Universidad Politécnica de Madrid–Escuela Técnica Superior de Ingenieros Industriales
Grado en Ingeniería en Tecnologías Industriales. Curso 2017-2018-3º
Matemáticas de Especialidad–Ingeniería Eléctrica

Programación no lineal con condiciones

José Luis de la Fuente O'Connor
jldelafuente@etsii.upm.es
joseluis.delafuente@upm.es

Índice

Formulación del problema

- El problema objeto de estudio de la programación no lineal con condiciones es este:

$$\begin{aligned} &\underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimizar}}\ f(\boldsymbol{x}) \\ &\text{sujeta a} \quad \boldsymbol{c}_i(\boldsymbol{x}) = \boldsymbol{0}, \quad i \in \mathcal{E}, \\ &\qquad\qquad \boldsymbol{c}_j(\boldsymbol{x}) \geq \boldsymbol{0}, \quad j \in \mathcal{I}. \end{aligned}$$

  Las función objetivo f y las condiciones —las funciones vectoriales $\boldsymbol{c}_i$ y $\boldsymbol{c}_j$— son, en general, no lineales, continuas y tienen derivadas parciales continuas hasta al menos primer orden. Los conjuntos $\mathcal{E}$ y $\mathcal{I}$ contienen los índices de las condiciones de igualdad y de desigualdad, o inecuaciones.

- La región factible o conjunto de puntos factibles de un problema de Programación No Lineal —**PNL** o NLP— es

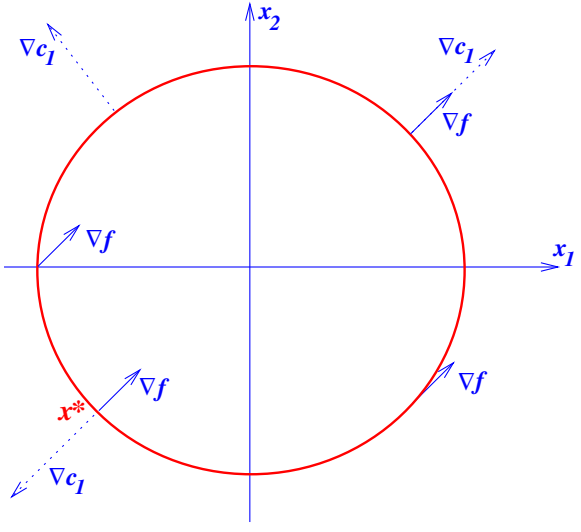
$$\Omega = \left\{\boldsymbol{x} : \boldsymbol{c}_i(\boldsymbol{x}) = \boldsymbol{0}, i \in \mathcal{E};\ \ \boldsymbol{c}_j(\boldsymbol{x}) \geq \boldsymbol{0}, j \in \mathcal{I}\right\}.$$

- **Ejemplo** Se trata de resolver

$$\begin{aligned} &\text{minimizar} \quad x_1 + x_2 \\ &\text{sujeta a} \quad x_1^2 + x_2^2 - 2 = 0. \end{aligned}$$

En este caso $f(\boldsymbol{x}) = x_1 + x_2$, $c_1(\boldsymbol{x}) = x_1^2 + x_2^2 - 2$, $\mathcal{E} = \{1\}$ y $\mathcal{I} = \emptyset$.

- La representación geométrica del problema es la de la figura.



El punto óptimo es $\boldsymbol{x}^* = [-1\ -1]^T$. Con el criterio de minimizar la función objetivo otros puntos de la circunferencia hacen que $f(\boldsymbol{x})$ sea peor.

En el óptimo, el vector gradiente de la condición, $\nabla c_1(\boldsymbol{x}^*)$, es paralelo (de hecho, colineal) al de la función objetivo, $\nabla f(\boldsymbol{x}^*)$, aunque con sentido opuesto. Existe un escalar λ_1^*, en este caso $\lambda_1^* = -1/2$, tal que $\nabla f(\boldsymbol{x}^*) = \lambda_1^* \nabla c_1(\boldsymbol{x}^*)$.

- Otra manera de ver el óptimo. Si en cualquier punto factible $\boldsymbol{x}$ del proceso de optimización se lleva a cabo un pequeño desplazamiento $\boldsymbol{s}$, para conservar la factibilidad deberá cumplirse que $\boldsymbol{c(x+s)=0}$.
  Por desarrollo en serie de Taylor,

$$\boldsymbol{0 = c(x+s) \approx c(x) + \nabla c(x)^T s = \nabla c(x)^T s.}$$

- Ese mismo paso debería producir una mejora de $f(\boldsymbol{x})$, es decir

$$0 > f(\boldsymbol{x}+\boldsymbol{s}) - f(\boldsymbol{x}) \approx \nabla f(\boldsymbol{x})^T \boldsymbol{s}.$$

  Cualquier dirección $\boldsymbol{d} \approx \boldsymbol{s}/\|\boldsymbol{s}\|$ en el sentido de ese paso que mejore la función objetivo y verifique la condición deberá cumplir simultáneamente que

$$\nabla \boldsymbol{c}(\boldsymbol{x})^T \boldsymbol{d} = 0 \quad \text{y } \nabla f(\boldsymbol{x})^T \boldsymbol{d} < 0.$$

  Si no existe, el punto $\boldsymbol{x}$ podría ser un óptimo (al menos local).

- La única manera de que no exista una $\boldsymbol{d}$ que satisfaga simultáneamente estos dos requerimientos es que $\nabla f(\boldsymbol{x})$ y $\nabla \boldsymbol{c}(\boldsymbol{x})$ sean paralelas.

- Volviendo al problema general de PNL, su función de Lagrange es

$$L(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) - \lambda \boldsymbol{c}(\boldsymbol{x}).$$

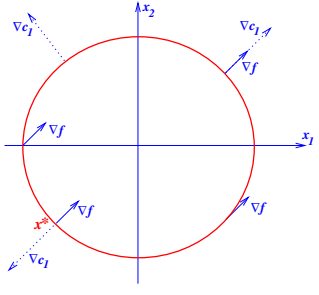
El gradiente de ésta con respecto a $\boldsymbol{x}$, $\nabla_{\boldsymbol{x}} L(\boldsymbol{x}, \lambda)$, es $\nabla f(\boldsymbol{x}) - \lambda \nabla \boldsymbol{c}(\boldsymbol{x})$.

- La condición de óptimo —necesaria pero no suficiente— se puede enunciar para el ejemplo que estudiamos como

$$\nabla_{\boldsymbol{x}} L(\boldsymbol{x}^*, \lambda_1^*) = \mathbf{0}.$$

A λ_1 se le denomina multiplicador de Lagrange de la condición $c_1(\boldsymbol{x}) = 0$.

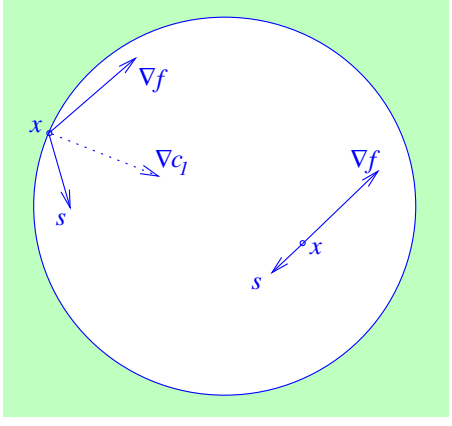
- La condición es necesaria, pero no suficiente. En efecto, en el ejemplo, el punto $[1 \quad 1]^T$ la cumple (con $\lambda_1 = 1/2$), pero no es el óptimo: de hecho es el máximo.



- **Ejemplo** Consideremos la siguiente variante del ejemplo anterior:

$$\text{min. } x_1 + x_2$$
$$\text{s. a } \; 2 - x_1^2 - x_2^2 \geq 0.$$

El óptimo sigue siendo el mismo. Su región factible es la circunferencia anterior y su interior: la superficie geométrica plana (círculo) contenida en ella.

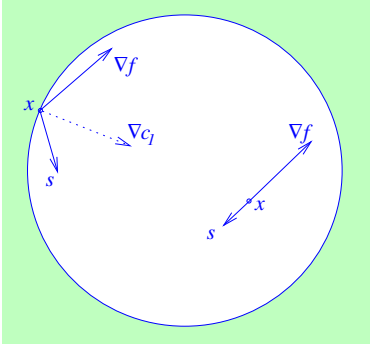


El gradiente de la condición en cualquier punto del extremo exterior de la región factible apunta hacia su interior.
La condición de óptimo $\nabla f(\boldsymbol{x}^*) = \lambda_1^* \nabla c_1(\boldsymbol{x}^*)$ se satisface con $\lambda_1^* = 1/2$, signo opuesto al del problema anterior.

- Razonando como antes, cualquier $\boldsymbol{x}$ no será óptimo si se puede dar un paso $\boldsymbol{s}$ al final del cual se conserve la factibilidad de la condición y se mejore la función objetivo. Es decir, si $\nabla f(\boldsymbol{x})^T \boldsymbol{s} < 0$ y se cumple que

$$\mathbf{0} \leq \boldsymbol{c}(\boldsymbol{x} + \boldsymbol{s}) \approx \boldsymbol{c}(\boldsymbol{x}) + \nabla \boldsymbol{c}(\boldsymbol{x})^T \boldsymbol{s}.$$

Consideraremos dos casos según se aprecia en la figura.

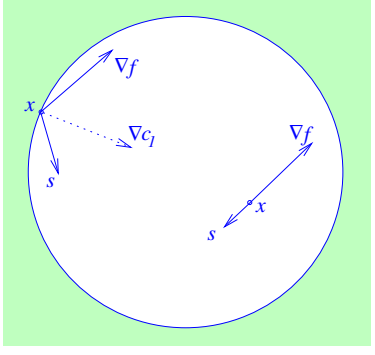


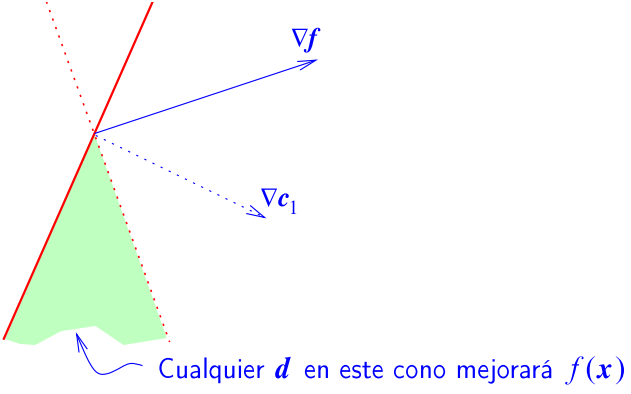
- **Caso 1** El punto $\boldsymbol{x}$ está estrictamente dentro de la región factible, $c_1(\boldsymbol{x}) > 0$. Cualquier paso $\boldsymbol{s}$ suficientemente pequeño seguirá cumpliendo esta inecuación (como ratifica la figura).
  De hecho, supuesta $\nabla f(\boldsymbol{x}) \neq \mathbf{0}$, para cualquier α positivo suficientemente pequeño el paso $\boldsymbol{s} = -\alpha \nabla f(\boldsymbol{x})$ satisface la condición anterior y que $\nabla f(\boldsymbol{x})^T \boldsymbol{s} < 0$.

- **Caso 2** Si $\boldsymbol{x}$ esta en el extremo o frontera de la región factible, $c_1(\boldsymbol{x}) = 0$.
  Las condiciones para mejorar la función objetivo serán

$$\nabla f(\boldsymbol{x})^T \boldsymbol{s} < 0 \quad \text{y} \quad \nabla c_1(\boldsymbol{x})^T \boldsymbol{s} \geq 0.$$

La primera determina un subespacio abierto y la segunda uno cerrado.
Cualquier dirección en el cono que genera la intersección de ambos, como se ve en la figura que sigue, mejorará la función objetivo.





Si $\nabla f(\boldsymbol{x})$ y $\nabla c_1(\boldsymbol{x})$ apuntan en la misma dirección la intersección es el vacío y

$$\nabla f(\boldsymbol{x}) = \lambda \nabla c_1(\boldsymbol{x}), \quad \text{para algún } \lambda_1 \geq 0.$$

- El signo del multiplicador es importante en este caso pues si la condición anterior se cumpliese con un valor negativo de λ_1, $\nabla f(\boldsymbol{x})$ y $\nabla c_1(\boldsymbol{x})$ apuntarían en direcciones opuestas y constituirían el subespacio completo.

- Por lo razonado, las condiciones de óptimo se pueden plasmar en

$$\nabla_{\boldsymbol{x}} L(\boldsymbol{x}^*, \lambda_1^*) = \mathbf{0} \qquad \text{para algún} \quad \lambda_1 \geq 0$$

$$\lambda_1^* c_1(\boldsymbol{x}^*) = 0.$$

  La segunda es la de complementariedad. Plasma el hecho de que el multiplicador de Lagrange λ_1^* sólo debe ser estrictamente positivo cuando la condición esté activa: es decir, $c_1(\boldsymbol{x}) = 0$.

- Para el ejemplo que consideramos:

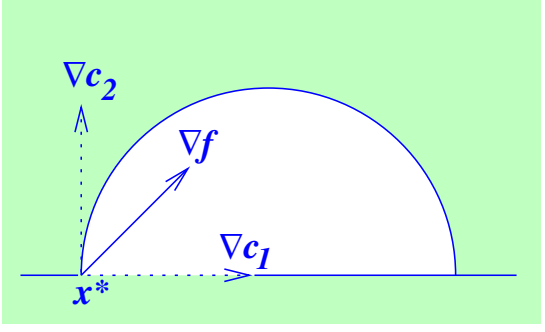
  En el **Caso 1** $c_1(\boldsymbol{x}) > 0$, por lo que se requiere que $\lambda_1^* = 0$. La otra condición de óptimo hace que $\nabla f(\boldsymbol{x}^*) = \mathbf{0}$.

  En el **Caso 2** la segunda condición permite que λ_1^* tome valores no negativos y la primera hace $\nabla f(\boldsymbol{x}) = \lambda_1 \nabla c_1(\boldsymbol{x})$.

- **Ejemplo** Consideremos una variante más del ejemplo que estudiamos:

$$\text{minimizar } x_1 + x_2$$
$$\text{sujeta a } \quad 2 - x_1^2 - x_2^2 \geq 0, \quad x_2 \geq 0.$$

La región factible es el semicírculo de la figura.



La solución es $\boldsymbol{x}^* = [-\sqrt{2},\ 0]^T$: Las dos condiciones están activas
Desde el óptimo, cualquier dirección $\boldsymbol{d}$ a lo largo de la cual la función objetivo mejore, deberá cumplir que $\nabla c_i(\boldsymbol{x})^T \boldsymbol{d} \geq 0, \quad i \in \mathcal{I} = \{1, 2\}$ y $\nabla f(\boldsymbol{x})^T \boldsymbol{d} < 0$.
La condición $\nabla c_i(\boldsymbol{x})^T \boldsymbol{d} \geq 0$, $i = 1, 2$, se cumple si $\boldsymbol{d}$ está dentro del cuadrante que definen $\nabla c_1(\boldsymbol{x})$ y $\nabla c_2(\boldsymbol{x})$, pero cualquier vector en él cumple que $\nabla f(\boldsymbol{x})^T \boldsymbol{d} \geq 0$.

- La función de Lagrange del problema es

$$L(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) - \lambda_1 c_1(\boldsymbol{x}) - \lambda_2 c_2(\boldsymbol{x}).$$

La condición de óptimo, $\nabla_{\boldsymbol{x}} L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = \boldsymbol{0}$ para algún $\boldsymbol{\lambda}^* \geq \boldsymbol{0}$.
La de complementariedad, $\lambda_1^* c_1(\boldsymbol{x}^*) = 0$ y $\lambda_2^* c_2(\boldsymbol{x}^*) = 0$.

