Optimización y búsqueda

M. Sc. Saúl Calderón Ramírez
Instituto Tecnológico de Costa Rica,
Escuela de Computación.

13 de marzo de 2019

El presente documento introduce conceptos básicos de la inteligencia artificial y algoritmos básicos de optimización.

1. Optimización

Sea una función de una variable independiente $x \in \mathbb{R}$:

$$y = f(x)$$

Supóngase el problema de encontrar el punto $x_{\text{máx}}$ o punto óptimo x_{opt}el cual haga que la función en tal punto $f(x_{\text{máx}})$ tome su valor máximo, de modo que:

$$\forall x,\ f(x) \leq f(x_{\text{máx}}).$$

A tal valor $x_{\text{máx}}$ se le denomina el **óptimo global** de la función a optimizar, usualmente referida como **función de costo**, **función objetivo**, o en el aprendizaje automático, **función de pérdida**. Los **algoritmos de optimización** tienen entonces por objetivo encontrar tal punto óptimo. La Figura 1 ilustra los conceptos de óptimo global y local. La derivada y segunda derivada pueden ser usados como criterio de optimalidad:

$$f'(x) = 0 \qquad f''(x) > 0$$

Muchas veces, la evaluación de la función $f(x)$ es computacionalmente muy costosa, por lo que muchos algoritmos se conforman con buscar un **óptimo local**. Un óptimo local, es un valor óptimo en un vecindario de la función que va desde el punto x_a al punto x_b, lo cual se expresa como:

$$x \in [x_a, x_b],\ f(x) \leq f(x_{\text{máx}}).$$

El encontrar un óptimo local es computacionalmente más viable en muchos casos, y la solución encontrada suficiente. A continuación se detallan distintas variantes del problema de optimización, según características de la función a optimizar.

1.1. Optimización convexa y no convexa

Una función convexa cumple que $\forall x, y \in \mathbb{R}$ y $\forall t \in [0, 1]$ lo siguiente:

$$f(tx + (1-t)y) < tf(x) + (1-t)f(y)$$

Donde $tx + (1-t)y$ corresponde a todos los puntos en el intervalo $[x, y]$, por lo que la evaluación en la función convexa de tales puntos $f(tx + (1-t)y)$ es menor a los puntos en la recta dada entre $f(x)$ y $f(y)$, con ecuación $tf(x) + (1-t)f(y)$, como lo ilustra la Figura 2.

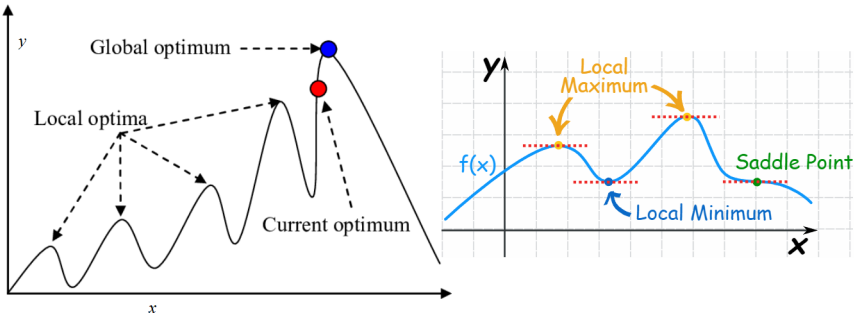

Figura 1: Ilustración del problema de optimización: encontrar al menos un óptimo local, en nuestro caso, máximo local. A la derecha la ilustración del concepto de **punto silla o llanura**.

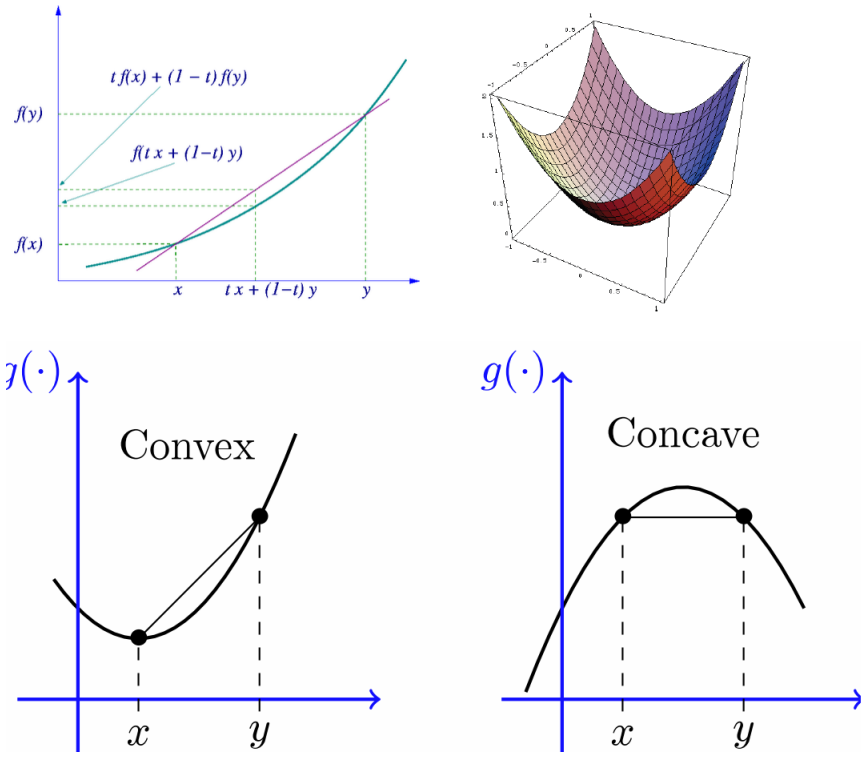

Figura 2: Función en una variable y en dos variables convexas.

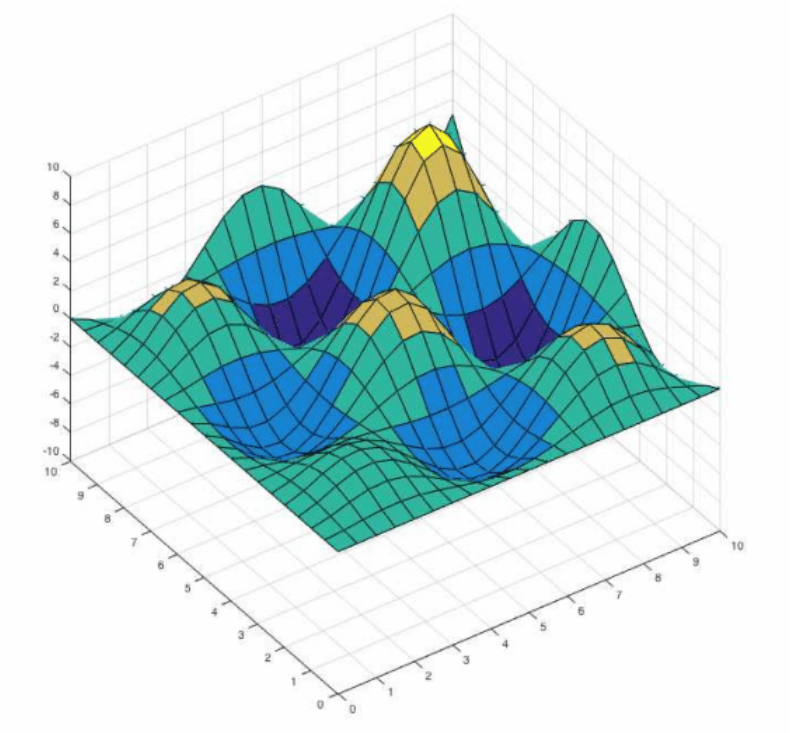

Figura 3: Función no convexa.

El negativo $-f$ de una función convexa f es una función cóncava. Observe como el punto óptimo (mínimo en el caso de la función convexa, máximo en la función cóncava) es el óptimo global, por lo que en una función convexa sólo existe un mínimo global. Los algoritmos diseñados para encontrar el punto óptimo en una función tienen por objetivo encontrar un único mínimo global. Los algoritmos basados en la **programación lineal** usualmente están diseñados para encontrar el punto óptimo en funciones convexas con **restricciones en el dominio**. En los casos con los que contamos con una **expresión analítica** de la función $f(x)$, y la misma es **derivable**, al realizar

$$f'(x) = 0$$

es decir, encontrar la raíz de la derivada, la cual si es única corresponde entonces a una optimización de una función convexa.

Las funciones no convexas, no cumplen la condición $f(tx + (1-t)y) < tf(x) + (1-t)f(y)$, con múltiples *valles y montañas*, correspondientes a múltiples mínimos y máximos locales, como se ilustra en la Figura 3. Los problemas más frecuentes en aplicaciones cotidianas forman funciones no convexas con múltiples óptimos locales, por lo que los algoritmos diseñados para la optimización de funciones no convexas tienen por objetivo buscar un punto óptimo local lo suficientemente *satisfactorio*.

Los algoritmos de optimización como el enjambre de partículas, el calentamiento simulado, los algoritmos genéticos, el Newton-Raphson o el descenso de gradiente son todos algoritmos diseñados para encontrar mínimos locales en funciones no convexas, donde incluso presciden de una expresión analítica de la función.

1.2. Optimización de funciones con restricciones y sin restricciones

Frecuentemente, la función a optimizar $f(\overrightarrow{x}) \in \mathbb{R}$, $\overrightarrow{x} \in \mathbb{R}^n$, necesita de implementar restricciones tanto en su dominio, como en su codominio, por lo que de forma general, cada componente del dominio debe pertenecer a un

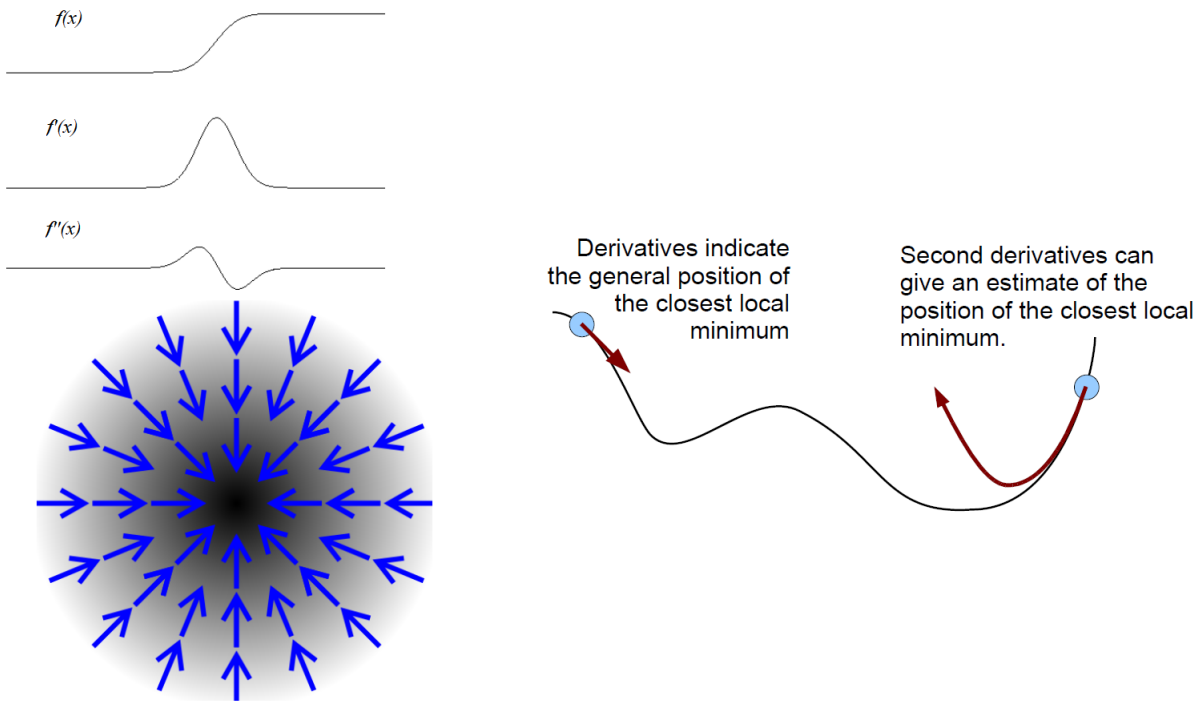

Figura 4: Significado de primera y segunda derivada.

espacio definido, de forma que:

$$\begin{aligned} x_1 &\in \mathbb{D}_1 \\ x_2 &\in \mathbb{D}_2 \\ &\vdots \\ x_n &\in \mathbb{D}_n \end{aligned}$$

y para el caso del codominio, su salida puede estar restriginda, a pertenecer a un espacio $\mathbb{C}$,por lo que en tal caso $f\left(\overrightarrow{x}\right) \in \mathbb{C}$. Algunos métodos populares de optimización con restricciones son los multiplicadores de Lagrange y los algoritmos de optimización primal y dual populares en la investigación de operaciones.

