# Optimización sin Restricciones e Introducción a la Complejidad

Dr. Ing. Claudio Risso (crisso@fing.edu.uy)

**Instituto de Computación (FING - UDELAR)**

Curso Optimización Continua y Aplicaciones (OCA)
(Setiembre 2023)

# Optimización sin Restricciones (óptimos)

## Definición

*Dado un conjunto $S \subseteq \mathbb{R}^n$ y una función $f : S \to \mathbb{R}$, decimos que:*

- *El punto $\bar{x} \in S$ es un mínimo global de $f$ en $S$ si y sólo si $f(x) \geq f(\bar{x})$ para todo $x \in S$.*
- *El punto $\bar{x} \in S$ es un mínimo local de $f$ sii existe $\epsilon > 0$ tal que $f(x) \geq f(\bar{x})$ para todo $x \in S$ con $||x - \bar{x}|| < \epsilon$.*

Substituyendo "$\geq$" por "$\leq$" podemos definir máximo global y local.

Tomaremos como problema de referencia la búsqueda del mínimo.

Si buscamos un máximo, basta con multiplicar $f$ por -1 para transformar el problema en la búsqueda de un mínimo.

Nos referiremos a óptimo o mínimo local/global indistintamente.

# Propiedades de los Óptimos

## Propiedades

- *Si $\bar{x} \in S$ es un óptimo global de $f : S \to \mathbb{R}$, también se cumple que $\bar{x}$ es óptimo local.*
- *Lo opuesto a lo anterior no es cierto en general.*
- *Si la función $f$ es convexa, un óptimo local también es global.*
- *Si la función $f$ es diferenciable, $S$ es abierto, y $\bar{x} \in S$ es un óptimo local, se cumple que $\nabla f(\bar{x}) = \vec{0}$ (necesario).*
- *Si $f$ es doblemente diferenciable, $\nabla f(\bar{x}) = \vec{0}$ y $H[f(\bar{x})]$ es semidefinida positiva, $\bar{x}$ es un óptimo local (suficiente).*

El óptimo de $f(x, y) = x^2 + 4y^2 - 12x - 8y + 10$ es $(6, 1)$, porque $\nabla f(\bar{x}) = \vec{0}$ y $f$ es convexa. El valor óptimo es -30.

# Búsqueda de Óptimos

No todas las funciones tienen óptimo. Pensar que aun estando acotada inferiormente, $f(x) = e^x$ no tiene mínimo en $\mathbb{R}$.

Conocido que un problema de optimización sin restricciones tiene óptimo para una función doblemente diferenciable, una estrategia para hallarlo es encontrar las soluciones al sistema $\nabla f(x) = \vec{0}$, quedarse con las que tiene $H[f(x)]$ semidefinido positivo y comparar los valores de $f$ hasta encontrar el menor, para lo cual, el conjunto de soluciones debe ser finito.

Lo anterior no es posible o práctico en general, así que se usan métodos de descenso (iterativos), para construir una sucesión de puntos con valores decrecientes de $f$.

Los métodos de descenso buscan mínimos locales. Si la función es convexa y el método converge a un óptimo local, podemos estar seguros que también es global.

# Algoritmos de Descenso

## Definición

*Dada $f : \mathbb{R}^n \to \mathbb{R}$ y un punto $\bar{x} \in \mathbb{R}^n$ cualquiera, diremos que $\vec{d}$ es una dirección de descenso de la función $f$ en $\bar{x}$ si y sólo si, existe $\epsilon > 0$ tal que $f(\bar{x} + \tau \cdot \vec{d}) < f(\bar{x})$ para todo $0 < \tau < \epsilon$.*

Observar que una dirección de descenso en un punto define una función en una variable $h(\tau) = f(\bar{x} + \tau \cdot \vec{d})$.

## Propiedad

*Sea $f : \mathbb{R}^n \to \mathbb{R}$ una función diferenciable en $\mathbb{R}^n$, y una dirección $\vec{d} \in \mathbb{R}^n$ cualquiera. Si $\nabla f(x) \cdot \vec{d} < 0$ (producto escalar), entonces $\vec{d}$ es una dirección de descenso de $f$ en $x$.*

# Algoritmos de Descenso

Buscamos el mínimo de una función $f : \mathbb{R}^n \to \mathbb{R}$, a través del límite $\lim_n x_n \to \bar{x}$ de una sucesión de puntos $\{x_n \in \mathbb{R}^n\}$.

El esquema general de un Algoritmo de Descenso es el siguiente:

1. Se tiene un punto inicial $x_0$.
2. A partir de un punto $x_n$ cualquiera, se toma una dirección $\vec{d}_n$ de descenso en ese punto.
3. Se busca $x_{n+1}$ en la dirección $\vec{d}_n$ que cumpla $f(x_{n+1}) < f(x_n)$, y se actualiza el punto (i.e. $x_n \leftarrow x_{n+1}$, sobrescribir $x_n$).
4. Si no se cumple la *condición de salida*, volver al punto 2.

El algoritmo admite muchas variantes, según cómo se elige: la dirección $\vec{d}$, el paso $\tau$ en esa dirección, e incluso el valor de $x_0$.

Al paso 3 se lo conoce como *búsqueda lineal*.

# Steepest Descent

El algoritmo de descenso más clásico elige $\vec{d_n} = -\nabla f(x_n)$.

Observar que $\nabla^T f(x_n) \cdot \vec{d_n} = -||\nabla f(x_n)||^2 < 0$, porque si $\nabla f(x_n) = \vec{0}$ ya estaría en un candidato a óptimo local.

Tomemos $f(x,y) = x^2 + 4y^2 - 12x - 8y + 10$, $\nabla f = \begin{bmatrix} 2x - 12 \\ 8y - 8 \end{bmatrix}$, $\vec{d} = -\nabla f$ (steepest descent) y usamos la búsqueda lineal exacta.

Si $z_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $\vec{d_0} = \begin{bmatrix} 12 \\ 8 \end{bmatrix}$, $h(\tau) = f(\begin{bmatrix} 0 \\ 0 \end{bmatrix} + \tau \begin{bmatrix} 12 \\ 8 \end{bmatrix})$, de donde

$h(\tau) = (12\tau)^2 + 4(8\tau)^2 - 12(12\tau) - 8(8\tau) + 10 = 400\tau^2 - 208\tau + 10$.

Podemos resolver $h'(\tau) = 0$ (búsqueda lineal exacta) para hallar

$\tau = \frac{208}{800} = \frac{13}{50} = 0.26$, con lo cual $z_1 = \begin{bmatrix} 3.12 \\ 2.08 \end{bmatrix}$ $(f(z_1) = -17.04)$.

# Steepest Descent

El algoritmo de descenso más clásico elige $\vec{d_n} = -\nabla f(x_n)$.

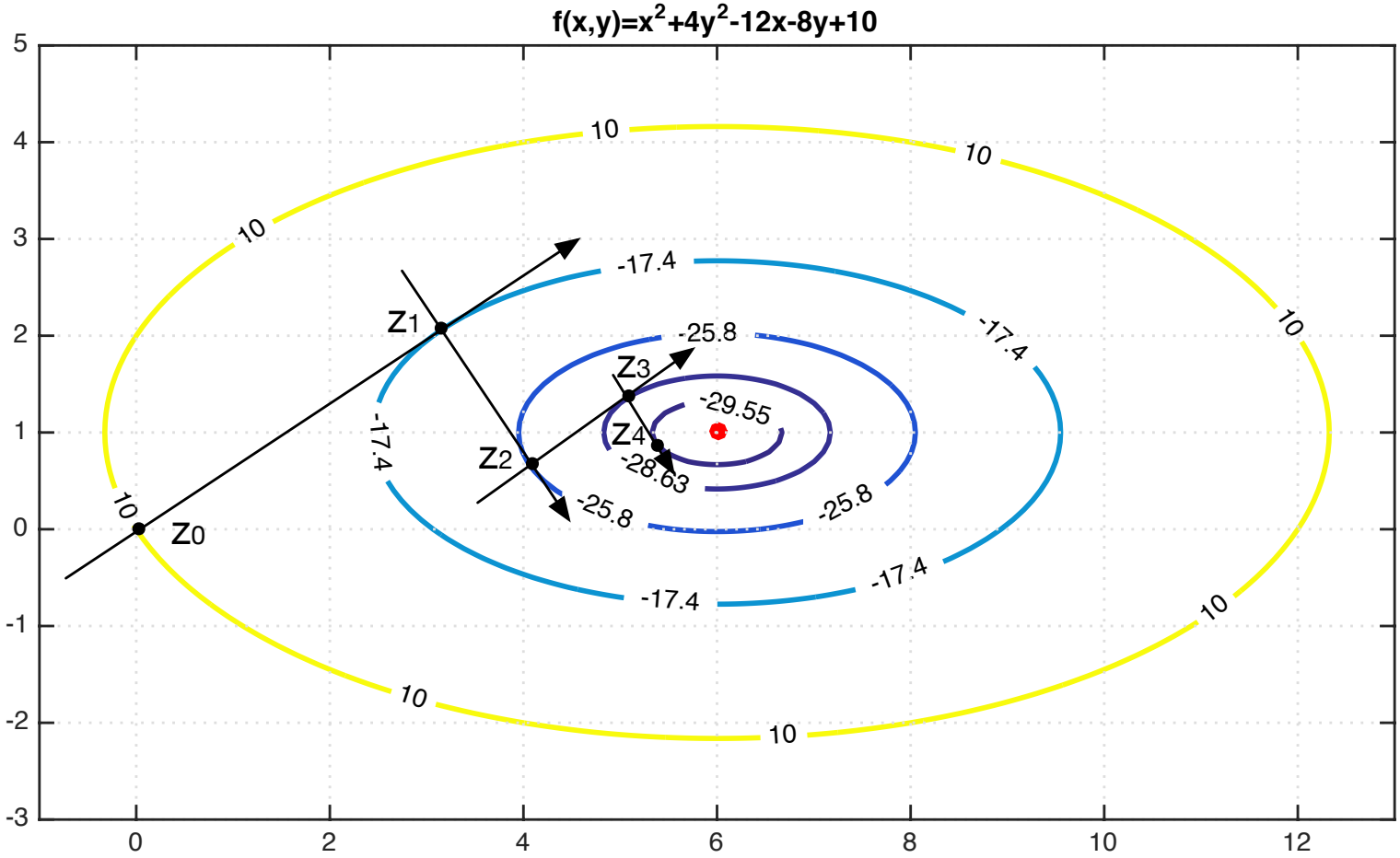
Observar que $\nabla^T f(x_n) \cdot \vec{d_n} = -||\nabla f(x_n)||^2 < 0$, porque si $\nabla f(x_n) = \vec{0}$ ya estaría en un candidato a óptimo local.

Tomemos $f(x, y) = x^2 + 4y^2 - 12x - 8y + 10$, $\nabla f = \begin{bmatrix} 2x - 12 \\ 8y - 8 \end{bmatrix}$, $\vec{d} = -\nabla f$ (steepest descent) y usamos la búsqueda lineal exacta.

Repitiendo, construimos la secuencia: $z_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $z_1 = \begin{bmatrix} 3.12 \\ 2.08 \end{bmatrix}$, $z_2 = \begin{bmatrix} 4.06 \\ 0.67 \end{bmatrix}$, $z_3 = \begin{bmatrix} 5.06 \\ 1.35 \end{bmatrix}$, $z_4 = \begin{bmatrix} 5.37 \\ 0.89 \end{bmatrix}$, cuyos valores son: 10, -17.04, -25.80, -28.63 y -29.55 respectivamente.

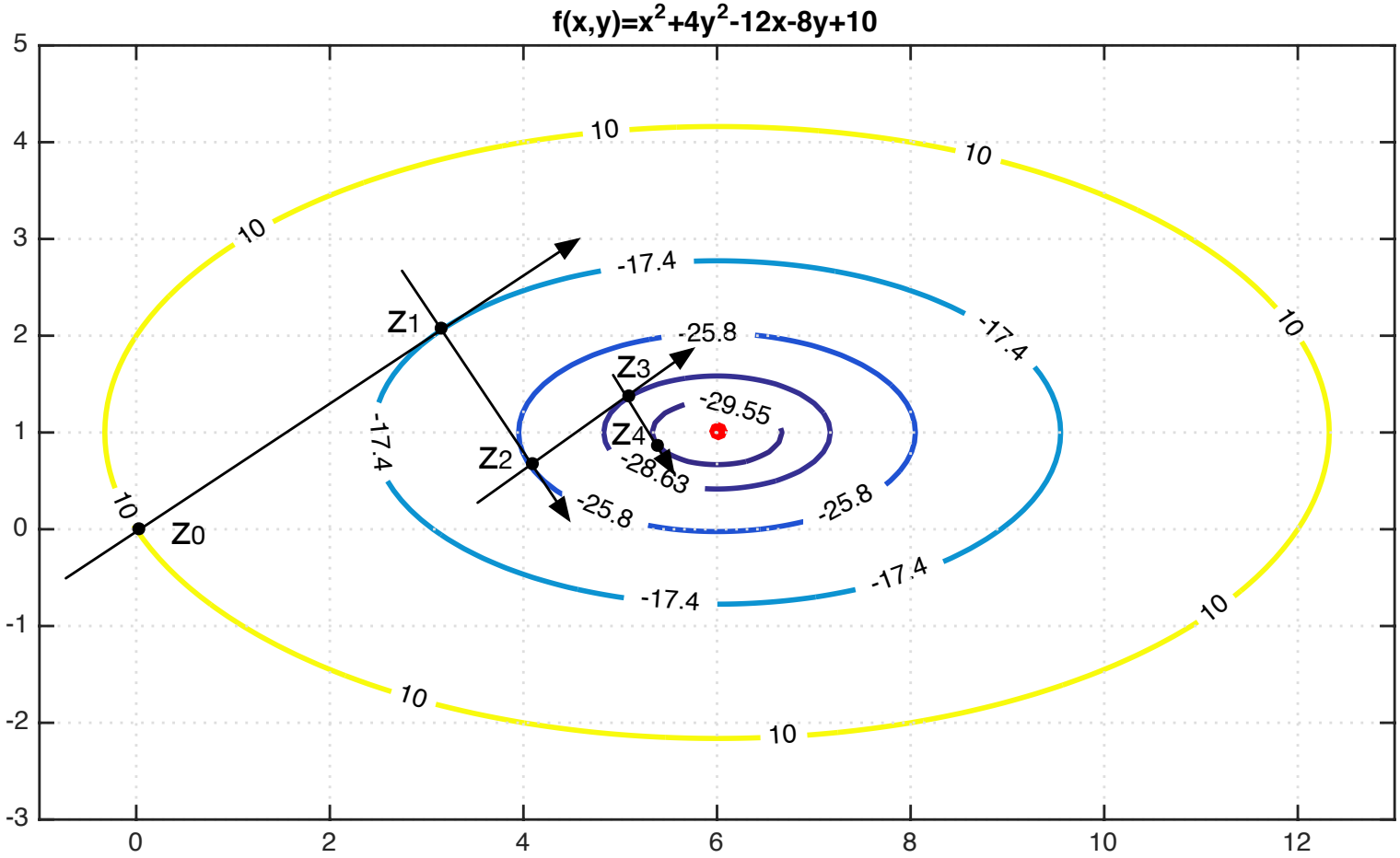
Recordar que el óptimo era -30 y se alcanza en $(6, 1)$.

# Steepest Descent




Observar que las direcciones de descenso son ortogonales a las curvas de nivel de partida, y tangentes a las de llegada.

