ENTRENAMIENTO DE UNA RED NEURONAL ARTIFICIAL MEDIANTE EL MÉTODO BFGS ESTRUCTURADO

JHOVANNY ALEXANDER GUTIÉRREZ CABALLERO

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE CIENCIAS
ESCUELA DE MATEMÁTICAS
BUCARAMANGA
2023

ENTRENAMIENTO DE UNA RED NEURONAL ARTIFICIAL MEDIANTE EL MÉTODO BFGS ESTRUCTURADO

JHOVANNY ALEXANDER GUTIÉRREZ CABALLERO

Trabajo de Grado para optar al título de
Matemático

Director
GIOVANNI ERNESTO CALDERÓN SILVA
Doctor en Matemáticas
Codirector
FAVIÁN ENRIQUE ARENAS APARICIO
Magister en Matemáticas

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE CIENCIAS
ESCUELA DE MATEMÁTICAS
BUCARAMANGA
2023

A Yerli, quien decidió acompañarme
en mi formación académica.

A mi abuela, quien me formó y
me enseñó acerca de la vida.

AGRADECIMIENTOS

A mi compañera Yerli Tatiana Prada, por su apoyo e inspiración en la elaboración de este trabajo.

A mis asesores, Favian Enrique Arenas de la Universidad del Cauca, por su inestimable apoyo, y al Doctor Giovanni Calderón por creer en este proyecto.

A mi compañero de universidad, Luis Mantilla, conocido en la academia como **Luis trivial**, por su amistad.

CONTENIDO

LISTA DE FIGURAS

pág.

[1] Marcin Andrychowicz et al. «Learning to learn by gradient descent by gradient descent». En: *Advances in neural information processing systems* 29 (2016).

RESUMEN

TÍTULO: ENTRENAMIENTO DE UNA RED NEURONAL ARTIFICIAL MEDIANTE EL MÉTODO BFGS ESTRUCTURADO.

AUTOR: JHOVANNY ALEXANDER GUTIÉRREZ CABALLERO [**]

PALABRAS CLAVE: MÍNIMOS CUADRADOS NO LINEALES, MÉTODO SECANTE ESTRUCTURADO, REDES NEURONALES, APRENDIZAJE PROFUNDO.

DESCRIPCIÓN: Los problemas de mínimos cuadrados no lineales son comunes en diversas áreas de la ciencia y la ingeniería, donde se busca ajustar modelos matemáticos a datos experimentales de manera que minimicen la diferencia entre los valores observados y los predichos por el modelo. Estos problemas suelen ser no lineales debido a la presencia de parámetros desconocidos en el modelo. Para abordar estos desafiantes problemas, se han desarrollado varios métodos de optimización. Tres de los enfoques más utilizados son el método de Gauss-Newton, el método de Levenberg-Marquardt y el secante estructurado BFGS. Estos métodos de optimización desempeñan un papel fundamental en la resolución de problemas de mínimos cuadrados no lineales en una amplia gama de aplicaciones. La elección del método más adecuado depende de la naturaleza específica del problema y de las características de los datos, y cada uno de estos enfoques ofrece ventajas y desventajas que deben considerarse cuidadosamente en la selección del algoritmo óptimo. En este trabajo se presenta el método secante estructurado BFGS, se analiza la convergencia del método y se presenta una aplicación del mismo en el contexto de las redes neuronales.

[**] Facultad de Ciencias. Escuela de Matemáticas. Director: Giovanni Ernesto Calderón Silva, Doctor en Matemáticas. Codirector: Favián Enrique Arenas Aparicio, Magister en Matemáticas.

ABSTRACT

TITLE: TRAINING OF AN ARTIFICIAL NEURAL NETWORK USING THE STRUCTURED BFGS METHOD. *

AUTHOR: JHOVANNY ALEXANDER GUTIÉRREZ CABALLERO **

KEYWORDS: NONLINEAR LEAST SQUARE, STRUCTURED SECANT METHOD, NEURONAL NETWORKS, DEEP LEARNING.

DESCRIPTION: Nonlinear least squares problems are common in various fields of science and engineering, where the goal is to fit mathematical models to experimental data in a way that minimizes the difference between the observed values and those predicted by the model. These problems often become nonlinear due to the presence of unknown parameters in the model. To address these challenging problems, several optimization methods have been developed. Three of the most commonly used approaches are the Gauss-Newton method, the Levenberg-Marquardt method, and the BFGS structured secant method. These optimization methods play a fundamental role in solving nonlinear least squares problems across a wide range of applications. The choice of the most suitable method depends on the specific nature of the problem and the characteristics of the data, and each of these approaches offers advantages and disadvantages that should be carefully considered when selecting the optimal algorithm. This paper presents the BFGS structured secant method, analyzes the convergence of the method, and provides an application of it within the context of neural networks.

---

* Bachelor Thesis

** Faculty of Science. School Of Mathematics. Advisor: Giovanni Ernesto Calderón Silva, Ph.D in Mathematics. Co-advisor: Favián Enrique Aparicio, Master in Mathematics.