1.3. Optimización de funciones diferenciables y no diferenciables

La primera derivada en funciones de una dimensión $f\left(x\right)$, denotada como $\frac{df}{dx}$ o $f'\left(x\right)$, y su extensión conceptual en funciones multivariable $f\left(\overrightarrow{x}\right)$ con $\overrightarrow{x} \in \mathbb{R}^n$, el vector gradiente $\nabla_{\overrightarrow{x}} f$ indica la dirección hacia donde crece la función, con $\overrightarrow{x} = \left[x_1, x_2, \ldots, x_n\right]^T$, y el vector gradiente dado más explicitamente por $\nabla_{\overrightarrow{x}} f = \left[\frac{df}{dx_1}, \frac{df}{dx_2}, \ldots, \frac{df}{dx_n}\right]^T$. La segunda derivada de una función en una variable se expresa como $\frac{df}{dx^2}$ o $f''\left(x\right)$ describe la curvatura o concavidad de la función, y los puntos de inflexión o cortes en cero de la función f, como lo ilustra la Figura 4. De este modo, si $f''\left(x\right) > 0$ la función es convexa, y si $f''\left(x\right) < 0$ es cóncava.

Para el caso de funciones multivariable $f\left(\overrightarrow{x}\right)$ con $\overrightarrow{x} \in \mathbb{R}^n$, la extensión del concepto de la segunda derivada

en un espacio de dimensión n se define en la matriz Hessiana $H_{\overrightarrow{x}} f$:

$$H_{\overrightarrow{x}} f = \begin{bmatrix} \frac{df}{dx_1^2} & \frac{df}{dx_1 dx_2} & \cdots & \frac{df}{dx_1 dx_n} \\ \frac{df}{dx_2 dx_1} & \frac{df}{dx_2^2} & \cdots & \frac{df}{dx_2 dx_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{df}{dx_n dx_1} & \cdots & \cdots & \frac{df}{dx_n^2} \end{bmatrix}$$

La segunda derivada es entonces más costosa de computar en general. Los métodos basados en la primera derivada como el **descenso de gradiente**, y en la segunda derivada como el algoritmo de **Newton-Raphson** necesitan entonces de una derivada computable en tiempo reazonable, ya sea analítica o numéricamente. Los métodos basados en la segunda derivada necesitan de menos iteraciones, pues la **información de la curvatura representada en la matriz Hessiana da una mayor exactitud de la localización del óptimo**, sin embargo, es necesario mayor cómputo por iteración, y muchas veces el cálculo de la segunda derivada en expresiones analíticas no es tratable. Ambos algoritmos son estocásticos por cuanto su convergencia depende del punto inicial x_0, elegido usualmente al azar.

Cuando no es posible el computo del gradiente o la matriz Hesiana en general, existen métodos simples como la búsqueda por bisección, la cual se explica en la siguiente sección, posteriormente se detallan los algoritmos de descenso de gradiente y Newton-Raphson.

2. Búsqueda

La búsqueda se refiere a encontrar un punto óptimo $\overrightarrow{x} \in \mathbb{N}^n$ en un espacio discreto, para una función de costo u objetivo $f(\overrightarrow{x})$ la cual puede ser continua o discreta. Los problemas de búsqueda clásicos estudiados en la inteligencia artificial y en las ciencias de la computación fijan además restricciones en el dominio para las **soluciones candidatas**, de modo que:

$$\begin{gathered} x_1 \in \mathbb{D}_1 \\ x_2 \in \mathbb{D}_2 \\ \vdots \\ x_n \in \mathbb{D}_n \end{gathered}$$

o de forma más compacta en el espacio $\overrightarrow{x} \in \mathbb{D}^n$. Algunos de estos problemas clásicos son el problema de las N Reinas (el cual consiste en fijar en un tablero N reinas y que estas no puedan atacarse mutuamente), la búsqueda de la ruta más corta entre dos puntos A y B, y el rombecabezas de números (el cual tiene por objetivo ordenar los números de forma ascendente en un tablero de $N \times N$), ilustrados en la Figura 5.

Sin embargo, la diferencia más importante de un problema de búsqueda respecto a los problemas de optimización, es la definición de restricciones de igualdad usualmente para la solución $\overrightarrow{x}_{\text{solucion}} \in \mathbb{A}^n$, de forma que la solución encontrada debe pertenecer a otro espacio $\mathbb{A}$. En el ejemplo del rompecabezas de 8 piezas, el objetivo es encontrar el estado solución $\overrightarrow{x}_{\text{solucion}} \in \mathbb{N}^9$, con los valores en la secuencia ilustrada en la Figura 5, que de forma vectorizada corresponden a $\overrightarrow{x}_{\text{solucion}} = [1, 2, 3, 4, 5, 6, 7, 8, 0]$ donde la ficha 0 corresponde al espacio vacío. Cada punto $\overrightarrow{x} \in \mathbb{D}^n$ en el espacio $\mathbb{D}$ se denomina **estado.** Es usual que las restricciones de tal dominio restrinjan también los **estados sucesores** o a los que se puede llegar a partir de un **estado origen**. En el ejemplo del rompecabezas de 8 piezas, para el estado $\overrightarrow{x}_1 = [1, 2, 3, 8, 0, 4, 7, 6, 5]$, sólo son posibles los estados sucesores $\overrightarrow{x}_{2,1} = [1, 0, 3, 8, 2, 4, 7, 6, 5]$, $\overrightarrow{x}_{2,2} = [1, 2, 3, 8, 4, 0, 7, 6, 5]$, $\overrightarrow{x}_{2,3} = [1, 2, 3, 8, 6, 4, 7, 0, 5]$ y $\overrightarrow{x}_{2,4} = [1, 2, 3, 0, 8, 4, 7, 6, 5]$, dada las restricciones físicas de las fichas, las cuales son deslizantes. Esto genera el **árbol de búsqueda de estados** ilustrado en la Figura 6.

En general, los problemas de búsqueda en un espacio de estados tienen el objetivo de a partir de un **estado inicial** $\overrightarrow{x}_1 \in \mathbb{D}^n$, encontrar un **estado solución o meta** $\overrightarrow{x}_{\text{solucion}} \in \mathbb{A}^n$. Alternativamente, la verificación de que el estado es un estado solución, es decir, pertenece al espacio de soluciones $\mathbb{A}^n$, se implementa en la **función de verificación de solución**,

$$g(\overrightarrow{x}) = \begin{cases} 1 & \overrightarrow{x} \in \mathbb{A}^n \\ 0 & \text{sino} \end{cases}$$

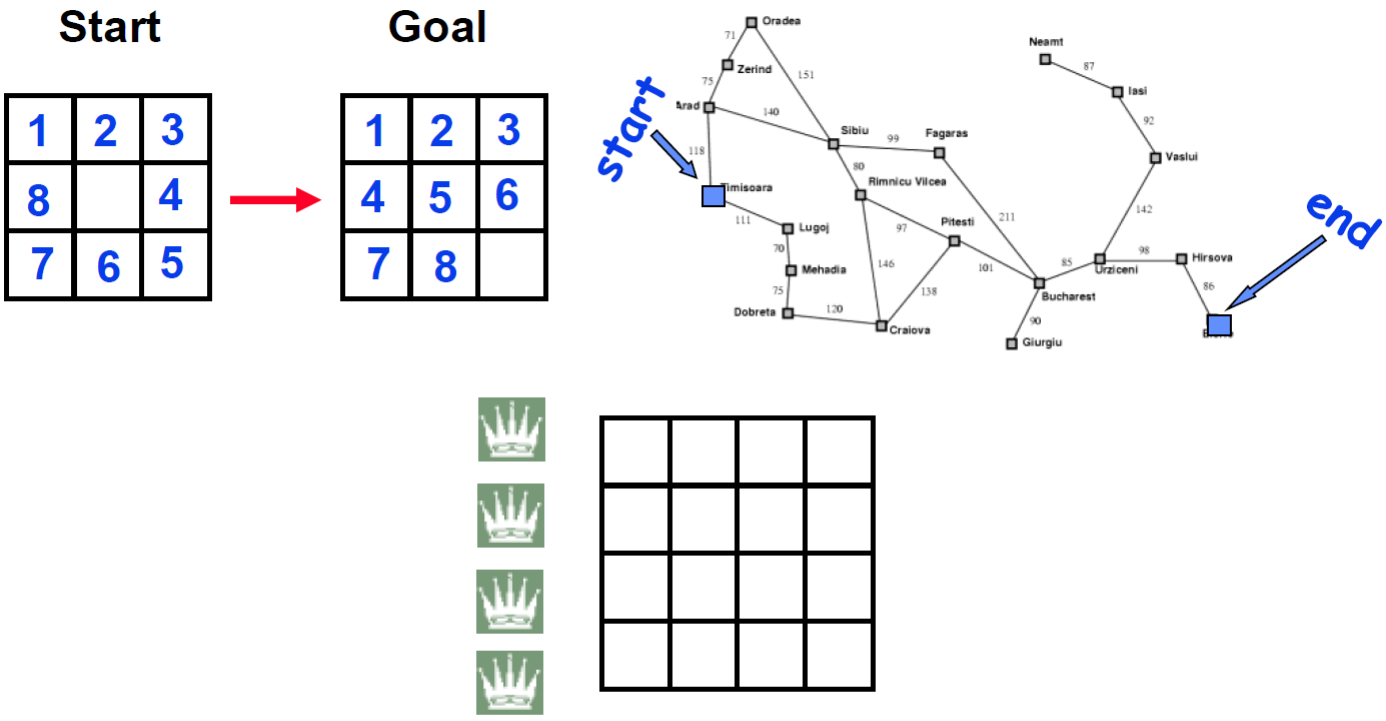

Figura 5: Problemas clásicos en la inteligencia artificial, tomado de *Chapter 3 Problem Solving using Search, CSE AI Faculty*.