# Steepest Descent



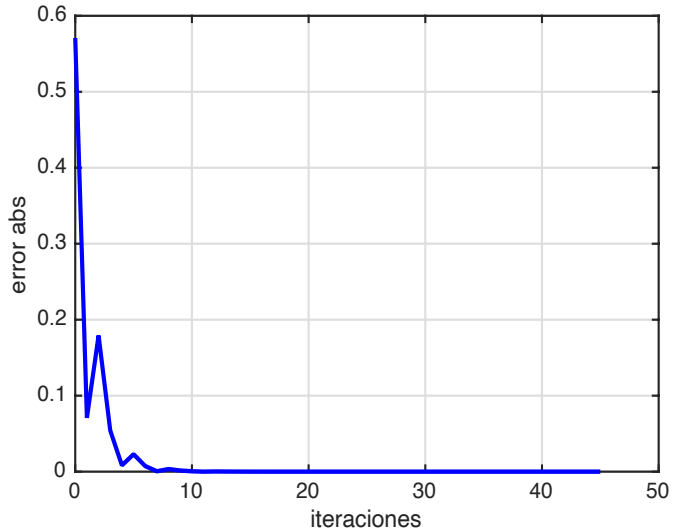

Por lo que el tour seguido hacia el óptimo es un zig-zag a 90º.

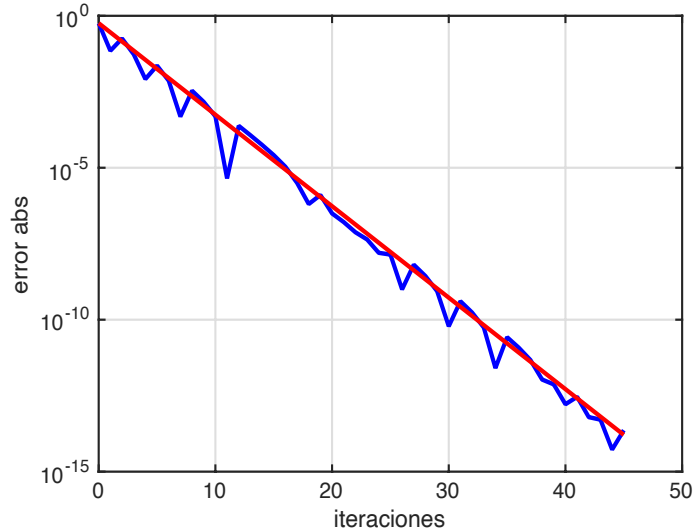
# Velocidad y Orden de Convergencia

## Definición

*Supongamos una secuencia* $\{x_n\}$ *de vectores que converge a* $\bar{x}$*, y una norma dada* $||,||$*. Decimos que la secuencia converge a* $\bar{x}$ *con orden* $p$ *y velocidad* $\mu$ *cuando* $\frac{||x_{n+1}-\bar{x}||}{||x_n-\bar{x}||^p} \to \mu$.

Ejemplo: Método de bipartición con $f(x) = \cos(x)e^{x/2}$ (raíz es $\frac{\pi}{2}$).




Es lineal ($p$=1) y velocidad $\frac{1}{2}$ (divide a la mitad en cada paso).

# Búsquedas Lineales

Hasta el momento hemos sostenido dos hipótesis:

i) Conocemos la función $f$ y también su gradiente $\nabla f$;

ii) Tenemos alguna expresión para hacer la búsqueda lineal en forma exacta.

Es normal disponer del gradiente (siempre se puede calcular analíticamente), pero incluso en caso de no tenerlo, hay técnicas para estimarlo numéricamente con precisión.

Con lo que en general no se cuenta es con una forma analítica para resolver la búsqueda lineal.

La función $h_n(\tau) = f(x_n + \tau\, d_n)$ es en definitiva un problema de optimización en $\mathbb{R}$ (tiene menos dimensiones), y puede usarse alguno de los algoritmos vistos hasta ahora o los siguientes.

Observar que $h_n'(\tau) = \nabla^T f(x_n + \tau\, d_n) \cdot d_n$ (producto escalar), y en el $\tau$ óptimo $d_n$ y $\nabla^T f(x_n + \tau\, d_n)$ son ortogonales.

# Búsquedas Lineales

En general, la búsqueda lineal exacta no es posible para funciones complejas, dado que no existen métodos para hallar sus raíces.

Aun en los casos que sí existen, el esfuerzo de cómputo de una búsqueda exacta no se justifica, y se usa alguna regla eficiente, incluso sub-óptima.

Pensar que en última instancia, estamos eligiendo el $x_0$ arbitrariamente, así que es como reiniciar desde un punto mejor.

Entre las búsquedas lineales asintóticamente óptimas están

- Bipartición (precisa, robusta, eficiencia media)
- Newton (inestable, performance muy alta)

Aun siendo sub-óptima, la Regla de Wolfe es un heurística popular, porque es robusta y tiene un balance costo/precisión.

# Búsquedas Lineales (Regla de Wolfe)

La Regla de Wolfe establece que cualquier $\tau$ que cumpla las siguientes dos condiciones es un buen paso de descenso.

I) $h(\tau) \leq h(0) + m \cdot h'(0) \cdot \tau$ (estar a la izquierda de $\tau_R$)

II) $h'(\tau) \geq m' \cdot h'(0)$ (estar a la derecha de $\tau_L$)

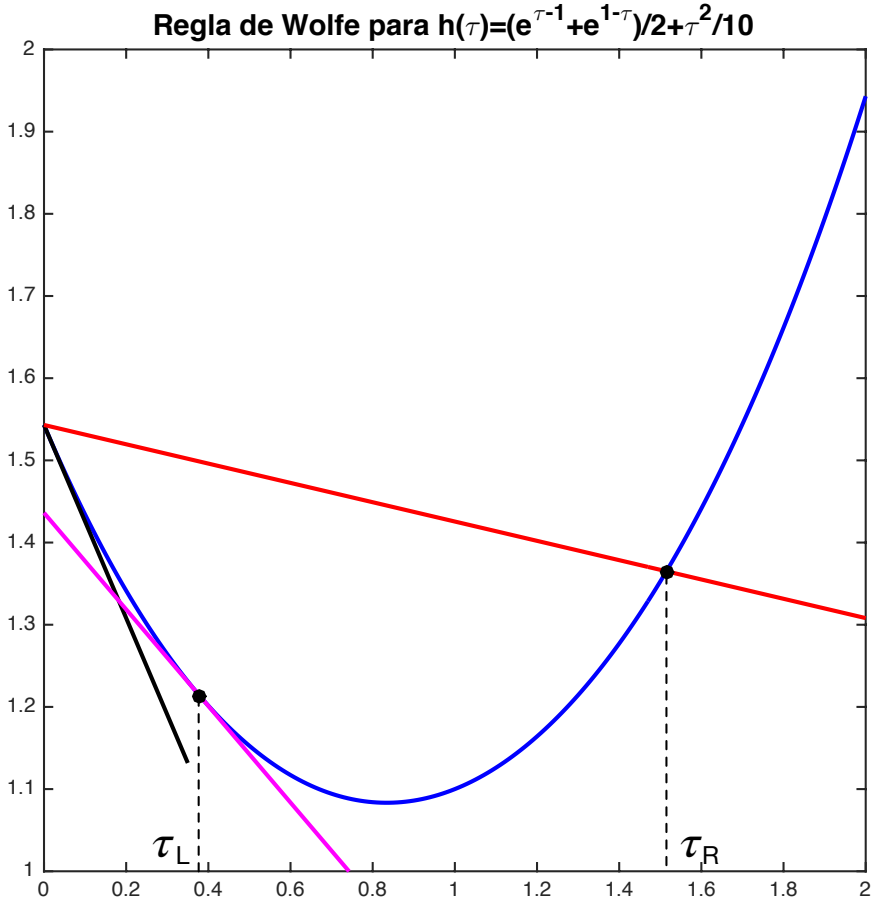
Los valores deben cumplir $0 < m < m' < 1$.

Recordar que $h'(0) < 0$ porque la dirección es de descenso, así que la regla está bien definida.

Tomemos con referencia $h(\tau) = \dfrac{e^{\tau-1} + e^{1-\tau}}{2} + \dfrac{\tau^2}{10}$, y vamos a usar $m = 0.1$ y $m' = 0.5$ como valores típicos.

# Búsquedas Lineales (Regla de Wolfe)

I) $h(\tau) \leq h(0) + m \cdot h'(0) \cdot \tau$ (estar a la izquierda de $\tau_R$)
II) $h'(\tau) \geq m' \cdot h'(0)$ (estar a la derecha de $\tau_L$)



# Búsquedas Lineales (Regla de Wolfe)

En este caso, $\tau_L = 0.378$ y $\tau_R = 1.5148$, pero no lo sabemos de antemano, del mismo modo que no sabemos que el paso óptimo era $\bar{\tau} = 0.834$. Se infiere la posición por los valores de $h$ y $h'$.

Un algoritmo para implementar la regla es:

1. Tomar $t_g$=0 y $t_d$=1. Mientras $h(t_d) < h(0) + m \cdot h'(0) \cdot t_d$, hacer $t_d$=2$t_d$ (también se usa $t_d$=10$t_d$).
2. Tomar $t = (t_g + t_d)/2$ (punto medio a $t_g$ y $t_d$).
3. Si $t$ cumple las reglas I) y II) ése es el paso (FIN).
4. Si $t$ no cumple I) (muy grande), hacemos $t_d = t$.
5. Si $t$ no cumple II) (muy chico), hacemos $t_g = t$.
6. Volver al punto 2.

En este ejemplo, habríamos llegado al paso 2 con $[t_g, t_d] = [0, 2]$, luego $t = 1$ cumple las dos reglas y ése habría sido el paso elegido.

# Gradiente Conjugado

En el problema anterior se busca el mínimo de un polinomio de segundo grado convexo, que es equivalente a resolver $\nabla f(x) = \vec{0}$, que siempre es lineal en este caso. Asumimos en el análisis que $f$ es definida positiva, para tener un sistema compatible determinado que garantice la unicidad del óptimo.

También podemos expresar $f(x)$ como $\frac{1}{2}x^T Ax - b^T x + c$, siendo la matriz $A$ la Hessiana de $f$, que es simétrica. Hallar el óptimo es resolver $Ax = b$, para $A$ simétrica y definida positiva.

El Gradiente Conjugado (M. Hestenes, E. Stiefel) es un método iterativo **"finito"** para encontrar soluciones de sistemas de ecuaciones lineales en esas condiciones.

Usa que los gradientes en el descenso son ortogonales y elige la nueva dirección efectiva como una combinación lineal en esa base, que captura mejor la "tendencia" de descenso.

# Gradiente Conjugado (ej1)

Considero nuevamente $f(x, y) = x^2 + 4y^2 - 12x - 8y + 10$: cuya solución óptima es $\bar{x} = (6, 1)$ y $\nabla f = \begin{bmatrix} 2x - 12 \\ 8y - 8 \end{bmatrix}$.

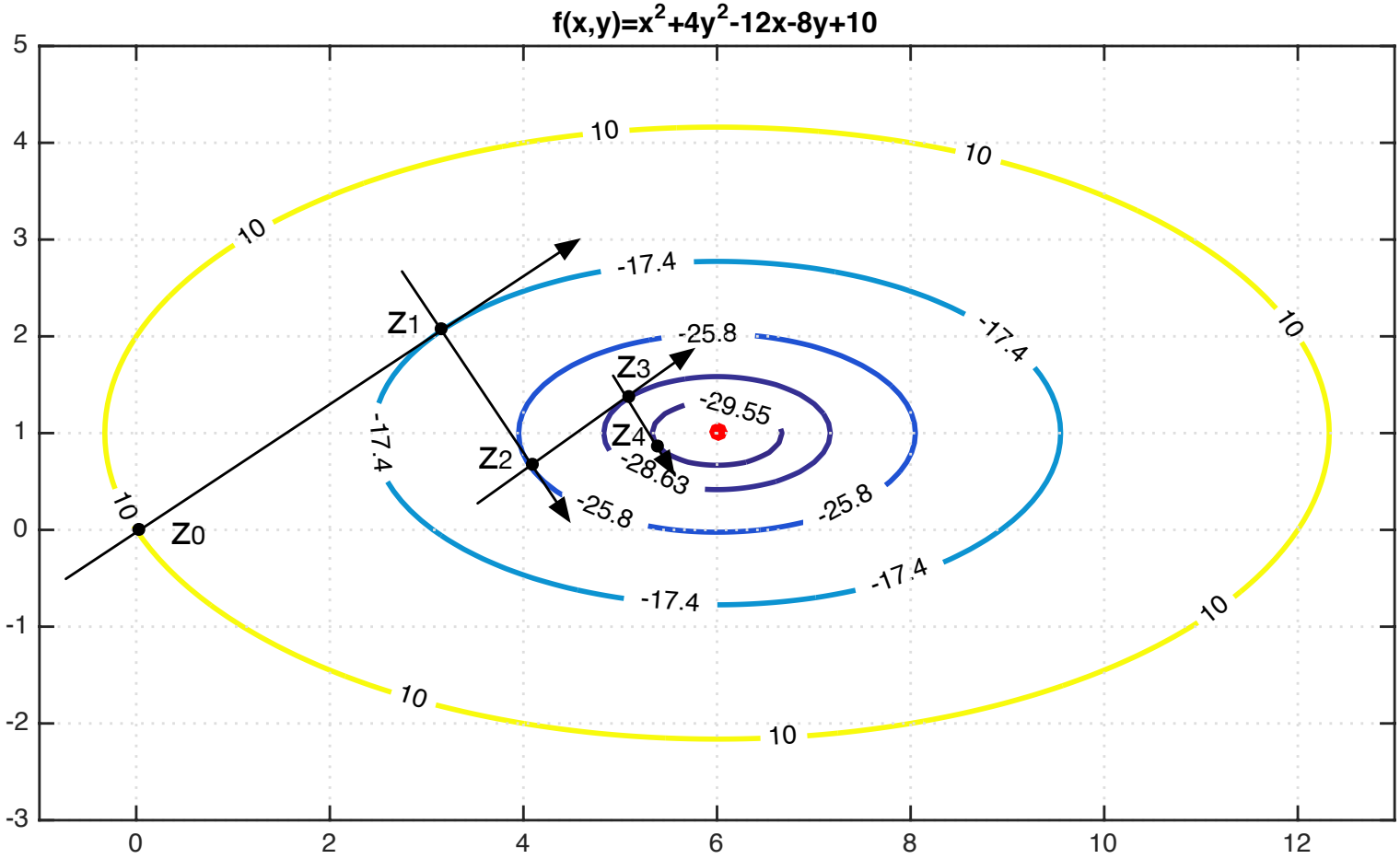
Podríamos tomar $z_0 = [0, 0]^T$ (nuevamente), y $d_0 = r_0 = -\nabla f(z_0)$ como referencia inicial (máximo descenso), para realizar una búsqueda lineal y hallar $z_1 = [3.12, 2.08]^T$ (también nuevamente).

Como segunda referencia de descenso tomamos $r_1 = -\nabla f(z_1)$, pero la usamos ahora para calcular $\beta = (r_1^T \cdot r_1)/(r_0^T \cdot r_0)$, y tomamos la nueva dirección como $d_1 = r_1 + \beta d_0$, que en este caso vale $[11.98078, -4.4928]^T$.