INTRODUCCIÓN

El aprendizaje automático (*machine learning*) es una disciplina del campo de la inteligencia artificial, que a través de algoritmos le da la capacidad a las computadoras de identificar patrones mediante un conjunto de datos, tanto estructurados como no estructurados, con la finalidad de realizar un análisis predictivo y además hacer tareas de forma autónoma, sin necesidad de ser programados[1]. En los últimos años con el aumento de las capacidades de cómputo de las computadoras, las técnicas de aprendizaje automático han sido una parte fundamental en el manejo de los volúmenes de datos (*big data*)[2]. Por lo general, un algoritmo de aprendizaje automático debe usar un conjunto de datos para aprender a realizar una tarea específica bajo un problema de aprendizaje supervisado. Este se basa en un sistema de etiquetas por su parte, asociado a los datos que le permite tomar decisiones o hacer sus predicciones[3]. Una red neuronal es un algoritmo de aprendizaje automático que se define como un grafo dirigido, con las siguientes propiedades: para cada nodo i se le asocia una variable x que se denomina variable de estado; para cada conexión i, j de los nodos, se les asocia un peso $w_{i,j} \in \mathbb{R}$, con $i, j \in \mathbb{N}$; para cada nodo i, se le asocia un *bias* o sesgo b_i; y para cada nodo i, se define una función f_i que depende de $w_{i,j}$, x_j, b_i y los estados de los nodos j a los que está conectado, la función f_i proporciona un nuevo estado del nodo[4].

---

1 Antonio Moreno et al. *Aprendizaje automático*. 1994.

2 Luis Joyanes Aguilar. *Big Data, Análisis de grandes volúmenes de datos en organizaciones*. Alfaomega Grupo Editor, 2016.

3 Ignacio Ros Gómez. «Introducción al aprendizaje supervisado e implementación de una red neuronal en Python». En: (2018).

4 B Martín et al. «Redes neuronales y sistemas borrosos: un libro de texto en español». En: (1970).

En terminología de las redes neuronales, los nodos son las neuronas y las conexiones son las sinapsis, ver en **Figura 7** el grafo dirigido con las propiedades mencionadas.

Los dos conceptos que caracterizan a una red neuronal son el tipo de aprendizaje y su arquitectura, que se determina mediante las siguientes propiedades: se denomina neurona de entrada y salida a las neuronas sin sinapsis entrantes y salientes, respectivamente; a las que no son de entrada ni salida se les denominan neuronas ocultas. Cuando una red es unidireccional no presenta bucles cerrados de conexiones, y es recurrente cuando el flujo de información puede encontrar un bucle de atrás hacia adelante, es decir, una retroalimentación. Los modelos de redes neuronales no realimentadas y de aprendizaje supervisado son los más numerosos y algunos de ellos son el perceptrón simple (*adaline*)[5] y el perceptrón multicapa (*multilayer perceptron*)[6]. Estos dos modelos son muy importantes por su interés histórico y su amplia forma de desempeñarse en aspectos que se encuentran en el campo de las redes neuronales (memoria asociativa, clasificación, aproximación funcional), de tal manera que tienen la capacidad de resolver problemas que no son linealmente separables[7]. En el contexto matemático *adaline* y *multilayer perceptron* son problemas de mínimos cuadrados lineales y no lineales sin restricciones respectivamente. El entrenamiento de una red neuronal, puede hacerse utilizando un conjunto de datos y ajustando los pesos, de tal manera que que el error cometido entre la

---

[5] Donald F Specht. «Probabilistic neural networks and the polynomial adaline as complementary techniques for classification». En: *IEEE Transactions on Neural Networks* 1.1 (1990), págs. 111-121.

[6] Jiexiong Tang, Chenwei Deng y Guang-Bin Huang. «Extreme learning machine for multilayer perceptron». En: *IEEE transactions on neural networks and learning systems* 27.4 (2015), págs. 809-821.

[7] Roberto García. «El perceptrón: una red neuronal artificial para clasificar datos». En: *Revista de investigación en modelos matematicos aplicados a la gestion la economia* (2021).

respuesta obtenida y la predicha sea mínima. Este proceso responde a uno de los problemas de optimización matemática llamado mínimos cuadrados no lineales. Por lo anterior, escoger un método de optimización es una decisión muy importante a la hora de diseñar el algoritmo de aprendizaje. Para resolver un problema de Mínimos cuadrados no lineales existen diversos métodos, entre ellos: el método de máximo descenso, que utiliza la dirección opuesta al gradiente de la función objetivo, con la finalidad de disminuir el error gradualmente hasta llegar a alcanzar un óptimo aproximado[8]. Este método, aunque es fácil de programar, converge en general lentamente[9]. El método de Newton se considera un modelo cuadrático basado en el polinomio de Taylor, y este requiere tanto el gradiente como su matriz Hessiana, que es positiva definida en vecindades cercanas al óptimo y va en dirección del máximo descenso hasta alcanzar un óptimo aproximado. Este método es costoso computacionalmente por las operaciones que involucran el cálculo de las segundas derivadas, y teóricamente el orden de convergencia de Newton es mayor que el máximo descenso[10]. El método de Gauss-Newton es una variación del método de Newton donde involucra el Jacobiano y la Hessiana, el cual hace una aproximación de la matriz Hessiana mediante el Jacobiano y su transpuesto. Su convergencia es cuadrática con respecto a su óptimo y va en dirección del máximo descenso[11]. El método de Levenberg-Marquardt presenta una ligera modificación sobre el método