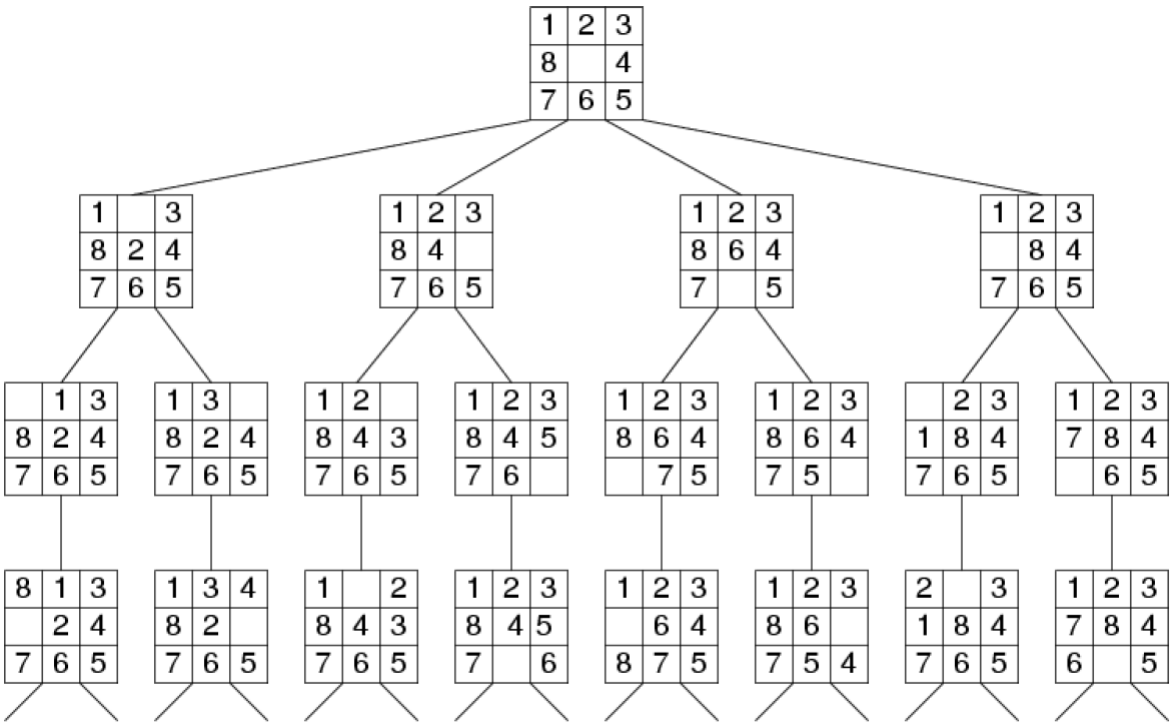

Figura 6: Arbol de búsqueda de estados, para el estado inicial $\overrightarrow{x}_1 = [1, 2, 3, 8, 0, 4, 7, 6, 5]$.

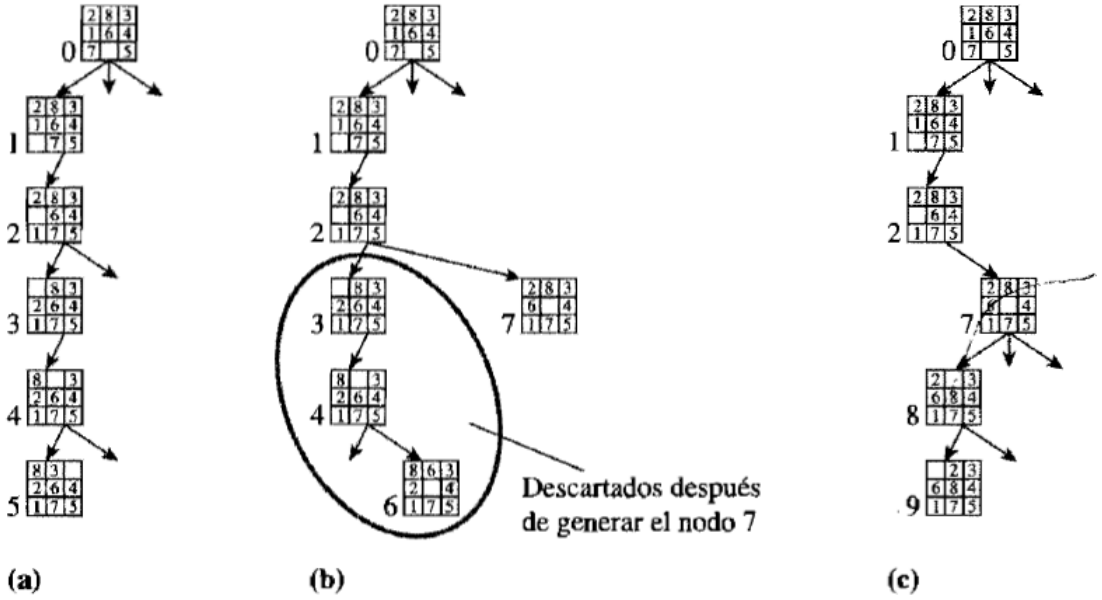

Figura 7: Búsqueda a profundidad primero, con $\tau = 5$.

La generación de estados sucesores de un estado $\overrightarrow{x}(t)$ se implementa en la **función de sucesores,** la cual genera un conjunto de estados sucesores:

$$s(\overrightarrow{x}(t)) = \{\overrightarrow{x}(t+1)_1\ \overrightarrow{x}(t+1)_2, \ldots, \overrightarrow{x}(t+1)_r\}$$

En algunos problemas específicos, la cantidad de aplicaciones de la función de sucesores tiene un costo asociado, por lo que además es neceario minimizar la profundidad del árbol de búsqueda. La función de costo en los problemas de búsqueda está asociada a una **heurística** la cual estima la cercanía del estado solución a partir del estado actual. Las búsquedas sin funciones heurísticas se refieren comúnmente como **búsquedas a ciegas**.

Dos variantes de la búsqueda a ciegas son la **búsqueda a profundidad con vuelta atrás o fuerza bruta** y la **búsqueda en ancho primero.** La búsqueda a profundidad primero explora el primer sucesor desde el estado inicial $\overrightarrow{x}(1)$, de forma similar para los demás sucesores, de forma que el árbol de búsqueda desarrolla una sola rama. Cuándo se encuentra un estado ya visitado, o en caso de no contar con una función de sucesores que genere sólo sucesores correctos, encontrar un estado inválido, el algoritmo sube un nivel en el árbol de búsqueda para visitar el siguiente estado sucesor. Es usual la implementación de una **profundidad máxima** τ, dada la limitación de memoria para guardar los estados anteriores. La Figura 7 ilustra la búsqueda a profundidad primero para el problema del rompecabezas. La búsqueda a fuerza bruta tiene alta demanda de memoria y no garantiza encontrar la ruta más corta en el árbol de búsqueda.

La búsqueda por ancho primero, ejecuta la función de verificación en todos los sucesores, y si en todos los sucesores no se encuentra el estado solución, se aplica la función de sucesor al primer estado según un orden predefinido. La Figura 8 muestra el procedimiento de búsqueda por ancho primero, descartando los estados ya visitados.

3. Algoritmos estocásticos de optimización y búsqueda

Los algoritmos estocásticos de optimización a estudiar inicializan usualmente un punto candidato óptimo de forma aleatoria. A continuación se estudiarán distintas variantes basadas en la diferenciación, evolutivos y de enjambre.

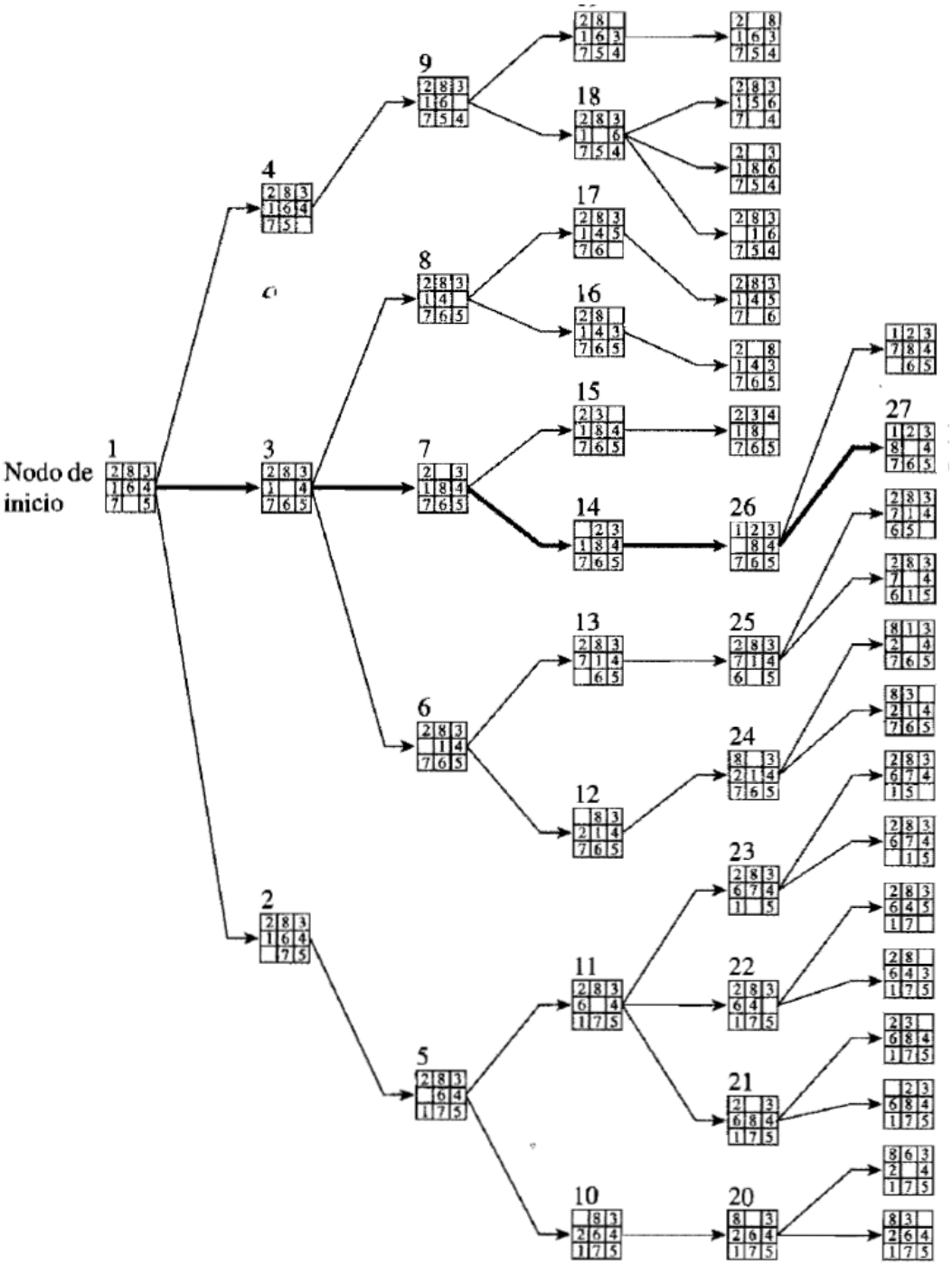

Figura 8: Búsqueda por ancho primero.

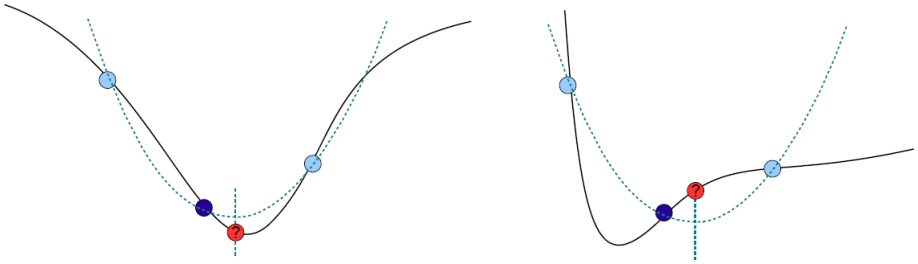

Figura 9: Casos distintos para el algoritmo de las interpolaciones cuadráticas sucesivas.