- En el punto óptimo, $\boldsymbol{x}^* = [-\sqrt{2},\ 0]^T$, se tiene que

$$\nabla f(\boldsymbol{x}^*) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \nabla c_1(\boldsymbol{x}^*) = \begin{bmatrix} 2\sqrt{2} \\ 0 \end{bmatrix}, \quad \nabla c_2(\boldsymbol{x}^*) = \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

  por lo que si

$$\boldsymbol{\lambda}^* = \begin{bmatrix} \frac{1}{2\sqrt{2}} \\ 1 \end{bmatrix},$$

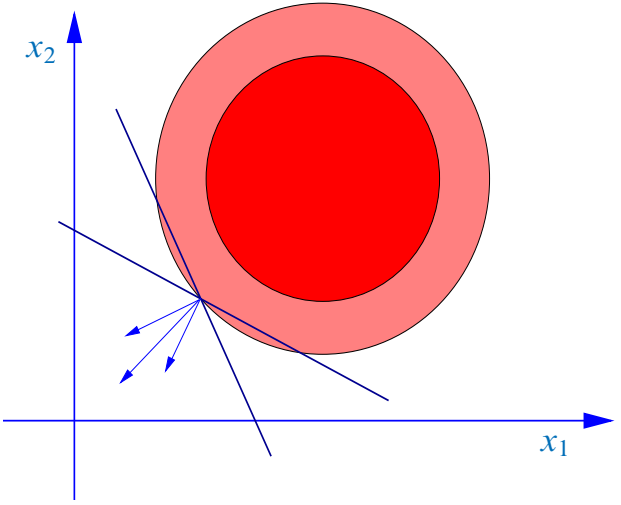
  se cumple que $\nabla_{\boldsymbol{x}} L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = \mathbf{0}$. Todos los coeficientes de $\boldsymbol{\lambda}^*$ también son positivos.

- Generalizando, en el óptimo, el gradiente de la función objetivo está (debe estar) en el cono que generan los vectores gradiente de las condiciones.

- **Ejemplo** Veamos este problema

$$\begin{aligned} \text{min. } & (x_1-1)^2+(x_2-1)^2 \\ \text{s. a } & -x_1-2x_2 \geq -1 \\ & -2x_1-x_2 \geq -1, \end{aligned}$$

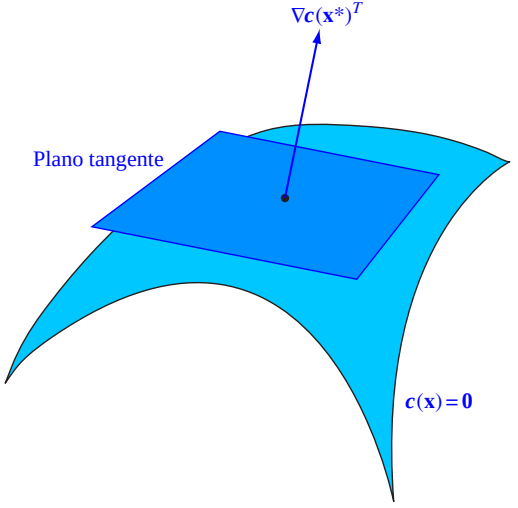
cuyo óptimo es $\left[\frac{1}{3}\ \frac{1}{3}\right]^T$.



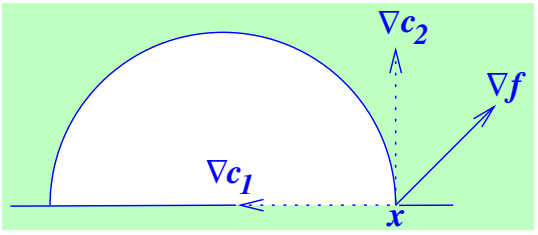
Definición Un punto $\boldsymbol{x}$ que satisfaga todas las condiciones activas se dice **regular** si los vectores gradiente del conjunto de condiciones activas en ese punto, $\mathcal{A}(\boldsymbol{x}) = \mathcal{E} \cup \{i \in \mathcal{I} : \boldsymbol{c}_i(\boldsymbol{x}) = \boldsymbol{0}\}$, son linealmente independientes.

- Por otro lado, el plano o subespacio tangente, $\mathcal{T}$, es

$$\mathcal{T} = \left\{\boldsymbol{v} : \nabla \boldsymbol{c}_i(\boldsymbol{x})^T \boldsymbol{v} = 0, \quad i \in \mathcal{E} \cup \left\{j \in \mathcal{I} : \boldsymbol{c}_j(\boldsymbol{x}) = \boldsymbol{0}\right\}\right\}.$$



- Consideremos ahora el punto $\boldsymbol{x} = [\sqrt{2} \;\; 0]^T$ del ejemplo del semicírculo.



minimizar $x_1 + x_2$
sujeta a $2 - x_1^2 - x_2^2 \geq 0, \quad x_2 \geq 0.$

Las dos condiciones están activas. La dirección $\boldsymbol{d} = [-1 \;\; 0]^T$, como muchas otras, en $\boldsymbol{x}$ cumple que

$$\nabla c_i(\boldsymbol{x})^T \boldsymbol{d} \geq 0, \;\; i \in \mathcal{I} = \{1, 2\}, \qquad \nabla f(\boldsymbol{x})^T \boldsymbol{d} < 0.$$

La condición $\nabla_{\boldsymbol{x}} L(\boldsymbol{x}, \boldsymbol{\lambda}) = 0$ sólo se cumple cuando $\boldsymbol{\lambda} = [-1/(2\sqrt{2}), \;\; 1]^T$ por lo que no se cumple la no negatividad.

- En el punto $\boldsymbol{x} = [1 \;\; 0]^T$, en el que sólo está activa la segunda condición, una $\boldsymbol{d}$ tal que $\nabla c_2(\boldsymbol{x})^T \boldsymbol{d} \geq 0$ y mejore $f(\boldsymbol{x})$, es decir, $\nabla f(\boldsymbol{x})^T \boldsymbol{d} < 0$, dado que $\nabla f(\boldsymbol{x}) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \nabla c_2(\boldsymbol{x}) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, sólo lo cumple $\boldsymbol{d} = \begin{bmatrix} -1/2 \\ 1/4 \end{bmatrix}$.
No obstante, no hay un λ_2 tal que $\nabla f(\boldsymbol{x}) - \lambda_2 \nabla c_2(\boldsymbol{x}) = \boldsymbol{0}$, por lo que no se cumplen tampoco todas las condiciones de óptimo.

Condiciones de Karush-Kuhn-Tucker

- Sea $\boldsymbol{x}^*$ un punto regular y mínimo local del problema

$$\begin{aligned} &\underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimizar}}\ f(\boldsymbol{x}) \\ &\text{sujeta a} \quad \boldsymbol{c}_i(\boldsymbol{x}) = \boldsymbol{0}, \quad i \in \mathcal{E}, \\ &\qquad\qquad \boldsymbol{c}_i(\boldsymbol{x}) \geq \boldsymbol{0}, \quad i \in \mathcal{I}. \end{aligned}$$

Existe un vector de multiplicadores de Lagrange, $\boldsymbol{\lambda}^*$, con coeficientes λ_i, $i \in \mathcal{E} \cup \mathcal{I}$, tal que se cumple que

$$\begin{aligned} \nabla_{\boldsymbol{x}} L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = \nabla f(\boldsymbol{x}^*) - \boldsymbol{\lambda}^T \boldsymbol{c}(\boldsymbol{x}^*) &= \boldsymbol{0}, \\ \boldsymbol{c}_i(\boldsymbol{x}^*) &= \boldsymbol{0}, \text{ para todo } i \in \mathcal{E}, \\ \boldsymbol{c}_i(\boldsymbol{x}^*) &\geq \boldsymbol{0}, \text{ para todo } i \in \mathcal{I}, \\ \lambda_i^* &\geq \boldsymbol{0}, \text{ para todo } i \in \mathcal{I}, \\ \lambda_i^* \boldsymbol{c}_i(\boldsymbol{x}^*) &= 0, \text{ para todo } i \in \mathcal{E} \cup \mathcal{I}. \end{aligned}$$

- **¡OJO CON LOS SIGNOS DE LOS MULTIPLICADORES!** Si se mantiene la coherencia con la notación y convención de signos adoptada para los multiplicadores λ, si hubiese inecuaciones o condiciones del tipo $\leq \mathbf{0}$, éstas se introducirían en la función de Lagrange también con signo negativo, pero las condiciones de punto óptimo exigirían que fuesen $\lambda^* \leq \mathbf{0}$.

- Es por esto que en muchas publicaciones la función de Lagrange incorpora las condiciones con el multiplicador precedido del signo $+$, pero luego exigen que si la inecuación es $\geq \mathbf{0}$, el multiplicador correspondiente debe ser $\leq \mathbf{0}$. Igualmente, si la inecuación es $\leq \mathbf{0}$, su multiplicador ha de ser $\geq \mathbf{0}$.

Condiciones de óptimo de segundo orden

- Sea $\boldsymbol{x}^*$ un punto regular y mínimo local del problema no lineal que estamos estudiando y $\boldsymbol{\lambda}^*$ un vector de multiplicadores de Lagrange tales que $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)$ cumplen las condiciones de **KKT**. Si $\mathcal{T}$ es el subespacio tangente en $\boldsymbol{x}^*$, para todo $\boldsymbol{z} \in \mathcal{T}$ se cumple que

$$\boldsymbol{z}^T \nabla^2_{\boldsymbol{x}} L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) \boldsymbol{z} \geq 0.$$

- Si en $\boldsymbol{x}^*$ la matriz $\nabla^2_{\boldsymbol{x}} L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)$ es definida positiva, ese punto es el mínimo local único del problema.

Algoritmos de PNL con Condiciones
Programación Cuadrática

- Tiene como objeto el estudio de problemas de optimización no lineales como

$$\begin{aligned} \underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimizar}} \quad & \tfrac{1}{2}\boldsymbol{x}^T\boldsymbol{G}\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{g} \\ \text{sujeta a} \quad & \boldsymbol{a}_i^T\boldsymbol{x} = \boldsymbol{b}_i, \quad i \in \mathcal{E}, \\ & \boldsymbol{a}_i^T\boldsymbol{x} \geq \boldsymbol{b}_i, \quad i \in \mathcal{I}, \end{aligned}$$

  donde $\boldsymbol{G}^{n\times n}$ es una matriz simétrica y $\boldsymbol{g}$, $\boldsymbol{x}$ y $\boldsymbol{a}_i, i \in \mathcal{E}\cup\mathcal{I}$, vectores $\mathbb{R}^n \to \mathbb{R}$. La Programación Cuadrática guarda una relación muy estrecha con la Programación Lineal

- Si $\boldsymbol{G}$ es definida positiva, el problema puede resolverse algebraicamente.

- Si $\boldsymbol{G}$ es semidefinida positiva, el problema en convexo y puede resolverse casi como un problema de Programación Lineal.

Programación Cuadrática con condiciones de igualdad

Método del subespacio imagen de la matriz A

- Se trata de resolver

$$\begin{aligned} &\underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimizar}} && \tfrac{1}{2}\boldsymbol{x}^T\boldsymbol{G}\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{g} \\ &\text{sujeta a} && \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \end{aligned}$$

  donde $\boldsymbol{A}$ es una matriz $m \times n$, $m \leq n$, que supondremos de rango completo.

- La función de Lagrange de este problema es

$$L(\boldsymbol{x}, \boldsymbol{\lambda}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{G}\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{g} - \boldsymbol{\lambda}^T(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}).$$

- Las condiciones de punto óptimo de Karush, Kuhn y Tucker son

$$\nabla_{\boldsymbol{x}} L(\boldsymbol{x}, \boldsymbol{\lambda}) = \boldsymbol{G}\boldsymbol{x} + \boldsymbol{g} - \boldsymbol{A}^T\boldsymbol{\lambda} = \boldsymbol{0}$$
$$\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} = \boldsymbol{0}.$$

  En forma matricial,

$$\begin{bmatrix} \boldsymbol{G} & -\boldsymbol{A}^T \\ -\boldsymbol{A} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} -\boldsymbol{g} \\ -\boldsymbol{b} \end{bmatrix}.$$

- Es fácilmente demostrable que si $\boldsymbol{G}$ es definida positiva y $\boldsymbol{A}$ de rango completo, la solución única de este problema es

$$\begin{bmatrix} \boldsymbol{x}^* \\ \boldsymbol{\lambda}^* \end{bmatrix} = -\begin{bmatrix} \boldsymbol{G} & -\boldsymbol{A}^T \\ -\boldsymbol{A} & \boldsymbol{0} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{g} \\ \boldsymbol{b} \end{bmatrix}.$$

  De aquí se deduce con cierta facilidad que

$$\boldsymbol{\lambda}^* = \left(\boldsymbol{A}\boldsymbol{G}^{-1}\boldsymbol{A}^T\right)^{-1}\left(\boldsymbol{A}\boldsymbol{G}^{-1}\boldsymbol{g} + \boldsymbol{b}\right)$$

  y que $\boldsymbol{x}^* = \boldsymbol{G}^{-1}\left(\boldsymbol{A}^T\boldsymbol{\lambda}^* - \boldsymbol{g}\right)$.

- **Ejemplo** Resolvamos

$$\begin{aligned} \text{min.} \quad & 3x_1^2 + 2x_1x_2 + x_1x_3 + 2{,}5x_2^2 + 2x_2x_3 + 2x_3^2 - 8x_1 - 3x_2 - 3x_3 \\ \text{s. a} \quad & x_1 + x_3 = 3 \\ & x_2 + x_3 = 0 \end{aligned}$$

- Las diversas matrices y vectores son

$$\boldsymbol{G} = \begin{bmatrix} 6 & 2 & 1 \\ 2 & 5 & 2 \\ 1 & 2 & 4 \end{bmatrix}, \quad \boldsymbol{g} = \begin{bmatrix} -8 \\ -3 \\ -3 \end{bmatrix}, \quad \boldsymbol{A} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 3 \\ 0 \end{bmatrix}.$$

- La solución mediante **Matlab**:

```
>> G=[6 2 1;2 5 2;1 2 4];
>> c=[-8;-3;-3];
>> A=[1 0 1;0 1 1];
>> b=[3;0];
>> Gm1=inv(G);
>> l=(A*Gm1*A')\(A*Gm1*c+b)
l =
   3.000000000000001
  -2.000000000000000
>> x=G\(A'*l-c)
x =
   2.000000000000000
  -1.000000000000000
   1.000000000000000
```

- Con el "solver" de Matlab para este propósito y fmincon:

```
» G=[6 2 1;2 5 2;1 2 4]; c=[-8;-3;-3];
A=[1 0 1;0 1 1];
b=[3;0];
» x=quadprog(G,c,[],[],A,b)
Optimization terminated: relative (projected) residual
 of PCG iteration <= OPTIONS.TolFun.
x =
   2.000000000000000
  -1.000000000000000
   1.000000000000000
» options=optimoptions('fmincon','Display','iter-detailed');
» x = fmincon(@Q1fun,[0;0;0],[],[],A,b,[],[],[],options)
                                          First-order      Norm of
 Iter F-count            f(x)  Feasibility   optimality         step
    0       4    0.000000e+00    3.000e+00    2.667e+00
    1       9    1.347222e+00    1.500e+00    1.117e+01    2.614e+00
    2      13   -3.198427e+00    8.882e-16    2.430e+00    2.949e+00
    3      17   -3.497660e+00    4.441e-16    3.369e-01    4.059e-01
    4      21   -3.500000e+00    0.000e+00    1.850e-07    3.286e-02

Optimization completed: The relative first-order optimality measure, 6.165186e-08,
is less than options.TolFun = 1.000000e-06, and the relative maximum constraint
violation, 0.000000e+00, is less than options.TolCon = 1.000000e-06.

Optimization Metric                                      Options
relative first-order optimality =   6.17e-08       TolFun =   1e-06 (default)
relative max(constraint violation) =   0.00e+00    TolCon =   1e-06 (default)
x =
    2.0000
   -1.0000
    1.0000
```

La función objetivo:

```
function f=Q1fun(x)
% Ejemplo optimización cuadrática 1
 f=0.5*[x(1) x(2) x(3)]*[6 2 1;2 5 2; 1 2 4]*[x(1); x(2); x(3)]+...
       [x(1) x(2) x(3)]*[-8;-3;-3];
end
```

- Si utilizamos un *software* disponible en Internet

OPTI Toolbox, https://www.inverseproblem.co.nz/OPTI/

de Jonathan Currie, del *Industrial Information & Control Centre* (I^2C^2), de la AUT University, Auckland, New Zealand, muy utilizado como plataforma general de pruebas, la sesión de **Matlab** correspondiente es la que sigue.

```
» fun = @(x) 0.5*[x(1) x(2) x(3)]*[6 2 1;2 5 2; 1 2 4]*[x(1); x(2); x(3)]+...
      [x(1) x(2) x(3)]*[-8;-3;-3];
» A=[1 0 1;0 1 1];
» b=[3;0];
» x0=[0;0;0];
» opts = optiset('solver','ipopt');
» Opt = opti('fun',fun,'eq',A,b,'options',opts)
---------------------------------------------------------
Nonlinear Program (NLP) Optimization
 min  f(x)
 s.t. rl <= Ax <= ru
---------------------------------------------------------
   Problem Properties:
# Decision Variables:        3
# Constraints:               2
  # Linear Equality:         2
---------------------------------------------------------
  Solver Parameters:
Solver:                  IPOPT
Objective Gradient:      @(x)mklJac(prob.fun,x,1) [n
Lagrangian Hessian:      Not Supplied
Hessian Structure:       Not Supplied
---------------------------------------------------------
```

```
» [x,fval,exitflag,info] = solve(Opt,x0)
iter  objective    inf_pr inf_du lg(mu)||d|| lg(rg) alpha_du alpha_pr  ls
 0 0.0000000e+000 3.00e+000 2.67e+000   0.0 0.00e+000    -  0.00e+000 0.00e+000
 1 4.2722222e+001 4.44e-016 1.16e+001 -11.0 4.67e+000    -  1.00e+000 1.00e+000
 2-2.5109957e+000 0.00e+000 1.69e+000 -11.0 2.28e+000    -  1.00e+000 1.00e+000
 3-3.5000000e+000 0.00e+000 1.44e-009 -11.0 3.90e-001    -  1.00e+000 1.00e+000
x =
    2.0000
   -1.0000
    1.0000
fval =
   -3.5000
exitflag =
     1
info =
    Iterations: 3
     FuncEvals: [1x1 struct]
          Time: 0.0034
     Algorithm: 'IPOPT: Interior Point NL Solver'
        Status: 'Success'
        Lambda: [1x1 struct]
```

Método del subespacio núcleo de la matriz A

- Este método se usa cuando la matriz G es semidefinida positiva.

