [3.4641    4.1231    4.4721    4.5826    4.4721    4.1231    3.4641]];
```

Resultados del ejemplo

```
y = 7.5;
x = 7.5;
e = [0.5 0.5 0 0 0 0];
ed = [1 0 0 0 0];
n = [0.5 0.5 0 0 0 0 0];
nd = [   1   0 0 0 0 0];
ld = [[0.5   0 0 0 0 0 0]
      [  0 0.5 0 0 0 0 0]
      [  0   0 0 0 0 0 0]
      [  0   0 0 0 0 0 0]
      [  0   0 0 0 0 0 0]
      [  0   0 0 0 0 0 0]];
z = 3.3845;
```

```
x == 7.5;
y == 7.5;
//z == 3.464;libre
```

g(7.5, 7.5) = 3.5355

Valor real de la función

```
y = 10;
x = 10;
e = [0 0 -6.865e-16 1 0 -1.1102e-16];
ed = [0 0 1 0 0];
n = [5.1e-16 -1.8e-18 -1.8e-17 1 3.3e-16 0 0];
nd = [0 0 0 1 0 0];
ld = [[0 0 0 0 0 0 0]
      [0 0 0 0 0 0 0]
      [0 -1.8e-18 -1.8e-17 -6.6e-16 0 0 0]
      [5.1e-16 0 0 1 4.4e-16 0 0]
      [0 0 0 0 0 0 0]
      [0 0 0 0 -1.1e-16 0 0]];
z = 5;
```

```
x == 7.5;
y == 7.5;
//z == 3.464;libre
```

g(10, 10) = 5.000

Valor real de la función. Coincide con el aproximado por ser un punto de la malla

```
y  = 7;
x  = 7.5;
e  = [1 0 0 0 0 0];
ed = [1 0 0 0 0];
n  = [0.5 0.5 0 0 0 0 0];
nd = [  1   0 0 0 0 0];
ld = [[0.5 0.5 0 0 0 0 0]
      [  0   0 0 0 0 0 0]
      [  0   0 0 0 0 0 0]
      [  0   0 0 0 0 0 0]
      [  0   0 0 0 0 0 0]
      [  0   0 0 0 0 0 0]];
z = 3.0549;
```

```
x == 7.5;
y == 7;
//z == 3.464;libre
```

g(7.5, 7) = 3.1225

Valor real de la función

```
y = 7.5973;
x = 7.5;
e = [0.40266 0.59734 0 0 0 0];
ed = [1 0 0 0 0];
n = [0.5 0.5 0 0 0 0 0];
nd = [1 0 0 0 0 0];
ld = [[0.40266 0 0 0 0 0 0]
      [0.097336 0.5 0 0 0 0 0]
      [0 0 0 0 0 0 0]
      [0 0 0 0 0 0 0]
      [0 0 0 0 0 0 0]
      [0 0 0 0 0 0 0]];
z = 3.4641;
```

```
x == 7.5;
//y == 7.5;libre
z == 3.464;
```

En este caso hemos fijado el valor de **x** y el valor de **z** y el programa calcula el valor de **y**

g(7.5, y) = 3.4641