3.1. Optimización convexa: Método de las interpolaciones cuadráticas sucesivas

El algoritmo de la bisección cuadrática es un algoritmo simple en el cual se tiene por objetivo buscar por ejemplo el valor mínimo de una función convexa $f(x)$, en el intervalo $[a_0, b_0]$. Para ello se biseca tal intervalo por la mitad, haciendo:

$$c_0 = a_0 + \frac{a_0 - b_0}{2}$$

y se calculan los parámetros de la función cuadrática $w_0x^2+y_0x+z_0$ que pase por los tres puntos $f(a_0), f(b_0), f(c_0)$, de forma que debe resolverse el sistema de ecuaciones:

$$\begin{aligned} w_0a_0^2 + y_0a_0 + z_0 &= f(a_0) \\ w_0b_0^2 + y_0b_0 + z_0 &= f(b_0) \\ w_0b_0^2 + y_0b_0 + z_0 &= f(c_0) \end{aligned}$$

Luego de resolverse tal sistema y conocer los valores de los parámetros de la función cuadrática, se calcula de nuevo el corte o bisección c_0', como el valor mínimo de tal ecuación, haciendo:

$$\frac{d}{dx}\left(w_0x^2 + y_0x + z_0\right) = 0$$

despejando x, y haciendo $c_0' = x$. Tal valor mínimo $f(c_0')$ es guardado. Nuevamente se hace una bisección en el intervalo que contenga el punto mínimo, para repetir el proceso anterior. El proceso se detiene cuando el mínimo nuevo encontrado sea mayor al mínimo en la iteración anterior. La Figura 9 muestra el caso donde la *heurística* de trazar una función cuadrática funciona correctamente donde el punto mínimo de la parabólica interpolada es el mejor de los puntos, pero otro en el que no, por lo que es necesario probar un intervalo distinto.

El algoritmo no garantiza su convergencia, pero si converge, lo hace relativamente rápido, sin necesidad de calcular gradientes para la función a optimizar. Usualmente para garantizar su convergencia, se combina con el algoritmo *golden rate.*

3.2. Optimización no convexa basdada en la diferenciación: Algoritmo de descenso de gradiente

El algoritmo del descenso de gradiente sigue la idea de modificar el punto óptimo estimado de forma iterativa. Para una función en una variable $f(x)$, la estimación del punto óptimo en una iteración $i+1$ está dada por:

$$x(t+1) = x(t) + \alpha f'(x(t))$$

donde el coeficiente α determina el *grado de confianza o velocidad* con la que el proceso de optimización iterativa sigue la dirección de la derivada. Para la optimización de una función multivariable $f(\overrightarrow{x}(t))$ con $\overrightarrow{x} \in \mathbb{R}^n$, la posición óptima se estima usando el vector gradiente:

$$\overrightarrow{x}(t+1) = \overrightarrow{x}(t) + \alpha \nabla_{\overrightarrow{x}} f(\overrightarrow{x}(t))$$

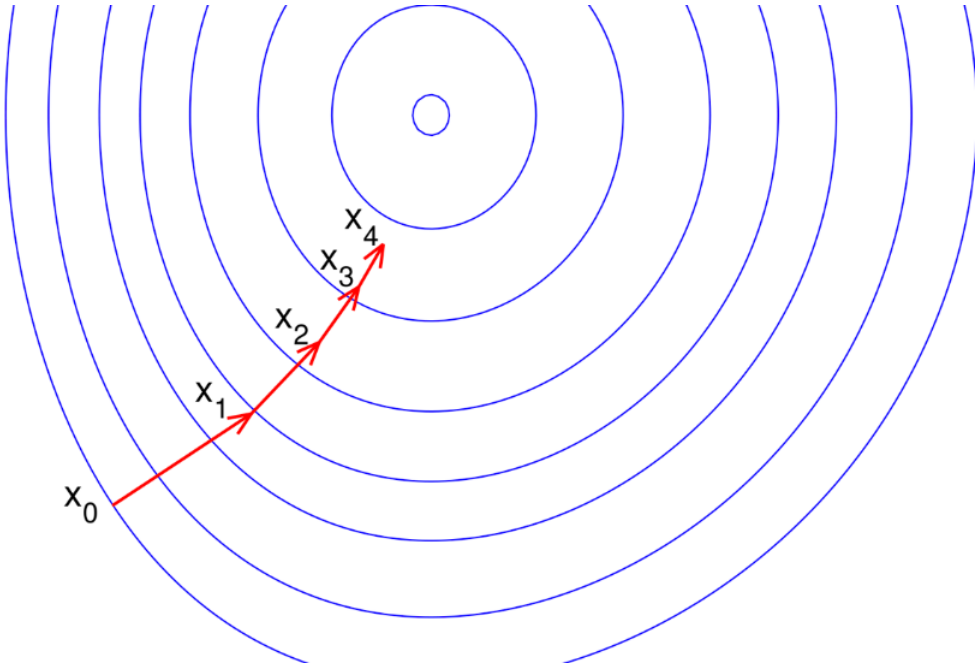

Figura 10: Proceso de optimización con el descenso de gradiente.

La Figura 10 muestra los pasos rectilíneos del algoritmo de descenso de gradiente, el cual sigue una dirección *recta.*

3.2.1. Ejemplo

Tómese una función con dominio $\overrightarrow{x} \in \mathbb{R}^2$,

$$f\left(\overrightarrow{x}\right) = x_1^2 - x_2^2,$$

la cual se grafica en la Figura 11.

Su gradiente se obtiene al hacer:

$$\nabla_{\overrightarrow{x}} f = \begin{bmatrix} 2x_1 \\ -2x_2 \end{bmatrix}$$

Para un punto $\overrightarrow{x}\left(t\right)$ en la iteración $t = 1$ elegido aleatoriamente, se actualiza la estimación del punto óptimo de la siguiente forma:

$$\overrightarrow{x}\left(t+1\right) = \overrightarrow{x}\left(t\right) + \alpha \nabla_{\overrightarrow{x}} f\left(\overrightarrow{x}\left(t\right)\right) = \begin{bmatrix} x_1\left(t\right) \\ x_2\left(t\right) \end{bmatrix} - \alpha \begin{bmatrix} 2x_1\left(t\right) \\ -2x_2\left(t\right) \end{bmatrix} = \begin{bmatrix} x_1\left(t\right) - \alpha 2x_1\left(t\right) \\ x_2\left(t\right) + 2x_2\left(t\right) \end{bmatrix}$$

Suponiendo por ejemplo que $\alpha = 0{,}25$ y que $\overrightarrow{x}\left(1\right) = \begin{bmatrix} 200 \\ 0 \end{bmatrix}$, en tal caso $\overrightarrow{x}\left(2\right) = \begin{bmatrix} 100 \\ 0 \end{bmatrix}$, por lo que son necesarias más iteraciones para converger en un punto mínimo satisfactorio. Sin embargo observe que si el coeficiente fuera más alto, por ejemplo $\alpha = 0{,}5$, lo acerca a un punto más satisfactorio, pero aún no el mínimo $\overrightarrow{x}\left(2\right) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, y un coeficiente más alto $\alpha = 1$ hace que «brinquemos» el punto silla.

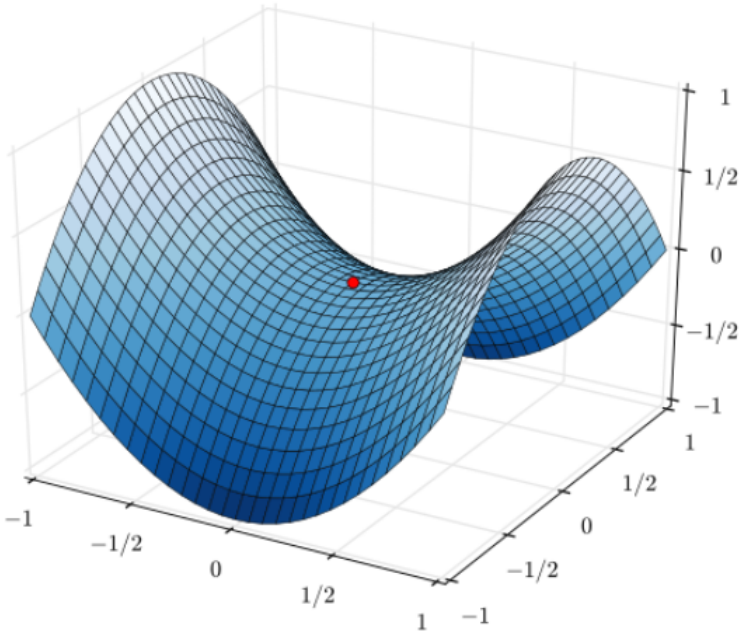

Figura 11: Función $f\left(\overrightarrow{x}\right) = x_1^2 - x_2^2$.

3.3. Variantes del algoritmo del descenso del gradiente: Descenso de gradiente con momentum

Tómese la siguiente función multivariable con dominio $\overrightarrow{x} \in \mathbb{R}^2$,

$$f\left(\overrightarrow{x}\right) = 0{,}1x_1^2 + 2x_2^2,$$

graficada en la Figura 12.

Al calcular el vector gradiente $\nabla_{\overrightarrow{x}} f = \begin{bmatrix} 0{,}2x_1 \\ 4x_2 \end{bmatrix}$, podemos notar que el desplazamiento será siempre más fuerte hacia la dimensión x_2. Esto ocasionará «saltos» más fuertes en tal dimensión, sobre todo con coeficientes de actualización α altos, como se ilustra en la Figura 13.

Los saltos hacen que la trayectoria necesite de más puntos para arribar al óptimo, por lo que es últil la idea de incorporar una **inercia** que reduzca las oscilaciones de grandes magnitudes en las dimensiones de mayor decaimiento. La **inercia o momentum** se expresa usualmente como un término de velocidad $\overrightarrow{v}\left(t\right)$, la cual se puede actualizar, de forma simple, como sigue:

$$\overrightarrow{v}\left(t\right) = \gamma\overrightarrow{v}\left(t-1\right) + \alpha\nabla_{\overrightarrow{x}} f\left(\overrightarrow{x}\left(t\right)\right)$$

$$\overrightarrow{x}\left(t+1\right) = \overrightarrow{x}\left(t\right) - \overrightarrow{v}\left(t\right)$$

donde el coeficiente γ controla la cantidad de momentum a utilizar en la actualización de la solución. La Figura 14 muestra como el recorrido del proceso de optimización al usar el descenso de gradiente con moméntum, el cual tiene menos oscilaciones (es más suave), y tarda menos en converger, al moverse más rápido en la dimensión x_1, para una superficie similar a la de la función $f\left(\overrightarrow{x}\right)$.

Usualmente se implementa la ecuación de actualización de la inercia, usando lo que se conoce como el **ponderado móvil exponencial,** el cual básicamente trata de mantener la magnitud de la velocidad independiente del coeficiente de moméntum γ, haciendo que:

$$\overrightarrow{v}\left(t\right) = \gamma\overrightarrow{v}\left(t-1\right) + \alpha\left(1-\gamma\right)\nabla_{\overrightarrow{x}} f\left(\overrightarrow{x}\left(t\right)\right)$$

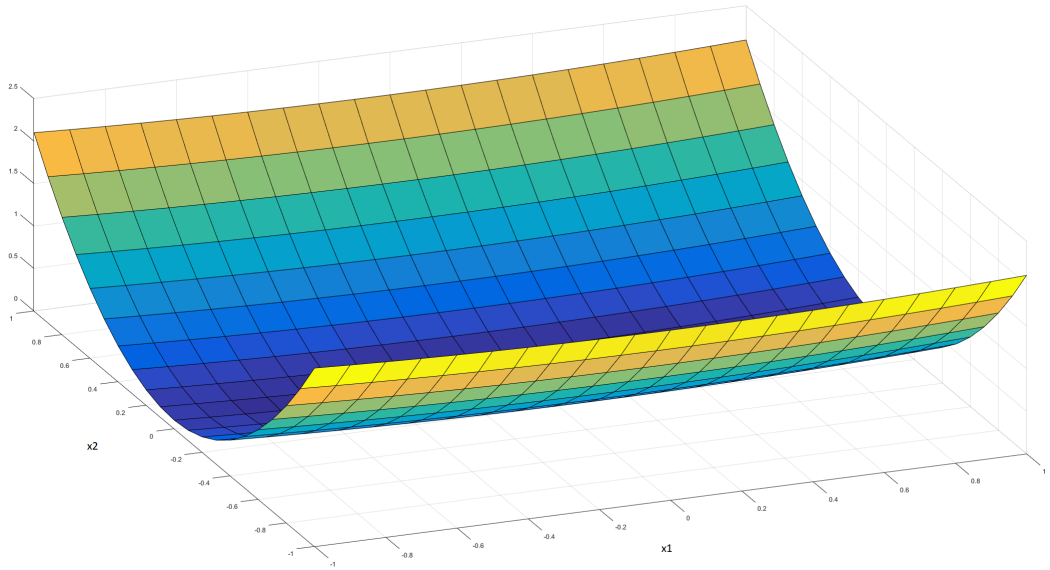

Figura 12: Superficie de $f\left(\overrightarrow{x}\right) = 0{,}1x_1^2 + 2x_2^2$.