- Se supone que $A \in \mathbb{R}^{m \times n}$, $n \geq m$, es de rango completo y se utiliza una matriz $Z^T G Z$, la hessiana reducida.

- La matriz Z la conforman los vectores de una base del subespacio núcleo de A, $\ker(A)$, obtenida por ejemplo a partir de[1]

$$A^T \Pi = Q \begin{bmatrix} R \\ 0 \end{bmatrix} = [Y \; Z] \begin{bmatrix} R \\ 0 \end{bmatrix}$$

  por lo que $AZ = 0$, y se supone que $Z^T G Z$ es definida positiva.

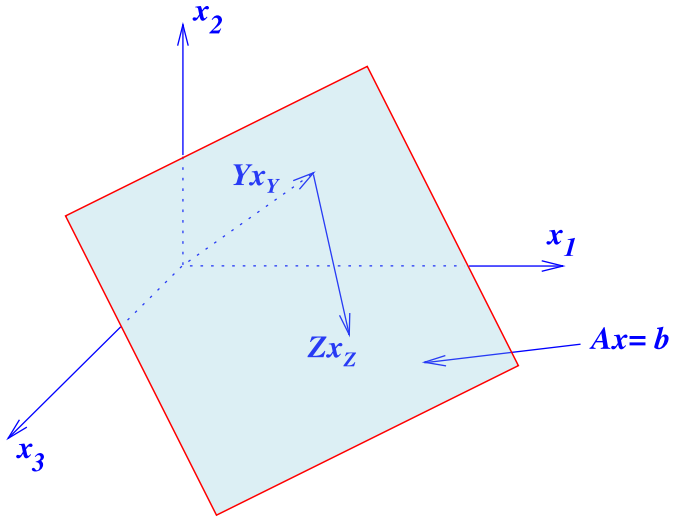
[1] La matriz Π es $m \times m$, Q, $n \times n$, R, $m \times m$, Y, $n \times m$, Z, $n \times (n-m)$.

- Generalizando la idea de PL de ordenar $\boldsymbol{A} = [\boldsymbol{B}\ \boldsymbol{N}]$, se puede hacer que el vector de dimensión n, $\boldsymbol{x}$, se estructure en dos partes, $\boldsymbol{x}_y$ y $\boldsymbol{x}_z$, tales que

$$\boldsymbol{x} = \boldsymbol{Y}\boldsymbol{x}_y + \boldsymbol{Z}\boldsymbol{x}_z$$

donde $\boldsymbol{Z}$ es la matriz $n \times (n-m)$ que conforma una base del $\ker(\boldsymbol{A})$ e $\boldsymbol{Y}$, $n \times m$, una base de $\mathrm{Im}(\boldsymbol{A}^T)$. Entonces $[\boldsymbol{Y}\ \ \boldsymbol{Z}]$ es regular.

- Así, $\boldsymbol{Y}\boldsymbol{x}_y$ es una solución de $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ y $\boldsymbol{Z}\boldsymbol{x}_z$ un desplazamiento a otro punto que verifique esas condiciones (en el subespacio núcleo de $\boldsymbol{A}$), pues $\boldsymbol{A}\boldsymbol{Z} = \boldsymbol{0}$.



- Si sustituimos $\boldsymbol{x} = \boldsymbol{Y}\boldsymbol{x}_y + \boldsymbol{Z}\boldsymbol{x}_z$ en las condiciones Karush, Kuhn y Tucker del problema de Programación Cuadrática, $\boldsymbol{G}\boldsymbol{x} - \boldsymbol{A}^T\boldsymbol{\lambda} = -\boldsymbol{g}$, $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, resulta que

$$\boldsymbol{G}\boldsymbol{Y}\boldsymbol{x}_y + \boldsymbol{G}\boldsymbol{Z}\boldsymbol{x}_z - \boldsymbol{A}^T\boldsymbol{\lambda} = -\boldsymbol{g}$$
$$\boldsymbol{A}\boldsymbol{Y}\boldsymbol{x}_y = \boldsymbol{b}$$

- Como $\boldsymbol{A}$ es de rango m y $[\boldsymbol{Y} \;\; \boldsymbol{Z}]$ es regular, el producto $\boldsymbol{A}\,[\boldsymbol{Y} \;\; \boldsymbol{Z}] = [\boldsymbol{A}\boldsymbol{Y} \;\; \boldsymbol{0}]$ tiene rango m, por lo que $\boldsymbol{A}\boldsymbol{Y}$ es una matriz $m \times m$ regular y $\boldsymbol{x}_y$ está perfectamente definido: $\boldsymbol{x}_y = (\boldsymbol{A}\boldsymbol{Y})^{-1}\boldsymbol{b}$.

- Si $\boldsymbol{Y}$ se obtuvo por factorización QR de $\boldsymbol{A}^T$, entonces $\boldsymbol{A}\boldsymbol{Y} = \boldsymbol{R}^T$, por lo que $\boldsymbol{p}_y = \boldsymbol{R}^{-T}\boldsymbol{b}$.

- Si se premultiplica la primera condición KKT por $\boldsymbol{Z}^T$ se tiene que

$$\left(\boldsymbol{Z}^T \boldsymbol{G}\, \boldsymbol{Z}\right) \boldsymbol{x}_z = -\boldsymbol{Z}^T \boldsymbol{G}\, \boldsymbol{Y}\, \boldsymbol{x}_y - \boldsymbol{Z}^T \boldsymbol{g}.$$

- Como la hessiana reducida $\boldsymbol{Z}^T \boldsymbol{G}\, \boldsymbol{Z}$ es definida positiva, se puede factorizar por Cholesky y determinar $\boldsymbol{x}_z$. Luego se calcula $\boldsymbol{x}$ a partir de $\boldsymbol{x} = \boldsymbol{Y}\, \boldsymbol{x}_y + \boldsymbol{Z}\, \boldsymbol{x}_z$.

- Para obtener el vector de multiplicadores $\boldsymbol{\lambda}$ se premultiplica por $\boldsymbol{Y}^T$ la condición $\boldsymbol{G}\boldsymbol{x} - \boldsymbol{A}^T \boldsymbol{\lambda} = -\boldsymbol{g}$ y se resuelve el sistema

$$\left(\boldsymbol{A}\, \boldsymbol{Y}\right)^T \boldsymbol{\lambda} = \boldsymbol{Y}^T \left(\boldsymbol{g} + \boldsymbol{G}\boldsymbol{x}\right).$$

- **Ejemplo** Consideremos otra vez

$$\begin{aligned} \text{min.} \quad & 3x_1^2 + 2x_1x_2 + x_1x_3 + 2{,}5x_2^2 + 2x_2x_3 + 2x_3^2 - 8x_1 - 3x_2 - 3x_3 \\ \text{s. a} \quad & x_1 + x_3 = 3 \\ & x_2 + x_3 = 0. \end{aligned}$$

- Haciéndolo con **Matlab** a mano:

```
>> G=[6 2 1;2 5 2;1 2 4];
>> c=[-8;-3;-3];
>> A=[1 0 1;0 1 1];
>> b=[3;0];
>> [Q R]=qr(A')
Q =
   -0.7071    0.4082   -0.5774
         0   -0.8165   -0.5774
   -0.7071   -0.4082    0.5774
R =
   -1.4142   -0.7071
         0   -1.2247
         0         0
>> Y=Q(:,1:m)
Y =
   -0.7071    0.4082
         0   -0.8165
   -0.7071   -0.4082
>> Z=Q(:,m+1:n)
Z =
   -0.5774
   -0.5774
    0.5774
>> xy=(A*Y)\b
xy =
   -2.1213
    1.2247
```

```
>> xyy=R(1:m,:)'\b   % alternativa
xyy =
   -2.1213
    1.2247
>> xz=-(Z'*G*Z)\(Z'*G*Y*xy+Z'*c)
xz =
  -6.1489e-16
>> x=Y*xy+Z*xz
x =
    2.0000
   -1.0000
    1.0000
>> la=(A*Y)'\(Y'*(c+G*x))
la =
    3.0000
   -2.0000
>> la=R(1:m,:)\(Y'*(c+G*x))
la =
    3.0000
   -2.0000
```

- Con quadprog, el "solver" de **Matlab**, para ver $\boldsymbol{\lambda}^*$:

```
» [x fval exitflag output lambda] = quadprog(G,c,[],[],A,b)
Optimization terminated: relative (projected) residual
 of PCG iteration <= OPTIONS.TolFun.
x =
   2.000000000000000
  -1.000000000000000
   1.000000000000000
fval =
  -3.500000000000000
exitflag =
     1
output =
         iterations: 1
    constrviolation: 4.440892098500626e-016
          algorithm: [1x58 char]
      firstorderopt: 0
       cgiterations: 1
            message: [1x91 char]
lambda =
      eqlin: [2x1 double]
    ineqlin: []
      lower: []
      upper: []
» lambda.eqlin
ans =
  -2.999999999999999
   2.000000000000000
```

Programación Cuadrática con condiciones generales

- Estudiamos ahora

$$\begin{aligned} \underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimizar}} \quad & \tfrac{1}{2}\boldsymbol{x}^T\boldsymbol{G}\boldsymbol{x} + \boldsymbol{g}^T\boldsymbol{x} \\ \text{sujeta a} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{C}\boldsymbol{x} \geq \boldsymbol{d} \end{aligned}$$

  donde $\boldsymbol{A}$ es una matriz $m_A \times n$, $m_A \leq n$, que supondremos de rango completo, $\boldsymbol{C}$ otra matriz $m_C \times n$, también de rango completo y $m_C \leq n$.

- La función de Lagrange de este problema es

$$L(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{G}\boldsymbol{x} + \boldsymbol{g}^T\boldsymbol{x} - \boldsymbol{y}^T(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) - \boldsymbol{z}^T(\boldsymbol{C}\boldsymbol{x} - \boldsymbol{d}).$$

- Las condiciones de Karush, Kuhn y Tucker son

$$\begin{aligned} \boldsymbol{G}\boldsymbol{x} + \boldsymbol{g} - \boldsymbol{A}^T\boldsymbol{y} - \boldsymbol{C}^T\boldsymbol{z} &= \boldsymbol{0} \\ \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} &= \boldsymbol{0} \\ \boldsymbol{C}\boldsymbol{x} - \boldsymbol{d} &\geq \boldsymbol{0} \\ \boldsymbol{z} &\geq \boldsymbol{0} \\ z_i\,(\boldsymbol{C}\boldsymbol{x} - \boldsymbol{d})_i &= 0, \quad i = 1, \ldots, m_C. \end{aligned}$$

- Si se introduce el vector de variables de holgura $\boldsymbol{s}$ en $\boldsymbol{C}\boldsymbol{x} - \boldsymbol{d} \geq \boldsymbol{0}$ las condiciones KKT quedan así:

$$\begin{aligned} \boldsymbol{G}\boldsymbol{x} + \boldsymbol{g} - \boldsymbol{A}^T\boldsymbol{y} - \boldsymbol{C}^T\boldsymbol{z} &= \boldsymbol{0} \\ \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} &= \boldsymbol{0} \\ \boldsymbol{s} - \boldsymbol{C}\boldsymbol{x} + \boldsymbol{d} &= \boldsymbol{0} \\ s_i z_i &= 0 \\ (\boldsymbol{z}, \boldsymbol{s}) &\geq \boldsymbol{0}. \end{aligned}$$

- Ahora vamos a aplicar como hacíamos en PL, un procedimiento de punto interior predictor corrector para resolver el sistema de ecuaciones $f(x, y, z, s)$. Es decir, a

$$f(x, y, z, s) = \begin{bmatrix} Gx + g - A^T y - C^T z \\ Ax - b \\ s - Cx + d \\ SZe \end{bmatrix} = 0.$$
$$z,s>0$$

- La ecuación de Newton-Raphson para resolver este sistema es

$$J \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta s \end{bmatrix} = -f(x, y, z, s),$$

donde J es la matriz jacobiana del sistema de ecuaciones.

- Desarrollando queda

$$\begin{bmatrix} \boldsymbol{G} & -\boldsymbol{A}^T & -\boldsymbol{C}^T & \boldsymbol{0} \\ -\boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ -\boldsymbol{C} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{S} & \boldsymbol{Z} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{x} \\ \Delta \boldsymbol{y} \\ \Delta \boldsymbol{z} \\ \Delta \boldsymbol{s} \end{bmatrix} = - \begin{bmatrix} \boldsymbol{r}_L \\ \boldsymbol{r}_A \\ \boldsymbol{r}_C \\ \boldsymbol{r}_{sz} \end{bmatrix},$$

donde $\boldsymbol{r}_L = \boldsymbol{g} + \boldsymbol{G}\boldsymbol{x} - \boldsymbol{A}^T\boldsymbol{y} - \boldsymbol{C}^T\boldsymbol{z}$

$\boldsymbol{r}_A = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$

$\boldsymbol{r}_C = \boldsymbol{s} - \boldsymbol{C}\boldsymbol{x} + \boldsymbol{d}$

$\boldsymbol{r}_{sz} = \mu \boldsymbol{e} - \boldsymbol{S}\boldsymbol{Z}\boldsymbol{e}.$

$\mu \boldsymbol{e}$ evita que $\boldsymbol{s}$ y $\boldsymbol{z}$ se acerquen demasiado a 0.

- Como en el caso predictor-corrector de PL, primero se calcula la dirección de escalado afín, $[\Delta \boldsymbol{x}^{af}, \Delta \boldsymbol{y}^{af}, \Delta \boldsymbol{z}^{af}, \Delta \boldsymbol{s}^{af}]$, o dirección pura de Newton, o dirección predictor, resolviendo este último sistema.

- Una vez calculada esta dirección, se determina una amplitud de paso α^{af} a lo largo de la misma de tal manera que se cumpla la factibilidad de $\boldsymbol{z}$ y $\boldsymbol{s}$, es decir, que

$$\begin{aligned} \boldsymbol{z} + \alpha^{af}\,\Delta\boldsymbol{x}^{af} &\geq \boldsymbol{0} \\ \boldsymbol{s} + \alpha^{af}\,\Delta\boldsymbol{s}^{af} &\geq \boldsymbol{0}, \end{aligned}$$

  teniendo en cuenta las ideas del central path de PL.

- Luego, para seguir con la dirección corrector hay que obtener primero el parámetro de centrado, τ para aproximarse al central path.

- Para ello se calcula la brecha dual, o lo que no se cumple de la condición de complementariedad de $\boldsymbol{s}$ y $\boldsymbol{z}$, $\mu = \frac{\boldsymbol{s}^T\boldsymbol{z}}{m_C}$, así como la que predecía el paso avanzado, esto es,

$$\mu^{af} = \frac{\left(\boldsymbol{s} + \alpha^{af}\,\Delta\boldsymbol{s}^{af}\right)^T\left(\boldsymbol{z} + \alpha^{af}\,\Delta\boldsymbol{z}^{af}\right)}{m_C},$$

  haciéndose por fin $\tau = \left(\frac{\mu^{af}}{\mu}\right)^3$.