---

8 Paul Tseng y Sangwoon Yun. «A coordinate gradient descent method for nonsmooth separable minimization». En: *Mathematical Programming* 117 (2009), págs. 387-423.

9 Kwangjun Ahn, Jingzhao Zhang y Suvrit Sra. «Understanding the unstable convergence of gradient descent». En: *International Conference on Machine Learning*. PMLR. 2022, págs. 247-257.

10 Paul T Boggs, Jon W Tolle y Pyng Wang. «On the local convergence of quasi-Newton methods for constrained optimization». En: *SIAM journal on control and optimization* 20.2 (1982), págs. 161-171.

11 Yong Wang. «Gauss–newton method». En: *Wiley Interdisciplinary Reviews: Computational Statistics* 4.4 (2012), págs. 415-420.

de Newton, que consiste en aproximar la matriz Hessiana mediante el Jacobiano y su transpuesto con un término de error. Este método combina tanto el de máximo descenso como el de Gauss-Newton. Se han demostrado que en varias versiones el algoritmo tiene una convergencia global, como las dadas por Powell, Osborne y More[12]. Estos métodos de aprendizaje son muy populares y hay una gran investigación sobre otros métodos enfocados en el contexto de las redes neuronales[13][14].

Como objetivo principal en el trabajo se desarrollará el método BFGS (Broyden, Fletcher, Golfard y Shano) estructurado, para esto se utilizará el artículo de *Hector Jaimes Martínez* e *Ivonne Rivas*[13]: *Método secante estructurado para el entrenamiento del perceptrón multicapa*, el cual se aprovecha de la estructura de la matriz Hessianna calculándola mediante la factorización de Cholesky debido a que la Hessiana tiene la propiedad de ser simétrica definida positiva (SPD). Así no es necesario el cálculo analítico de la matriz Hessiana (segundas derivadas) lo cual es costosa de obtener computacionalmente.

La monografía queda estructurada de la siguiente forma:

En Preliminares se definirán y estudiarán los componentes básicos como definiciones y propiedades de la topología del espacio euclidiano en $\mathbb{R}^n$, continuidad, diferenciabilidad, conjuntos convexos, normas vectoriales, normas matriciales, definiciones de convergencia y además, se enunciarán teoremas que garantizan condiciones necesarias y suficientes tanto de primer y segundo orden para funciones

---

[12] Jorge J Moré. «The Levenberg-Marquardt algorithm: implementation and theory». En: *Numerical Analysis: Proceedings of the Biennial Conference Held at Dundee, June 28–July 1, 1977*. Springer. 2006, págs. 105-116.

[13] Hevert Vivas, Héctor Jairo Martínez y Rosana Pérez. «Método secante estructurado para el entrenamiento del perceptrón multicapa». En: (2018).

[14] Favián Arenas y Hevert Vivas. «Un software interactivo para el entrenamiento de redes neuronales multicapa usando el método secante estructurado». En: *Revista colombiana de tecnologías de avanzada (RCTA)* 2.34 (2019), págs. 85-90.

continuamente diferenciables de $\mathbb{R}^n \to \mathbb{R}$. Las definiciones, propiedades y teoremas se tomaron del libro Dennis Jr, John E and Schnabel, Robert B[15]. Finalmente, se usará la terminología mencionada en los preliminares para hablar de los métodos de optimización sin restricciones y en particular del método secante estructurado BFGS (Broyden-Fletcher-Golfarb-Shano).

En el Capítulo 2 se desarrollan técnicas para encontrar la longitud de paso α_k de la sucesión de descenso $\boldsymbol{x_{k+1}} = \boldsymbol{x_k} + \alpha_k \boldsymbol{d_k}$ mediante la búsqueda de línea exacta e inexacta. Las técnicas de búsqueda de línea son muy utilizadas para hallar una aproximación del punto solución, minimizando la función (20).

En el Capítulo 3 se desarrollan los métodos de búsqueda direccional: Máximo Descenso, Método de Newton, Métodos Cuasi-Newton (SR1-DFP-BFGS) con el objetivo de minimizar la función (27); además, se enuncian propiedades de cada uno de los métodos, su convergencia local y la deducción de cada uno de ellos.

En el Capítulo 4 se desarrollan los métodos Cuasi-Newton estructurados y secantes, resolviendo el problema de Mínimos Cuadrados No Lineales (MCNL) y finalmente se introduce el método secante estructurado BFGS, el cual va a ser implementado en una red neuronal profunda o totalmente conectada.