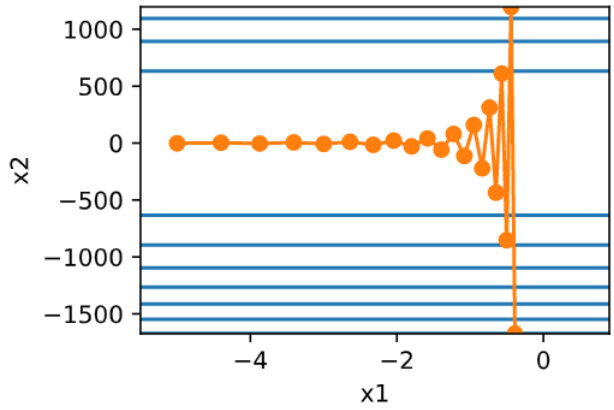

Figura 13: Distintos puntos en el recorrido de la optimización usando el descenso de gradiente, tomado de `http://d2l.ai/chapter_optimization/momentum.html`.

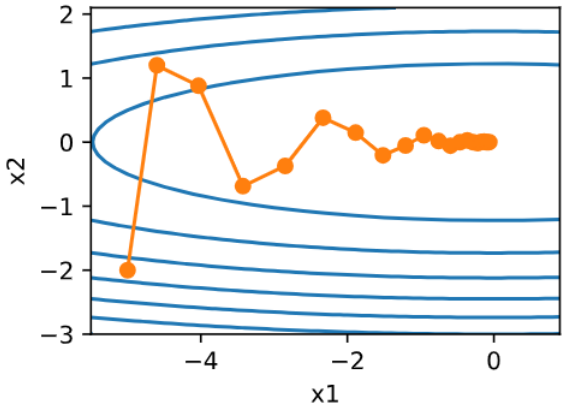

Figura 14: Recorrido en la optimización, usando el descenso de gradiente con moméntum, tomado de `http://d2l.ai/chapter_optimization/momentum.html`.

con lo cual al expandir tal **relación recurrente** obtenemos que:

$$\begin{aligned}
\overrightarrow{v}(t) &= \gamma\left(\gamma\overrightarrow{v}(t-2)+\alpha(1-\gamma)\nabla_{\overrightarrow{x}}f\left(\overrightarrow{x}(t-1)\right)\right)+\alpha(1-\gamma)\nabla_{\overrightarrow{x}}f\left(\overrightarrow{x}(t)\right)\\
&= \gamma^2\overrightarrow{v}(t-2)+\gamma\alpha(1-\gamma)\nabla_{\overrightarrow{x}}f\left(\overrightarrow{x}(t-1)\right)+\alpha(1-\gamma)\nabla_{\overrightarrow{x}}f\left(\overrightarrow{x}(t)\right)\\
= \gamma^2\left(\gamma\overrightarrow{v}(t-3)+\alpha(1-\gamma)\nabla_{\overrightarrow{x}}f\nabla_{\overrightarrow{x}}f\left(\overrightarrow{x}(t-2)\right)\right)&+\gamma\alpha(1-\gamma)\nabla_{\overrightarrow{x}}f\left(\overrightarrow{x}(t-1)\right)+\alpha(1-\gamma)\nabla_{\overrightarrow{x}}f\left(\overrightarrow{x}(t)\right)\\
&= \gamma^3\overrightarrow{v}(t-3)+\gamma^2\alpha(1-\gamma)\nabla_{\overrightarrow{x}}f\left(\overrightarrow{x}(t-2)\right)+\gamma\alpha(1-\gamma)\nabla_{\overrightarrow{x}}f\left(\overrightarrow{x}(t-1)\right)+\alpha(1-\gamma)\nabla_{\overrightarrow{x}}f\left(\overrightarrow{x}(t)\right)
\end{aligned}$$

Con lo cual se observa que términos más antiguos, reciben un peso exponencialmente menor, por lo cual se le conoce como **pesado exponencial móvil**.

3.4. Variantes del algoritmo del descenso del gradiente: Descenso de gradiente adaptativo o AdaGrad y Propagación de la raíz media

Otro enfoque para solucionar el problema de distintas magnitudes de gradiente a distintas direcciones, y sus consecuentes oscilaciones en el espacio de búsqueda, es el propuesto en el artículo *Duchi, John; Hazan, Elad; Singer, Yoram (2011). "Adaptive subgradient methods for online learning and stochastic optimization"*. La técnica conocida popularmente como AdaGrad, propone utilizar un coeficiente de aprendizaje particular por cada dimensión

$$\overrightarrow{\alpha} = \begin{bmatrix}\alpha_1\\ \vdots\\ \alpha_n\end{bmatrix},$$

$$\overrightarrow{x}(t+1) = \overrightarrow{x}(t) - \overrightarrow{\alpha}.*\nabla_{\overrightarrow{x}}f\left(\overrightarrow{x}(t)\right),$$

donde el operador $.*$ multiplica los vectores por elemento (*element-wise*), a diferencia del algoritmo del descenso de gradiente original, que utiliza un sólo coeficiente de aprendizaje escalar, lo que implica el uso del mismo valor en todas las dimensiones $\overrightarrow{x}(t+1) = \overrightarrow{x}(t) - \alpha\nabla_{\overrightarrow{x}}f\left(\overrightarrow{x}(t)\right)$. Los elementos del vector $\overrightarrow{\alpha}$, AdaGrad propone calcularlos adaptativamente, normalizando el coeficiente de aprendizaje para la dimensión i, por la sumatoria de todas las derivadas parciales evaluadas en cada punto $\overrightarrow{x}(t)$ hasta la iteración actual t:

$$\alpha_i = \frac{\rho}{\sqrt{s_i(t)+\epsilon}}$$

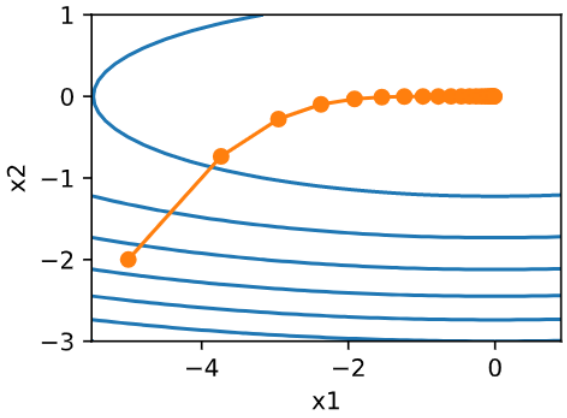

Figura 15: Algoritmo RMS Prop, tomado de `http://d2l.ai/chapter_optimization/momentum.html`.

con un valor pequeño ϵ para evitar las divisiones por cero (estabilidad numérica). El término $s_i(\tau)$ es el que contiene las magnitudes de las derivadas parciales en la solución candidata actual:

$$s_i(t) = \sum_{k=1}^{t} \left(\frac{df(\overrightarrow{x}(k))}{dx_i} \right)^2$$

El coeficiente ρ controla el aprendizaje global del algoritmo. De esta forma, si un componente del gradiente tiene una alta magnitud, Una desventaja del método AdaGrad es que cada coeficiente α_i tiende a hacerse más chico conforme pasan más iteraciones, al dar igual peso a los valores de la derivada parcial anteriores.

El algoritmo de ***Root Mean Square Propagation (RMS Prop)***, introducido en Tieleman, *Tijmen and Hinton, Geoffrey (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude, coursera*, propone entonces usar el **pesado exponencial móvil,** con lo que entonces, se calcula el término $s_i(\tau)$ de la siguiente forma:

$$s_i(t) = \gamma s_i(t-1) + (1-\gamma) \sum_{t=1}^{\tau} \left(\frac{df(\overrightarrow{x}(k))}{dx_i} \right)^2$$

La Figura 15 muestra los puntos óptimos candidatos recorridos por el algoritmo RMS Prop, para una superficie cuadrática.

3.5. Optimización no convexa basada en la diferenciación: Algoritmo de Newton-Raphson

Antes de discutir el método de Newton-Raphson, es necesario repasar el concepto de series, específicamente, las series de Taylor.

3.5.1. Series de Taylor

Para una función $f(x)$, las series de Taylor permiten aproximar a una función alrededor de un punto de operación a de la siguiente forma:

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)(x-a)^k}{k!} = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \dots$$

Con lo que entonces la aproximación de primer orden o linear alrededor del punto a toma sólo dos términos:

$$f(x) \approx f(a) + f'(a)(x-a).$$

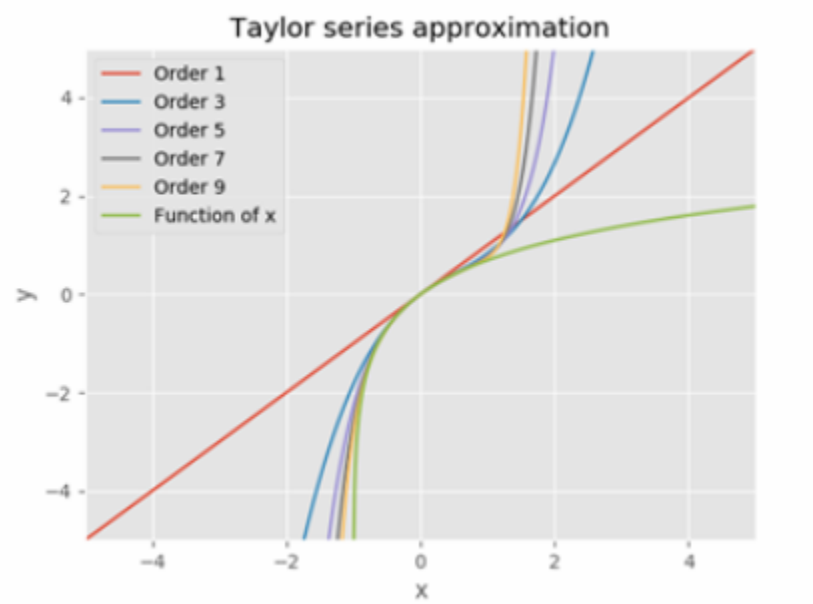

Figure 16: Aproximación por series de Taylor alrededor del punto de operación $x = 0$.

Entre más alejados estén los valores de x respecto al punto a, mayor será la diferencia entre la función real y la aproximación por series de Taylor. Para una función multivariable con $\vec{x} = [x_1, x_2, \ldots, x_n]$ la aproximación de Taylor de primer orden alrededor de un punto $\overrightarrow{a} = [a_1, a_2, \ldots, a_n]^T$ está dada por:

$$f(\vec{x}) = f(\overrightarrow{a}) + \frac{\partial f}{\partial x_1}(x_1 - a_1) + \frac{\partial f}{\partial x_2}(x_2 - a_2) + \ldots + \frac{\partial f}{\partial x_n}(x_n - a_n).$$

La Figura 16 muestra la aproximación de Taylor de distintos órdenes alrededor del punto de operación $x = 0$.