- La dirección corrector, con información de segundo orden, se calcula resolviendo

$$\begin{bmatrix} \boldsymbol{G} & -\boldsymbol{A}^T & -\boldsymbol{C}^T & \boldsymbol{0} \\ -\boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ -\boldsymbol{C} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{S} & \boldsymbol{Z} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{x} \\ \Delta \boldsymbol{y} \\ \Delta \boldsymbol{z} \\ \Delta \boldsymbol{s} \end{bmatrix} = - \begin{bmatrix} \boldsymbol{r}_L \\ \boldsymbol{r}_A \\ \boldsymbol{r}_C \\ \boldsymbol{r}_{sz} + \Delta \boldsymbol{S}^{af} \Delta \boldsymbol{Z}^{af} \boldsymbol{e} - \tau \mu \boldsymbol{e} \end{bmatrix}.$$

- Por último se calcula una amplitud de paso global de la iteración, α, que cumpla la factibilidad de $\boldsymbol{s}$ y $\boldsymbol{z}$ llegándose, mediante un último parámetro de seguridad o amortiguación, η, al nuevo punto

$$[\boldsymbol{x}_{k+1}\, \boldsymbol{y}_{k+1}\, \boldsymbol{z}_{k+1}\, \boldsymbol{s}_{k+1}]^T = [\boldsymbol{x}_k\, \boldsymbol{y}_k\, \boldsymbol{z}_k\, \boldsymbol{s}_k]^T + \eta\alpha[\Delta \boldsymbol{x}\, \Delta \boldsymbol{y}\, \Delta \boldsymbol{z}\, \Delta \boldsymbol{s}]^T.$$

- La mayor cantidad de trabajo de este procedimiento está en resolver los sistemas de ecuaciones que definen las direcciones predictor y corrector.

- En este sentido, la cuarta ecuación del sistema matricial predictor anterior es

$$\boldsymbol{S}\,\Delta\boldsymbol{z} + \boldsymbol{Z}\,\Delta\boldsymbol{s} = -\boldsymbol{r}_{sz} \quad \text{y de ahí} \quad \Delta\boldsymbol{s} = -\boldsymbol{Z}^{-1}\left(\boldsymbol{r}_{sz} + \boldsymbol{S}\,\Delta\boldsymbol{z}\right).$$

Despejando en la tercera

$$-\boldsymbol{C}^T\Delta\boldsymbol{x} - \Delta\boldsymbol{s} = -\boldsymbol{r}_C \quad \Rightarrow -\boldsymbol{C}^T\Delta\boldsymbol{x} - \boldsymbol{Z}^{-1}\left(\boldsymbol{r}_{sz} + \boldsymbol{S}\,\Delta\boldsymbol{x}\right) = -\boldsymbol{r}_C$$

por lo que

$$-\boldsymbol{C}^T\Delta\boldsymbol{x} - \boldsymbol{Z}^{-1}\boldsymbol{S}\,\Delta\boldsymbol{z} = -\boldsymbol{r}_C + \boldsymbol{Z}^{-1}\boldsymbol{r}_{sz}.$$

- El sistema queda así

$$\begin{bmatrix} \boldsymbol{G} & -\boldsymbol{A}^T & -\boldsymbol{C}^T \\ -\boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} \\ -\boldsymbol{C} & \boldsymbol{0} & -\boldsymbol{Z}^{-1}\boldsymbol{S} \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{x} \\ \Delta\boldsymbol{y} \\ \Delta\boldsymbol{z} \end{bmatrix} = -\begin{bmatrix} \boldsymbol{r}_L \\ \boldsymbol{r}_A \\ \boldsymbol{r}_C - \boldsymbol{Z}^{-1}\boldsymbol{r}_{sz} \end{bmatrix}.$$

```
function [x_f, y_f, z_f, s_f, fo, k] = pcQP_gen_2(G, g, A, b, C, d, x, y, z, s)
% Resuelve el problema de Programación Cuadrática
% min 0.5x'Gx + g'x
% s. a    Ax =  b         A=nA x n
%         Cx >= d         C=nC x n
% mediante el método predictor corrector de punto interior. Parte de (x,y,z,s)
 [nA n]=size(A); [nC ncc]=size(C); e=ones(n,1);         % ¡OJO! n==ncc
 k=0 ; maxk=200; eps_L=1e-9; bc=1+max(norm(b),norm(d));
 if nargin<7, s=ones(nC,1); y=ones(nA,1); z=ones(nC,1); x=sqrt(n)*ones(n,1); end
 % Residuos
 rL  = G*x + g - A'*y - C'*z;
 rA  = -A*x + b;
 rC  = -C*x + s + d;
 rsz = s.*z;
 mu  = mean(rsz);               residuo=norm([rL;rA;rC;rsz])/bc;

 fprintf('\n              No fac.           Error         Brecha     Error
 fprintf('\n              primal           Lagrang.        dual      relati
 fprintf('  Iter    A*x-b, C*x-s-d   G*x+g-A''*y-C''*z     z''*s      total
 fprintf('  ------------------------------------------------------------\
 while k<=maxk & residuo>eps_L
 % Se resuelve el sistema para determinar dirección de Newton pura: predict
    Mat = [G          -A'          -C'                ; ...
           -A         zeros(nA,nA) zeros(nA,nC)       ; ...
           -C         zeros(nC,nA) sparse(-diag(s./z))];
    rhs = [-rL; -rA; -rC+rsz./z];
    [L D P] = ldl(Mat);
    dxyz_a  = P*(L'\(D\(L\(P'*rhs))));
    dx_a    = dxyz_a(1:length(x));
    dy_a    = dxyz_a(length(x)+1:length(x)+length(y));
    dz_a    = dxyz_a(length(x)+length(y)+1:length(x)+length(y)+length(z));
    ds_a    = -((rsz+s.*dz_a)./z);
 % Cálculo de alpha_af
    alpha_a = 1;
    idx_z = find(dz_a<0);
    if isempty(idx_z)==0, alpha_a = min(alpha_a,min(-z(idx_z)./dz_a(idx_z))); end
    idx_s = find(ds_a<0);
    if isempty(idx_s)==0, alpha_a = min(alpha_a,min(-s(idx_s)./ds_a(idx_s))); end
    mu_a  = ((z+alpha_a*dz_a)'*(s+alpha_a*ds_a))/nC;    % Affine duality gap
    sigma = (mu_a/mu)^3;                                % Centering parameter
```

```
  % Se resuelve dirección corrector
    rsz = rsz + ds_a.*dz_a - sigma*mu*e;
    rhs = [-rL; -rA; -rC+rsz./z];
    dxyz = P*(L'\(D\(L\(P'*rhs))));
    dx = dxyz(1:length(x));
    dy = dxyz(length(x)+1:length(x)+length(y));
    dz = dxyz(length(x)+length(y)+1:length(x)+length(y)+length(z));
    ds = -((rsz+s.*dz)./z);
  % Cálculo de alpha
    alpha = 1;
    idx_z = find(dz<0);
    if isempty(idx_z)==0, alpha = min(alpha,min(-z(idx_z)./dz(idx_z))); end
    idx_s = find(ds<0);
    if isempty(idx_s)==0, alpha = min(alpha,min(-s(idx_s)./ds(idx_s))); end
  % Obtención nuevo punto
    tau=max(.9995,1-mu)*alpha;      % Con factor eta=0.9995
    x = x+tau*dx; y = y+tau*dy; z = z+tau*dz; s = s+tau*ds; k = k+1;
  % Recalcular residuos en nuevo punto, brecha dual e imprimir iteración
    rL  = G*x + g - A'*y - C'*z;
    rA  = -A*x + b;
    rC  = -C*x + s + d;             nrC=norm([rA;rC]);
    rsz = s.*z;
    mu  = mean(rsz);                residuo=norm([rL;rA;rC;rsz])/bc;
    fprintf('%5i %15.2e %17.2e %13.2e %10.2e\n',k,nrC,norm(rL),mu,residuo);
end
x_f = x; y_f = y; z_f = z; s_f = s; fo = 0.5*x'*G*x+g'*x;
```

- **Ejemplo** Resolvamos el problema

$$\text{minimizar} \quad \frac{1}{2}\boldsymbol{x}^T \begin{bmatrix} 4 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{bmatrix} \boldsymbol{x} + \boldsymbol{x}^T \begin{bmatrix} -8 \\ -6 \\ -6 \end{bmatrix}$$

$$\text{sujeta a} \quad x_1 + x_2 + x_3 = 3$$

$$\boldsymbol{x} \geq \boldsymbol{0}$$

- Utilizamos el `script` anterior

```
>> G=[4 0 0;0 1 -1;0 -1 1];
>> g=[-8;-6;-6];
>> A=[1 1 1];
>> b=3;
>> C=eye(3);
>> d=zeros(3,1);
>> x=[1;1;1];
>> y=1;
>> s=y;
>> z=x;
```

```
>> [x1 y1 z1 s1 fo k]=pcQP_gen_2(G, g, A, b, C, d, x, y, z, s)

            No fac.          Error         Brecha      Error
            primal         Lagrang.         dual       relati.
  Iter    A*x-b, C*x-s-d   G*x+g-A'*y-C'*z     z'*s       total
  ---------------------------------------------------------------
    1       0.00e+000         8.66e-002     4.13e-002  3.76e-002
    2       3.14e-016         4.33e-005     6.75e-004  4.84e-004
    3       3.33e-016         2.17e-008     3.42e-007  2.45e-007
    4       2.22e-016         6.65e-015     1.17e-013  8.38e-014
x1 =
   0.500000000000166
   1.249999999999917
   1.249999999999917
y1 =
  -6.000000000000002
z1 =
  1.0e-012 *
   0.670272053531211
   0.006169231801483
   0.006169231801483
s1 =
   0.500000000000166
   1.249999999999917
   1.249999999999917
fo =
 -18.500000000000000
k =
     4
```

Con el "solver" quadprog de **Matlab**:

```
>> [x fval exitflag output lambda] =quadprog(G,g,-C,d,A,b,[0;0;0])
Warning: Large-scale algorithm does not currently solve this problem
formulation,
using medium-scale algorithm instead.
> In quadprog at 293
Optimization terminated.
x =
   0.500000000000001
   1.249999999999999
   1.249999999999999
fval =
 -18.499999999999996
exitflag =
     1
output =
         iterations: 1
    constrviolation: -0.500000000000001
          algorithm: 'medium-scale: active-set'
      firstorderopt: 3.552713678800501e-015
       cgiterations: []
            message: 'Optimization terminated.'
lambda =
      lower: [3x1 double]
      upper: [3x1 double]
      eqlin: 5.999999999999998
    ineqlin: [3x1 double]
```

- Con el `fmincon` de **Matlab**:

```
x = fmincon(@Q2fun,[0;0;0],[],[],A,b,[],[],[],options)
                                                First-order      Norm of
 Iter F-count            f(x)  Feasibility   optimality         step
    0       4    0.000000e+00    3.000e+00    1.333e+00
    1       8   -1.177778e+01    0.000e+00    7.000e+00    2.380e+00
    2      12   -1.831556e+01    0.000e+00    9.448e-01    1.873e+00
    3      16   -1.850000e+01    0.000e+00    1.164e-07    3.719e-01

Optimization completed: The relative first-order optimality measure, 1.939322e-08,
is less than options.TolFun = 1.000000e-06, and the relative maximum constraint
violation, 0.000000e+00, is less than options.TolCon = 1.000000e-06.

Optimization Metric                                          Options
relative first-order optimality =   1.94e-08       TolFun =   1e-06 (default)
relative max(constraint violation) =   0.00e+00    TolCon =   1e-06 (default)
x =
    0.5000
    1.2500
    1.2500
```

- La función que calcula los valores de la función objetivo para `fmincon` es esta.

```
function f=Q2fun(x)
% Ejemplo optimización cuadrática 1
 f=0.5*[x(1) x(2) x(3)]*[4 0 0;0 1 -1; 0 -1 1]*[x(1); x(2); x(3)]+...
       [x(1) x(2) x(3)]*[-8;-6;-6];
end
```

Programación Cuadrática Secuencial

- Volvemos al problema general de Programación No Lineal

$$\begin{aligned} &\underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimizar}}\ f(\boldsymbol{x}) \\ &\text{sujeta a}\quad c_i(\boldsymbol{x}) = \mathbf{0}, \quad i \in \mathcal{E}, \\ &\qquad\qquad c_i(\boldsymbol{x}) \geq \mathbf{0}, \quad i \in \mathcal{I}. \end{aligned}$$

- La idea de los métodos SQP (*Sequential Quadratic Programming*) es desarrollar un proceso iterativo en el que en cada punto $(\boldsymbol{x}_k, \boldsymbol{\lambda}_k)$ se modeliza el problema mediante uno de programación cuadrática

$$\begin{aligned} &\underset{\boldsymbol{p}\in\mathbb{R}^n}{\text{minimizar}}\ f_k + \nabla f_k^T \boldsymbol{p} + \tfrac{1}{2}\boldsymbol{p}^T \nabla^2_{\boldsymbol{x}\boldsymbol{x}} L_k \boldsymbol{p} \\ &\text{sujeta a}\quad \nabla c_i(\boldsymbol{x}_k)^T \boldsymbol{p} + c_i(\boldsymbol{x}_k) = \mathbf{0}, \quad i \in \mathcal{E}, \\ &\qquad\qquad \nabla c_i(\boldsymbol{x}_k)^T \boldsymbol{p} + c_i(\boldsymbol{x}_k) \geq \mathbf{0}, \quad i \in \mathcal{I}, \end{aligned}$$

  cuyo óptimo llevará a $(\boldsymbol{x}_k + \boldsymbol{p}, \boldsymbol{\lambda}_{k+1})$, el nuevo punto.

Dados Un punto de partida $(\boldsymbol{x}_0, \boldsymbol{\lambda}_0)$ y una tolerancia de óptimo.
Hacer $k \leftarrow 0$

Repetir Hasta que se satisfaga la convergencia:

Calcular f_k, ∇f_k, $\nabla^2_{\boldsymbol{x}\boldsymbol{x}} L$, c_k y ∇c_k.

Resolver el subproblema cuadrático

$$\begin{aligned} \underset{\boldsymbol{p}\in\mathbb{R}^n}{\text{minimizar}} \quad & f_k + \nabla f_k^T \boldsymbol{p}_k + \tfrac{1}{2} \boldsymbol{p}_k^T \nabla^2_{\boldsymbol{x}\boldsymbol{x}} L_k \boldsymbol{p}_k \\ \text{sujeta a} \quad & \nabla c_i(\boldsymbol{x}_k)^T \boldsymbol{p}_k + c_i(\boldsymbol{x}_k) = \boldsymbol{0}, \quad i \in \mathcal{E}, \\ & \nabla c_i(\boldsymbol{x}_k)^T \boldsymbol{p}_k + c_i(\boldsymbol{x}_k) \geq \boldsymbol{0}, \quad i \in \mathcal{I}, \end{aligned}$$

para determinar $\boldsymbol{p}_k$ y nuevo vector $\boldsymbol{\lambda}_k$.

Hacer $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{p}_k$, $\boldsymbol{\lambda}_{k+1} \leftarrow \boldsymbol{\lambda}_k$ y $k \leftarrow k + 1$

end

Algoritmo de Programación Cuadrática Secuencial

SQP

SQP sólo con condiciones de igualdad

- **Ejemplo** Resolvamos el siguiente problema:

$$\begin{aligned} \text{minimizar } & -x_1^4 - 2x_2^4 - x_3^4 - x_1^2 x_2^2 - x_1^2 x_3^2 \\ \text{sujeta a } & x_1^4 + x_2^4 + x_3^4 - 25 = 0 \\ & 8x_1^2 + 14x_2^2 + 7x_3^2 - 56 = 0 \end{aligned}$$

- La función de Lagrange es $L_k = -x_1^4 - 2x_2^4 - x_3^4 - x_1^2 x_2^2 - x_1^2 x_3^2 - \\ -\lambda_1 \left(x_1^4 + x_2^4 + x_3^4 - 25\right) - \lambda_2 \left(8x_1^2 + 14x_2^2 + 7x_3^2 - 56\right)$.