En el Capítulo 5 se desarrollará la implementación matemática del algoritmo *feed-forward* o propagación hacia adelante, además se introduce la implementación matemática del *backpropagation* o retropropagación, después se presenta el algoritmo del máximo descenso, encargado de minimizar la función objetivo con respecto a

---

[15] John E Dennis Jr y Robert B Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations*. SIAM, 1996.

los errores de la diferencia de los valores reales y los valores predichos por la red. Finalmente, se ilustrará un ejemplo de una red neuronal profunda, realizando el proceso de aprendizaje en un problema de interpolación lineal.

En el Capítulo 6, se incluye el código en el lenguaje de programación MATLAB[16] que describe la estructura y la implementación del algoritmo secante estructurado BFGS, el cual se utiliza como el método para llevar a cabo el proceso de aprendizaje en las redes neuronales.

[16] Starting Matlab. «Matlab». En: *The MathWorks, Natick, MA* (2012).

1. Preliminares

En este capítulo se presentan algunos conceptos y resultados necesarios para la comprensión del resto del documento.

1.1. Topología en el espacio Euclidiano

Definición 1.1. *(Conjunto acotado) Un conjunto A es acotado en $\mathbb{R}^n$ si existe algún número real $M > 0$ tal que*

$$||\boldsymbol{x}|| \leq M, \text{ para todo } \boldsymbol{x} \in A.$$

Definición 1.2. *(Conjunto abierto y cerrado) Un conjunto $A \subset \mathbb{R}^n$ es abierto, si para cada $\boldsymbol{x} \in A$, existe un número real positivo $\varepsilon > 0$ tal que toda bola de radio ε alrededor de $\boldsymbol{x}$ esta contenida en A, es decir*

$$\{\boldsymbol{y} \in \mathbb{R}^n : ||\boldsymbol{y} - \boldsymbol{x}|| < \varepsilon\} \subset A.$$

Un conjunto A es cerrado si para cualquier sucesión $\{x_k\}$ de puntos en A, todos los puntos límites de la sucesión $\{x_k\}$ son elementos de A, en otras palabras

$$\boldsymbol{x} \in cl(A) \text{ si, y solo si, } \lim_{k\to\infty} x_k = \boldsymbol{x} \text{ para alguna sucesión } \{x_k\} \text{ en } A.$$

Note que un conjunto es abierto si, y solo si, $int(A) = A$ y un conjunto es cerrado si, y solo si, $cl(A) = A$, donde $int(A)$ y $cl(A)$ representan el interior y la clausura del conjunto A respectivamente.

Definición 1.3. *(Vecindad) Dado un punto $\boldsymbol{x} \in \mathbb{R}^n$, $\mathcal{N} \in \mathbb{R}^n$ es una vecindad de $\boldsymbol{x}$, si es un conjunto abierto que contiene $\boldsymbol{x}$. Una vecindad en particular es una bola*

abierta de radio $\varepsilon > 0$ *alrededor de* $\boldsymbol{x}$ *denotada por* $B(\boldsymbol{x};\varepsilon)$ *donde*

$$B(\boldsymbol{x};\varepsilon) = \{\boldsymbol{y} : ||\boldsymbol{y} - \boldsymbol{x}|| < \varepsilon\}.$$

Dado un conjunto $A \subset \mathbb{R}^n$, $\mathcal{N}$ *es una vecindad de* A *si existe un* $\varepsilon > 0$ *tal que*

$$\bigcup_{\boldsymbol{x}\in A} B(\boldsymbol{x};\varepsilon) \subset \mathcal{N}.$$

1.2. Normas vectoriales y matriciales

Definición 1.4. *Sea* $\boldsymbol{v} \in \mathbb{R}^n$, *la función* $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}_+$ *es llamada norma vectorial si satisface las siguientes condiciones:*

b1) $\|\boldsymbol{v}\| \geq 0$ *para cada* $\boldsymbol{v} \in \mathbb{R}^n$, *y* $\|\boldsymbol{v}\| = 0$ *si, y solo si,* $\boldsymbol{v} = 0 \in \mathbb{R}^n$.

b2) $\|\alpha\boldsymbol{v}\| = |\alpha|\|\boldsymbol{v}\|$ *para cada* $\boldsymbol{v} \in \mathbb{R}^n$ y $\alpha \in \mathbb{R}$.

b3) $\|\boldsymbol{v} + \boldsymbol{w}\| \leq \|\boldsymbol{v}\| + \|\boldsymbol{w}\|$ *para cada* $\boldsymbol{w}, \boldsymbol{v} \in \mathbb{R}^n$.