Con una nueva búsqueda lineal exacta en la dirección $d_1$ llegamos a $z_2 = [6, 1]^T$, que es exactamente $\bar{x}$... **el óptimo!!**

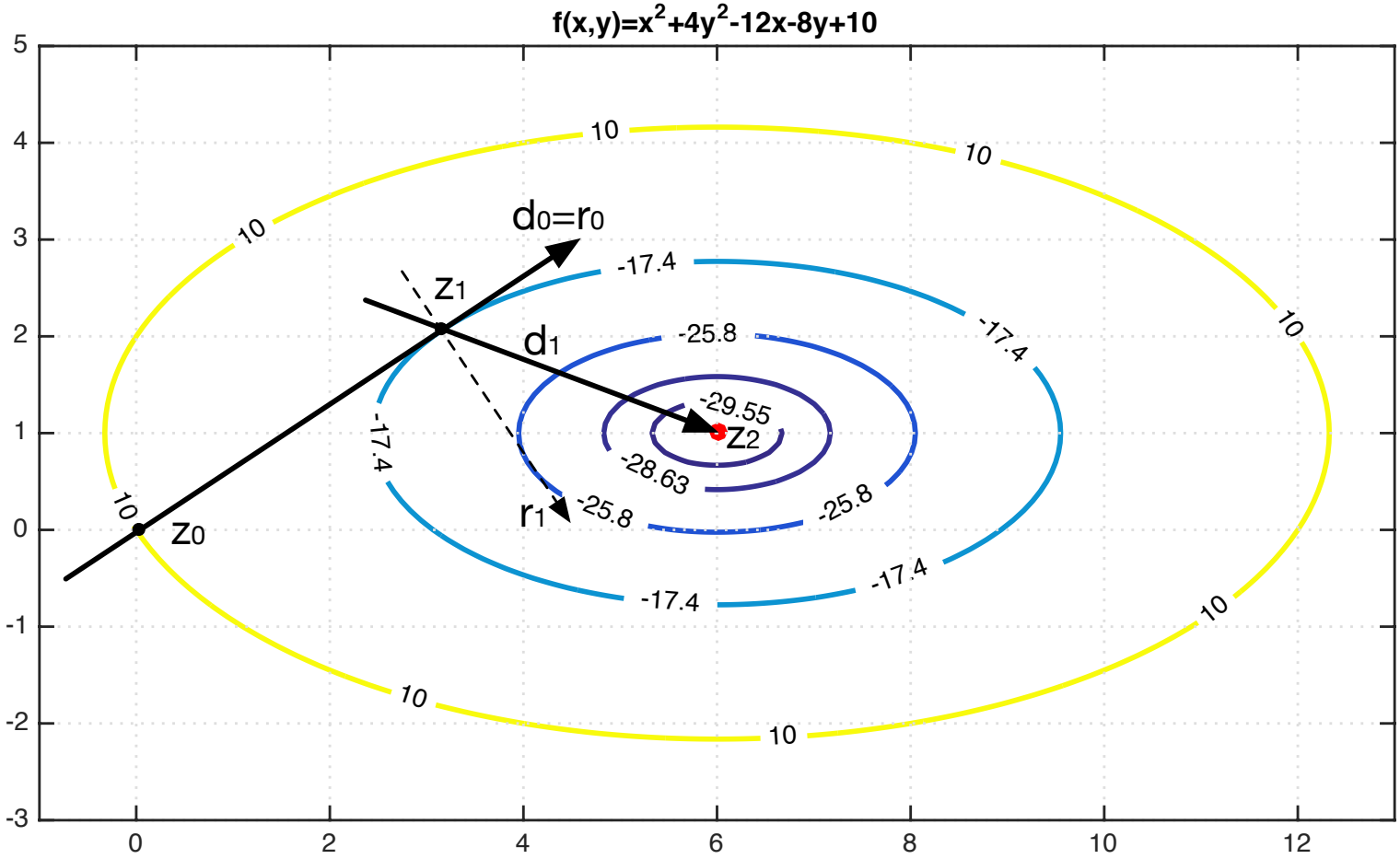
# Gradiente Conjugado (ej1)

Considero nuevamente $f(x, y) = x^2 + 4y^2 - 12x - 8y + 10$:



# Gradiente Conjugado (ej1)

Considero nuevamente $f(x, y) = x^2 + 4y^2 - 12x - 8y + 10$:



# Gradiente Conjugado (justificación en $\mathbb{R}^2$)

Como hipótesis se tiene:

- $f(x) = \frac{1}{2}x^T Ax - b^T x + c$ y $r(x) = -\nabla f(x) = b - Ax$, siendo $A$ simétrica y definida positiva.
- Los $\{r_k\}$ son ortogonales en el producto escalar $\langle u, v\rangle = u^T \cdot v$.
- La matriz $A$ induce un producto interno $\langle u, v\rangle_A = u^T \cdot A \cdot v$.
  1. $\langle u, v\rangle_A = u^T Av = v^T A^T u = v^T Au = \langle v, u\rangle_A$ ya que $A$ es simétrica.
  2. $\alpha\langle u, v\rangle_A = \alpha(u^T Av) = (\alpha u)^T Av = \langle \alpha u, v\rangle_A$.
  3. $\langle u + v, w\rangle_A = (u + v)^T Aw = u^T Aw + v^T Aw = \langle u, w\rangle_A + \langle v, w\rangle_A$.
  4. $\langle u, u\rangle_A = u^T Au \geq 0$ y $\langle u, u\rangle_A = 0$ si y sólo si $u = 0$, porque la matriz $A$ se definida positiva.
- Luego, usando por ejemplo Gram-Schmidt, se concluye que cualquier base se puede transformar en una ortogonal para el producto $\langle , \rangle_A$, que se denomina de vectores conjugados.

# Gradiente Conjugado (justificación en $\mathbb{R}^2$)

Asumo que la base es $D = \{d_0, d_1\}$, donde $d_0 = r_0 = r(0) = b$.

Si $x$ resuelve $Ax = b$ es el óptimo ($\nabla f(x) = 0$).

Existen $\alpha_0$, $\alpha_1$ tales que: $x = \alpha_0 \cdot d_0 + \alpha_1 \cdot d_1$ ($D$ es una base), y $Ax{=}\alpha_0 Ad_0{+}\alpha_1 Ad_1{=}b$, de donde: $\langle d_0, x\rangle_A{=}\alpha_0\langle d_0, d_0\rangle_A = d_0^T b$. Análogamente se llega a que $\langle d_1, x\rangle_A{=}\alpha_1\langle d_1, d_1\rangle_A = d_1^T b$.

Luego $\alpha_0 = \frac{d_0^T b}{d_0^T A d_0}{=}\frac{r_0^T r_0}{r_0^T A r_0}$. Como $h_0(\tau){=}\frac{1}{2}(r_0^T A r_0)\tau^2{-}(b^T r_0)\tau{+}c$, $h_0'(\bar{\tau}) = 0 \Leftrightarrow (r_0^T A r_0)\bar{\tau} = r_0^T r_0$, y se cumple que $\alpha_0 = \bar{\tau}$ (el primer paso óptimo es el coeficiente de $d_0$).

Se elige $\beta_1$ de forma que $\langle d_0, d_1\rangle_A{=}\langle r_0, r_1 + \beta_1 r_0\rangle_A{=}0$, de donde $\beta_1{=}-\frac{r_0^T A r_1}{r_0^T A r_0}{=}-\frac{r_1^T A r_0}{r_0^T A r_0}$. Como $r_1{=}r(r_0\bar{\tau}){=}b - \bar{\tau}Ar_0{=}r_0 - \bar{\tau}Ar_0$, se cumple $Ar_0 = \frac{r_0{-}r_1}{\bar{\tau}}$, y $\beta_1 = -\frac{r_1^T(r_0{-}r_1)}{r_0^T(r_0{-}r_1)} = \frac{r_1^T r_1}{r_0^T r_0}$.

Como $\alpha_0$ es conocido, el siguiente paso óptimo debe ser $\alpha_1$ ($D$ es una base y el paso me lleva al óptimo).

# Gradiente Conjugado (ej2)

Considero $f(x,y) = 2x^2 + 5y^2 + 2xy - 12x - 8y + 10$. La Hessiana de $f$ es $H[f] = \begin{bmatrix} 4 & 2 \\ 2 & 10 \end{bmatrix}$, que tiene valores propios $7 \pm \sqrt{27/2} > 0$.

El óptimo de $f$ puede calcularse con $\nabla f = \begin{bmatrix} 4x + 2y - 12 \\ 2x + 10y - 8 \end{bmatrix} = \vec{0}$,

$\begin{bmatrix} 4 & 2 \\ 2 & 10 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 12 \\ 8 \end{bmatrix} = b$, cuya solución es $\bar{x} = (26/9, 2/9)$.

Podríamos haber tomado $z_0 = [0,0]^T$, y $d_0 = r_0 = b - H \cdot z_0$ como referencia inicial (máximo descenso), realizar una búsqueda lineal para hallar $z_1 = [1.56, 1.04]^T$, tomar $r_1 = b - H \cdot z_1$ (referencia), pero usarlo para calcular una nueva dirección $d_1 = r_1 + \beta_1 d_0$, donde $\beta_k = (r_k^T \cdot r_k)/(r_{k-1}^T \cdot r_{k-1})$. La búsqueda lineal en esa dirección $d_1 = [6.2192, -3.8272]^T$ termina en la solución $\bar{x}$.

# Algoritmo de Descenso por Gradiente Conjugado

Buscamos el mínimo de una función $f : \mathbb{R}^n \to \mathbb{R}$ diferenciable, e idealmente (aunque no necesariamente) convexa.

1. Tomamos $k = 0$, un punto inicial $x_0$ y $d_0 = r_0 = -\nabla f(x_0)$.
2. $k$=$k$+1 (inicio del loop).
3. calculo $x_k$ haciendo búsqueda lineal en dirección $d_{k-1}$.
4. $r_k = -\nabla f(x_k)$.
5. $\beta_k = (r_k^T \cdot r_k)/(r_{k-1}^T \cdot r_{k-1})$.
6. $d_k = r_k + \beta_k \cdot d_{k-1}$;
7. Si no se cumple la *condición de salida*, volver al punto 2.

El cálculo $\beta_k = (r_k^T \cdot r_k)/(r_{k-1}^T \cdot r_{k-1})$ se conoce como fórmula de Polak-Ribière, y surge del desarrollo analítico del método.

# Algoritmo de Descenso por Gradiente Conjugado

Buscamos el mínimo de una función $f : \mathbb{R}^n \to \mathbb{R}$ diferenciable, e idealmente (aunque no necesariamente) convexa.

1. Tomamos $k = 0$, un punto inicial $x_0$ y $d_0 = r_0 = -\nabla f(x_0)$.
2. $k$=$k$+1 (inicio del loop).
3. calculo $x_k$ haciendo búsqueda lineal en dirección $d_{k-1}$.
4. $r_k = -\nabla f(x_k)$.
5. $\beta_k = (r_k^T \cdot r_k)/(r_{k-1}^T \cdot r_{k-1})$.
6. $d_k = r_k + \beta_k \cdot d_{k-1}$;
7. Si no se cumple la *condición de salida*, volver al punto 2.

Es equivalente a $\beta_k = (r_k^T \cdot (r_k - r_{k-1}))/(r_{k-1}^T \cdot r_{k-1})$ (Fletcher-Reeves), porque $r_k$ y $r_{k-1}$ son ortogonales, pero esta formulación concreta es numéricamente más estable.

# Algoritmo de Descenso por Gradiente Conjugado

Como ya vimos, si $f$ es cuadrática y definida positiva (es convexa en este caso), el algoritmo converge en $n$ pasos.

Como una función doblemente diferenciable puede aproximarse por:

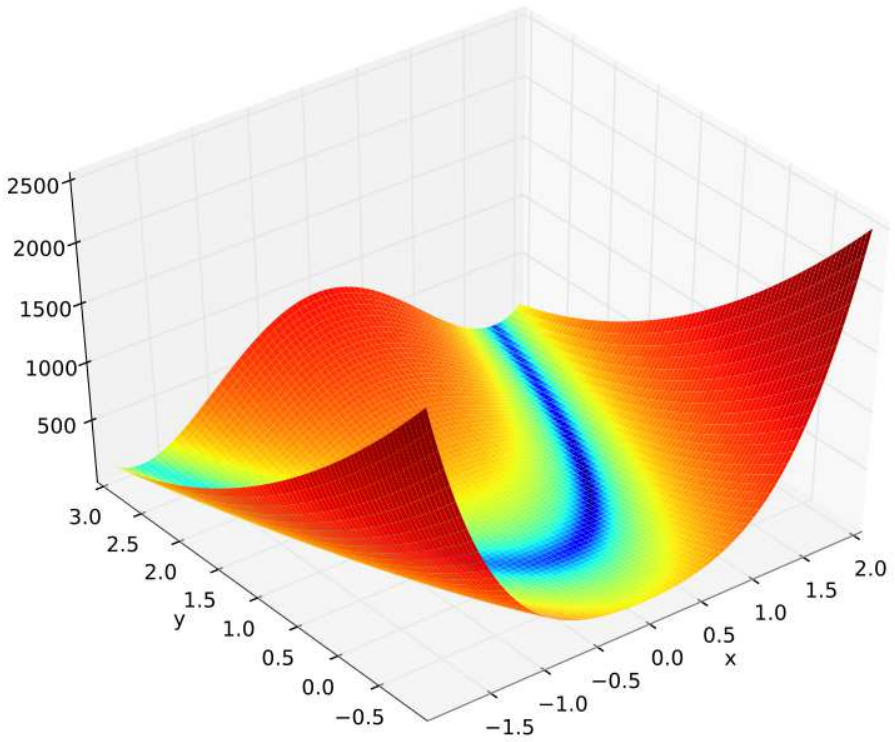
$$f(y) \approxeq f(x) + \nabla^T f(x) \cdot (y - x) + (y - x)^T \frac{H[f(x)]}{2}(y - x),$$

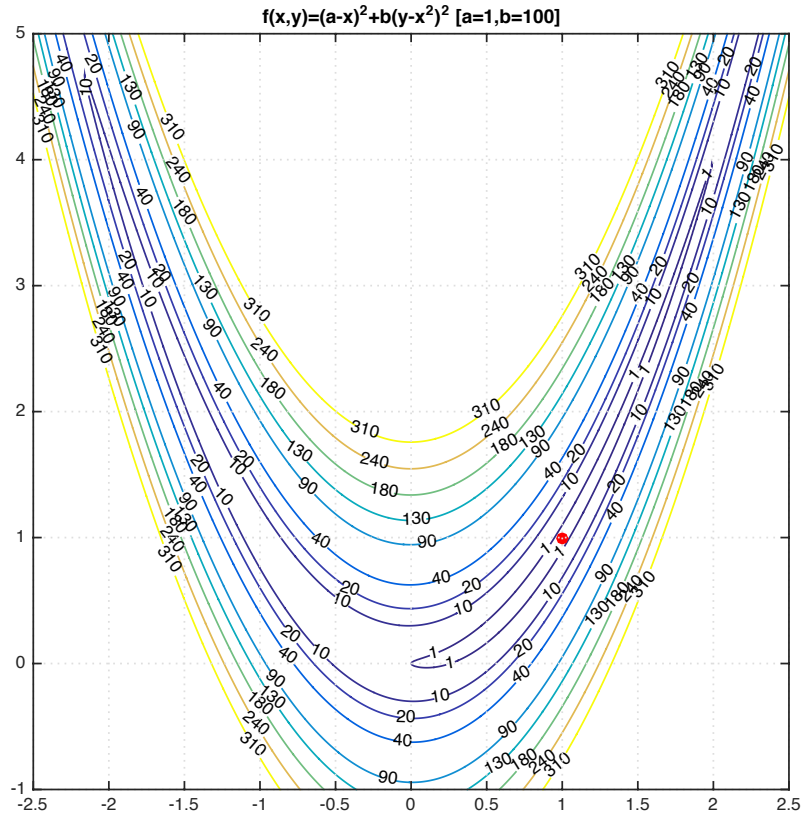
el método funciona localmente bien en una secuencia de descenso.

Converge más rápido y es mucho más robusto que steepest descent para sortear problemas como las curvas de nivel en banana.

La función de Rosenbrock $f(x, y) = (a - x)^2 + b(y - x^2)^2$ tiene óptimo global en $(a, a^2)$, pero no es convexa, y de hecho, es un desafío para los métodos de descenso greedy.