- Los diversos vectores y matrices del problema son

$$\nabla f_k = \begin{bmatrix} -4x_1^3 - 2x_1 x_2^2 - 2x_1 x_3^2 \\ -8x_2^3 - 2x_1^2 x_2 \\ -4x_3^3 - 2x_1^2 x_3 \end{bmatrix}, \quad \boldsymbol{J}_k = \begin{bmatrix} 4x_1^3 & 4x_2^3 & 4x_3^3 \\ 16x_1 & 28x_2 & 14x_3 \end{bmatrix}, \quad \boldsymbol{J}_{L_k} = \begin{bmatrix} -4x_1^3 - 2x_1 x_2^2 - 2x_1 x_3^2 - 4\lambda_1 x_1^3 - 16\lambda_2 x_1 \\ -8x_2^3 - 2x_1^2 x_2 - 4\lambda_1 x_2^3 - 28\lambda_2 x_2 \\ -4x_3^3 - 2x_1^2 x_3 - 4\lambda_1 x_3^3 - 14\lambda_2 x_3 \end{bmatrix}$$

$$\nabla^2_{\boldsymbol{xx}} L_k = \begin{bmatrix} -12x_1^2 - 2x_2^2 - 2x_3^2 - 12\lambda_1 x_1^2 - 16\lambda_2 & -4x_1 x_2 & -4x_1 x_3 \\ -4x_1 x_2 & -24x_2^2 - 2x_1^2 - 12\lambda_1 x_2^2 - 28\lambda_2 & 0 \\ -4x_1 x_3 & 0 & -12x_3^2 - 2x_1^2 - 12\lambda_1 x_3^2 - 14\lambda_2 \end{bmatrix}$$

- Para resolverlo, utilicemos este script de Matlab.

```
function [xs,lmdk,fs,k] = SQP_e(faname,gname,wname,x0,lmd0)
% sqp_e.m: SQP algorithm for nonlinear problems with equality constraints
%
% faname, gname, wname: funciones para f.o., gradientes y Hessiana Lag.
% x0, initial point; lmd0; initial Lagrange multiplier;
% xs, solución; lmdk, mult. de Lagrange; fs, función objetivo; k, iter
 xk = x0(:);     lmdk = lmd0;
 n = length(xk); p = length(lmdk);
 p1 = p + 1; k = 0; d = 1; ze = zeros(p,p);
 while d >= sqrt(eps)
    fk = feval(faname,xk);
    Gk = feval(gname,xk);
    ak = fk(2:p1); gk = Gk(:,1); Ak = Gk(:,2:p1)';
    xlmdk = [xk; lmdk];
    Lk    = feval(wname,xlmdk);
    xz    = [Lk -Ak'; -Ak ze]\[Ak'*lmdk-gk; ak];
    d_x   = xz(1:n);
    xk    = xk + d_x;
    lmdk  = Ak'\(Lk*d_x+gk);
    d = norm(d_x); k = k + 1;
 end
 disp('Vector solución:'); xs = xk
 disp('Función objetivo:'); fk = feval(faname,xs); fs = fk(1)
 disp('Número de iteraciones:'); k
end

function z = f_ejsqp1(x)
% Función objetivo y condiciones de Ejemplo SQP1
 x1 = x(1); x2 = x(2); x3 = x(3);
 fk = -x1^4-2*x2^4-x3^4-x1^2*x2^2-x1^2*x3^2;
 ak = [x1^4+x2^4+x3^4-25; 8*x1^2+14*x2^2+7*x3^2-56];
 z = [fk; ak];
end

function z = g_ejsqp1(x)
% Gradientes de f.o. y condiciones de Ejemplo SQP1
 x1 = x(1); x2 = x(2); x3 = x(3);
 gk = [-4*x1^3-2*x1*x2^2-2*x1*x3^2; -8*x2^3-2*x1^2*x2; -4*x3^3-2*x1^2*x3];
 Ak = [4*x1^3 4*x2^3 4*x3^3; 16*x1 28*x2 14*x3];
 z = [gk Ak'];
end
```

```
function L = w_ejsqp1(xlmdk)
% Hessiana de f.o. y condiciones de Ejemplo SQP1
 x = xlmdk(1:3); lmd = xlmdk(4:5);
 x1 = x(1); x2 = x(2); x3 = x(3); lm1 = lmd(1); lm2 = lmd(2);
 w11 = -12*x1^2-2*x2^2-2*x3^2-lm1*12*x1^2-16*lm2; w12 = -4*x1*x2; w13 = -4*x1*x3;
 w22 = -24*x2^2-2*x1^2-12*lm1*x2^2-28*lm2; w33 = -12*x3^2-2*x1^2-12*lm1*x3^2-14*lm2;
 L = [w11 w12 w13; w12 w22 0; w13 0 w33];
end
```

El subproblema cuadrático se resuelve con el procedimiento del subespacio imagen de A para programación cuadrática con condiciones de igualdad.

- Apliquémosle los datos del problema.

```
>> x0=[3;1;3];
>> lmd0 = [-1 -1]';
>> [xs,lmds,fs,k] = SQP_e('f_ejsqp1','g_ejsqp1','w_ejsqp1',x0,lmd0)
Vector solución:
xs =
   1.874065458268392
   0.465819644836092
   1.884720444741611
Función objetivo:
fs =
 -38.284827869947819
Número de iteraciones:
k =
     7
xs =
   1.874065458268392
   0.465819644836092
   1.884720444741611
lmds =
  -1.223463560484408
  -0.274937102065629
fs =
 -38.284827869947819
k =
     7
```

- Comprobemos ahora que el punto obtenido es un mínimo local.

```
>> A=[4*xs(1)^3 4*xs(2)^3 4*xs(3)^3; 16*xs(1) 28*xs(2) 14*xs(3)]   % Jacobiana en el óptimo
A =
  26.327781168218593   0.404308983401344  26.779398394895583
  29.985047332294279  13.042950055410584  26.386086226382556
>> kA=null(A)                                                     % Subespacio núcleo de A
kA =
  -0.696840008344593
   0.222860949715932
   0.681723550907565
>> L = w_ejsqp1([xs;lmds])                                        % Hessiana en el óptimo
L =
   6.278649119268822  -3.491906024680684 -14.128357935929984
  -3.491906024680684  -1.347992348176401                   0
 -14.128357935929984                   0   6.350246509661261
>> kA'*L*kA
ans =
  20.441122695387470
```

¿Qué se hace en el cuadro?

- Vamos a resolver el mismo caso con el software IPOPT, que pasa por ser el mejor de los actualmente en el mercado para Programación No Lineal.

- Tendremos que informarle de dónde están las funciones que calculan la función objetivo, y su gradiente, si queremos, y las condiciones y los gradientes de estas. Usaremos este pequeño programa para hacerlo.

```
function x = IPOPT_ejsqp1
   x0=[3; 1; 3];                          % Punto de partida
   options.ub=[Inf;Inf;Inf];              % Cotas superiores de las variables
   options.lb=[-Inf;-Inf;-Inf];           % Cotas inferiores de las variables

   options.cl = zeros(1,2);    % Condiciones igual a cero
   options.cu = zeros(1,2);

   options.ipopt.print_level           = 5;
   options.ipopt.hessian_approximation = 'exact';        % Matriz Hessiana si
   options.ipopt.derivative_test       = 'first-order';
   options.ipopt.derivative_test       = 'second-order';

   funcs.objective         = @f_ejsqp1;                % Dónde están funciones
   funcs.constraints       = @const_ejsqp1;
   funcs.gradient          = @g_ejsqp1;
   funcs.jacobian          = @jacobian_ejsqp1;
   funcs.jacobianstructure = @() sparse(ones(2,3));
   funcs.hessian           = @hessian_ejsqp1;
   funcs.hessianstructure  = @() sparse(tril(ones(3)));

   [x info] = ipopt(x0,funcs,options);            % Se ejecuta IPOPT
end

function f = f_ejsqp1(x)             % Función objetivo
 x1 = x(1); x2 = x(2); x3 = x(3);
 f  = -x1^4-2*x2^4-x3^4-x1^2*x2^2-x1^2*x3^2;
end
```

```
function g = g_ejsqp1(x)             % Gradiente de la función objetivo
 x1 = x(1); x2 = x(2); x3 = x(3);
 g  = [-4*x1^3-2*x1*x2^2-2*x1*x3^2; -8*x2^3-2*x1^2*x2; -4*x3^3-2*x1^2*x3];
end

function c = const_ejsqp1(x)         % Condiciones del problema
 x1 = x(1); x2 = x(2); x3 = x(3);
 c = [x1^4+x2^4+x3^4-25; 8*x1^2+14*x2^2+7*x3^2-56];
end

function J = jacobian_ejsqp1(x)      % Jacobiana de las constraints
 x1 = x(1); x2 = x(2); x3 = x(3);
 J1 = [4*x1^3 4*x2^3 4*x3^3; 16*x1 28*x2 14*x3];
 J  = sparse(J1);
end

function H = hessian_ejsqp1(x, sigma, lambda)   % Matriz hessiana de f. Laplace
 x1 = x(1); x2 = x(2); x3 = x(3);
 w11 = -12*x1^2-2*x2^2-2*x3^2; w12 = -4*x1*x2; w13 = -4*x1*x3;
 w22 = -24*x2^2-2*x1^2; w33 = -12*x3^2-2*x1^2;
 H  = [w11 w12 w13; w12 w22 0; w13 0 w33];
 H = sigma*H+lambda(1)*[12*x1^2 0 0; 0 12*x2^2 0; 0 0 12*x3^2]+...
                    lambda(2)*[16 0 0; 0 28 0; 0 0 14];
 H = sparse(tril(H));
end
```

- Partiendo del mismo punto x_0, Los resultados que se obtienen son los que siguen.

```
>> IPOPT_ejsqp1
This is Ipopt version trunk, running with linear solver mumps.

Starting derivative checker for first derivatives.

Starting derivative checker for second derivatives.


No errors detected by derivative checker.

Number of nonzeros in equality constraint Jacobian...:        6
Number of nonzeros in inequality constraint Jacobian.:        0
Number of nonzeros in Lagrangian Hessian.............:        6

Total number of variables............................:        3
                     variables with only lower bounds:        0
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints.................:        2
Total number of inequality constraints...............:        0
        inequality constraints with only lower bounds:        0
   inequality constraints with only lower and upper bounds:        0
        inequality constraints with only upper bounds:        0

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0 -2.5400000e+002 1.38e+002 4.28e-001  -1.0 0.00e+000    -  0.00e+000 0.00e+000   0
   1 -9.8397946e+001 4.06e+001 6.25e+000  -1.0 1.32e+000    -  1.00e+000 1.00e+000h  1
   2 -5.1217894e+001 8.96e+000 3.08e+000  -1.0 5.83e-001    -  1.00e+000 1.00e+000h  1
   3 -3.9660992e+001 9.26e-001 2.85e+000  -1.0 2.12e-001    -  1.00e+000 1.00e+000h  1
   4 -3.8369140e+001 6.58e-002 2.29e-001  -1.0 5.09e-002    -  1.00e+000 1.00e+000h  1
   5 -3.8292391e+001 5.59e-003 4.10e-003  -1.7 1.25e-002    -  1.00e+000 1.00e+000h  1
   6 -3.8284834e+001 4.41e-006 2.80e-006  -3.8 4.07e-004    -  1.00e+000 1.00e+000h  1
   7 -3.8284828e+001 1.49e-012 7.46e-013  -8.6 2.48e-007    -  1.00e+000 1.00e+000h  1

Number of Iterations....: 7
```

```
                                   (scaled)                 (unscaled)
Objective...............: -2.2788588017827323e+001  -3.8284827869949901e+001
Dual infeasibility......:  7.4606987254810520e-013   1.2533973858808168e-012
Constraint violation....:  1.3816108750890836e-012   1.4921397450962104e-012
Complementarity.........:  0.0000000000000000e+000   0.0000000000000000e+000
Overall NLP error.......:  1.3816108750890836e-012   1.4921397450962104e-012


Number of objective function evaluations             = 8
Number of objective gradient evaluations             = 8
Number of equality constraint evaluations            = 8
Number of inequality constraint evaluations          = 0
Number of equality constraint Jacobian evaluations   = 8
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations             = 7
Total CPU secs in IPOPT (w/o function evaluations)   =      0.000
Total CPU secs in NLP function evaluations           =      0.000

EXIT: Optimal Solution Found.
ans =
    1.8741
    0.4658
    1.8847
>> IPOPT_ejsqp1
```

- Vamos a resolver el mismo caso con la función `fmincon` de **Matlab**. Le informaremos dónde están las funciones que calculan la función objetivo, y su gradiente si queremos, y las condiciones y los gradientes de estas.

- En este caso la función objetivo y su gradiente están en `fg_ejsqp1.m` y las condiciones y sus gradientes en `cg_ejsqp1.m`.

- Estas son esas funciones:

```
function [fo gfo] = fg_ejsqp1(x)
% Función objetivo y su gradiente de Ejemplo SQP1
 x1 = x(1); x2 = x(2); x3 = x(3);
 fo  = -x1^4-2*x2^4-x3^4-x1^2*x2^2-x1^2*x3^2;
 gfo = [-4*x1^3-2*x1*x2^2-2*x1*x3^2; -8*x2^3-2*x1^2*x2; -4*x3^3-2*x1^2*x3];
end

function [c ceq cg ceqg] = cg_ejsqp1(x)
% Condiciones y gradientes de Ejemplo SQP1
 x1 = x(1); x2 = x(2); x3 = x(3);
 c    = [];
 ceq  = [x1^4+x2^4+x3^4-25; 8*x1^2+14*x2^2+7*x3^2-56];
 cg   = [];
 ceqg = [4*x1^3 4*x2^3 4*x3^3; 16*x1 28*x2 14*x3]';
end
```

- Partiendo del mismo punto x_0, Los resultados que se obtienen son los que siguen.

```
>> options = optimset('Algorithm','sqp',...
'Display','iter','GradObj','on','GradConstr','on'...
);
>> [x fval mflag output lambda grad hessian]=fmincon(@fg_ejsqp1,[3;1;3],...
[],[],[],[],[],[],@cg_ejsqp1,options)
                                                    Norm of  First-order
 Iter F-count            f(x) Feasibility  Steplength        step  optimality
    0       1  -2.540000e+002  1.380e+002                          4.166e+002
    1       3  -1.031438e+002  4.531e+001  1.000e+000  1.703e+000  2.546e+001
    2       5  -5.555541e+001  2.211e+001  1.000e+000  1.085e+000  3.733e+001
    3       7  -3.931698e+001  4.846e+000  1.000e+000  6.290e-001  2.616e+001
    4       9  -3.443752e+001  3.801e-001  1.000e+000  1.880e-001  1.320e+001
    5      16  -3.451286e+001  5.603e-001  2.401e-001  2.620e-001  1.133e+001
    6      23  -4.743444e+001  6.339e+000  1.681e-001  7.912e-001  3.875e+000
    7      25  -3.884668e+001  4.724e-001  1.000e+000  1.616e-001  2.638e+000
    8      27  -3.837025e+001  6.995e-002  1.000e+000  6.153e-002  4.865e-001
    9      29  -3.830612e+001  1.596e-002  1.000e+000  2.845e-002  4.268e-002
   10      31  -3.828490e+001  5.415e-005  1.000e+000  1.961e-003  4.825e-003
   11      33  -3.828483e+001  2.474e-009  1.000e+000  1.083e-005  3.709e-005

Local minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the default value of the function tolerance,
and constraints were satisfied to within the default value of the constraint tolerance.

<stopping criteria details>

x =
   1.874064803822328
  -0.465819854015786
   1.884721085084425
fval =
 -38.284827873204847
```

```
mflag =
     1
output =
         iterations: 11
          funcCount: 33
          algorithm: 'sequential quadratic programming'
            message: [1x782 char]
    constrviolation: 2.473605320574279e-009
           stepsize: 1
      firstorderopt: 3.709210533962449e-005
lambda =
          eqlin: [0x1 double]
       eqnonlin: [2x1 double]
        ineqlin: [0x1 double]
          lower: [3x1 double]
          upper: [3x1 double]
     ineqnonlin: [0x1 double]
grad =
 -40.455060168650469
   4.080648472401320
 -40.018154736754951
hessian =
  14.871537376429519  -4.562119880847577 -10.877223066124762
  -4.562119880847577   7.568783635756974  -0.927972279190895
 -10.877223066124762  -0.927972279190895  10.903957285682973
```

SQP sólo con condiciones de desigualdad

- Sigamos con SQP y apliquémosla ahora a una variante del problema en la que sólo hay condiciones de desigualdad.