Ejemplo 1.1. *(Normas vectoriales) Las siguientes son normas vectoriales y una de las que más se va a usar en este trabajo es (2).*

$$\|\boldsymbol{v}\|_1 = \sum_{n=1}^{n} |v_i|. \tag{1}$$

$$\|\boldsymbol{v}\|_2 = \left(\sum_{n=1}^{n} (v_i)^2\right)^{\frac{1}{2}}. \tag{2}$$

$$\|\boldsymbol{v}\|_\infty = \max_{1\leq i\leq n} |v_i|. \tag{3}$$

Definición 1.5. *(Normas equivalentes) Dos normas vectoriales* $\|\cdot\|_a$ *y* $\|\cdot\|_b$ *son equivalentes si existen constantes* $\alpha, \beta \in \mathbb{R}$ *tales que*

$$\alpha\|\boldsymbol{x}\|_a \leq \|\boldsymbol{x}\|_b \leq \beta\|\boldsymbol{x}\|_a \ \textit{para todo } \boldsymbol{x} \in \mathbb{R}^n.$$

Ejemplo 1.2. *(Normas vectoriales equivalentes) Se presentan algunas equivalencias de las normas dadas en el ejemplo (1.1)*

$$\begin{aligned}
\|\boldsymbol{x}\|_2 &\leq \|\boldsymbol{x}\|_1 \leq \sqrt{n}\ \|\boldsymbol{x}\|_2 \ \textit{para todo } \boldsymbol{x} \in \mathbb{R}^n. \\
\|\boldsymbol{x}\|_\infty &\leq \|\boldsymbol{x}\|_2 \leq \sqrt{n}\ \|\boldsymbol{x}\|_\infty \ \textit{para todo } \boldsymbol{x} \in \mathbb{R}^n. \\
\|\boldsymbol{x}\|_\infty &\leq \|\boldsymbol{x}\|_1 \leq n\ \|\boldsymbol{x}\|_\infty \ \textit{para todo } \boldsymbol{x} \in \mathbb{R}^n.
\end{aligned}$$

Definición 1.6. *Sea $A \in \mathbb{R}^{m\times n}$, la función $|||\cdot||| : \mathbb{R}^{m\times n} \to \mathbb{R}_+$ es llamada norma matricial si satisface las siguientes condiciones:*

c1) $|||A||| \geq 0$ *para cada* $A \in \mathbb{R}^{m\times n}$*, y* $|||A||| = 0$ *si, y solo si,* $A = 0 \in \mathbb{R}^{m\times n}$*.*

c2) $|||\alpha A||| = |\alpha|\ |||A|||$ *para cada* $A \in \mathbb{R}^{m\times n}$ *y* $\alpha \in \mathbb{R}$*.*

c3) $|||A + B||| \leq |||A||| + |||B|||$ *para todo* $A, B \in \mathbb{R}^{m\times n}$*.*

Definición 1.7. *(Normas matriciales equivalentes) Dos normas matriciales $|||\cdot|||_a$ y $|||\cdot|||_b$ son equivalentes , si existen escalares $\alpha, \beta \in \mathbb{R}$ tal que*

$$\alpha\ |||A|||_a \leq |||A|||_b \leq \beta\ |||A|||_a \ \textit{para todo} \quad A \in \mathbb{R}^{m\times n}.$$

Definición 1.8. *(Normas subordinadas) Dada $A \in \mathbb{R}^{m\times n}$, y una norma vectorial $\|\cdot\|_p$, se define una norma matricial como:*

$$|||A|||_p = \max_{x\neq 0} \frac{\|A\boldsymbol{x}\|_p}{\|\boldsymbol{x}\|_p},$$

la cual se conoce como norma p o norma subordinada a la norma vectorial $\|\cdot\|_p$.

Note que si para todo $\boldsymbol{x} \in \mathbb{R}^n \setminus \{0\}$ y $A \in \mathbb{R}^{m\times n}$, si $|||\cdot|||_p$ es una norma matricial subordinada a la norma $\|\cdot\|_p$, entonces

$$\|A\boldsymbol{x}\|_p \leq |||A|||_p \, \|\boldsymbol{x}\|_p.$$

Ejemplo 1.3. *(Normas matriciales) Las siguientes son normas matriciales y se definen como:*
Sea $A \in \mathbb{R}^{m\times n}$ y $\boldsymbol{x} \in \mathbb{R}^n \setminus \{0\}$

$$|||A|||_1 = \max_{x\neq 0} \frac{\|A\boldsymbol{x}\|_1}{\|\boldsymbol{x}\|_1} = \max_{1\leq j\leq n} \sum_{i=1}^{m} |a_{ij}|. \tag{4}$$

$$|||A|||_\infty = \max_{\boldsymbol{x}\neq 0} \frac{\|A\boldsymbol{x}\|_\infty}{\|\boldsymbol{x}\|_\infty} = \max_{1\leq i\leq m} \sum_{j=1}^{n} |a_{ij}|. \tag{5}$$

$$|||A|||_F = \left(\sum_{ij} |a_{ij}|^2 \right)^{\frac{1}{2}}. \tag{6}$$

La norma matricial (6) es conocida como la norma de Frobenius y va hacer la más usada en esté trabajo.