# Función de Rosenbrock $[a = 1, b = 100]$




El descenso por gradiente conjugado tiene una buena performance en este problema, mientras que steepest descent requiere muchas iteraciones.

# Métodos de Mayor Orden (Newton)

Buscar óptimos y resolver ecuaciones son problemas parecidos.

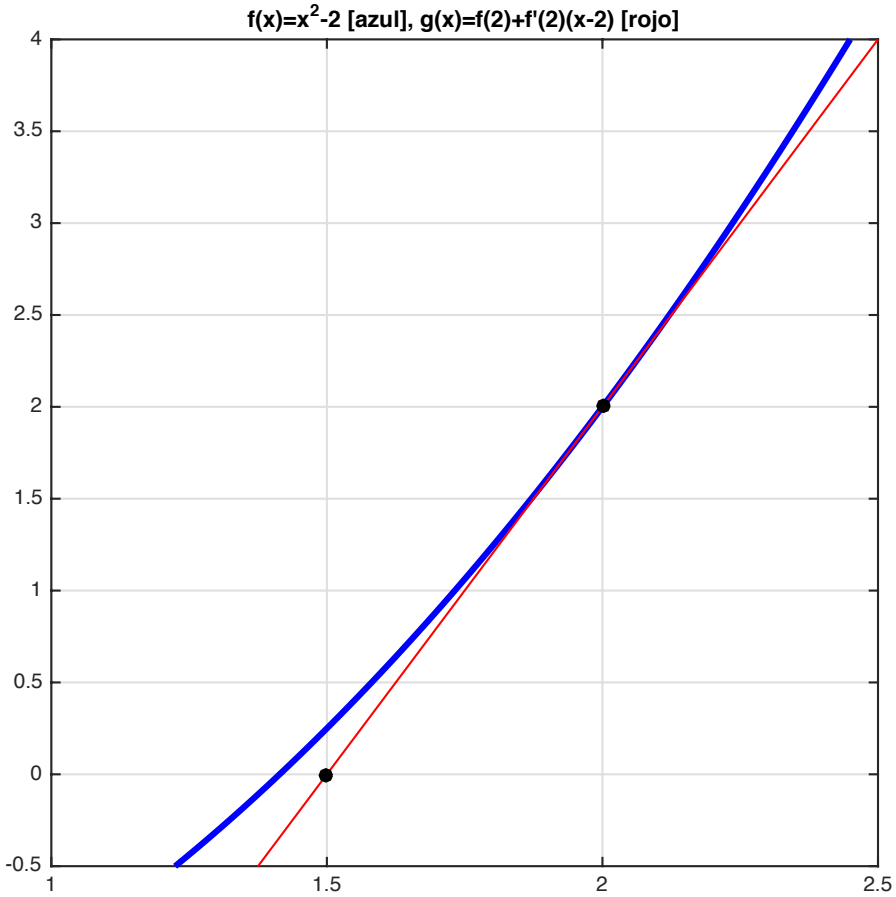
Supongamos que quiero calcular $\bar{x} = \sqrt{2}$ con mucha precisión y no tengo una calculadora a mano.

Podría hallar la raíz positiva de la función $f(x)=x^2-2$ mediante un método iterativo que busque la raíz $x_{n+1}$ de la tangente a $f$ en $x_n$.

# Métodos de Mayor Orden (Newton)

Buscar óptimos y resolver ecuaciones son problemas parecidos.

Supongamos que quiero calcular $\bar{x} = \sqrt{2}$ con mucha precisión y no tengo una calculadora a mano.



# Métodos de Mayor Orden (Newton)

Buscar óptimos y resolver ecuaciones son problemas parecidos.

Supongamos que quiero calcular $\bar{x} = \sqrt{2}$ con mucha precisión y no tengo una calculadora a mano.

Podría hallar la raíz positiva de la función $f(x)=x^2-2$ mediante un método iterativo que busque la raíz $x_{n+1}$ de la tangente a $f$ en $x_n$.

La tangente es $g(y) = f(x) + f'(x)(y - x)$, así que la iteración sería: $g(x_{n+1}) = f(x_n) + f'(x_n)(x_{n+1} - x_n) = 0$.

Despejando, llegamos a $x_{n+1} = x_n - \frac{x_n^2-2}{2x_n}$ y tomemos $x_0 = 2$.

La secuencia $x_n$ queda: $x_0 = 2$, $x_1 = \frac{3}{2}$, $x_2 = \frac{17}{12}$, $x_3 = \frac{577}{408}$ que vale 1.414215686274510, con lo cual, en tres iteraciones alcanzamos una precisión de $1.5 \cdot 10^{-6}$ ($\sqrt{2} \approx 1.414213562373095$), y habríamos llegado a 12 dígitos con otra iteración más.

# Descenso por Método de Newton

Para optimizar con este método, usamos la iteración que surge de la ecuación $\nabla f(x - x_n) = \vec{0}$. Si $f$ es doblemente diferenciable: $f(x) \approxeq f(x_n) + \nabla^T f(x_n) \cdot (x - x_n) + (x - x_n)^T \frac{H[f(x_n)]}{2}(x - x_n)$, de donde buscamos: $\nabla f(x_n) + H[f(x_n)](x_{n+1} - x_n) = \vec{0}$. La recursión es entonces: $x_{n+1} = x_n - H^{-1}[f(x_n)] \cdot \nabla f(x_n)$.

Si $f(x, y) = 2x^2 + 5y^2 + 2xy - 12x - 8y + 10$, $\nabla f = \begin{bmatrix} 4x + 2y - 12 \\ 2x + 10y - 8 \end{bmatrix}$, $H[f] = \begin{bmatrix} 4 & 2 \\ 2 & 10 \end{bmatrix}$ y $H^{-1}[f] = \frac{1}{18}\begin{bmatrix} 5 & -1 \\ -1 & 4 \end{bmatrix}$.

La iteración es: $z_{n+1} = z_n - \begin{bmatrix} x - \frac{26}{9} \\ y - \frac{2}{9} \end{bmatrix}$, con lo cual, tomo $z_0 = [0, 0]$ y llegamos al óptimo en un paso $z_1 = [26/9, 2/9]$.

# Descenso por Método de Newton

El método de Newton está entre los más rápidos en lo que a iteraciones refiere si $x_0$ está próximo al óptimo, pero suele tener problemas de convergencia global.

No es tan robusto como gradiente conjugado, así que podríamos usar: GC para acercarse y MN para conseguir la solución final.

Además, el método de newton requiere muchas operaciones en cada paso (hay que invertir una matriz), y necesita no solamente el gradiente, sino la Hessiana de la función.

Veremos a continuación que lo usado en la práctica son los métodos Cuasi-Newton, que generan una sucesión $B_n$ de matrices que aproximan a las $H^{-1}[f(x_n)]$.

# Métodos de Mayor Orden (cuasi-Newton)

No tengo una expresión para la Hessiana, pero sí para el gradiente, y cuando los pasos en la sucesión $\delta_n = x_{n+1} - x_n$ son cortos, puedo usar la expresión $\gamma_n = \nabla f(x_{n+1}) - \nabla f(x_n) \approxeq H[f(x_n)] \cdot \delta_n$.

Como lo que busco es el paso, planteo $\delta_n = B_n \cdot \gamma_n$, y debo construir una sucesión $\{B_n\}$ de matrices simétricas definidas positivas, que convergen a $H^{-1}[f(x_n)]$ si el método funciona.

Durante las iteraciones, procedemos en forma similar al método de Newton: $x_{n+1} = x_n - H^{-1}[f(x_n)] \cdot \nabla f(x_n)$, aunque usando $B_n$ en lugar de $H^{-1}[f(x_n)]$. La dirección elegida es siempre $-B_n \cdot \nabla f(x_n)$.

Así, la matriz $B_{n+1}$ se calcula usando $B_n$, $\delta_n$ y $\gamma_n$, mediante una *fórmula de actualización*.

Todos los métodos cuasi-Newton tienen el mismo esquema, diferenciándose sólo por la fórmula de actualización elegida.

# Algoritmo de Descenso cuasi-Newton

Buscamos el mínimo de una función $f : \mathbb{R}^n \to \mathbb{R}$ doblemente diferenciable, y convexa (idealmente).

1. Tomamos $k$=0, un punto $x_0$, $g_0$=$\nabla f(x_0)$, $d_0$=$-g_0$ y $B_0 = Id$.
2. $k$=$k$+1 (inicio del loop, tengo $x_{k-1}$, $g_{k-1}$, $d_{k-1}$ y $B_{k-1}$).
3. Calculo $x_k$ haciendo búsqueda lineal en dirección $d_{k-1}$.
4. Calculo $\delta_k = x_k - x_{k-1}$, $g_k = \nabla f(x_k)$ y $\gamma_k = g_k - g_{k-1}$.
5. $B_k$ = alguna_fórmula_actualización ($B_{k-1}$, $\delta_k$, $\gamma_k$).
6. Tomo la dirección de descenso $d_k = -B_k \cdot g_k$.
7. Si no se cumple la *condición de salida*, volver al punto 2.

Método DFP: $B_k = B_{k-1} + \frac{\delta_k \delta_k^T}{\delta_k^T \gamma_k} - \frac{B_{k-1}\gamma_k \gamma_k^T B_{k-1}}{\gamma_k^T B_{k-1} \gamma_k}$, recibe su nombre por Davidon-Fletcher-Powell.

# Algoritmo de Descenso cuasi-Newton

Buscamos el mínimo de una función $f : \mathbb{R}^n \to \mathbb{R}$ doblemente diferenciable, y convexa (idealmente).

1. Tomamos $k$=0, un punto $x_0$, $g_0$=$\nabla f(x_0)$, $d_0$=$-g_0$ y $B_0 = Id$.
2. $k$=$k$+1 (inicio del loop, tengo $x_{k-1}$, $g_{k-1}$, $d_{k-1}$ y $B_{k-1}$).
3. Calculo $x_k$ haciendo búsqueda lineal en dirección $d_{k-1}$.
4. Calculo $\delta_k = x_k - x_{k-1}$, $g_k = \nabla f(x_k)$ y $\gamma_k = g_k - g_{k-1}$.
5. $B_k$ = alguna_fórmula_actualización ($B_{k-1}$, $\delta_k$, $\gamma_k$).
6. Tomo la dirección de descenso $d_k = -B_k \cdot g_k$.
7. Si no se cumple la *condición de salida*, volver al punto 2.

$$B_k = B_{k-1} + \left[1 + \frac{\gamma_k^T B_{k-1} \gamma_k}{\delta_k^T \gamma_k}\right] \frac{\delta_k \delta_k^T}{\delta_k^T \gamma_k} - \frac{\delta_k \gamma_k^T B_{k-1} + B_{k-1} \gamma_k \delta_k^T}{\delta_k^T \gamma_k},$$ es conocido como BFGS (por Broyden-Fletcher-Goldfarb-Shannon).

# Ejemplo BFGS

$f(x,y)=2x^2+5y^2+2xy-12x-8y+10$, $\nabla f = \begin{bmatrix} 4x+2y-12 \\ 2x+10y-8 \end{bmatrix}$,
$z_0 = [0,0]$, $d_0 = -g_0 = [12,8]^T$ (máximo descenso), y $B_0 = Id(2)$.

Con una búsqueda lineal llegamos a $z_1 = [1.56, 1.04]^T = \delta_1$,
$\gamma_1 = [8.32, 13.52]^T$ y $B_1 = \begin{bmatrix} 0.9688 & -0.4808 \\ -0.4808 & 0.3728 \end{bmatrix}$.

En el paso 2, $z_2 = [2.889, 0.222]^T$, $\delta_2 = [1.329, -0.818]^T$,
$\gamma_2 = [3.680, -5.521]^T$ y $B_2 = \frac{1}{18}\begin{bmatrix} 5 & -1 \\ -1 & 2 \end{bmatrix}$.

Ya en $z_3 = [26/9, 2/9]^T$ alcanzamos el óptimo.

# ¿Cuál es la eficiencia de un algoritmo?

Sean $f(x)$ y $g(x)$ dos funciones reales definidas sobre un mismo conjunto de entrada, cuyas salidas (positivas) respectivamente representan el tiempo que le tomaría a dos algoritmos $\mathcal{A}_f$ and $\mathcal{A}_g$, corriendo en el mismo hardware, para encontrar una solución asociada a la instancia determinada por los datos de entrada.

## Definición

*Decimos que $g$ es de orden más alto que $f$ (representado con $f(x) = O(g(x))$), sii, existen $x_0$ y $M$ positivos tal que para toda entrada $x$ suficientemente grande: $f(x) \leq Mg(x)$, $x > x_0$.*

Los coeficientes y términos de menor orden son usualmente excluidos. Si, por ejemplo, el tiempo de ejecución se estima en $5n^3 + 3n$, decimos que la complejidad asintótica es $O(n^3)$.

# ¿Qué representa en términos prácticos?

Suponga que el poder de cómputo (en operaciones por segundo) se incrementa en un factor de 1000 por década, y que debemos elegir entre cuatro algoritmos para una aplicación crítica, que requiere de respuesta en el día. Todos encuentran actualmente soluciones para entradas de tamaño 100. Además, sabemos que sus complejidades asintóticas respectivas son: $n^2$, $n^{10}$, $10^n$ y $n!$.

¿Cuáles son los tamaños de los problemas que podemos resolver en un día?

# ¿Qué representa en términos prácticos?

Suponga que el poder de cómputo (en operaciones por segundo) se incrementa en un factor de 1000 por década, y que debemos elegir entre cuatro algoritmos para una aplicación crítica, que requiere de respuesta en el día. Todos encuentran actualmente soluciones para entradas de tamaño 100. Además, sabemos que sus complejidades asintóticas respectivas son: $n^2$, $n^{10}$, $10^n$ y $n!$.

| Año | pod. cómp. | $O(n^2)$ | $O(n^{10})$ | $O(10^n)$ | $O(n!)$ |
|---|---|---|---|---|---|
| 2020 | $10^0$ | 100 | 100 | 100 | 100 |
| 2030 | $10^3$ | 3,162 | 200 | 103 | 102 |
| 2040 | $10^6$ | 100,000 | 398 | 106 | 104 |
| 2050 | $10^9$ | 3,162,278 | 794 | 109 | 105 |
| 2060 | $10^{12}$ | 100,000,000 | 1,585 | 112 | 106 |

Década tras década el tamaño de una instancia resoluble en un algoritmo polinomial se incrementa por un factor constante ($\sqrt{1000} \approxeq 31.62$ en el primer caso, $\sqrt[10]{1000} \approxeq 1.995$ en el segundo)

# ¿Qué representa en términos prácticos?

Suponga que el poder de cómputo (en operaciones por segundo) se incrementa en un factor de 1000 por década, y que debemos elegir entre cuatro algoritmos para una aplicación crítica, que requiere de respuesta en el día. Todos encuentran actualmente soluciones para entradas de tamaño 100. Además, sabemos que sus complejidades asintóticas respectivas son: $n^2$, $n^{10}$, $10^n$ y $n!$.

| Año | pod. cómp. | $O(n^2)$ | $O(n^{10})$ | $O(10^n)$ | $O(n!)$ |
|---|---|---|---|---|---|
| 2020 | $10^0$ | 100 | 100 | 100 | 100 |
| 2030 | $10^3$ | 3,162 | 200 | 103 | 102 |
| 2040 | $10^6$ | 100,000 | 398 | 106 | 104 |
| 2050 | $10^9$ | 3,162,278 | 794 | 109 | 105 |
| 2060 | $10^{12}$ | 100,000,000 | 1,585 | 112 | 106 |

Cuando el orden es exponencial, el tamaño de las instancias resolubles se vuelve "inmutable" a la potencia de cómputo a partir de instancias de cierto tamaño.