- **Ejemplo** Resolvamos este problema:

$$\text{minimizar } \tfrac{1}{2}\left((x_1 - x_3)^2 + (x_2 - x_4)^2\right)$$

$$\text{sujeta a } \quad -\begin{bmatrix} x_1 & x_2 \end{bmatrix}\begin{bmatrix} \frac{1}{4} & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} x_1 & x_2 \end{bmatrix}\begin{bmatrix} \frac{1}{2} \\ 0 \end{bmatrix} + \tfrac{3}{4} \geq 0$$

$$-\tfrac{1}{8}\begin{bmatrix} x_3 & x_4 \end{bmatrix}\begin{bmatrix} 5 & 3 \\ 3 & 5 \end{bmatrix}\begin{bmatrix} x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} x_3 & x_4 \end{bmatrix}\begin{bmatrix} \frac{11}{2} \\ \frac{13}{2} \end{bmatrix} - \tfrac{35}{2} \geq 0$$

- Los diversos vectores y matrices del problema son

$$\nabla f_k = \begin{bmatrix} x_1 - x_3 \\ x_2 - x_4 \\ x_3 - x_1 \\ x_4 - x_2 \end{bmatrix}$$

$$J_k = \begin{bmatrix} -x_1/2 + 0{,}5 & -2x_2 & 0 & 0 \\ 0 & 0 & -(5x_3 + 3x_4)/4 + 11/2 & -(3x_3 + 5x_4)/4 + 13/2 \end{bmatrix} \text{ y}$$

$$\nabla^2_{\boldsymbol{x}\boldsymbol{x}} L_k = \begin{bmatrix} 1 + \lambda_1/2 & 0 & -1 & 0 \\ 0 & 1 + 2\lambda_1 & 0 & -1 \\ -1 & 0 & 1 + 5\lambda_2/4 & 3\lambda_2/4 \\ 0 & -1 & 3\lambda_3/4 & 1 + 5\lambda_2/4 \end{bmatrix}$$

- Para resolver los subproblemas cuadráticos utilizaremos un método de punto interior primal dual un poco más simple que el predictor-corrector que acabamos de ver.

- Este `script` puede resolver este tipo de problemas.

```
function [xs,fs,k] = SQP_ie_1(fun,x0,mu0,epsi)
% SQP algorithm for nonlinear problems with inequality constraints.
% x0, punto de partida; mu0, multiplicadores de Lagrange; epsi, tolerancia óptimo
% xs, óptimo; fs, función objetivo solución; k, iteraciones
 xk = x0(:); muk = mu0(:); n = length(x0); q = length(muk);
 q1 = q + 1; k = 0; In = eye(n); d = 1;
 while d >= epsi
   [fck gAk Lk] = fun([xk;muk]);
   ck = fck(2:q1);
   gk = gAk(:,1);
   Ak = gAk(:,2:q1)';
   %options = optimset('Display','off');
   %[d_x,fval,exitflag,output,lambda] = quadprog(Lk,gk,-Ak,ck,[],[],[],[],[],options);
   [d_x muk] = qp_path_ie(Lk,gk,Ak,-ck,zeros(n,1),epsi);
   xk  = xk + d_x;
   d   = norm(d_x); k = k + 1;
 end
 disp('Óptimo:'); xs = xk, muk      % lambda.ineqlin si quadprod
 disp('Función objetivo:'); fck = f_ejsqp2([xs;muk]); fs = fck(1)
 disp('Iteraciones:'); k
end
```

```
function [fck gAk Lk] = f_ejsqp2(x)
% Función objetivo, condiciones gradientes y der_2 Lag del ejemplo sqpej_2
 x1  = x(1); x2 = x(2); x3 = x(3); x4 = x(4);
 fk  = 0.5*((x1-x3)^2+(x2-x4)^2);
 ck  = [-(x1^2/4+x2^2)+x1/2+0.75; -(5*x3^2+6*x3*x4+5*x4^2)/8+(11*x3+13*x4)/2-35/2];
 fck = [fk; ck];
 gk  = [x1-x3, x2-x4, x3-x1, x4-x2]';
 Ak  = [-x1/2+.5, -2*x2, 0, 0; 0, 0, -(5*x3+3*x4)/4+11/2, -(3*x3+5*x4)/4+13/2];
 gAk = [gk Ak'];
 mu  = x(5:6); mu1 = mu(1); mu2 = mu(2);
 y11 = 1+mu1/2; y22 = 1+2*mu1; y33 = 1+5*mu2/4; y34 = 3*mu2/4;
 Lk  = [y11, 0, -1, 0; 0, y22, 0, -1; -1, 0, y33, y34; 0, -1, y34, y33];
end
```

- El procedimiento de punto interior lo realiza este otro `script`

```
function [xs muk] = qp_path_ie(H,p,A,b,x0,epsi)
% Punto interior primal-dual path-following con partida no factible para prog. cua.
% x0, punto de partida A*x0 - b > 0;
% xs, óptimo; fs, función objetivo en la solución; muk, multiplicadores en óptimo
 n = length(x0); q  = length(b); a_max = 1-1e-5; x = x0(:);
 s = ones(q,1);  mu = ones(q,1); k = 0;
 rsz = s.*mu;
 gap = mean(rsz);
 while gap > epsi                                    % Proceso iterativo
    rs  = -A*x + b + s;
    rl   = H*x + p - A'*mu;
    d_x  = -(H + A'*diag(mu./s)*A)\(A'*((rsz - rs.*mu)./s) + rl); % Newton KKT.
    d_s  = A*d_x-rs;
    d_mu = -(rsz + mu.*d_s)./s;
 % Amplitud de paso
    ind = find(d_s < 0); a_p=1;
    if isempty(ind)==0, a_p = min(a_p,min(-s(ind)./d_s(ind)));   end
    ind = find(d_mu < 0); a_d=1;
    if isempty(ind)==0, a_d = min(a_d,min(-mu(ind)./d_mu(ind))); end
    a_k = a_max*min([1 a_p a_d]);
 % Nuevo punto
    x   = x + a_k*d_x; mu  = mu + a_k*d_mu; s = s+a_k*d_s;
    rsz = s.*mu;       gap = mean(rsz); k = k + 1;
 end
 xs = x; muk = mu;
end
```

- Apliquémosle los datos del problema.

```
>> x0=[1 0.5 2 3]';
>> mu0=[1 1]';
>> [xs,fs,k] = SQP_ie_1(@f_ejsqp2,x0,mu0,1e-5)
Óptimo:
xs =
    2.0447
    0.8527
    2.5449
    2.4856
muk =
    0.9575
    1.1001
Función objetivo:
fs =
    1.4583
Iteraciones:
k =
     7
xs =
    2.0447
    0.8527
    2.5449
    2.4856
fs =
    1.4583
```

- Resolvamos el mismo problema con la función `fmincon` de Matlab.

- La función objetivo y su gradiente están en `fg_ejsqp2.m` y las condiciones y sus gradientes en `cg_ejsqp2.m`.

```
function [fo fog] = fg_ejsqp2(x)
% Función objetivo y gradiente del ejemplo ejsqp2
 x1  = x(1); x2 = x(2); x3 = x(3); x4 = x(4);
 fo  = 0.5*((x1-x3)^2+(x2-x4)^2);
 fog = [x1-x3, x2-x4, x3-x1, x4-x2]';
end

function [c ceq cg ceqg] = cg_ejsqp2(x)
% Función objetivo, condiciones gradientes y der_2 Lag del ejemplo sqpej_2
 x1 = x(1); x2 = x(2); x3 = x(3); x4 = x(4);
 c   = -[-(x1^2/4+x2^2)+x1/2+0.75; -(5*x3^2+6*x3*x4+5*x4^2)/8+(11*x3+13*x4)/2-35/2];
 cg = -[-x1/2+.5, -2*x2, 0, 0; 0, 0, -(5*x3+3*x4)/4+11/2, -(3*x3+5*x4)/4+13/2]';
 ceq  = [];
 ceqg = [];
end
```

- Partiendo del mismo punto x_0, y con los mismos parámetros de optimización, los resultados que se obtienen son los que siguen.

```
>> [x fval mflag output lambda grad hessian]=fmincon(@fg_ejsqp2,[1;0.5;2;3],...
[],[],[],[],[],[],@cg_ejsqp2,options)
                                                        Norm of  First-order
 Iter F-count            f(x) Feasibility  Steplength        step  optimality
    0       1   3.625000e+000  0.000e+000                          1.500e+000
    1       3   8.841912e-001  8.125e-001  1.000e+000  1.346e+000  3.066e+000
    2       5   1.292252e+000  1.222e-001  1.000e+000  5.150e-001  3.150e-001
    3       7   1.455222e+000  3.965e-003  1.000e+000  9.376e-002  7.193e-002
    4      10   1.456116e+000  1.796e-003  7.000e-001  4.723e-002  1.198e-002
    5      12   1.458272e+000  1.488e-005  1.000e+000  6.361e-003  9.076e-004
    6      14   1.458290e+000  6.578e-009  1.000e+000  1.439e-004  3.903e-005
    7      16   1.458290e+000  8.607e-012  1.000e+000  6.809e-006  2.100e-007

Local minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the default value of the function tolerance,
and constraints were satisfied to within the default value of the constraint tolerance.

<stopping criteria details>
x =
   2.044749645910814
   0.852715981057535
   2.544913047857301
   2.485632846451933
fval =
   1.458290458968075
mflag =
     1
```

```
output =
         iterations: 7
          funcCount: 16
          algorithm: 'sequential quadratic programming'
            message: [1x782 char]
    constrviolation: 8.606670931499139e-012
           stepsize: 1
      firstorderopt: 2.099667071320255e-007
lambda =
          eqlin: [0x1 double]
        eqnonlin: [0x1 double]
         ineqlin: [0x1 double]
           lower: [4x1 double]
           upper: [4x1 double]
      ineqnonlin: [2x1 double]
grad =
  -0.500163401946486
  -1.632916865394398
   0.500163401946486
   1.632916865394398
hessian =
   1.379465461685063  -0.173824495237544  -0.999742693536608  -0.132078313168329
  -0.173824495237544   2.586140075166582   0.067423868800302  -0.902783042682583
  -0.999742693536608   0.067423868800302   2.367844422245836   0.727009339021801
  -0.132078313168329  -0.902783042682583   0.727009339021801   1.431944391971096
```

SQP con condiciones de todo tipo

- Tratemos por último el problema general

$$\begin{aligned} &\underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimizar}}\ f(\boldsymbol{x}) \\ &\text{sujeta a} \quad c_i(\boldsymbol{x}) = \mathbf{0}, \quad i \in \mathcal{E}, \\ &\qquad\qquad c_i(\boldsymbol{x}) \geq \mathbf{0}, \quad i \in \mathcal{I}. \end{aligned}$$

- La estrategia sigue siendo la misma, pero con tres mejoras:
  - Una función de mérito, que indique lo que mejora la función objetivo de un paso a otro y cómo se van cumpliendo las condiciones.
  - El cálculo de una amplitud de paso en la dirección calculada en el subproblema cuadrático.
  - La adaptación de $\nabla^2_{\boldsymbol{x}\boldsymbol{x}} L$ en cada iteración, no cálculo de nuevo.

- Las funciones de mérito, y filtros, pretenden ponderar adecuadamente en un proceso de optimización el mejorar la función objetivo y que se cumplan las condiciones. Hay muchas en las referencias de la literatura; la que emplearemos aquí es

$$\psi(\alpha) = f(\boldsymbol{x}_k + \alpha \boldsymbol{p}_k) + \beta \sum_{i \in \mathcal{E}} c_i^2 (\boldsymbol{x}_k + \alpha \boldsymbol{p}_k) - \sum_{i \in \mathcal{I}} \lambda_i c_i (\boldsymbol{x}_k + \alpha \boldsymbol{p}_k)$$

  donde β es un número suficientemente grande.

- Reducir esta función reducirá la función objetivo a lo largo de $\boldsymbol{p}$ y lo que no se cumplen las condiciones de igualdad y de desigualdad ($- \geq 0$).

- El cálculo de la amplitud de paso se hará mediante backtracking.

- La adaptación de $\nabla^2_{\boldsymbol{x}\boldsymbol{x}} L$ se hará con la fórmula BFGS, que conocemos.

```
function [xk,lak,muk,fk,k] = sqp_general_1(fn,gn,x0,p,q,epsi)
% SQP algorithm for general nonlinear programming problems
% x0, punto inicial; lmd0, mu0, multiplicadores de Lagrange; epsi, tolerancia
% xs, solución; fs, función objetivo; k, iteraciones
 xk   = x0(:); n = length(xk); In = eye(n); Zk = In; p1 = p + 1; muk=ones(q,1);
 [fk ak ck]   = fn(xk);
 [gk Aek Aik] = gn(xk);
 k = 0; d = 1; beta=0.5; rho=0.01;
 while d >= epsi
    options = optimset('Display','off');
    %[d_x,fval,exitflag,output,lambda] = quadprog(Zk,gk,-Aik,ck,-Aek,ak,[],[],zeros(n,1),options);
    [d_x d_y d_z] = pcQP_gen_2(Zk, gk, Aek, -ak, Aik, -ck); % Subproblema por punto int.
    alpha=1;
    for kk=1:10                                          % Backtracking linesearch
       xnew      = xk+alpha*d_x; [fkn akn ckn] = fn(xnew);
       merit_old = fk+100*sum(ak.^2)-sum(ck.*muk);
       merit_gra = gk+2*100*Aek'*ak-Aik'*muk;
       merit_new = fkn+100*sum(akn.^2)-sum(ckn.*d_z);
       if merit_new < merit_old+alpha*rho*merit_gra'*d_x, break
       else alpha=alpha*beta;
       end
    end
    d_x = alpha*d_x; xk  = xk + d_x;
    [gk1 Aek1 Aik1] = gn(xk);
    gama_k = (gk1-gk)-(Aek1-Aek)'*d_y-(Aik1-Aik)'*d_z;  % Fórmula BFGS
    qk     = Zk*d_x; dg = d_x'*gama_k; ww = d_x'*qk;
    if dg >= 0.2*ww, thet = 1; else thet = 0.8*ww/(ww-dg); end
    eta = thet*gama_k + (1-thet)*qk;
    phi = 1/ww; cta = 1/(d_x'*eta);
    Zk  = Zk + cta*(eta*eta') - phi*(qk*qk');           % Nueva hessiana fun. de Lagrange
    Aek = Aek1; Aik = Aik1; gk = gk1;
    [fk ak ck] = fn(xk);
    muk = d_z; lak = d_y; d = norm(d_x); k = k + 1;
 end

function [fk ak ck] = f_ex15_4(x)
 x1 = x(1); x2 = x(2);
 fk = x1^2 + x2;
 ak = x1^2 + x2^2 - 9;
 ck = [x1-1; -x1+5; x2-2; -x2+4];
end

function [gk Aek Aik] = g_ex15_4(x)
 x1 = x(1); x2 = x(2);
 gk = [2*x1; 1];
 Aek = [2*x1 2*x2]; Aik = [1 0; -1 0; 0 1; 0 -1];
end
```

- **Ejemplo** Resolvamos el siguiente problema no lineal:

$$\begin{array}{ll} \text{minimizar} & x_1^2 + x_2 \\ \text{sujeta a} & x_1^2 + x_2^2 - 9 = 0 \quad \text{donde} \quad \boldsymbol{A} = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \quad \text{y} \quad \boldsymbol{b} = \begin{bmatrix} 1 \\ -5 \\ 2 \\ -4 \end{bmatrix}. \\ & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \end{array}$$

- Los diversos vectores y matrices del problema son

$$\nabla f_k = \begin{bmatrix} 2x_1 \\ 1 \end{bmatrix} \quad \text{y} \quad J_k = \begin{bmatrix} 2x_1 & 2x_2 \\ 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix}.$$

Apliquemos el programa:

```
>> p=1; q=4;
>> x0=[4 3]';
>> [xs lak muk fs k]=sqp_general_1(@f_ex15_4,@g_ex15_4,x0,p,q,1.e-10)
xs =
   1.000000000000160
   2.828427124751375
lak =
   0.176776695296306
muk =
   1.646446609401296
   0.000000000000000
   0.000000000000000
   0.000000000000000
fs =
   3.828427124751695
k =
     6
```

- Las funciones a las que llama el `script` anterior para calcular la función objetivo, las condiciones, gradientes y jacobianas de este ejemplo son estas.

```
function [fk ak ck] = f_ex15_4(x)
 x1 = x(1); x2 = x(2);
 fk = x1^2 + x2;
 ak = x1^2 + x2^2 - 9;
 ck = [x1-1; -x1+5; x2-2; -x2+4];
end

function [gk Aek Aik] = g_ex15_4(x)
 x1 = x(1); x2 = x(2);
 gk = [2*x1; 1];
 Aek = [2*x1 2*x2]; Aik = [1 0; -1 0; 0 1; 0 -1];
end
```

- Si resolvemos el mismo problema con `fmincon` de **Matlab**. La función objetivo y su gradiente están en `fg_ejsqp3.m` y las condiciones y sus gradientes en `cg_ejsqp3.m`.

```
function [fo fog] = fg_ejsqp3(x)
 x1 = x(1); x2 = x(2);
 fo  = x1^2 + x2;
 fog = [2*x1; 1];
end

function [c ceq cg ceqg] = cg_ejsqp3(x)
 x1 = x(1); x2 = x(2);
 c    = -[x1-1; -x1+5; x2-2; -x2+4];
 ceq  = x1^2 + x2^2 - 9;
 ceqg = [2*x1 2*x2]';
 cg   = -[1 0; -1 0; 0 1; 0 -1]';
end
```