Ejemplo 1.4. *(Normas matriciales equivalentes) Las siguientes normas matriciales equivalentes se encuentran en*[17]*.*

---

[17] Richard L Burden et al. «Análisis numérico». En: (2017).

Sea $B \in \mathbb{R}^{m\times n}$ se tiene que

$$\begin{aligned}
\frac{1}{\sqrt{n}}\,|||B|||_\infty &\leq |||B|||_2 \leq \sqrt{m}\,|||B|||_\infty.\\
|||B|||_2 &\leq |||B|||_F \leq \sqrt{n}\,|||B|||_2.\\
\frac{1}{\sqrt{m}}\,|||B|||_1 &\leq |||B|||_2 \leq \sqrt{n}\,|||B|||_1.\\
|||B|||_2 &\leq \sqrt{|||B|||_1\,|||B|||_\infty}.
\end{aligned}$$

Propiedades de las normas matriciales

Se enuncian algunas propiedades de las normas matriciales

- Una norma matricial $|||\cdot|||_m$ es compatible con una norma vectorial $\|\cdot\|_v$ si

$$\|A\boldsymbol{x}\|_v \leq |||A|||_m\,\|\boldsymbol{x}\|_v.$$

- Toda norma matricial $|||\cdot|||_p$ subordinada a la norma vectorial $\|\cdot\|_p$ es compatible con $\|\cdot\|_p$.

- Para toda norma subordinada $|||\cdot|||_p$, se tiene que $|||I_n||| = 1$

$$|||I_n|||_p \leq \max_{x\neq 0}\frac{\|I_n\boldsymbol{x}\|_p}{\|\boldsymbol{x}\|_p} = \max_{x\neq 0}\frac{\|\boldsymbol{x}\|_p}{\|\boldsymbol{x}\|_p} = \max_{\boldsymbol{x}\neq 0}\{1\} = 1.$$

  Además $|||I_n|||_F = \sqrt{n}$

$$|||I_n|||_F = \left(\sum_{ii}|a_{ii}|^2\right)^{\frac{1}{2}} = \sqrt{n}$$

  Note que $|||A|||_F^2 = \text{Traza}(A^TA)$, donde $A = (a_{ii})$, entonces $\text{Traza}(B) = b_{11} + b_{22} + \ldots + b_{nn}$.

- La norma de Frobenius es compatible con la norma vectorial $||\cdot||_2$, entonces

se tiene que

$$||A\boldsymbol{x}||_2 \leq |||A|||_F||\boldsymbol{x}||_2 = |||A|||_2||\boldsymbol{x}||_2.$$

Definición 1.9. *Una norma matricial* $|||\cdot|||$ *es consistente si*

$$|||AB||| \leq |||A||| \; |||B|||.$$

Ejemplo 1.5. *(Norma que no es consistente) La norma del máximo definida por* $|||A|||_{max} = \max_{ij}|a_{ij}|$ *no es consistente.*

Definición 1.10. *Sea* $A \in \mathbb{R}^{n\times n}$*, si existe otra matriz* $B \in \mathbb{R}^{n\times n}$ *tal que* $AB = BA = I_n$ *decimos que* A *es No singular o también invertible.*

Lema 1.11. *(Lema de Banach) Para cualquier* $|||\cdot|||$ *norma de* $\mathbb{R}^{n\times n}$ *que cumple la condición de ser consistente (1.9) y* $|||I_n||| = 1$*. Si* $F \in \mathbb{R}^{n\times n}$ *y* $|||F||| < 1$*, entonces* $(I-F)^{-1}$ *existe y se tiene que*

$$|||(I-F)^{-1}||| \leq \frac{1}{1-|||F|||}. \tag{7}$$

Si A *es no singular y* $|||A^{-1}(B-A)||| < 1$*, entonces* B *es no singular y se tiene que*

$$|||B^{-1}||| \leq \frac{|||A^{-1}|||}{1-|||A^{-1}(B-A)|||}. \tag{8}$$

Demostración. (7) Supongamos que $I-F$ es singular, es decir existe $\boldsymbol{x} \in \mathbb{R}^n \setminus \{0\}$ tal que $(I-F)\,\boldsymbol{x} = 0$, se tiene que $F\boldsymbol{x} = \boldsymbol{x}$, luego $\|\boldsymbol{x}\| = \|F\boldsymbol{x}\| \leq |||F||| \; \|\boldsymbol{x}\|$. Se concluye que $1 \leq |||F|||$ lo cual es una contradicción, por lo tanto $(I-F)^{-1}$ existe.