# Problemas computacionalmente tratables

## Definición

*Decimos que un algoritmo es eficiente o tratable, sii, su complejidad asintótica es un monomio. Existe $p \in \mathbb{N}$ tal que, el tiempo requerido para que el algoritmo encuentre una solución (i.e. $f(x)$) es de complejidad computaciones menor que $n^p$ ($f(x) = O(n^p)$).*

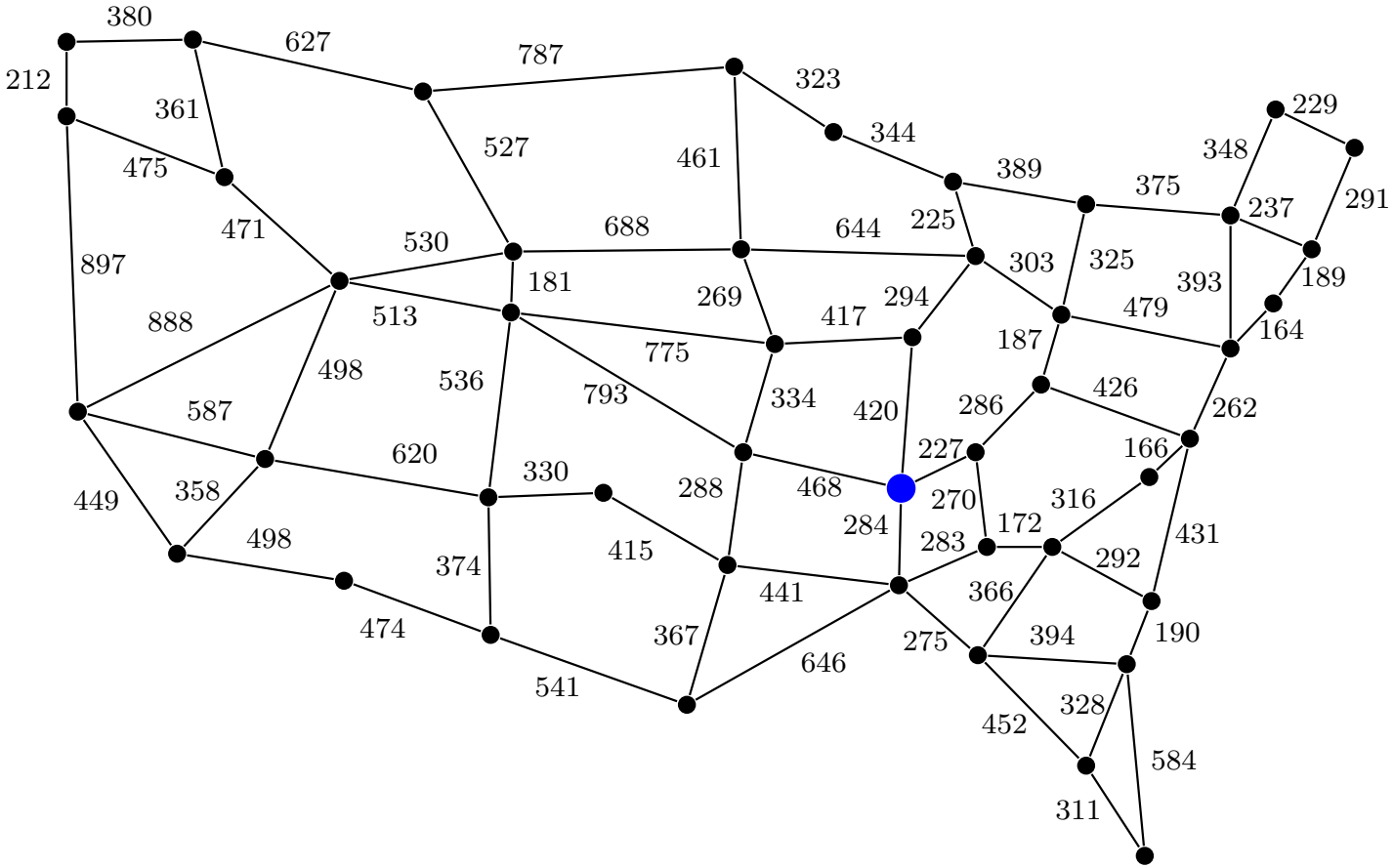
En otras palabras, para cualquier instancia $x$ de tamaño $n$, el algoritmo puede encontrar soluciones en *tiempo polinomial*.

Cualquier algoritmo polinomial para un problema ya es considerado una opción razonable. Aunque encontrado uno, se investiga intensamente cómo mejorarlo (bajarle el orden o la constante multiplicativa del término de mayor orden).

Un problema actual es que hay muchos "problemas importantes", para los cuales no se ha encontrado un algoritmo polinomial.

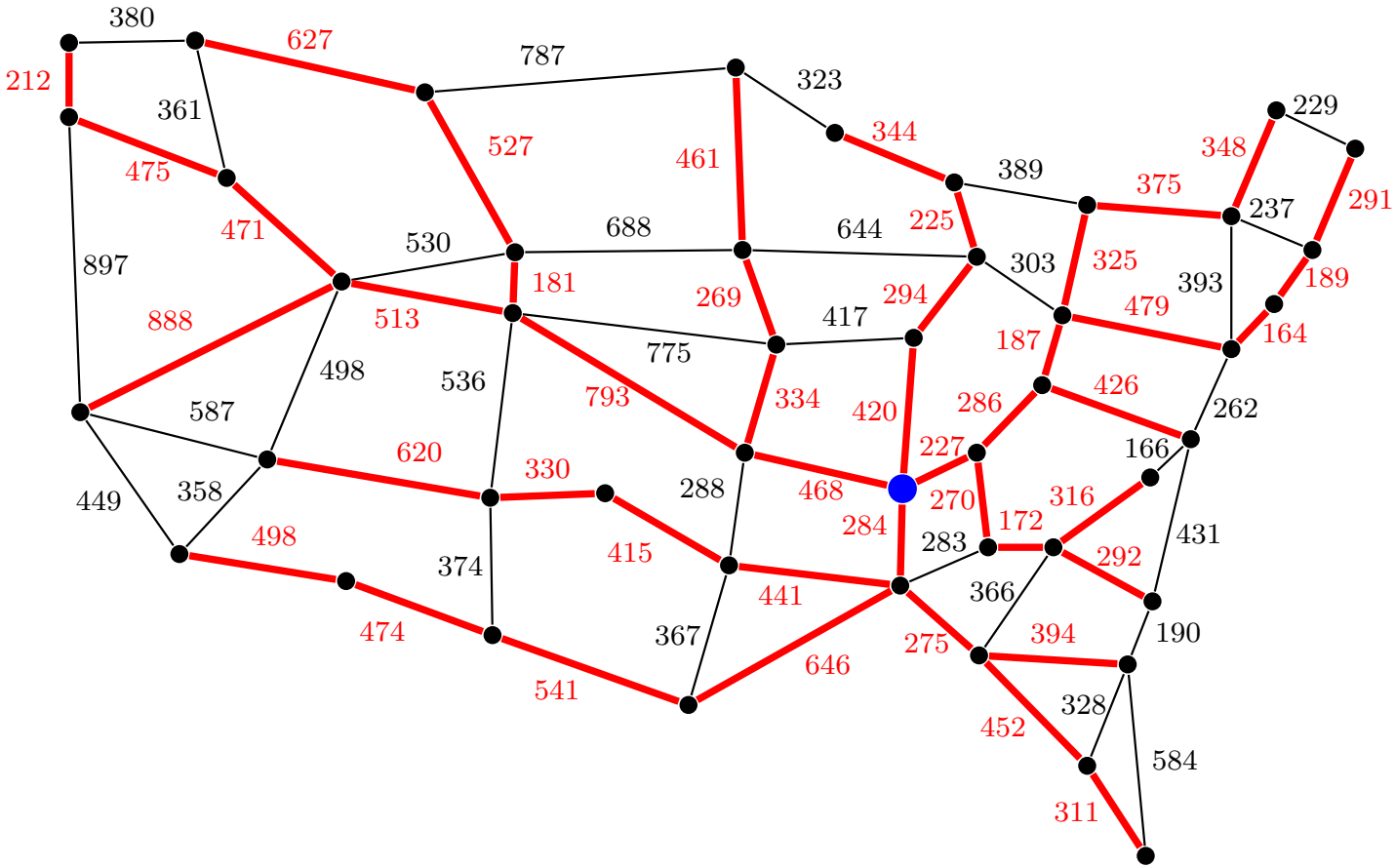
# Problemas tratables/fáciles: Camino más corto (SPP)

¿Cuáles son los caminos más cortos desde el nodo azul a los demás, dentro USNet?



# Problemas tratables/fáciles: Camino más corto (SPP)

¿Cuáles son los caminos más cortos desde el nodo azul a los demás, dentro USNet?




La solución es el árbol remarcado en la figura.

# Problemas tratables/fáciles: Camino más corto (SPP)

El *Shortest Path Problem* tiene múltiples aplicaciones en:

- Problemas de ruteo (telecomunicaciones, transporte/GPS)
- Problemas de ordenamiento (planificación de actividades)
- Análisis de Redes Sociales
- Mantenimiento y remplazo de componentes

Se conocen muchos algoritmos eficientes según el tipo de grafo:

- Grafos no-dirigidos
- Grafos con aristas de igual peso
- Grafos dirigidos acíclicos
- Grafos dirigidos con pesos no-negativos
- Grafos planares
- Grafos dirigidos sin ciclos negativos

# Problemas tratables/fáciles: Camino más corto (SPP)

Ejemplos de algoritmos más eficientes conocidos:

## UNDIRECTED GRAPHS

- Dijkstra (1959), $O(V^2)$ cuando pesos en $\mathbb{R}^+$
- Fredman-Tarjan (1984) $O(E + V \log(V))$ con Fibonnacci Heap cuando pesos en $\mathbb{R}^+$
- Thorup (1999) $O(E)$ cuando pesos en $\mathbb{N}$

## UNWEIGHTED GRAPHS

- Moore (1958) Breadth-first search (BFS), $O(E + V)$

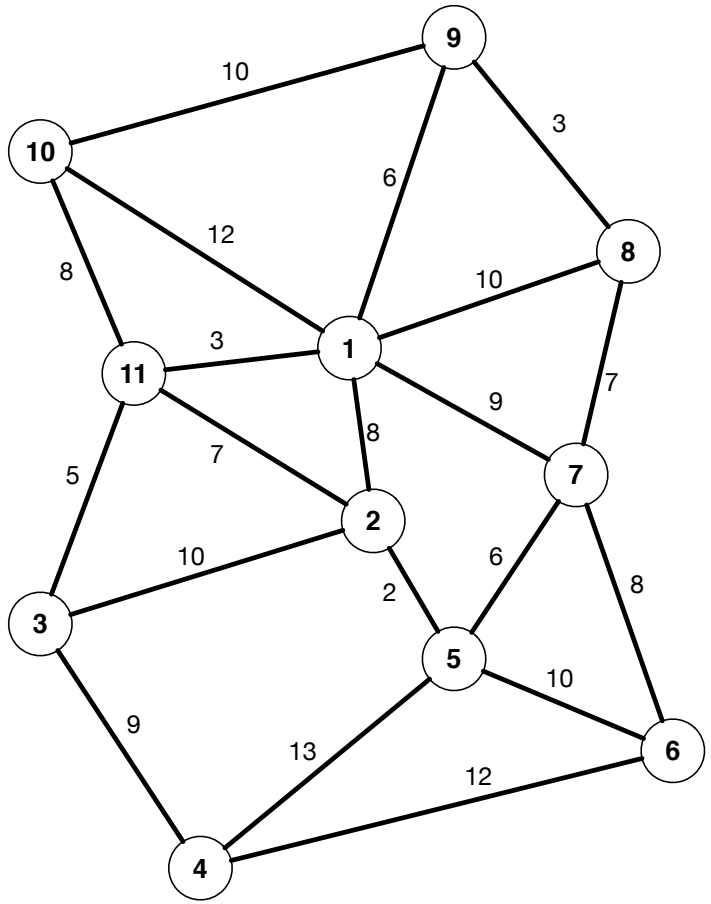
## DG with arbitrary weights, without negative cycles

- Bellman-Ford (1958) / Moore (59), $O(VE)$

La lista completa es mucho más extensa.

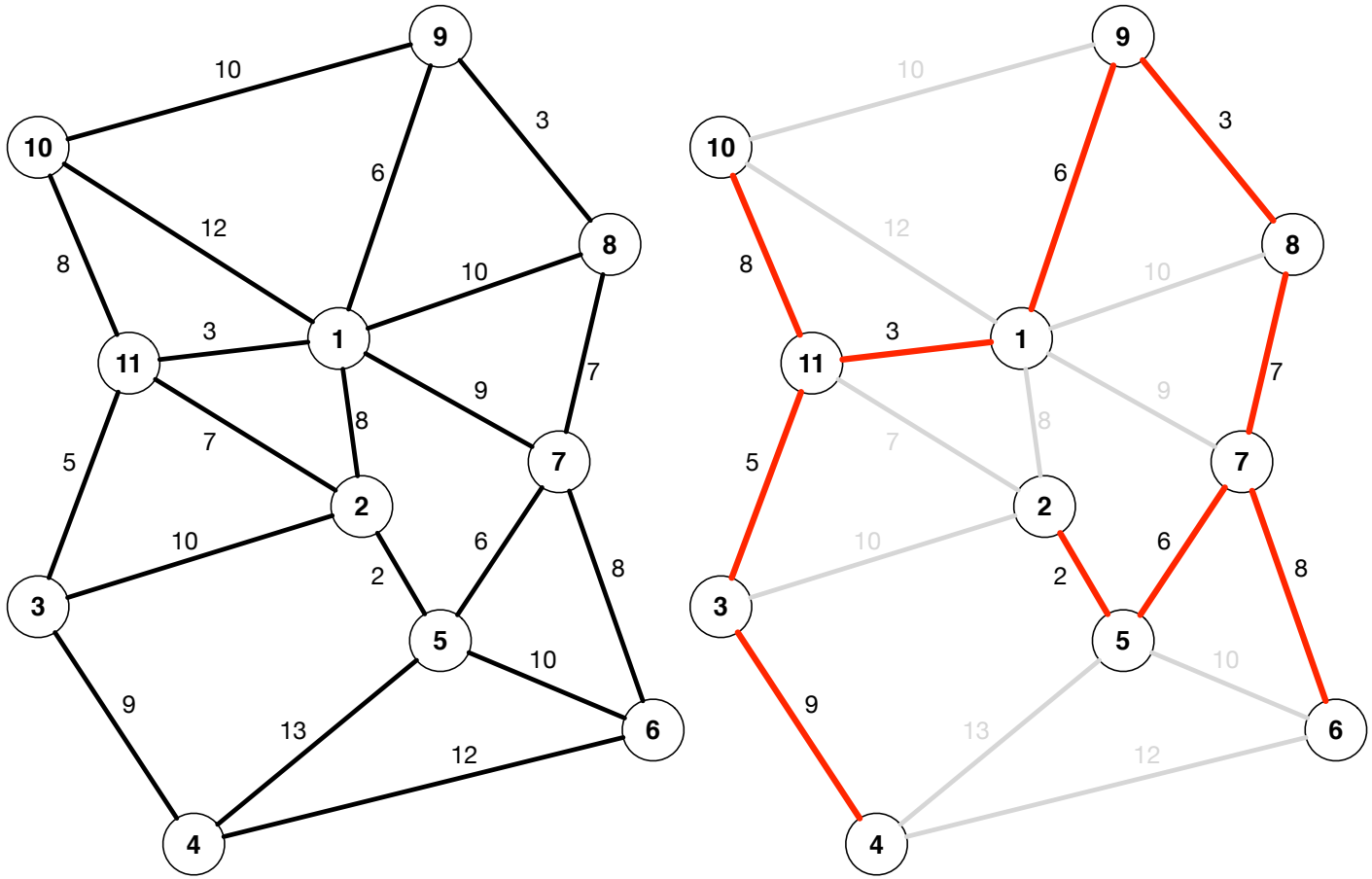
# Problemas tratables/fáciles: árbol de Cubrimiento (MST)

¿Cuál es un subgrafo conexo minimal (el árbol de cubrimiento) de costo mínimo para un grafo dado?