- Partiendo del mismo punto x_0, y con los mismos parámetros de optimización, los resultados que se obtienen son los que siguen.

```
>> [x fval mflag output lambda grad hessian]=fmincon(@fg_ejsqp3,[4;3],...
[],[],[],[],[],[],@cg_ejsqp3,options)
                                                          Norm of  First-order
 Iter F-count            f(x) Feasibility  Steplength        step  optimality
    0       1   1.900000e+001  1.600e+001                          5.000e+000
    1       3   5.562500e+000  8.563e+000  1.000e+000  2.926e+000  2.313e+000
    2       5   4.007813e+000  1.047e+000  1.000e+000  1.023e+000  1.010e+000
    3       7   3.833776e+000  3.029e-002  1.000e+000  1.740e-001  8.322e-002
    4       9   3.828432e+000  2.856e-005  1.000e+000  5.344e-003  2.051e-003
    5      11   3.828427e+000  2.549e-011  1.000e+000  5.049e-006  1.816e-006

Local minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the default value of the function tolerance,
and constraints were satisfied to within the default value of the constraint tolerance.

<stopping criteria details>
x =
   1.000000000000000
   2.828427124750697
fval =
   3.828427124750697
mflag =
     1
output =
         iterations: 5
          funcCount: 11
          algorithm: 'sequential quadratic programming'
            message: [1x782 char]
    constrviolation: 2.549249700223299e-011
           stepsize: 1
      firstorderopt: 1.816258994868925e-006
lambda =
          eqlin: [0x1 double]
       eqnonlin: -0.176776374224092
        ineqlin: [0x1 double]
          lower: [2x1 double]
          upper: [2x1 double]
    ineqnonlin: [4x1 double]
grad =
     2
     1
hessian =
   7.282321097662638   0.203510246567942
   0.203510246567942   0.006185913353082
```

Métodos de puntos interiores

- Consideraremos el problema general:

$$\begin{aligned} &\underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimizar}}\ f(\boldsymbol{x}) \\ &\text{sujeta a} \quad c_i(\boldsymbol{x}) = \mathbf{0}, \quad i \in \mathcal{E}, \\ &\qquad\qquad c_i(\boldsymbol{x}) \geq \mathbf{0}, \quad i \in \mathcal{I}. \end{aligned}$$

Escrito con variables de holgura:

$$\begin{aligned} &\underset{\boldsymbol{x},\boldsymbol{s}}{\text{minimizar}}\ f(\boldsymbol{x}) \\ &\text{sujeta a} \quad \boldsymbol{c}_{\mathcal{E}}(\boldsymbol{x}) = \mathbf{0} \\ &\qquad\qquad \boldsymbol{c}_{\mathcal{I}}(\boldsymbol{x}) - \boldsymbol{s} = \mathbf{0} \\ &\qquad\qquad \boldsymbol{s} \geq \mathbf{0}. \end{aligned}$$

- Las condiciones KKT de este problema se escriben así

$$\begin{aligned} \nabla f(\boldsymbol{x}) - \boldsymbol{A}_{\mathcal{E}}^T(\boldsymbol{x})\boldsymbol{y} - \boldsymbol{A}_{\mathcal{I}}^T(\boldsymbol{x})\boldsymbol{z} &= \boldsymbol{0} \\ \boldsymbol{S}\boldsymbol{z} - \mu\boldsymbol{e} &= \boldsymbol{0} \\ \boldsymbol{c}_{\mathcal{E}}(\boldsymbol{x}) &= \boldsymbol{0} \\ \boldsymbol{c}_{\mathcal{I}}(\boldsymbol{x}) - \boldsymbol{s} &= \boldsymbol{0}, \end{aligned}$$

  debiendo ser $\mu = 0$, $\boldsymbol{s} \geq \boldsymbol{0}$ y $\boldsymbol{z} \geq \boldsymbol{0}$.

  Las matrices $\boldsymbol{A}_{\mathcal{E}}$ y $\boldsymbol{A}_{\mathcal{I}}$ son las jacobianas de las condiciones $\boldsymbol{c}_{\mathcal{E}}(\boldsymbol{x})$ y $\boldsymbol{c}_{\mathcal{I}}(\boldsymbol{x})$, respectivamente y $\boldsymbol{y}$ y $\boldsymbol{z}$ sus multiplicadores de Lagrange. $\boldsymbol{S}$ es una matriz diagonal cuyos coeficientes son los de $\boldsymbol{s}$.

- La segunda ecuación, con $\mu = 0$, es la de complementariedad.

- Como razonábamos en Programación Lineal, para solventar la dificultad que supone la no negatividad de esas variables, se introduce un $\mu > 0$ –denominado parámetro barrera– que haga que $\boldsymbol{S}\boldsymbol{z} = \mu \boldsymbol{e}$ a lo largo del proceso de optimización, lo que permitirá que $\boldsymbol{s}$ y $\boldsymbol{z}$ se mantengan estrictamente positivas. Recalculando μ de iteración en iteración, y haciendo que tienda a cero en el global del proceso se acabará consiguiendo cumplir las condiciones KKT.

- Si esto lo acompasamos con la mejora de una función de mérito que equilibre la minimización de $f(\boldsymbol{x})$ con el que se cumplan las condiciones, esa convergencia llevará a un mínimo local o global del problema.

- A este mismo enfoque se llega si, como en Programación lineal una vez más, se considera el problema barrera del original en el que la función objetivo es $f(\boldsymbol{x}) - \mu \sum_{i=1}^{m} \log s_i$.

- Si se aplica el método de Newton para resolver el sistema de ecuaciones que determinan las condiciones KKT del problema, se tiene en cada iteración un sistema como éste

$$\begin{bmatrix} \nabla^2_{xx} L & \mathbf{0} & -A_{\mathcal{E}}^T(x) & -A_{\mathcal{I}}^T(x) \\ \mathbf{0} & Z & \mathbf{0} & S \\ A_{\mathcal{E}}(x) & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ A_{\mathcal{I}}(x) & -I & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} p_x \\ p_s \\ p_y \\ p_z \end{bmatrix} = - \begin{bmatrix} \nabla f(x) - A_{\mathcal{E}}^T(x) y - A_{\mathcal{I}}^T(x) z \\ Sz - \mu e \\ c_{\mathcal{E}}(x) \\ c_{\mathcal{I}}(x) - s \end{bmatrix},$$

donde L es la función de Lagrange del problema

$$L(x, s, y, z) = f(x) - y^T c_{\mathcal{E}}(x) - z^T \left(c_{\mathcal{I}}(x) - s\right).$$

- Después de obtenida la solución de ese sistema de ecuaciones, se determina un nuevo punto del proceso iterativo (x^+, s^+, y^+, z^+) tal que

$$\begin{aligned} x^+ &= x + \alpha_s^{max} p_s, & s^+ &= s + \alpha_s^{max} p_s \\ y^+ &= y + \alpha_z^{max} p_y, & z^+ &= z + \alpha_z^{max} p_z, \end{aligned}$$

donde

$$\begin{aligned} \alpha_s^{max} &= \text{máx}\left\{\alpha \in (0,1] : s + \alpha p_s \geq (1-\tau)s\right\}, \\ \alpha_z^{max} &= \text{máx}\left\{\alpha \in (0,1] : z + \alpha p_z \geq (1-\tau)z\right\}, \end{aligned}$$

con $\tau \in (0,1)$: normalmente $0{,}995$.

- El proceso pararía cuando el error

$$E(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{y}, \boldsymbol{z}; \mu) = \max \left\{ \|\nabla f(\boldsymbol{x}) - \boldsymbol{A}_{\mathcal{E}}^T(\boldsymbol{x})\boldsymbol{y} - \boldsymbol{A}_{\mathcal{I}}^T(\boldsymbol{x})\boldsymbol{z}\|, \ \|\boldsymbol{S}\boldsymbol{z} - \mu\boldsymbol{e}\|, \right.$$
$$\left. \|\boldsymbol{c}_{\mathcal{E}}(\boldsymbol{x})\|, \ \|\boldsymbol{c}_{\mathcal{I}}(\boldsymbol{x}) - \boldsymbol{s}\| \right\}$$

  estuviese dentro de una tolerancia adecuada.

- Para facilitar la resolución del sistema de ecuaciones anterior, la matriz se reescribe en forma simétrica de esta manera:

$$\begin{bmatrix} \nabla_{xx}^2 L & \mathbf{0} & \boldsymbol{A}_{\mathcal{E}}^T(\boldsymbol{x}) & \boldsymbol{A}_{\mathcal{I}}^T(\boldsymbol{x}) \\ \mathbf{0} & \Psi & \mathbf{0} & -\boldsymbol{I} \\ \boldsymbol{A}_{\mathcal{E}}(\boldsymbol{x}) & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{A}_{\mathcal{I}}(\boldsymbol{x}) & -\boldsymbol{I} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{p}_{\boldsymbol{x}} \\ \boldsymbol{p}_{\boldsymbol{s}} \\ -\boldsymbol{p}_{\boldsymbol{y}} \\ -\boldsymbol{p}_{\boldsymbol{z}} \end{bmatrix} = -\begin{bmatrix} \nabla f(\boldsymbol{x}) - \boldsymbol{A}_{\mathcal{E}}^T(\boldsymbol{x})\boldsymbol{y} - \boldsymbol{A}_{\mathcal{I}}^T(\boldsymbol{x})\boldsymbol{z} \\ \boldsymbol{z} - \mu\boldsymbol{S}^{-1}\boldsymbol{e} \\ \boldsymbol{c}_{\mathcal{E}}(\boldsymbol{x}) \\ \boldsymbol{c}_{\mathcal{I}}(\boldsymbol{x}) - \boldsymbol{s} \end{bmatrix},$$

  donde $\Psi = \boldsymbol{S}^{-1}\boldsymbol{Z}$.

- El algoritmo básico de punto interior para Programación No lineal es este.

Dados Una $f(\boldsymbol{x})$, un punto inicial, una $\boldsymbol{B}_0$ si es el caso, un $\mu > 0$,
unos $\eta, \sigma \in (0,1)$, una tol. ϵ_μ y ϵ_{tol}

Repetir hasta que $E(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{y}, \boldsymbol{z}, 0) \leq \epsilon_{tol}$

  Repetir hasta que $E(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{y}, \boldsymbol{z}, \mu) \leq \epsilon_\mu$

    Resolver sistema KKT: obtener $\boldsymbol{p} = (\boldsymbol{p_x}, \boldsymbol{p_s}, \boldsymbol{p_y}, \boldsymbol{p_z})$

    Calcular α_s^{max}, α_z^{max}; hacer $\boldsymbol{p}_w = (\boldsymbol{p_x}, \boldsymbol{p_s})$.

    Calcular amplitudes de paso α_s y α_z que cumplan
    $\Phi_\nu(\boldsymbol{x}_k + \alpha_s \boldsymbol{p_x}, \boldsymbol{s}_k + \alpha_s \boldsymbol{p_s}) \leq \Phi_\nu(\boldsymbol{x}_k, \boldsymbol{s}_k) + \eta \alpha_s \nabla \Phi_\nu(\boldsymbol{x}_k, \boldsymbol{s}_k, \boldsymbol{p}_w)^T \boldsymbol{p}$

    Obtener $(\boldsymbol{x}_{k+1}, \boldsymbol{s}_{k+1}, \boldsymbol{y}_{k+1}, \boldsymbol{z}_{k+1})$

    Si cuasi Newton, adaptar $\boldsymbol{B}_k$, $k \leftarrow k+1$

  end

  Hacer $\mu \leftarrow \sigma\mu$ y adaptar ϵ_μ

end

Aceptabilidad del paso de Newton

- Uno de las cuestiones siempre presente en todos los procedimientos que estudiamos es el dar el paso más adecuado en la dirección de búsqueda.

- En el caso del método primal-dual, para calcular esa amplitud de paso, se usa una función de mérito en términos de la función barrera:

$$\Phi_\nu(\boldsymbol{x}, \boldsymbol{s}) = f(\boldsymbol{x}) - \mu \sum_{i=1}^{m} \log s_i + \nu \|\boldsymbol{c}_{\mathcal{E}}(\boldsymbol{x})\| + \nu \|\boldsymbol{c}_{\mathcal{I}}(\boldsymbol{x}) - \boldsymbol{s}\|,$$

  donde las normas pueden ser a 1 o la 2 y ν un parámetro que pondere el peso del incumplimiento de los factores a los que afecta con el de los logaritmos.

- En la determinación de la amplitud de paso, una vez calculado el paso $\boldsymbol{p(x,s)}$ y los α_s^{max} y α_z^{max}, mediante backtracking u otra estrategia similar, se calculan

$$\alpha_s \in (0, \alpha_s^{max}], \qquad \alpha_z \in (0, \alpha_z^{max}]$$

que proporcionen un decremento suficiente en la función de mérito anterior.

- El nuevo paso será

$$\boldsymbol{x}^+ = \boldsymbol{x} + \alpha_s \boldsymbol{p_s}, \qquad \boldsymbol{s}^+ = \boldsymbol{s} + \alpha_s \boldsymbol{p_s}$$
$$\boldsymbol{y}^+ = \boldsymbol{y} + \alpha_z \boldsymbol{p_y}, \qquad \boldsymbol{z}^+ = \boldsymbol{z} + \alpha_z \boldsymbol{p_z}.$$

Aproximación cuasi Newton

- La idea es adaptar en cada paso del proceso iterativo la matriz $\nabla^2_{\boldsymbol{x}\boldsymbol{x}} L$ mediante una fórmula adecuada: BFGS, por ejemplo, u otras.

- Si es BFGS, la aproximación $\boldsymbol{B}$ de esa matriz estaría dada, de iteración en iteración, por la expresión

$$\boldsymbol{B}_{k+1} = \boldsymbol{B}_k - \frac{\boldsymbol{B}_k \boldsymbol{x}_k \boldsymbol{x}_k^T \boldsymbol{B}_k}{\boldsymbol{x}_k^T \boldsymbol{B}_k \boldsymbol{x}_k} + \frac{\boldsymbol{l}_k \boldsymbol{l}_k^T}{\boldsymbol{l}_k^T \boldsymbol{x}_k},$$

donde

$$\boldsymbol{x}_k = \boldsymbol{x}_k - \boldsymbol{x}_{k-1} \quad \text{y}$$
$$\boldsymbol{l}_k = \nabla_{\boldsymbol{x}} L(\boldsymbol{x}_k, \boldsymbol{s}_k, \boldsymbol{y}_k, \boldsymbol{z}_k) - \nabla_{\boldsymbol{x}} L(\boldsymbol{x}_{k-1}, \boldsymbol{s}_k, \boldsymbol{y}_k, \boldsymbol{z}_k).$$

Implementación práctica en Matlab

- Vamos a implementar en Matlab un caso particular del problema:

$$\begin{aligned} &\underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimizar}}\ f(\boldsymbol{x}) \\ &\text{sujeta a} \quad \boldsymbol{c}(\boldsymbol{x}) \geq \boldsymbol{0}. \end{aligned}$$

  El caso en el que haya además condiciones de igualdad es esencialmente similar, aunque complica un poco la formulación.

- Lo que sigue es una adaptación del conocido programa LOQO.

- Escrito con variables de holgura, el problemas es

$$\begin{aligned} &\underset{\boldsymbol{x},\boldsymbol{s}}{\text{minimizar}}\ f(\boldsymbol{x}) \\ &\text{sujeta a } \boldsymbol{c}(\boldsymbol{x}) - \boldsymbol{s} = \boldsymbol{0} \\ &\qquad\qquad\quad \boldsymbol{s} \geq \boldsymbol{0}. \end{aligned}$$

  Su función de Lagrange $L(\boldsymbol{x},\boldsymbol{s},\boldsymbol{\lambda},\boldsymbol{\mu}) = f(\boldsymbol{x}) - \boldsymbol{\lambda}^T(\boldsymbol{c}(\boldsymbol{x}) - \boldsymbol{s}) - \boldsymbol{\mu}^T\boldsymbol{s}$.