3.5.2. El método de Newton-Raphson

El método de Newton-Raphson fue **diseñado originalmente para encontrar la raíz de ecuaciones no lineales igualadas a cero**:

$$f(x) = 0$$

y para el caso de una función multivariable, $f(\overrightarrow{x}) = 0$. Su extensión para el problema de encontrar un óptimo en tal función, viene dado al usar el algoritmo para encontrar la raíz de la primera derivada, resolviendo entonces $f'(\overrightarrow{x}) = 0$ en general.

El algoritmo de Newton-Raphson es iterativo, de forma que a partir de un punto inicial $x(1)$, calcula una **secuencia de valores** $x(1), x(2), x(3), \ldots, x(p)$, con $x(p) \to x_{\text{opt}}$. Por cada iteración i, el punto óptimo estimado se calcula como sigue:

$$x(t+1) = x(t) + \Delta x$$

Para obtener el valor de Δx se descompone la función $f(x)$ **utilizando una aproximación por series de Taylor, para hallar el Δx óptimo igualando tal aproximación de Taylor para igualarla a cero**. Recordemos que la serie de Taylor está dada en general por:

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots$$

en este caso de segundo grado, con $a = x(t)$ como punto de operación, sobre el intervalo $[x_i, x_i + \Delta x]$:

$$f(x(t)) \approx f(x(t)) + f'(x(t))((x(t) + \Delta x) - x(t)) + \frac{f''(x(t))}{2}((x(t) + \Delta x) - x(t))^2 = f(x(t)) + f'(x(t))\Delta x + \frac{f''(x(t))}{2}\Delta x^2$$

Observe que esta aproximación es sólo válida en tal intervalo. **Derivando respecto al Δx e igualando a cero, en búsqueda del punto óptimo**, se tiene que:

$$\frac{d}{d\Delta x}\left(f(x(t)) + f'(x(t))(\Delta x) + \frac{f''(x(t))}{2}\Delta x^2\right) = 0$$

$$\Rightarrow f'(x(t)) + f''(x(t))\Delta x = 0 \Rightarrow \Delta x = -\frac{f'(x(t))}{f''(x(t))}$$

Al realizar este cálculo, hallaremos el Δx que se acerque más a la solución de $f'(x) = 0$. Para el caso de la optimización de una función multivariable $f(\overrightarrow{x}(t))$ con $\overrightarrow{x}(t) \in \mathbb{R}^{n\times 1}$, el cambio en el vector $\overrightarrow{x}_i$ viene dado en términos de la matriz Hessiana $H_{\overrightarrow{x}} f(\overrightarrow{x}(t)) \in \mathbb{R}^{n\times n}$evaluada en tal punto $\overrightarrow{x}(t)$:

$$\Delta\overrightarrow{x} = -\left[H_{\overrightarrow{x}} f(\overrightarrow{x}(t))\right]^{-1} \nabla_{\overrightarrow{x}} f(\overrightarrow{x}(t))$$

donde el vector gradiente tiene dimensiones $\nabla_{\overrightarrow{x}} f(\overrightarrow{x}(t)) \in \mathbb{R}^{n\times 1}$, lo cual hace que $\Delta\overrightarrow{x} \in \mathbb{R}^{n\times 1}$. La ecuación de actualización del vector viene entonces dada por:

$$\overrightarrow{x}(t+1) = \overrightarrow{x}(t) + \Delta\overrightarrow{x}$$

3.5.3. Ejemplo

Tómese una función con dominio $\overrightarrow{x} \in \mathbb{R}^2$,

$$f(\overrightarrow{x}) = x_1^2 - x_2^2,$$

la cual se grafica en la Figura 17.

Formalmente, para encontrar el punto mínimo en una función usando el método de Newton-Raphson, se utiliza la aproximación por series de Taylor del gradiente para hacer $g = \nabla_{\overrightarrow{x}}(f) = 0$. Sin embargo, esto involucraría derivar dos veces más la función g, la cuál ya es vectorial, con lo que se complica el cómputo. Es por ello que nos quedaremos con la búsqueda con el punto $f(\overrightarrow{x}) = 0$, lo cuál nos permite estimar el Δx haciendo:

$$\Delta\overrightarrow{x} = -\left[H_{\overrightarrow{x}} f(\overrightarrow{x}(t))\right]^{-1} \nabla_{\overrightarrow{x}} f(\overrightarrow{x}(t))$$

para por cada iteración actualizar el punto óptimo estimado haciendo $\overrightarrow{x}(t+1) = \overrightarrow{x}(t) + \Delta\overrightarrow{x}$. Calculando ambos factores se obtiene:

$$\nabla_{\overrightarrow{x}} f = \begin{bmatrix} 2x_1 \\ -2x_2 \end{bmatrix}$$

y la matriz Hessiana está dada por:

$$H_{\overrightarrow{x}} = \begin{bmatrix} \frac{df}{dx_1^2} & \frac{df}{dx_1 dx_2} \\ \frac{df}{dx_2 dx_1} & \frac{df}{dx_2^2} \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & -2 \end{bmatrix}$$

y su inversa por $H_{\overrightarrow{x}}^{-1} = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & -\frac{1}{2} \end{bmatrix}$. Para un punto cualquiera en la iteración $t = 1$, se tiene que:

$$\Delta\overrightarrow{x} = -\left[H_{\overrightarrow{x}} f(\overrightarrow{x}(t))\right]^{-1} \nabla_{\overrightarrow{x}} f(\overrightarrow{x}(t)) = -\begin{bmatrix} \frac{1}{2} & 0 \\ 0 & -\frac{1}{2} \end{bmatrix}\begin{bmatrix} 2x_1(t) \\ -2x_2(t) \end{bmatrix} = \begin{bmatrix} -x_1(t) \\ -x_2(t) \end{bmatrix} = -\overrightarrow{x}(t)$$

y al hacer la actualización

$$\overrightarrow{x}(t+1) = \overrightarrow{x}(t) + \Delta\overrightarrow{x} \Rightarrow \overrightarrow{x}(t+1) = \overrightarrow{x}(t) - \overrightarrow{x}(t) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Lo cual significa que el algoritmo ha arribado al **punto silla,** el cual en este caso no es el óptimo, pero lo hizo en una iteración sin importar el punto inicial $\overrightarrow{x}(t)$. Las funciones de error usualmente no tienen partes negativas, pero si **muchos puntos silla**.

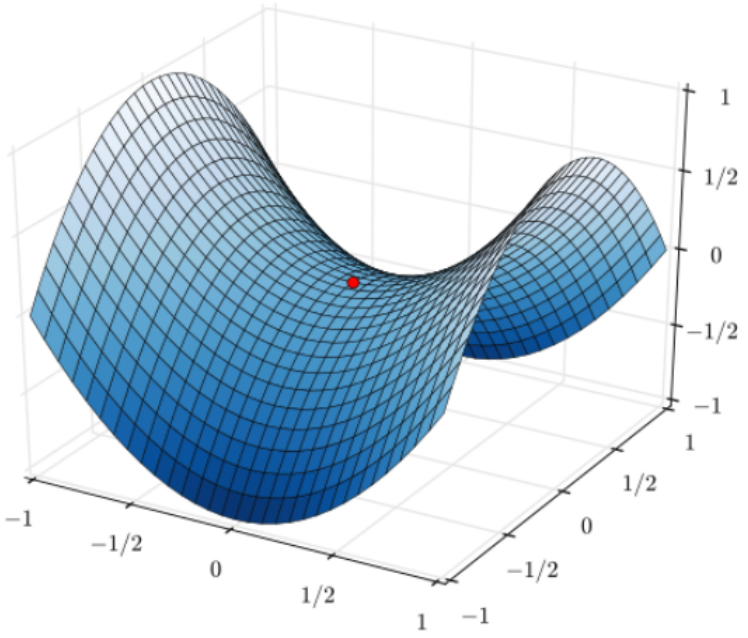

Figura 17: Función $f\left(\overrightarrow{x}\right) = x_1^2 - x_2^2$.

3.6. Algoritmo de optimización no convexo: *Simulated Annealing*

El recocido de material o *annealing* se refiere al **calentamiento o temperado** de ciertas aleaciones de metal, vidrio o cristal calentando por encima de su punto de fusión, manteniendo su temperatura y luego enfriándola muy lentamente hasta que se solidifica en una estructura cristalina. El recocido simulado o *simulated annealing* (SA) se inspira en este proceso físico [3]. Siguiendo tal analogía, el estado físico del material corresponde al punto actual $\overrightarrow{x}\left(t\right)$. La **energía total** en un estado corresponde a la evaluación de la función de costo o función a optimizar, por lo que entonces

$$E\left(\overrightarrow{x}\left(t\right)\right) = f\left(\overrightarrow{x}\left(t\right)\right),$$

para un tiempo t.

La **probabilidad de un cambio de estado** en el tiempo t está determinada por la distribución de Boltzmann de la **diferencia de energía de los dos estados**:

$$P\left(t\right) = \begin{cases} e^{\frac{-\Delta E}{T(t)}} & \Delta E > 0 \\ 1 & \Delta E \leq 0 \end{cases},$$

lo cual indica que si la energía es mayor con el nuevo estado $\overrightarrow{x}\left(t\right) + \Delta\overrightarrow{x}\left(t\right)$, el cambio a tal nuevo estado es 100 % seguro.

$T\left(t\right)$ corresponde a la **temperatura actual** y controla la magnitud de las perturbaciones de la función de energía $E\left(\overrightarrow{x}\left(t\right)\right)$.

$$\Delta E = E\left(\overrightarrow{x}\left(t\right)\right) - E\left(\overrightarrow{x}\left(t\right) + \Delta\overrightarrow{x}\left(t\right)\right)$$

está asociado al cambio en la función de energía, al modificar el estado en $\Delta\overrightarrow{x}\left(t\right)$ para generar lo que se denomina una **solución vecina** de $\overrightarrow{x}\left(t\right)$.

El cambio de solución candidato $\Delta\overrightarrow{x}\left(t\right)$ se puede definir mediante **dos enfoques**: **seleccionar un elemento al azar de todo el espacio de búsqueda o usar un subespacio dentro del espacio de búsqueda**. Se aconseja

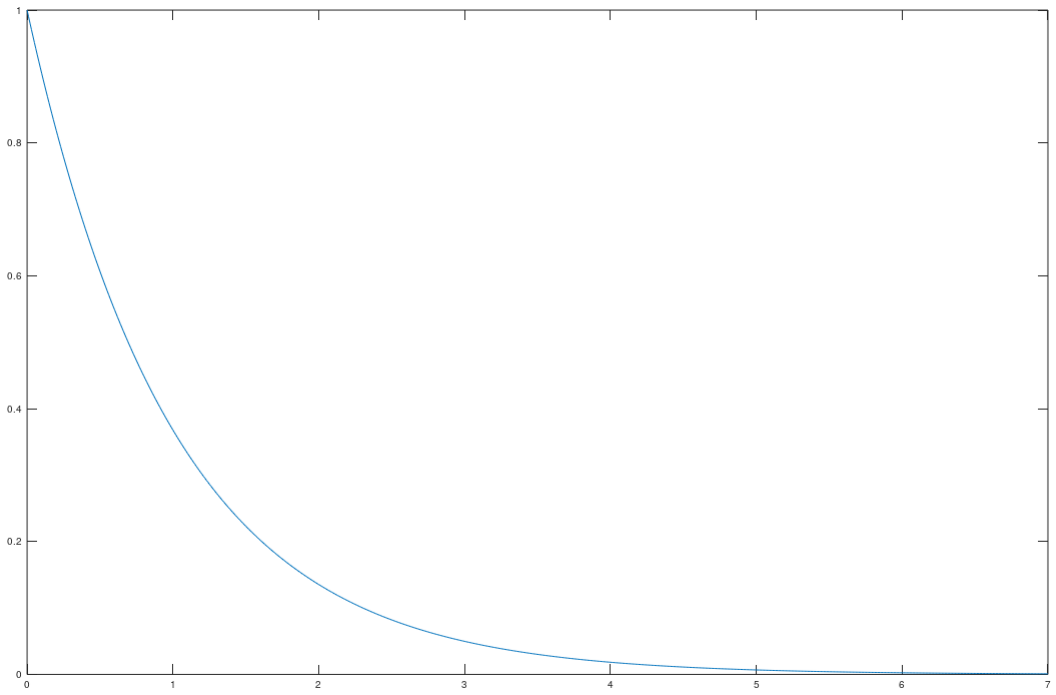

Figura 18: Función de densidad de probabiliad para el cambio de estado.

definir un subespacio para realizar el cambio $\Delta\overrightarrow{x}(t)$, con alrededor del $\pm 30\,\%$ del espacio de búsqueda total para aleatoriamente definir el vector de cambio $\Delta\overrightarrow{x}(t)$, usualmente con una distribución uniforme.