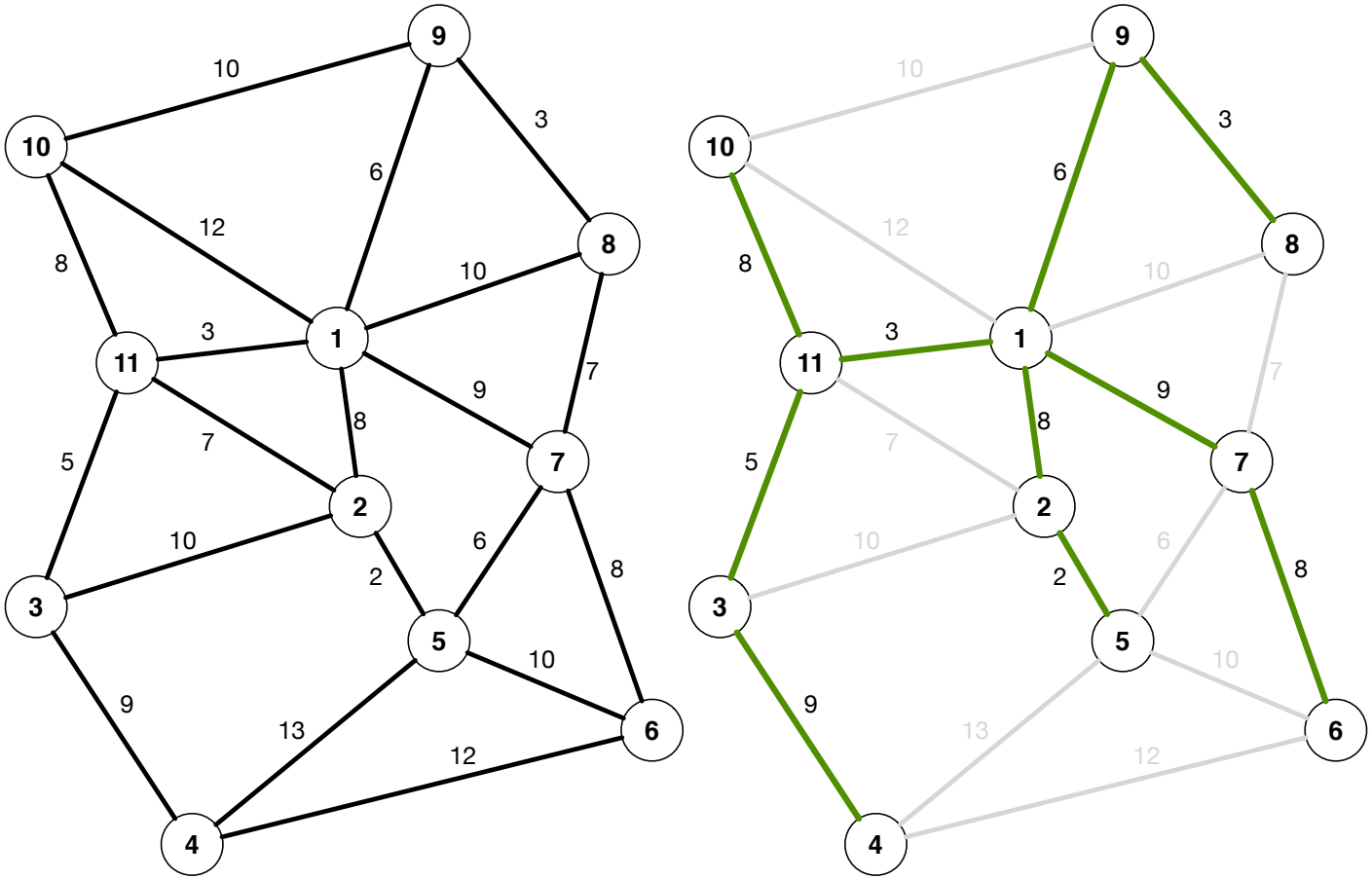
# Problemas tratables/fáciles: árbol de Cubrimiento (MST)

¿Cuál es un subgrafo conexo minimal (el árbol de cubrimiento) de costo mínimo para un grafo dado?



# Problemas tratables/fáciles: árbol de Cubrimiento (MST)

Observar que la solución (costo total 57) es distinta de la solución del SPP (costo 61).



# Problemas tratables/fáciles: árbol de Cubrimiento (MST)

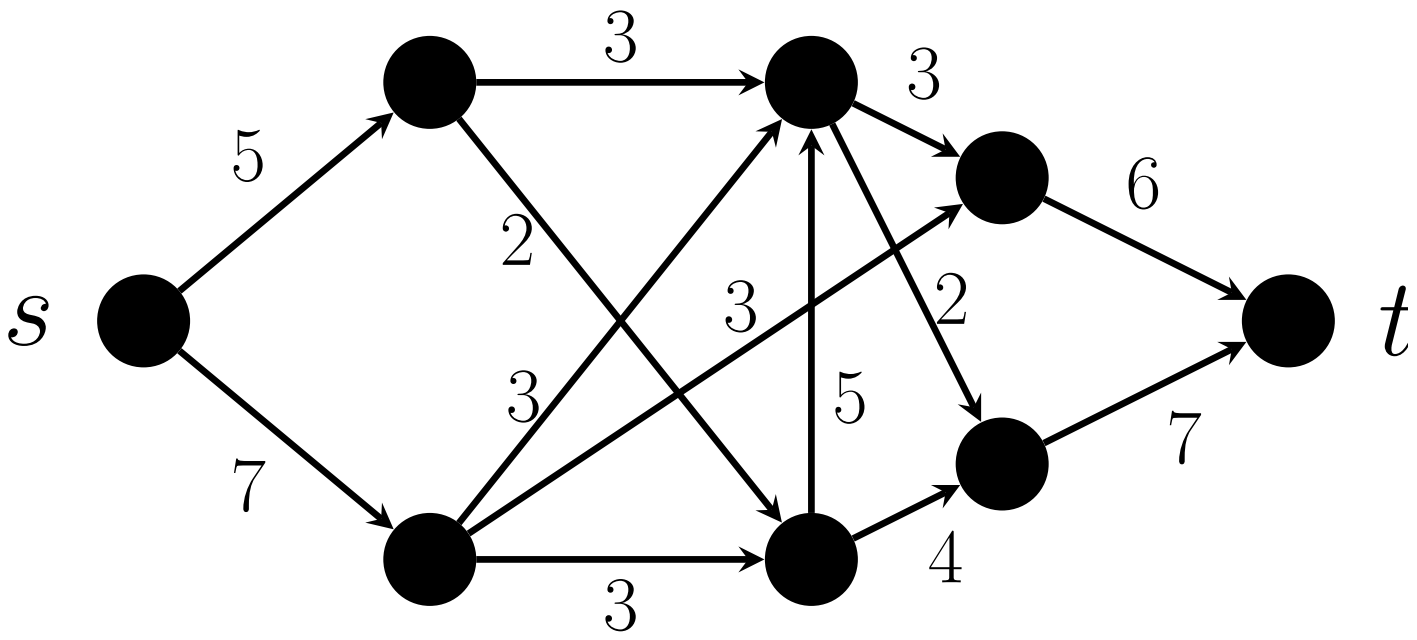
¿Cuál es un subgrafo conexo minimal (el árbol de cubrimiento) de costo mínimo para un grafo dado?

Los algoritmos polinomiales más famosos para resolver este problema son

- **Prim** Construye un grafo conexo con $|V| - 1$ arcos
- **Kruskal** Busca un grafo acíclico con $|V| - 1$ arcos
- Ambas propiedades garantizan que el resultado es un árbol
- Los órdenes de ejecución son $O(|E| + |V| \log |V|)$ y $O(|E| log |E|)$ respectivamente

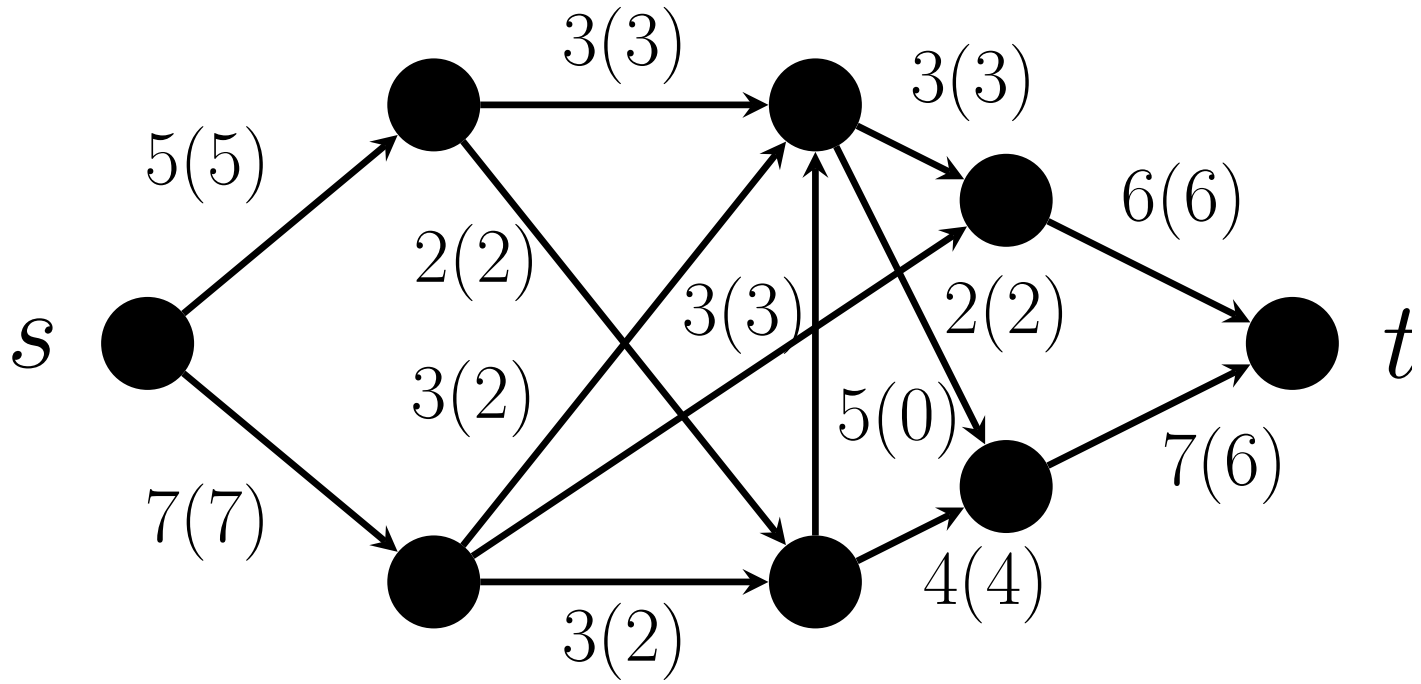
# Problemas tratables/fáciles: Flujo Máximo (MFP)

¿Cuál es el máximo flujo que puede circular de $s$ a $t$ en esta red con esas capacidades?



# Problemas tratables/fáciles: Flujo Máximo (MFP)

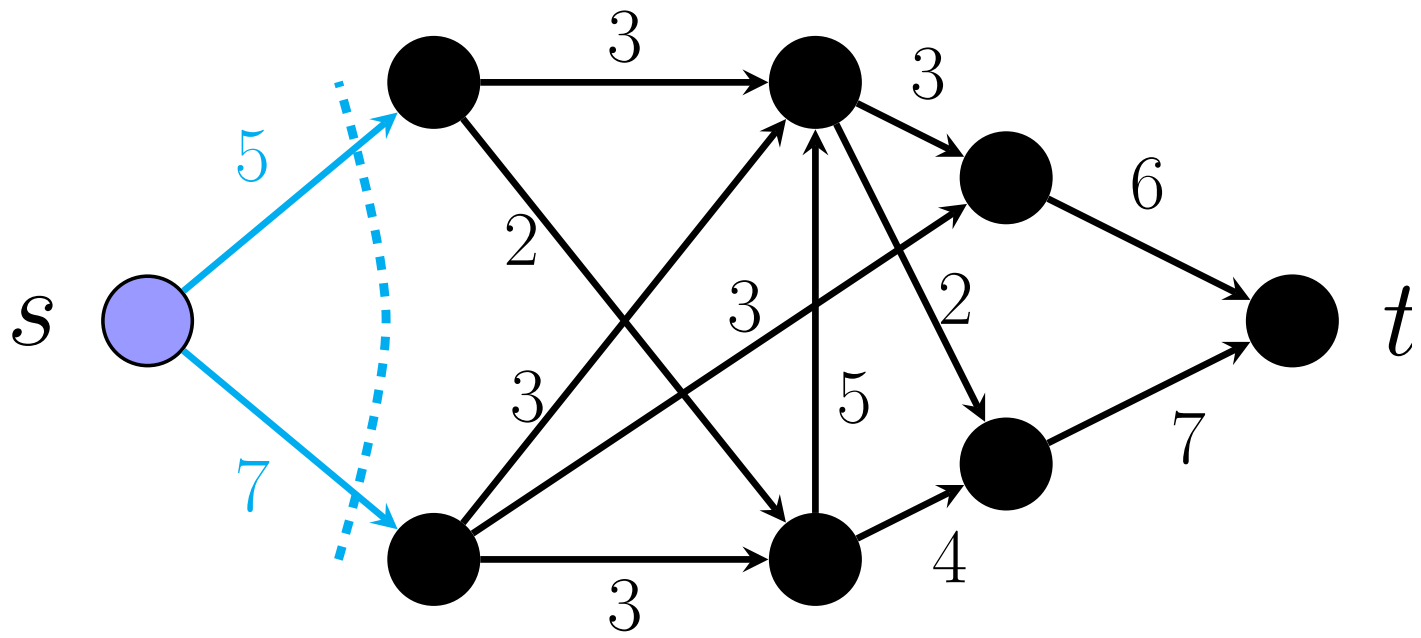
¿Cuál es el máximo flujo que puede circular de $s$ a $t$ en esta red con esas capacidades?




La respuesta es 12, y podría lograrse con esta configuración

# Problemas tratables/fáciles: Flujo Máximo (MFP)

¿Cuál es el máximo flujo que puede circular de $s$ a $t$ en esta red con esas capacidades?