Por otro lado, note que

$$\left(\sum_{i=0}^{N} F^{(i)}\right)(I-F) = \sum_{i=0}^{N}\left(F^{(i)}-F^{(i+1)}\right) = I - F^{(N+1)}$$

$$\lim_{N\to\infty}\left(F^{(i)}\right)(I-F) = I - \lim_{N\to\infty} F^{(N+1)}.$$

Veamos que $\lim_{N\to\infty} F^{(N+1)} = 0$, note que $|||F^{(N+1)}||| \leq |||F|||^{(N+1)}$ y como $0 \leq |||F^{(N+1)}||| \leq |||F|||^{(N+1)}$, luego

$$\lim_{N\to\infty} 0 \leq \lim_{N\to\infty} |||F^{(N+1)}||| \leq \lim_{N\to\infty} |||F|||^{(N+1)},$$

pero $|||F||| \leq 0$, así $\lim_{N\to\infty} |||F|||^{(N+1)} = 0$ por teorema de compresión, así

$$(I-F)^{-1} = \sum_{i=0}^{\infty} F^{(i)} \text{ (Newman serie)},$$

por lo tanto

$$|||I-F||| = ||| \lim_{N\to\infty} \sum_{i=0}^{N} F^{(i)}||| = \lim_{N\to\infty} ||| \sum_{i=0}^{N} F^{(i)}||| \leq$$

$$\leq \lim_{N\to\infty} \sum_{i=0}^{N} |||F^{(i)}||| \leq \lim_{N\to\infty} \sum_{i=0}^{N} |||F|||^{(i)} = \sum_{i=0}^{\infty} |||F|||^{(i)} = \frac{1}{1-|||F|||}$$

$$\sum_{i=0}^{\infty} |||F|||^{(i)} = \frac{1}{1-|||F|||} \text{ por hipótesis } |||F||| < 1.$$

Por lo tanto se concluye que

$$|||(I-F)^{-1}||| \leq \frac{1}{1-|||F|||}.$$

□

Demostración. (8) Sea $|||A^{-1}(B-A)||| < 1$ y A es no singular, se define $E = B - A$, así $B = A + E = A\,(I + A^{-1}E)$ es invertible, se tiene que $|||A^{-1}E||| < 1$, por (7) se tiene que

$$(A+E)^{-1} = \left(\sum_{i=0}^{\infty}(-A^{-1}E)^i\right)\ A^{-1} \quad \text{(Newman serie)}. \tag{9}$$

Luego

$$\begin{aligned}
|||(A+E)^{-1} - A^{-1}||| &= ||| \left(\sum_{i=0}^{\infty}(-A^{-1}E)^i\right)\ A^{-1} - A^{-1}||| \\
&= ||| \left(\sum_{i=0}^{\infty}(-AE)^i - I\right)\ A^{-1}||| \\
&\leq |||A^{-1}||| \ ||| \sum_{i=0}^{\infty}(-A^{-1}E)^i - I||| \\
&\leq |||A^{-1}||| \ \frac{|||A^{-1}E|||}{1 - |||A^{-1}E|||} \\
&= \frac{|||A^{-1}|||}{1 - |||A^{-1}E|||}.
\end{aligned}$$

Note que

$$\begin{aligned}
B^{-1} &= A^{-1}(I + A^{-1}E)^{-1} \\
&= \sum_{i=0}^{\infty}(-A^{-1}E)^{-1}\ A^{-1} \\
&= A^{-1} + \sum_{i=1}^{\infty}(-A^{-1}E)^{-1}\ A^{-1} \\
&= A^{-1} - (A+E)^{-1}.
\end{aligned}$$

Por lo tanto se concluye que

$$|||B^{-1}||| \leq \frac{|||A^{-1}|||}{1 - |||A^{-1}(B-A)|||}.$$

□

Definición 1.12. *Una matriz $A \in \mathbb{R}^{n\times n}$ es definida positiva, si para todo $\boldsymbol{x} \in \mathbb{R}^n$ no nulo, se tiene que*

$$\boldsymbol{x}^T A\boldsymbol{x} \geq 0. \tag{10}$$

Si la desigualdad anterior se cumple estrictamente; es decir, $\boldsymbol{x}^T A\boldsymbol{x} > 0$, se dice que la matriz A es definida positiva.

Definición 1.13. *Si una matriz $A \in \mathbb{R}^{n\times n}$ es simétrica y definida positiva, sus valores propios son todos reales positivos.*

Definición 1.14. *Sea $\lambda \in \mathbb{R}$ es un valor propio de una matriz $A \in \mathbb{R}^{n\times n}$ si existe un vector $\mathbf{0} \neq \boldsymbol{x} \in \mathbb{R}^n$ tal que*

$$A\boldsymbol{x} = \lambda\boldsymbol{x},$$

al vector $\boldsymbol{x} \in \mathbb{R}^n$ se le conoce como el vector propio asociado a λ.

1.3. Continuidad

Definición 1.15. *(Función continua) Sea f una función que mapea algún dominio $\mathcal{D} \subset \mathbb{R}^n$ al espacio de $\mathbb{R}^m$, para algún punto $\boldsymbol{x_0} \in cl(\mathcal{D})$, y se escribe*

$$\lim_{x\to x_0} f(\boldsymbol{x}) = f_0,$$

en otras palabras, para todo $\varepsilon > 0$ existe $\delta > 0$ tal que

$$||\boldsymbol{x} - \boldsymbol{x_0}|| < \delta \text{ y } \boldsymbol{x} \in \mathcal{D}, \text{ entonces } ||f(\boldsymbol{x}) - f_0|| < \varepsilon.$$

Se dice que f es continua en $\boldsymbol{x_0}$ si $\boldsymbol{x_0} \in \mathcal{D}$, donde $f_0 = f(\boldsymbol{x_0})$.