La probabilidad de movimientos es grande con una temperatura T alta, y es menor en valores más bajos de temperatura T, o más *frío* es el cuerpo. Observe además que si ΔE es alto (es decir, el estado anterior tiene considerablemente mayor energía), la probabilidad de cambio es baja para un mismo T, como se ilustra en la gráfica de la función exponencial, en la Figura 18. De darse un cambio en el estado, se actualiza el estado para la nueva iteración:

$$\overrightarrow{x}(t+1) = \overrightarrow{x}(t) + \Delta\overrightarrow{x}(t)$$

La **temperatura decrece conforme el tiempo**, por lo que se puede utilizar un descenso logarítmico. Sin embargo, este descenso es muchas veces muy lento, especialmente para funciones de energía con costos computacionales altos, por lo que en la literatura es frecuente la implementación de un descenso lineal como el siguiente:

$$T(t) = \alpha T(t-1)$$

con $0{,}85 \leq \alpha \leq 0{,}96$. Por cada unidad de tiempo t es posible computar τ iteraciones de cambio de estado, con una misma temperatura $T(t)$, según [2], cuanto más τ iteraciones se ejecuten, mayor probabilidad del algoritmo de converger a un máximo global.

3.7. Algoritmos de enjambre: El algoritmo de enjambre de partículas

El algoritmo de enjambre de partículas propone inicializar un conjunto X de n **soluciones candidatas** o partículas:

$$X = \{x_1, x_2, \ldots, x_n\}.$$

A tal conjunto de soluciones candidatas se le llama **enjambre**. Para toda partícula $x_i(t)$ se realiza un cambio en su posición en el tiempo o *repetición del algoritmo* t, según la siguiente ecuación:

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

donde la función $v_i\,(t+1)$ se refiere a la velocidad de la partícula i en la repetición o iteración $t+1$. El vector de velocidad define el cambio que tendrá la posición de la partícula i, y se define según la siguiente ecuación:

$$v_i\,(t+1) = v_i\,(t) + c_1\,(p_i\,(t) - x_i\,(t))\,r_1 + c_2\,(g\,(t) - x_i\,(t))\,r_2,$$

donde se distinguen tres términos:

1. La influencia de la velocidad en la iteración anterior o inercia para la partícula i, correspondiente a la velocidad en el tiempo actual $v_i\,(t)$ para la partícula i.
2. El término de aprendizaje individual o componente cognitivo define el aporte del conocimiento individual de la partícula en el cambio de su velocidad, en el cual distinguen los siguientes componentes:
   a) $p_i\,(t)$: El valor de x para el cual se ha obtenido el mejor puntaje de la partícula i hasta el tiempo t. Se calcula la diferencia con el valor actual de x para la partícula i.
   b) c_1: **El coeficiente de peso al componente cognitivo**, es un valor que asigna el peso al componente cognitivo, y es definido por el usuario, recomendado por la literatura con el valor $c_1 = 2$.
   c) r_1: Coeficiente aleatorio de peso al componente cognitivo: Es un valor aleatorio $0 \leq r_1 \leq 1$.
3. El término de aprendizaje colectivo o componente social: identifica la tendencia de la partícula de seguir la mejor posición en todo el enjambre. El término está compuesto por los siguientes valores:
   a) $g\,(t)$: El valor de x para el cual se ha obtenido el mejor puntaje en todas la partículas hasta el tiempo t. Se calcula la diferencia con el valor actual de x para la partícula i.
   b) c_2: El **coeficiente de peso al componente social**, de forma similar a c_1, es definido por el usuario, recomendado por la literatura con el valor $c_2 = 2$.
   c) r_2: Coeficiente aleatorio de peso al componente social: Valor aleatorio $0 \leq r_2 \leq 1$.

La inicialización de las partículas para la primera iteración

$$X = \{x_1(0), x_2(0), \ldots, x_n(0)\}\,.$$

se realiza de forma aleatoria entre los M puntos de la función $f[x]$ a optimizar. La velocidad inicial de cada partícula $v_1(0), v_2(0), \ldots, v_n(0)$ se inicializa también de forma aleatoria, con valores entre 0 y 1, de forma que $0 \leq v_i(0) \leq 1$.

3.8. Algoritmos evolutivos: Algoritmos genéticos

Un algoritmo genético es un enfoque evolutivo que permite la optimización de problemas de caja negra no convexos, con poco conocimiento sobre sus propiedades matemáticas. Los algoritmos genéticos imitan la forma en que los organismos evolucionan y se adaptan a las condiciones externas en un entorno: los individuos se evalúan mediante una **función de aptitud**, correspondiente a la función de costo $f\,(\overrightarrow{x}\,(t))$ en un bucle finito que implementa operaciones de selección, cruce y mutación de los individuos.

De manera similar al enfoque de PSO, un individuo i es un vector de dimensión $\overrightarrow{x}_i\,(t) \in \mathbb{R}^n$ y representa una solución candidata para ser evaluada en la función de aptitud. Cada uno de los componentes del vector que conforma el individuo se le denomina **cromosoma**. Una generación t se define como un conjunto de individuos $X(t) = \{\overrightarrow{x}_1\,(t)\,, \overrightarrow{x}_2\,(t)\,, ..., \overrightarrow{x}_p\,(t)\}$. Cada individuo se inicializa aleatoriamente, en un subespacio definido $\overrightarrow{x}_i\,(t) \in \mathbb{D}^n$. Tal población sin puntuación que pertenece a la generación cero, $X(0)$.

Para una generación t se realizan los siguientes pasos:

1. Se evalúa la puntuación o aptitud de cada individuo $y_i = f\,(\overrightarrow{x}_i\,(t))$, y se almacena el mejor individuo encontrado hasta ahora según tal función de aptitud $\overrightarrow{x}_{\text{mejor}}$. Usualmente las puntuaciones son normalizadas para facilitar la selección de hiperparámetros

2. Se seleccionan los mejores individuos según el **hiperparámetro** referido como **umbral de selección** τ, de modo que todos los individuos seleccionados pertenencen al conjunto

$$X_{\text{seleccionados}}(t) = \{\overrightarrow{x}_1(t), \overrightarrow{x}_2(t), ..., \overrightarrow{x}_s(t)\}, f(\overrightarrow{x}_i(t)) > \tau$$

.

3. La creación de la población en la iteración $t+1$ se realiza al aplicar la **función de cruce** $c(\overrightarrow{x}_a(t), \overrightarrow{x}_b(t))$, donde $\overrightarrow{x}_a(t), \overrightarrow{x}_b(t) \in X_{\text{seleccionados}}(t)$. Tal función de cruce puede considerarse como otro hiperparámetro del modelo, además de la política de la selección de los individuos a cruzar. Un ejemplo de operador de cruce es el cruce ponderado, el cual toma un mayor porcentaje del individuo con mayor aptitud. Para ello se calculan los pesos de cruce para cada individuo, de la siguiente forma:

$$w_a(t) = \frac{f(\overrightarrow{x}_a(t))}{f(\overrightarrow{x}_a(t)) + f(\overrightarrow{x}_b(t))} \qquad w_b(t) = \frac{f(\overrightarrow{x}_b(t))}{f(\overrightarrow{x}_a(t)) + f(\overrightarrow{x}_b(t))}$$

de forma que el descendiente de tales padres, es calculado de la siguiente forma:

$$\overrightarrow{x}_i(t+1) = w_a(t)\overrightarrow{x}_a(t) + w_b(t)\overrightarrow{x}_b(t).$$

Esta combinación lineal conserva las características del padre más apto. Existen otros operadores de cruce, como el cruce a nivel de bits y de puntos múltiples. Además, para crear nuevos individuos en la siguiente iteración, se implementa el operador de mutación, el cual se aplica según el **hiperparámetro de probabilidad de mutación** p_m. Por cada individuo i en la población $X(t)$ se calcula un número real entre cero y uno. Si el factor es menor o igual que la probabilidad de mutación, se produce una mutación utilizando un enfoque de cambio de bit aleatorio, que altera un cromosoma en el individuo de forma aleatoria, con distintas políticas implementables. El operador de mutación tiene por objetivo aumentar el espacio explorado por el algoritmo.

4. Ejemplo de definición de un espacio de búsqueda: Segmentación de imágenes usando contornos activos

A continuación se presenta un problema de optimización para su estudio.

Los algoritmos de segmentación de imágenes tienen por objetivo distinguir la región correspondiente a un objeto de interés (foreground), y por ende encontrar la región de fondo (background). Los algoritmos más complejos de detección, conteo de objetos y rastreo de objetos usualmente necesitan implementar una etapa de segmentación previa [1, 4].

Existen distintos enfoques para implementar la segmentación. Por ejemplo, enfoques de aprendizaje automático que realizan una clasificación supervisada (usando redes neuronales, Bayesianas, etc) para clasificar los pixeles en dos clases (fondo y no fondo) o no supervisados (no necesitan de muestras de entrenamiento), los cuales usan algoritmos como K-medias o maximización de la esperanza, para encontrar autonomamente las etiquetas óptimas para los pixeles.