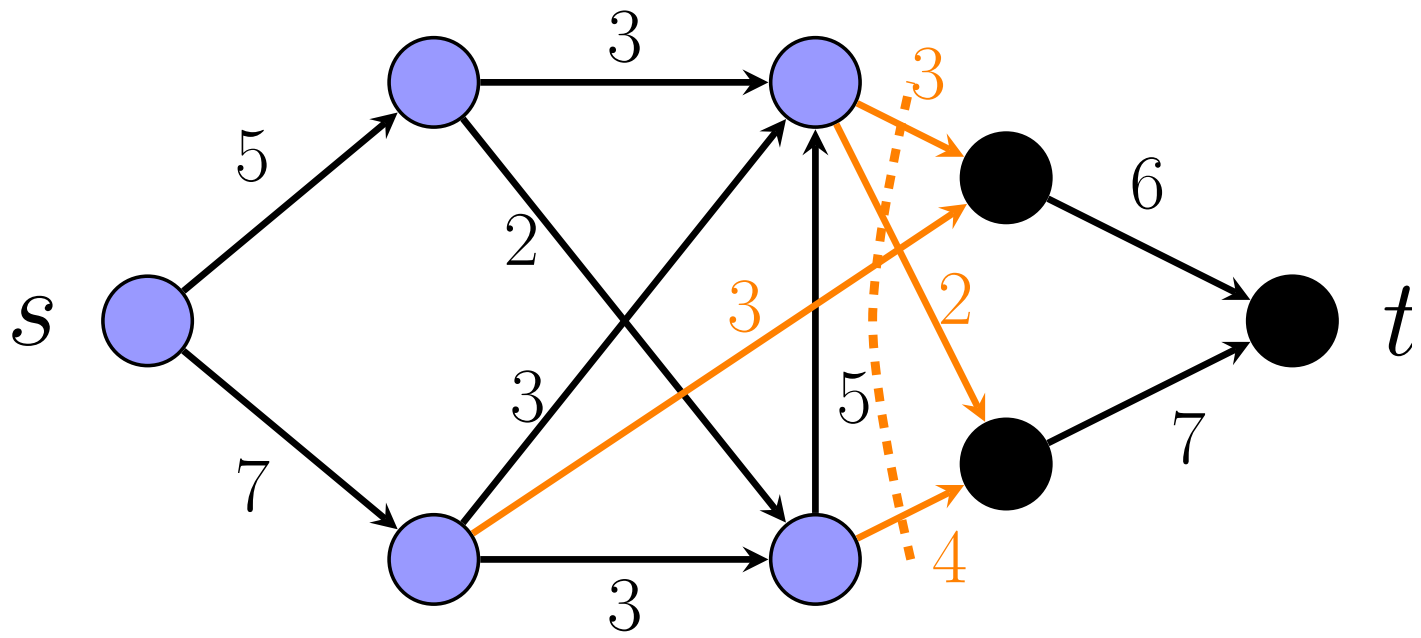



**Teorema (Ford-Fulkerson (1962))**

*El valor del flujo máximo entre s y t es igual a la capacidad del corte mínimo.*

# Problemas tratables/fáciles: Flujo Máximo (MFP)

¿Cuál es el máximo flujo que puede circular de $s$ a $t$ en esta red con esas capacidades?




**Teorema (Ford-Fulkerson (1962))**

*El valor del flujo máximo entre s y t es igual a la capacidad del corte mínimo.*

# Problemas tratables/fáciles: Flujo Máximo (MFP)

El *Maximum Flow Problem* tiene múltiples aplicaciones en:

- Logística y transporte (desde movimientos de mercancías a scheduling de aerolíneas)
- Resiliency (caminos disjuntos entre nodos, conjuntos de separación en una red)
- Cubrimientos jerárquicos (aplicaciones varias)

También hay muchas variantes y algoritmos:

Ford-Fulkerson (1956): Complejidad $O(E \max(f))$

Dinic (1970): Complejidad $O(VE \log(V))$

Malhotra, Pramodh, Kumar y Maheshwari (1978): $O(V^3)$

La lista completa es mucho más extensa.

# Problemas tratables/fáciles: Flujo Máximo (MFP)

El *Maximum Flow Problem* tiene múltiples aplicaciones en:

- Logística y transporte (desde movimientos de mercancías a scheduling de aerolíneas)
- Resiliency (caminos disjuntos entre nodos, conjuntos de separación en una red)
- Cubrimientos jerárquicos (aplicaciones varias)

También hay muchas variantes y algoritmos:

Ford-Fulkerson (1956): Complejidad $O(E \max(f))$

Dinic (1970): Complejidad $O(VE \log(V))$

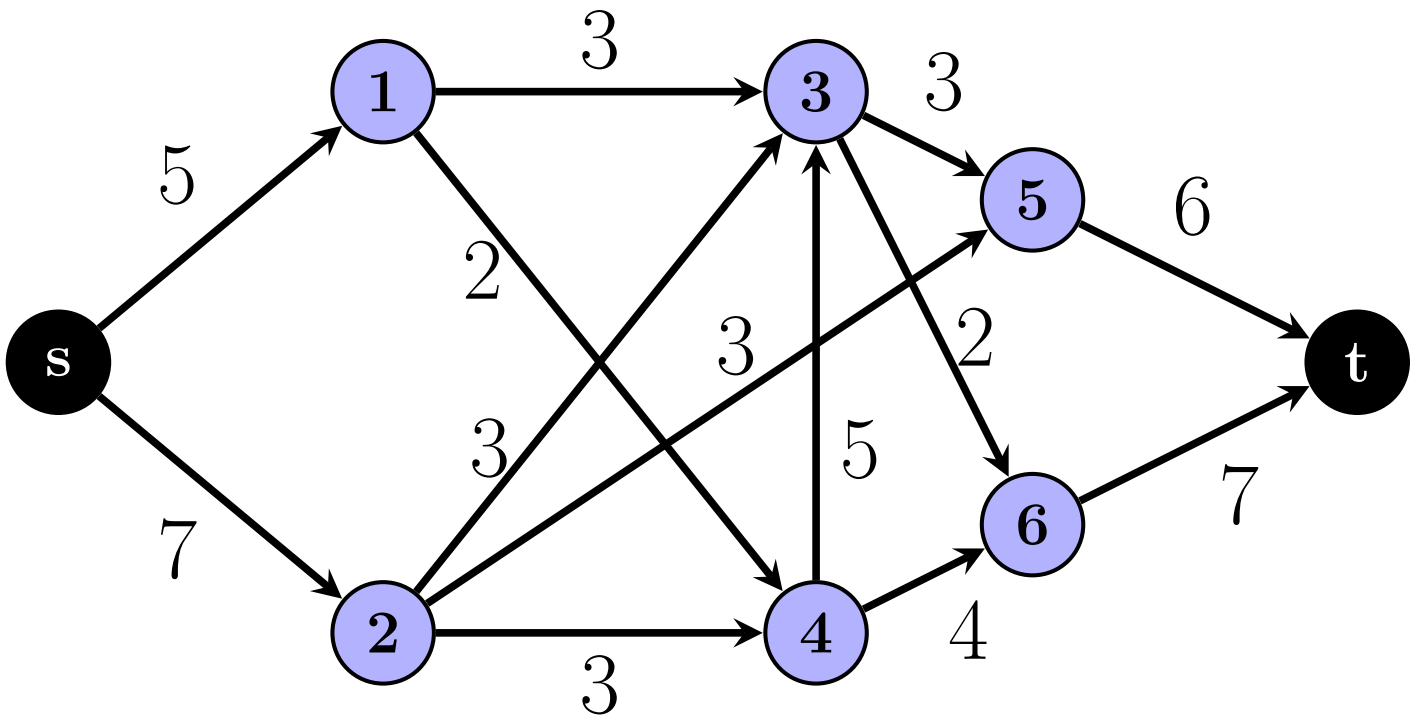
Malhotra, Pramodh, Kumar y Maheshwari (1978): $O(V^3)$

La lista completa es mucho más extensa.

**¿Qué pasa con los problemas de optimización en general?**

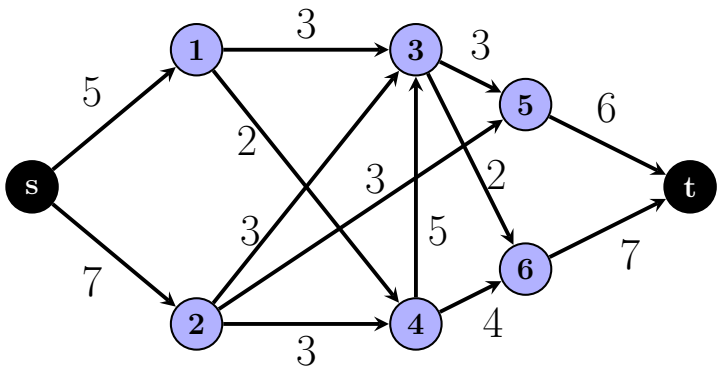
# El Problema de Programación Lineal

Los problemas anteriormente citados admiten formulaciones LP (Linear Programming). Por ejemplo el MFP:



# El Problema de Programación Lineal

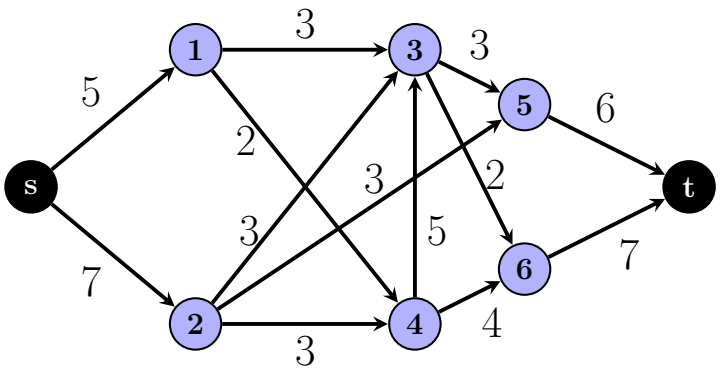
Los problemas anteriormente citados admiten formulaciones LP (Linear Programming). Por ejemplo el MFP:




$$
\begin{cases}
\max\ x_{s1} + x_{s2} \\
x_{s1} - x_{13} - x_{14} = 0 \\
x_{s2} - x_{23} - x_{24} - x_{25} = 0 \\
x_{13} + x_{23} + x_{43} - x_{35} - x_{36} = 0 \\
\dots \\
0 \le x_{s1} \le 5 \\
0 \le x_{s2} \le 7 \\
0 \le x_{13} \le 3 \\
\dots \\
0 \le x_{6t} \le 7
\end{cases}
$$

# El Problema de Programación Lineal

Los problemas anteriormente citados admiten formulaciones LP (Linear Programming). Por ejemplo el MFP:

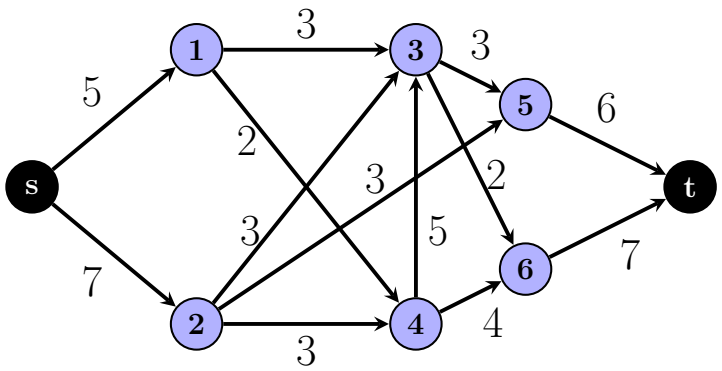



$$
\begin{cases}
\max\ x_{s1} + x_{s2} \\
x_{s1} - x_{13} - x_{14} = 0 \\
x_{s2} - x_{23} - x_{24} - x_{25} = 0 \\
x_{13} + x_{23} + x_{43} - x_{35} - x_{36} = 0 \\
\dots \\
0 \le x_{s1} \le 5 \\
0 \le x_{s2} \le 7 \\
0 \le x_{13} \le 3 \\
\dots \\
0 \le x_{6t} \le 7
\end{cases}
$$

Buscamos maximar el flujo que abandona $s$ (y por tanto circula por el grafo)

# El Problema de Programación Lineal

Los problemas anteriormente citados admiten formulaciones LP (Linear Programming). Por ejemplo el MFP:

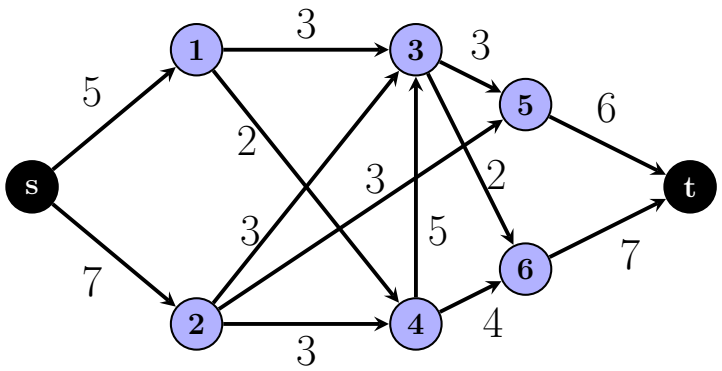



$$
\begin{cases}
\max\ x_{s1} + x_{s2} \\
x_{s1} - x_{13} - x_{14} = 0 \\
x_{s2} - x_{23} - x_{24} - x_{25} = 0 \\
x_{13} + x_{23} + x_{43} - x_{35} - x_{36} = 0 \\
\dots \\
0 \le x_{s1} \le 5 \\
0 \le x_{s2} \le 7 \\
0 \le x_{13} \le 3 \\
\dots \\
0 \le x_{6t} \le 7
\end{cases}
$$

Asegurando balance de flujo en los nodos intermedios (todos menos $s$ y $t$)

# El Problema de Programación Lineal

Los problemas anteriormente citados admiten formulaciones LP (Linear Programming). Por ejemplo el MFP:

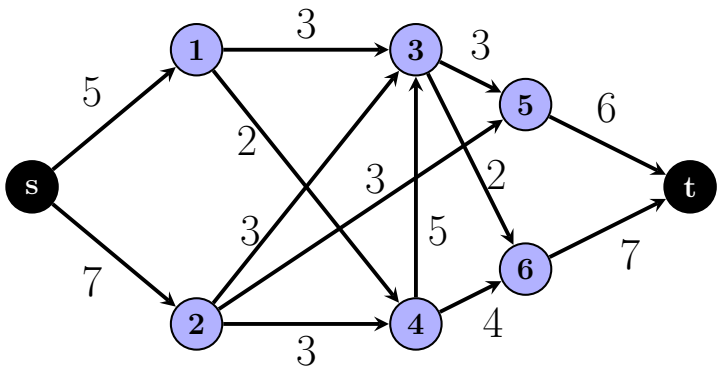



$$
\begin{cases}
\max\ x_{s1} + x_{s2} \\
x_{s1} - x_{13} - x_{14} = 0 \\
x_{s2} - x_{23} - x_{24} - x_{25} = 0 \\
x_{13} + x_{23} + x_{43} - x_{35} - x_{36} = 0 \\
\dots \\
0 \leq x_{s1} \leq 5 \\
0 \leq x_{s2} \leq 7 \\
0 \leq x_{13} \leq 3 \\
\dots \\
0 \leq x_{6t} \leq 7
\end{cases}
$$

Imponiendo al mismo tiempo que el valor del flujo en cada arco ($x_{ij}$) no viole la capacidad del enlace

# El Problema de Programación Lineal

Los problemas anteriormente citados admiten formulaciones LP (Linear Programming). Por ejemplo el MFP:




$$
\begin{cases}
\max\ x_{s1} + x_{s2} \\
x_{s1} - x_{13} - x_{14} = 0 \\
x_{s2} - x_{23} - x_{24} - x_{25} = 0 \\
x_{13} + x_{23} + x_{43} - x_{35} - x_{36} = 0 \\
\dots \\
0 \leq x_{s1} \leq 5 \\
0 \leq x_{s2} \leq 7 \\
0 \leq x_{13} \leq 3 \\
\dots \\
0 \leq x_{6t} \leq 7
\end{cases}
$$

**¡¡También puede verse que el dual de este problema, corresponde a encontrar el corte de capacidad mínima!!**

# Complejidad intrínseca en problemas de decisión

## Definición

*Dado un problema de decisión $\pi$, llamamos instancia de él, a un set concreto de parámetros que acotan unívocamente al problema para determinar una respuesta (Sí o No). Cada problema tiene asociado un dominio de instancias $D_\pi$, donde ese problema tiene sentido. Sea $Y_\pi \subseteq D_\pi$ el conjunto de las instancias cuya respuesta es Sí.*