- Las condiciones de óptimo de KKT de este problema son estrictamente

$$\begin{aligned} \nabla f(\boldsymbol{x}) - \boldsymbol{A}^T(\boldsymbol{x})\lambda &= \boldsymbol{0} \\ \boldsymbol{\lambda} &= \boldsymbol{\mu} \\ \boldsymbol{c}(\boldsymbol{x}) - \boldsymbol{s} &= \boldsymbol{0} \\ s_i \mu_i &= 0 \\ \boldsymbol{s} &\geq \boldsymbol{0}. \end{aligned}$$

- Para evitar la condición de no negatividad del vector $\boldsymbol{s}$ se introduce esta formulación equivalente del problema (tradicional como vimos en PL)

$$\begin{aligned} &\underset{\boldsymbol{x},\boldsymbol{s}}{\text{minimizar}} \; f(\boldsymbol{x}) - \mu \sum \ln s_i \\ &\text{sujeta a} \quad \boldsymbol{c}(\boldsymbol{x}) - \boldsymbol{s} = \boldsymbol{0}, \end{aligned}$$

cuya función de Lagrange es

$$L(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) - \mu \sum \ln s_i - \boldsymbol{\lambda}^T (\boldsymbol{c}(\boldsymbol{x}) - \boldsymbol{s}).$$

- Sus condiciones KKT son

$$\begin{aligned} \nabla f(\boldsymbol{x}) - \boldsymbol{A}^T(\boldsymbol{x})\boldsymbol{\lambda} &= \boldsymbol{0} \\ \boldsymbol{c}(\boldsymbol{x}) - \boldsymbol{s} &= \boldsymbol{0} \\ -\mu \boldsymbol{S}^{-1}\boldsymbol{e} + \boldsymbol{\lambda} &= \boldsymbol{0}, \end{aligned}$$

que, si nos fijamos bien, son idénticas a las anteriores, salvo el escalar μ (que no el vector $\boldsymbol{\mu}$) que es el que tiene como objetivo que las variables del vector $\boldsymbol{s}$ no se acerquen mucho a cero.

- Si a la tercera ecuación se le multiplica por $\boldsymbol{S}$ se llega al conocido sistema primal-dual, básico en el algoritmo que implementamos.

- Aplicándole Newton, en cada paso se resuelve el siguiente sistema

$$\begin{bmatrix} \nabla^2_{xx} L & \boldsymbol{0} & -\boldsymbol{A}_k^T \\ \boldsymbol{0} & \boldsymbol{\Lambda}_k & \boldsymbol{S}_k \\ \boldsymbol{A}_k & -\boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{x}_k \\ \Delta \boldsymbol{s}_k \\ \Delta \boldsymbol{\lambda}_k \end{bmatrix} = \begin{bmatrix} -\nabla f(\boldsymbol{x}) + \boldsymbol{A}_k^T \boldsymbol{\lambda}_k \\ \mu \boldsymbol{e} - \boldsymbol{S}_k \boldsymbol{\Lambda}_k \boldsymbol{e} \\ -\boldsymbol{c}_k + \boldsymbol{s}_k \end{bmatrix}.$$

- La matriz de este sistema no es simétrica pero puede hacerse así: multipliquemos la primera ecuación por $-\boldsymbol{I}$ y la segunda por $-\boldsymbol{S}_k^{-1}$:

$$\begin{bmatrix} -\boldsymbol{H}_k & \boldsymbol{0} & \boldsymbol{A}_k^T \\ \boldsymbol{0} & -\boldsymbol{S}_k^{-1}\boldsymbol{\Lambda}_k & -\boldsymbol{I} \\ \boldsymbol{A}_k & -\boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{x}_k \\ \Delta \boldsymbol{s}_k \\ \Delta \boldsymbol{\lambda}_k \end{bmatrix} = \begin{bmatrix} \boldsymbol{\sigma}_k \\ -\boldsymbol{\gamma}_k \\ \boldsymbol{\rho}_k \end{bmatrix},$$

donde $\boldsymbol{H}_k = \boldsymbol{H}(\boldsymbol{x}, \boldsymbol{\lambda}) = \nabla^2 f(\boldsymbol{x}) - \sum \lambda_i \nabla^2 c_i(\boldsymbol{x})$

$$\boldsymbol{A}_k = \nabla \boldsymbol{c}(\boldsymbol{x})$$

$$\boldsymbol{\sigma}_k = \nabla f(\boldsymbol{x}) - \boldsymbol{A}_k^T \boldsymbol{\lambda}_k = \boldsymbol{g}_k - \boldsymbol{A}_k^T \boldsymbol{\lambda}_k$$

$$\boldsymbol{\gamma}_k = \mu \boldsymbol{S}_k^{-1} \boldsymbol{e} - \boldsymbol{\lambda}_k$$

$$\boldsymbol{\rho}_k = \boldsymbol{s}_k - \boldsymbol{c}_k.$$

- El vector $\boldsymbol{\rho}_k$ mide la no factibilidad del problema primal. Por analogía con Programación Lienal, $\boldsymbol{\sigma}_k$ mide lo lejos que está $(\boldsymbol{x}_k, \boldsymbol{\lambda}_k)$ de ser un punto factible del problema dual.

- El algoritmo procedería iterativamente a partir de la dirección $[\Delta \boldsymbol{x}_k,\ \Delta \boldsymbol{s}_k,\ \Delta \boldsymbol{\lambda}_k]^T$, supuesto que sea de descenso, obteniéndose un nuevo punto

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k \Delta \boldsymbol{x}_k \\ \boldsymbol{s}_{k+1} &= \boldsymbol{s}_k + \alpha_k \Delta \boldsymbol{s}_k \\ \boldsymbol{\lambda}_{k+1} &= \boldsymbol{\lambda}_k + \alpha_k \Delta \boldsymbol{\lambda}_k \end{aligned}$$

  calculando para ello la amplitud de paso α_k que minimiza una función de mérito adecuada.

- La función de mérito que se puede usar es

$$\Psi_{\beta,\mu}(\boldsymbol{x},\boldsymbol{s}) = f(\boldsymbol{x}) - \mu \sum \ln s_i + \frac{\beta}{2} \|\boldsymbol{s} - \boldsymbol{c}(\boldsymbol{x})\|_2^2,$$

  donde μ cambia de iteración en iteración, tendiendo a cero, y β es una constante que penalice la no factibilidad del primal y que haga que la dirección sea realmente de descenso en $\Psi_{\beta,\mu}(\boldsymbol{x},\boldsymbol{s})$ si el problema no es convexo.

- Si el problema no es convexo, la matriz hessiana de la función de Lagrange del problema hay que modificarla en cada iteración para que sea definida positiva. Para ello se hace

$$\hat{\boldsymbol{H}}_k = \boldsymbol{H}_k(\boldsymbol{x}_k, \boldsymbol{\lambda}_k) + \eta \boldsymbol{I},$$

  escogiéndose η a partir del valor propio más pequeño de $\boldsymbol{H}_k(\boldsymbol{x}_k, \boldsymbol{\lambda}_k)$.

- El listado que sigue recoge todas las consideraciones que acabamos de hacer. Esta basado en los artículos de LOQO y unos `scripts` suministrados por Andreas Antoniou, Profesor de la Universidad de Victoria de Canadá.

```
function [xk,yk,lmdk,fs,ck,k,L] = NLP_IP_nc(fcname,gname,hname,x0,y0,lmd0,tau0)
% Método de punto interior para min. f(x)
%                              s. a g(x) >= 0
 xk = x0(:); yk = y0(:); q = length(y0); lmdk = lmd0(:);
 tau = tau0; n = length(x0); E2 = eye(n); ei = 5e-4; eo = 1e-5;
 q1 = q + 1; L = 0; K = 0; do = 1;
 % Bucle exterior
 while do >= eo,
    xk0 = xk; yk0 = yk; lmdk0 = lmdk;
    k = 0; di = 1;
  % Bucle interior
    while di >= ei,
       bt = 0;
       fck = feval(fcname,xk); Gk = feval(gname,xk);
       fk  = fck(1);    ck  = fck(2:q1);
       gk  = Gk(:,1);   Ak  = Gk(:,2:q1)';
       yki = 1./yk;     wwk = yki.*lmdk;
       Yki = diag(yki); Lk  = diag(lmdk);
       xlk = [n;q;xk;lmdk];
       Hk  = feval(hname,xlk);
       ew  = min(eig(Hk));
       if ew <= 0
          eb = -1.2*ew + 1e-6;
          Hk = Hk + eb*E2;
       end
       Nk  = Hk + Ak'*(Lk*Yki)*Ak;
       R   = cholinc(sparse(Nk),'inf');
       Nki = R\(R'\eye(n));                          % Mat = [-Hk         zeros(n,q)  Ak'      ;...
       gak = tau*yki - lmdk;                         %        zeros(q,n) -diag(wwk)  -eye(q)   ;...
       rk  = yk - ck;                                %        Ak         -eye(q)      zeros(q)];
       dx = Nki*(-gk + tau*Ak'*yki + Ak'*(wwk.*rk)); % [L D P] = ldl(Mat);
       dL = wwk.*(rk - Ak*dx) + gak;                 % rhs = [gk-Ak'*lmdk; -gak; rk];
       dy = Ak*dx - rk;                              % [dx; dy; dL] = P*(L'\(D\(L\(P'*rhs))));
    % Comprobar si la dirección obtenida es de descenso en función de mérito
       ksi = gk - tau*Ak'*yki;
       sk = -ksi'*(Nki*ksi) + tau*(yki'*rk) + ksi'*(Nki*Ak')*(wwk.*rk)...
           -bt*norm(rk)^2;
       if sk >= 0,
          bmin = (sk+bt*norm(rk)^2)/(norm(rk)^2);      % bmin = sk/(norm(rk)^2);
          bt   = 10*bmin;
       end
```

```
    % Cálculo de line search en x e y
      awk  = max([-dy./yk; -dL./lmdk]);
      ak   = 0.95/awk;
      aks  = lsearch_merit(fcname,xk,yk,dx,dy,ak,tau,bt);
      xk   = xk + aks*dx;
      yk   = yk + aks*dy;
      lmdk = lmdk + aks*dL;
      k  = k + 1;
      di = aks*(norm(dx) + norm(dy) + norm(dL));
   end
 % Recalcular parámetro de barrera tau.
   tsi = q*min(yk.*lmdk)/(yk'*lmdk);
   tau = 0.1*(min([0.05*(1-tsi)/tsi,2]))^3*(yk'*lmdk)/q;
   K  = k + K; L = L + 1;
   do = norm(xk-xk0) + norm(yk-yk0) + norm(lmdk-lmdk0);
end
fck = feval(fcname,xk); fs  = fck(1); k = K;

function [a psm] = lsearch_merit(fcname,xk,yk,dx,dy,ak,tau,bt)
 aa = (0:0.01:1)*ak;
 ps = zeros(101,1);
 for i = 1:101
    ai = aa(i);
    xki = xk + ai*dx;
    yki = yk + ai*dy;
    fci = feval(fcname,xki);
    fi = fci(1);
    cki = fci(2:end);
    yci = yki - cki;
    ps(i) = fi - tau*sum(log(yki)) + 0.5*bt*(yci'*yci); % F. mérito
 end
 [psm ind] = min(ps); a = min(1,aa(ind));
end
```

- **Ejemplo** Resolvamos el siguiente problema no lineal de la colección de K. Schittkowski:

$$\begin{aligned} \text{minimizar } & (x_1-2)^2+(x_2-1)^2 \\ \text{sujeta a } & -0{,}25x_1^2-x_2^2+1\geq 0 \\ & x_1-2x_2+1=0. \end{aligned}$$

  Partiremos del punto $\boldsymbol{x}_0=[2,\ 2]^T$.

- Como hay una condición de igualdad, vamos a resolver

$$\begin{aligned} \text{minimizar } & (x_1-2)^2+(x_2-1)^2 \\ \text{sujeta a } & -0{,}25x_1^2-x_2^2+1\geq 0 \\ & x_1-2x_2+1\geq 0 \\ & -x_1+2x_2-1\geq 0. \end{aligned}$$

- Los diversos vectores y matrices del problema son:

$$\nabla f_k = \begin{bmatrix} 2(x_1-2) \\ 2(x_2-1) \end{bmatrix}, \; J_k = \begin{bmatrix} -0{,}5x_1 & -2x_2 \\ 1 & -2 \end{bmatrix} \text{ y } \boldsymbol{H}_k = \begin{bmatrix} 2+0{,}5\boldsymbol{\lambda}_1 & 0 \\ 0 & 2+2\boldsymbol{\lambda}_1 \end{bmatrix}.$$

- Usemos el programa:

```
>> x0=[2;2];
>> y1=[1;1;1];
>> lmd1=[1;1;1];
>> mu1=0.001;
>> [xs,yk,lmdk,fs,ck,k,L] = NLP_IP_nc('f_ex15_8','g_ex15_8','h_ex15_8',x0,y1,lmd1,mu1)
xs =
   0.822466525719178
   0.911233262859589
yk =
  1.0e-003 *
   0.541144176123609
   0.000000000000000
   0.000000000000000
lmdk =
  1.0e+012 *
   0.000000000000000
   3.876880823233180
   3.876968062445849
fs =
   1.394464616674418
ck =
  1.0e-003 *
   0.541144176123609
                   0
                   0
k =
    59
L =
     3
```

- Las `functions` para calcular la función objetivo, condiciones, gradientes, jacobianas y hessianas son estas.

```
function z = f_ex15_8(x)
 x1 = x(1); x2 = x(2);
 ck = [-0.25*x1^2-x2^2+1; x1-2*x2+1; -x1+2*x2-1];
 fk = (x1-2)^2+(x2-1)^2;
 z  = [fk; ck];
end

function z = g_ex15_8(x)
 x1 = x(1); x2 = x(2);
 Ak = [-0.5*x1, -2*x2; 1, -2; -1, 2];
 gk = [2*(x1-2); 2*(x2-1)];
 z  = [gk,  Ak'];
end

function z = h_ex15_8(xlk)
 lmdk = xlk(5:7);
 lmd1 = lmdk(1); lmd2 = lmdk(2); lmd3 = lmdk(3);
 z = [2+0.5*lmd1, 0; 0, 2+2*lmd1];
end
```

- Resolvamos el mismo caso con la función `fmincon` de Matlab.

```
% Script de prueba de fmincon
% Resuelve el problema Ex15_8

x0=[2;2]; options = optimset('Algorithm','interior-point',...
        'Display','iter','GradObj','on','GradConstr','on',...
        'Hessian','user-supplied','HessFcn',@hess_ex158_M);

[x fval mflag output lambda grad hessian]=fmincon(@f_ex158_M,x0,...
          [],[],[],[],[],[],@g_ex158_M,options)
```

Las funciones `f_ex158_M`, `g_ex158_M` y `hess_ex158_M` que proporcionan los datos son:

```
function [f g] = f_ex158_M(x)
 x1 = x(1); x2 = x(2);
 f  = (x1-2)^2+(x2-1)^2;
 g  = [2*(x1-2); 2*(x2-1)];
end

function [c ceq cg ceqg] = g_ex158_M(x)
 x1 = x(1); x2 = x(2);
 c    = 0.25*x1^2+x2^2-1;
 ceq  = x1-2*x2+1;
 cg   = [0.5*x1; 2*x2];
 ceqg = [1;-2];
end

function h = hess_ex158_M(x,lambda)
 h = [2 0;0 2]+lambda.ineqnonlin(1)*[0.5 0;0 2];
end
```

- Estos son los resultados.

```
>> Script_fmincon_ex158
                                                 First-order      Norm of
 Iter F-count            f(x)  Feasibility   optimality         step
    0       1   1.000000e+000   4.000e+000   1.332e+000
    1       2   5.198962e-001   7.344e-001   6.048e-001   1.107e+000
    2       3   1.280727e+000   6.368e-002   2.827e-001   4.740e-001
    3       4   1.498705e+000   0.000e+000   5.490e-002   1.000e-001
    4       5   1.416079e+000   0.000e+000   1.215e-002   3.685e-002
    5       6   1.393937e+000   0.000e+000   2.554e-004   1.009e-002
    6       7   1.393468e+000   0.000e+000   1.417e-006   2.146e-004

Local minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the default value of the function tolerance,
and constraints were satisfied to within the default value of the constraint tolerance.

<stopping criteria details>

x =
   0.822874584569342
   0.911437292284671
fval =
   1.393467596850670
mflag =
     1

output =
         iterations: 6
          funcCount: 7
    constrviolation: 0
           stepsize: 2.146411271157580e-004
          algorithm: 'interior-point'
      firstorderopt: 1.416750245231135e-006
       cgiterations: 0
            message: [1x782 char]
lambda =
          eqlin: [0x1 double]
      eqnonlin: 1.594492450496799
        ineqlin: [0x1 double]
          lower: [2x1 double]
          upper: [2x1 double]
    ineqnonlin: 1.846594194644163
grad =
  -2.354250830861316
  -0.177125415430658
hessian =
   2.923297097322081                   0
                   0   5.693188389288325
```