Otro enfoque es el basado en la umbralización, con algoritmos como el de Kittler u Otsu, los cuales buscan el umbral que maximice una función de verosimilitud para una distribución de probabilidad específica. El enfoque de contornos activos o «snakes» a implementar en el presente proyecto es muy popular. Básicamente un contorno activo se define como una secuencia de puntos, ya sea discreta o continua. Las distintas variantes de los algoritmos «snakes» tienen por objetivo minimizar una función de energía la cuál está compuesta de tres términos: la **continuidad, la suavidad** (ambas de la curva), y un término de ***energía interna*** el cual mide distintas características que aproximan la probabilidad de que el contorno esté sobre los bordes de una imagen. El encontrar tal contornos puede entenderse como un problema de optimización, para cuya solución implementaremos las siguientes etapas básicas:

1. Búsqueda de una solución inicial

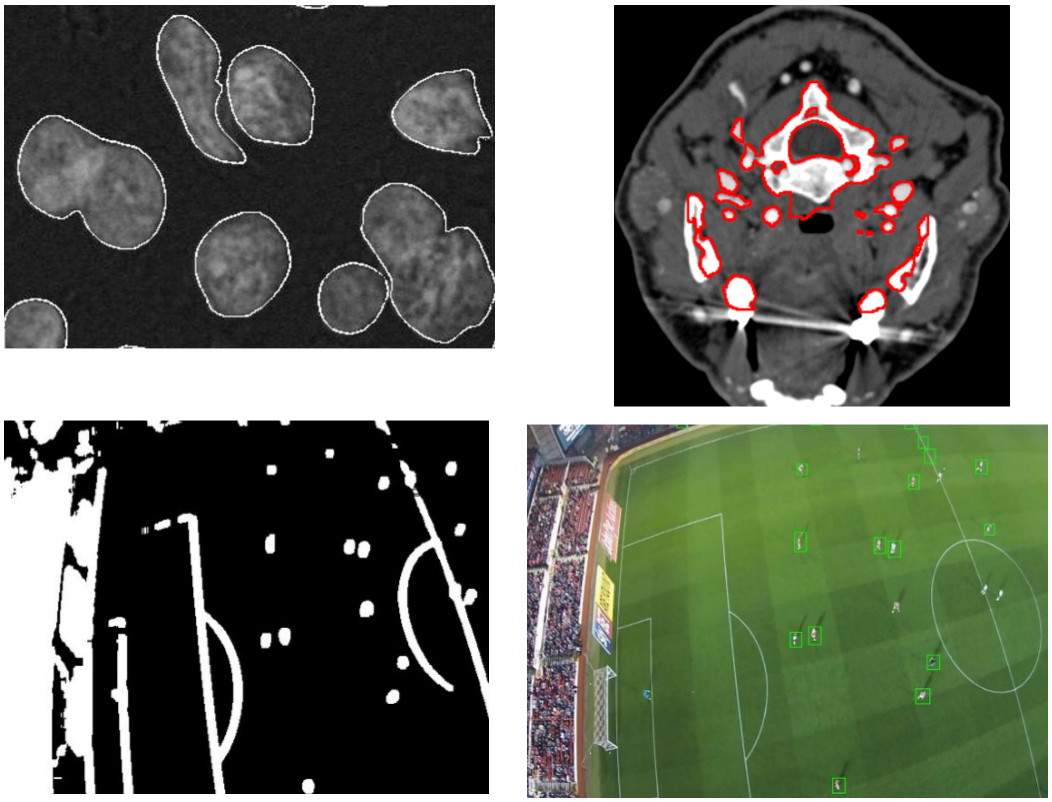

Figura 19: Ejemplos de algoritmos de segmentación. Arriba, contornos activos, abajo izquierda, algoritmos de umbralización. Abajo, derecha, algoritmo de detección de jugadores.

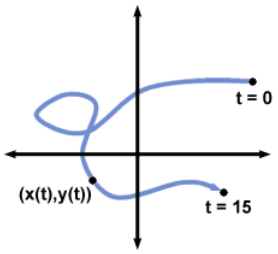

Figura 20: Contorno como función paramétrica.

2. Realización de iteraciones en búsqueda de un mínimo o máximo de una función de costo.

3. Parada del algoritmo bajo una o más condiciones específicas.

Es por ello que explorar la implementación del algoritmo de contornos activos es de interés para el presente curso, pues permite ilustrar de forma práctica la utilidad de los algoritmos de optimización.

4.1. Algortimo de contornos activos.

El algoritmo de contornos activos optimiza la posición de un conjunto de puntos, también referido como contorno. La Figura 20 presenta un contorno de ejemplo:

La implementación computacional de un contorno activo opera en el dominio discreto, donde un contorno se define simplemente como un arreglo de N puntos:

$$c = \langle p_1, p_2, \ldots p_{i-1}, p_i, p_{i+1} \ldots, p_N \rangle$$

donde cada punto está compuesto por el par ordenado de coordenadas x e y, por lo que entonces:

$$p_i = (x_i, y_i)\,.$$

En MATLAB por ejemplo, puede inicializar un contorno o «snake» circular, como sigue:

```
r = 170; %radius
x0 = 350;
y0 = 224;
% r = 230; %radius % x0 = 361;    % y0 = 317;
th = 0:2*pi/nPts:(2*pi)-(2*pi/nPts);
%column-vector with snake points
snake = [round(r*sin(th) + y0); round(r*cos(th) + x0)]';
```

Dos propiedades interesantes para un contorno son la continuidad y la curvatura, los cuales analizamos a continuación.

4.1.1. Continuidad de un contorno

La continuidad de un contorno se refiere a la presencia (o ausencia) de cambios abruptos en la dirección del contorno (cambios en el recorrido). Un vértice en un cuadrilátero por ejemplo, se puede asociar con una discontinuidad en su contorno. Para medir la continuidad en un contorno, basta con calcular la magnitud de su primer derivada:

$$F_{cont}(c) = \left\| \frac{dc}{ds} \right\|^2$$

la cual se eleva al cuadrado para simplificar los cálculos. La aproximación de la primer derivada en el dominio discreto, corresponde simplemente a la diferencia de dos puntos consecutivos, por lo que el termino de continuidad se reescribe como:

$$F_{cont}(c) = \sum_{i=1}^{N} E_{cont}\left(p_i\right)$$

con:

$$E_{cont}\left(p_i\right) = \left\|p_i - p_{i-1}\right\|^2 = \left(x_i - x_{i-1}\right)^2 + \left(y_i - y_{i-1}\right)^2$$

Otra manera de implementar el término de continuidad, es restando la norma entre los dos puntos a la distancia media $\bar{d}$ entre todos los puntos del contorno :

$$E_{cont}\left(p_i\right) = \left(\bar{d} - \left\|p_i - p_{i-1}\right\|\right)$$

4.1.2. Curvatura o suavidad de un contorno

La curvatura o suavidad de una curva define la concavidad o velocidad de los cambios en la dirección de tal curva, por lo cual en el dominio continuo, se puede definir la magnitud al cuadrado de la segunda derivada de una curva c como:

$$F_{curv}(c) = \left\|\frac{d^2c}{ds}\right\|^2$$

La aproximación de la segunda derivada en el dominio discreto, corresponde a lo siguiente:

$$F_{cont}(c) = \sum_{i=1}^{N} E_{curv}\left(p_i\right)$$

$$E_{curv}\left(p_i\right) = \left\|p_{i-1} - 2p_i + p_{i+1}\right\|^2 = \left(x_{i-1} - 2x_i + x_{i+1}\right)^2 + \left(y_{i-1} - 2y_i + y_{i+1}\right)^2$$

4.1.3. La función de energía

La función de energía z contiene entonces los términos correspondientes a la curvatura y la continuidad del contorno. La minimización de tales términos garantiza que el contorno se mantenga «coherente», es decir, no presente grandes discontinuidades ni concavidades, características usuales de los contornos en objetos. Sin embargo, estas características hasta ahora son «independientes» de la imagen a segmentar. Es deseable que los contornos tomen en cuenta características como las regiones «borde» de una imagen, de modo que los puntos en el contorno estén lo más cerca posible de los pixeles correspondientes a los bordes. Para ello, el algoritmo básico de contornos activos define el término de *borde atractores de contornos*, el cual se basa en el cálculo de la magnitud del gradiente de la imagen de entrada U:

$$E_{bordes}\left(U, p_i\right) = \left\|\nabla U\left(p_i\right)\right\|$$

y para un contorno:

$$F_{bordes}\left(U, c\right) = \sum_{i=1}^{N} \left\|\nabla U\left(p_i\right)\right\|$$

Los bordes básicamente se aproximan al calcular la magnitud de la primera de la imagen, pues se les puede conceptualizar como cambios en la intensidad de los pixeles. El gradiente básicamente se calcula usando una máscara como la Laplaciana, la cual aproxima la primer derivada de la imagen. En MATLAB puede calcular el gradiente con la siguiente función:

```
[Gmag,Gdir] = imgradient(U);
```

Figura 21: Gradiente.

donde *Gmag* contiene la magnitud del gradiente.

La Figura 21 muestra la magnitud del gradiente (normalizada entre 0 y 255) para una imagen de ejemplo.

De esta manera, la función de energía z a minimizar viene dada por:

$$z\left(c\right)=\sum_{i=1}^{N}\left(\alpha_{i}E_{cont}\left(p_{i}\right)+\beta_{i}E_{curv}\left(p_{i}\right)-\gamma_{i}E_{bordes}\left(U,p_{i}\right)\right)$$

Los términos α_i, β_i y γ_i pesan la continuidad y suavidad de la curva, además del gradiente de la imagen, respectivamente. Se recomienda fijar siempre:

$$\alpha_{i}+\beta_{i}+\gamma_{i}=1.$$

Observe que los coeficientes pueden ser particulares a cada punto p_i, sin embargo, la versión más simple y recomendada para este proyecto hace $\alpha_i=\alpha$, $\beta_i=\beta$ y $\gamma_i=\gamma$.

$$z\left(c\right)=\sum_{i=1}^{N}\left(\alpha F_{cont}\left(c\right)+\beta F_{curv}\left(c\right)-\gamma F_{bordes}\left(U,c\right)\right)$$

Para encontrar el mínimo de la función de coste z, se propone usar un enfoque «voraz» de optimización, el cual se detalla en la siguiente sección.

4.2. Minimización de la función de energía con algoritmo de búsqueda: Algoritmo voraz o A estrella

El algoritmo voraz realiza decisiones localmente óptimas, en la búsqueda ideal de un óptimo global. Para la minimización de la función z, el algoritmo voraz sigue el siguiente flujo de pasos, donde este conjunto de iteraciones se repite por K iteraciones previamente definidas:

1. Por cada punto p_i en los N puntos del contorno c realizar lo siguiente:

   a) Evaluar z en los $M\times M-1$ vecinos del punto p_i, definidos como puntos candidatos $v_j,\ j=1,2,\ldots,M\times M-1$. La Figura 22 ilustra los vecinos en el caso de $M=3$. Cada vecino será tomado como un punto candidato, para reemplazar al punto p_i en el contorno c, por lo que entonces, por cada candidato, se crea un nuevo contorno:

   $$c_{j}=\left\langle p_{1},p_{2},\ldots p_{i-1},v_{j},p_{i+1}\ldots,p_{N}\right\rangle .$$

2. Para cada contorno c_j se calculan los tres términos $F_{cont}\left(c_j\right)$, $F_{cont}\left(c_j\right)$ y $F_{bordes}\left(U,c_j\right)$, y cada uno se normaliza (divide) por el valor máximo obtenido para cada término en el vecindario: $F_{cont}\left(c_{max}\right)$, $F_{cont}\left(c_{max}\right)$ y $F_{bordes}\left(U,c_{max}\right)$, de esta manera, se obtienen términos normalizados$F'_{cont}\left(c_j\right)$, $F'_{cont}\left(c_j\right)$ y $F'_{bordes}\left(U,c_j\right)$ que van de cero a uno.

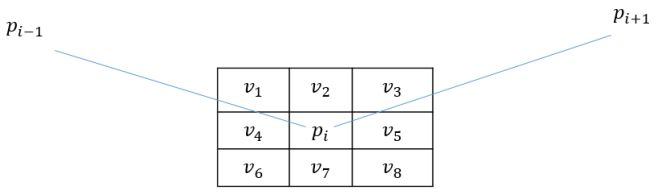

Figura 22: Ilustración de los vecinos para un punto p_i en un contorno.

3. Escoger el contorno c_{min} que minimiza la función:

$$z\left(c_{min}\right) = \sum_{i=1}^{N} \left(\alpha F'_{cont}\left(c_{min}\right) + \beta F'_{curv}\left(c_{min}\right) - \gamma F'_{bordes}\left(U, c_{min}\right)\right).$$

Referencias

[1] S Calderon and F Siles. Deceived bilateral filter for improving the classification of football players from tv broadcast. In *IEEE 3rd International Conference and Workshop on Bioinspired Intelligence*, 2014.

[2] K. Du and M. Swamy. *Search and optimization by metaheuristics*. Birkhauser Basel, 2016.

[3] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, et al. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[4] Mourad Oussalah Zahir Messaoudi, Abdelaziz Ouldali. Tracking objects in video sequence using active contour models and unscented kalman filter. In *2011 7th International Workshop on Systems, Signal Processing and their Applications (WOSSPA)*, 2011.