## Definición

*Decimos que un problema de decisión $\pi$ está en la clase* **P**, *sii, hay un algoritmo capaz de encontrar la respuesta en orden polinomial del tamaño de la instancia. Decimos que $\pi$ está en la clase* **NP**, *sii, existe un algoritmo capaz de chequear en tiempo polinomial las soluciones provistas por un oráculo externo.*

# Complejidad intrínseca en problemas de decisión

**¿Pertenecen a $P$ o $NP$ los problemas SPP, MST o MFP?**

# Complejidad intrínseca en problemas de decisión

**¿Pertenecen a $P$ o $NP$ los problemas SPP, MST o MFP?**
Sí, los tres están en **P** porque existen algoritmos de orden polinomial en el tamaño del problema ($|V|$), que permiten encontrar la solución (e.g. Dijkstra, Prim, Ford-Fulkerson)

# Complejidad intrínseca en problemas de decisión

**¿Pertenecen a $P$ o $NP$ los problemas SPP, MST o MFP?**
Sí, los tres están en **P** porque existen algoritmos de orden polinomial en el tamaño del problema ($|V|$), que permiten encontrar la solución (e.g. Dijkstra, Prim, Ford-Fulkerson)
**Sabiendo que SPP y MFP (también MST) pueden formularse como problemas LP ¿Podemos decir que LP está en $P$?**

# Complejidad intrínseca en problemas de decisión

**¿Pertenecen a *P* o *NP* los problemas SPP, MST o MFP?**
Sí, los tres están en **P** porque existen algoritmos de orden polinomial en el tamaño del problema ($|V|$), que permiten encontrar la solución (e.g. Dijkstra, Prim, Ford-Fulkerson)
**Sabiendo que SPP y MFP (también MST) pueden formularse como problemas LP ¿Podemos decir que LP está en *P*?**
No, eso sólo nos dice que *algunas instancias* pueden resolverse en tiempo polinomial. Para estar en **P** hay que tener certeza que cualquier instancia puede resolverse en tiempo polinomial

# Complejidad intrínseca en problemas de decisión

**¿Pertenecen a $P$ o $NP$ los problemas SPP, MST o MFP?**
Sí, los tres están en **P** porque existen algoritmos de orden polinomial en el tamaño del problema ($|V|$), que permiten encontrar la solución (e.g. Dijkstra, Prim, Ford-Fulkerson)
**Sabiendo que SPP y MFP (también MST) pueden formularse como problemas LP ¿Podemos decir que LP está en $P$?**
No, eso sólo nos dice que *algunas instancias* pueden resolverse en tiempo polinomial. Para estar en **P** hay que tener certeza que cualquier instancia puede resolverse en tiempo polinomial
**¿Se cumple que $P \subseteq NP$?**

# Complejidad intrínseca en problemas de decisión

**¿Pertenecen a *P* o *NP* los problemas SPP, MST o MFP?**
Sí, los tres están en **P** porque existen algoritmos de orden polinomial en el tamaño del problema ($|V|$), que permiten encontrar la solución (e.g. Dijkstra, Prim, Ford-Fulkerson)
**Sabiendo que SPP y MFP (también MST) pueden formularse como problemas LP ¿Podemos decir que LP está en *P*?**
No, eso sólo nos dice que *algunas instancias* pueden resolverse en tiempo polinomial. Para estar en **P** hay que tener certeza que cualquier instancia puede resolverse en tiempo polinomial
**¿Se cumple que $P \subseteq NP$?**
Sí, en el peor de los casos se resuelve el problema (es polinomial) y se comparan los resultados

# Complejidad intrínseca en problemas de decisión

**¿Pertenecen a $P$ o $NP$ los problemas SPP, MST o MFP?**
Sí, los tres están en **P** porque existen algoritmos de orden polinomial en el tamaño del problema ($|V|$), que permiten encontrar la solución (e.g. Dijkstra, Prim, Ford-Fulkerson)
**Sabiendo que SPP y MFP (también MST) pueden formularse como problemas LP ¿Podemos decir que LP está en $P$?**
No, eso sólo nos dice que *algunas instancias* pueden resolverse en tiempo polinomial. Para estar en **P** hay que tener certeza que cualquier instancia puede resolverse en tiempo polinomial
**¿Se cumple que $P \subseteq NP$?**
Sí, en el peor de los casos se resuelve el problema (es polinomial) y se comparan los resultados
**¿Se cumple que $NP \subseteq P$?**

# Complejidad intrínseca en problemas de decisión

**¿Pertenecen a $P$ o $NP$ los problemas SPP, MST o MFP?**
Sí, los tres están en **P** porque existen algoritmos de orden polinomial en el tamaño del problema ($|V|$), que permiten encontrar la solución (e.g. Dijkstra, Prim, Ford-Fulkerson)
**Sabiendo que SPP y MFP (también MST) pueden formularse como problemas LP ¿Podemos decir que LP está en $P$?**
No, eso sólo nos dice que *algunas instancias* pueden resolverse en tiempo polinomial. Para estar en **P** hay que tener certeza que cualquier instancia puede resolverse en tiempo polinomial
**¿Se cumple que $P \subseteq NP$?**
Sí, en el peor de los casos se resuelve el problema (es polinomial) y se comparan los resultados
**¿Se cumple que $NP \subseteq P$?**
**No se sabe. Éste es de hecho uno de los problemas abiertos más importantes de la informática**

# Complejidad intrínseca en problemas de decisión

El **Number Partition Problem (NPP)** consiste en encontrar una partición (dos subconjuntos) con casi la misma suma para un multiset de números $\mathcal{M}=\{a_1, \ldots, a_N\}$. Buscamos $\mathcal{A} \subset \{1, \ldots, N\}$ tal que se cumpla: $|\sum_{i \in \mathcal{A}} a_i - \sum_{i \notin \mathcal{A}} a_i| \leq 1$ ¿Está en **NP** el NPP?

# Complejidad intrínseca en problemas de decisión

El **Number Partition Problem (NPP)** consiste en encontrar una partición (dos subconjuntos) con casi la misma suma para un multiset de números $\mathcal{M}=\{a_1, \ldots, a_N\}$. Buscamos $\mathcal{A} \subset \{1, \ldots, N\}$ tal que se cumpla: $|\sum_{i \in \mathcal{A}} a_i - \sum_{i \notin \mathcal{A}} a_i| \leq 1$ ¿Está en **NP** el NPP?
No es fácil partir $\{62, 83, 121, 281, 486, 734, 771, 854, 885, 1003\}$

# Complejidad intrínseca en problemas de decisión

El **Number Partition Problem (NPP)** consiste en encontrar una partición (dos subconjuntos) con casi la misma suma para un multiset de números $\mathcal{M}=\{a_1, \ldots, a_N\}$. Buscamos $\mathcal{A} \subset \{1, \ldots, N\}$ tal que se cumpla: $|\sum_{i\in\mathcal{A}} a_i - \sum_{i\notin\mathcal{A}} a_i| \leq 1$ ¿Está en **NP** el NPP? No es fácil partir {62, 83, 121, 281, 486, 734, 771, 854, 885, 1003} pero sí lo es verificar que {281, 734, 771, 854}, {62, 83, 121, 486, 885, 1003} es una partición que satisface el NPP.

# Complejidad intrínseca en problemas de decisión

El **Number Partition Problem (NPP)** consiste en encontrar una partición (dos subconjuntos) con casi la misma suma para un multiset de números $\mathcal{M}=\{a_1, \ldots, a_N\}$. Buscamos $\mathcal{A} \subset \{1, \ldots, N\}$ tal que se cumpla: $|\sum_{i\in\mathcal{A}} a_i - \sum_{i\notin\mathcal{A}} a_i| \leq 1$ ¿Está en **NP** el NPP? No es fácil partir {62, 83, 121, 281, 486, 734, 771, 854, 885, 1003} pero sí lo es verificar que {281, 734, 771, 854}, {62, 83, 121, 486, 885, 1003} es una partición que satisface el NPP.

El **Boolean Satisfiability Problem (SAT)** consiste en determinar si existe una interpretación que satisfaga una fórmula Booleana cualquiera. Por ejemplo, encontrar valores $x_i \in \{0, 1\}$, $i = 1, 2, 3$, que verifquen $(x_1 \vee \neg x_3) \wedge (\neg x_1 \vee x_2) \wedge x_3 = 1$.
¿Es **NP** el SAT?

# Complejidad intrínseca en problemas de decisión

El **Number Partition Problem (NPP)** consiste en encontrar una partición (dos subconjuntos) con casi la misma suma para un multiset de números $\mathcal{M}=\{a_1,\ldots,a_N\}$. Buscamos $\mathcal{A}\subset\{1,\ldots,N\}$ tal que se cumpla: $|\sum_{i\in\mathcal{A}} a_i - \sum_{i\notin\mathcal{A}} a_i| \leq 1$ ¿Está en **NP** el NPP? No es fácil partir {62, 83, 121, 281, 486, 734, 771, 854, 885, 1003} pero sí lo es verificar que {281, 734, 771, 854}, {62, 83, 121, 486, 885, 1003} es una partición que satisface el NPP.

El **Boolean Satisfiability Problem (SAT)** consiste en determinar si existe una interpretación que satisfaga una fórmula Booleana cualquiera. Por ejemplo, encontrar valores $x_i \in \{0,1\}$, $i=1,2,3$, que verifquen $(x_1 \vee \neg x_3) \wedge (\neg x_1 \vee x_2) \wedge x_3 = 1$.
¿Es **NP** el SAT? Sí. Es fácil ver que $x_1 = x_2 = x_3 = 1$ es solución.

# Una regla para comparar las complejidades

Los padres de la Teoría de la Complejidad incluyen a: Alan Turing, Stephen Cook y Richard Karp. La teoría propone una forma de caracterizar/comparar problemas.

## Definición

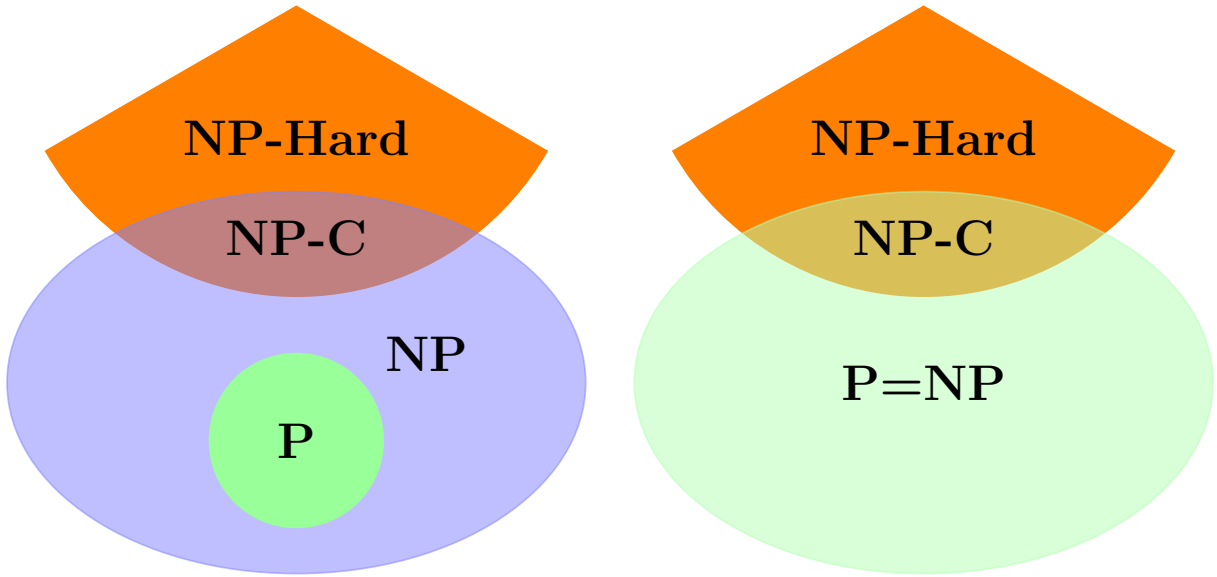
*Dados cualesquiera dos problemas de decisión $\pi$ y $\pi'$, con sus respectivos dominios $D_\pi$ y $D_{\pi'}$, llamamos reducción polinomial de $\pi'$ a $\pi$ ($\pi' \preccurlyeq \pi$) a cualquier función $f : D_{\pi'} \to D_\pi$ de complejidad polinomial, tal que, para cualquier $d \in D_{\pi'}$ se cumple que $d \in Y_{\pi'}$ si y sólo si $f(d) \in Y_\pi$.*

La existencia de una reducción polinomial de $\pi'$ a $\pi$ ($\pi' \preccurlyeq \pi$) significa que si alguien desarrollara un algoritmo eficiente para hallar soluciones a las instancias de $\pi$, a través de la reducción, ese algoritmo podría usarse como *kernel* de un algoritmo eficiente para resolver cualquier instancia de $\pi'$.

# Clasificación de problemas según su complejidad conocida

## Definición

*Dado un problema $\pi$, decimos que él es NP-Hard, si y sólo si, para todo $\pi' \in NP$ se cumple que $\pi' \preccurlyeq \pi$. Cuando además se cumple que $\pi \in NP$, decimos que $\pi$ es NP-Completo o NP-C.*




Cuál es el diagrama correcto depende de si $\mathbf{P} \neq \mathbf{NP}$ o $\mathbf{P} = \mathbf{NP}$.
Se conjetura que $\mathbf{P} \neq \mathbf{NP}$.

# Clasificación de problemas según su complejidad conocida

**Teorema (Cook's (1971))**

*El boolean SATisfiability (SAT) es NP-Completo. Así, el problema SAT es al menos tan difícil como cualquier problema en NP.*

En 1972, Richard Karp presentó una lista de 21 reducciones polinomiales del SAT a otros problemas en *NP*, mostrando entonces que los problemas en la lista eran *NP-completos* (Karp's list).

- Determinar si hay un corte de cierto tamaño en un grafo es *NP-Completo*, y también lo es encontrar el "Corte Máximo".
- El "Number Partitioning Problem" es *NP-Completo*.
- El Travelling Salesman Problem (ver Contraejemplo a la Programación Dinámica) es *NP-Hard* en general.
  Si las distancias son números enteros, el TSP se vuelve *NP-Complete*.

# Clasificación de problemas según su complejidad conocida

El procedimiento estándar para probar que un problema $\pi \in NP$ es *NP-Completo*, consiste en encontrar otro problema *NP-completo* conocido ($\pi'$) y una reducción polinomial de ese problema $\pi'$ al nuevo problema $\pi$ (i.e. probar que $\pi' \preccurlyeq \pi$).

Así, la transitividad de la relación "$\preccurlyeq$", garantiza que: $SAT \preccurlyeq \pi$.

Complementariamente y dado que el SAT es el problema NP más difícil (Cook's theorem), ambas complejidades deben ser equivalentes y $\pi$ está en la clase *NP-Completo*.

Se destaca que aun si no podemos probar que $\pi \in NP$, el procedimiento anterior garantiza al menos que $\pi$ es *NP-Hard*.

# El problema general de optimización

El problema de optimización cuadrática sin restricciones está en **P**, porque el Método del Gradiente Conjugado (Magnus Hestenes y Eduard Stiefel, 1952) encuentra el óptimo en $n$ pasos.

En 1984, Narendra Karmarkar probó que la programación lineal continua ($LP$) está en la clase **P**, para lo que usó un algoritmo de punto interior. La prueba se extiende a ($QP$) en el caso definido positivo. Estos problemas son computacionalmente fáciles.

Sin embargo, cuando $X \subseteq \mathbb{Z}^n$, incluso los problemas lineales son difíciles en general. Los problemas de optimización combinatoria o mixtos son más difíciles que los continuos.