Definición 1.16. *(Lipschitz continua) Sea $m, n > 0$, $F : \mathbb{R}^n \to \mathbb{R}^{m\times n}$, $\boldsymbol{x} \in \mathbb{R}^n$, sea $\|\cdot\|$ una norma vectorial sobre $\mathbb{R}^n$, y $|||\cdot|||$ una norma matricial sobre $\mathbb{R}^{m\times n}$, decimos que F es Lipschitz continua en $\boldsymbol{x}$, si existe un conjunto abierto $D \subset \mathbb{R}^n$, $\boldsymbol{x} \in D$ y una constante $\gamma > 0$ tal que para todo $\boldsymbol{v} \in D$ se tiene que*

$$||| F(\boldsymbol{v}) - F(\boldsymbol{x}) ||| \leq \gamma \, \|\boldsymbol{v} - \boldsymbol{x}\|. \tag{11}$$

La constante γ es conocida como la constante Lipschitz para F en $\boldsymbol{x}$ para cualquier D que contenga a $\boldsymbol{x}$.
Se dice que F es Lipschitz continua en una vecindad $\boldsymbol{x} \in D$, y se denota por $F \in Lip_\gamma(D)$ para cada $\boldsymbol{x} \in D$.

Definición 1.17. *(conjuntos de nivel) El conjunto de nivel de valor c de una función $f : \mathbb{R}^n \to \mathbb{R}$ se define como*

$$\mathcal{S}_c = \{\boldsymbol{x} \in \mathbb{R}^n : f(\boldsymbol{x}) = c\} \subset \mathbb{R}^n.$$

En el caso de $\mathbb{R}^2$ se trata de curvas, se les conoce como curvas de nivel y en el caso de $\mathbb{R}^3$ se trata de superficies, se les conoce como superficies de nivel.

1.4. Diferenciabilidad

Sea $\Phi : \mathbb{R} \to \mathbb{R}$ una función, la derivada Φ' es definida por

$$\frac{d\Phi}{d\alpha} = \Phi'(\alpha) = \lim_{\varepsilon \to 0} \frac{\Phi(\alpha + \varepsilon) - \Phi(\alpha)}{\varepsilon}.$$

La segunda derivada se obtiene sustituyendo Φ por $\Phi^{'}$ en la misma expresión de la definición de derivada; esto es

$$\frac{d^2\Phi}{d\alpha^2} = \Phi^{''} = \lim_{\varepsilon \to} \frac{\Phi^{'}(\alpha + \varepsilon) - \Phi^{'}(\alpha)}{\varepsilon}.$$

Definición 1.18. *Sea $f : \mathbb{R}^n \to \mathbb{R}$ donde $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T$. Decimos que f es diferenciable en x si existe un vector $\boldsymbol{g} \in \mathbb{R}^n$ tal que*

$$\lim_{\boldsymbol{y} \to 0} \frac{f(\boldsymbol{x} + \boldsymbol{y}) - f(\boldsymbol{x}) - \boldsymbol{g}^T\boldsymbol{y}}{||\boldsymbol{y}||} = 0, \tag{12}$$

donde $||\cdot||$ es la norma del vector $\boldsymbol{y}$ (este tipo de diferenciabilidad se conoce como la derivada de Frechet). Si $\boldsymbol{g}$ satisface (12) se conoce como el gradiente de f en $\boldsymbol{x}$ y se denota por $\nabla f(\boldsymbol{x})$ y se escribe

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} \dfrac{\partial f}{\partial x_1} \\ \vdots \\ \dfrac{\partial f}{\partial x_n} \end{bmatrix},$$

donde $\dfrac{\partial f}{\partial x_i}$ representa la derivada parcial de f con respecto a x_i. Tomando a $\boldsymbol{y} = \varepsilon \boldsymbol{e_i}$ en (12) donde $\boldsymbol{e_i} \in \mathbb{R}^n$ que consiste de ceros en la posición i, excepto para primera posición, así se tiene que

$$\begin{aligned} \frac{\partial f}{x_i} &= \lim_{\varepsilon \to 0} \frac{f(x_1, \ldots, x_{i-1}, x_i + \varepsilon, x_{i+1}, \ldots, x_n) - f(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n)}{\varepsilon} \\ &\approx \frac{f((\boldsymbol{x} + \varepsilon \boldsymbol{e_i}) - f(\boldsymbol{x}))}{\varepsilon}. \end{aligned}$$

La matriz de las segundas derivadas parciales de f se conoce como la Hessiana y

se define como

$$\nabla^2 f(\boldsymbol{x}) = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_1 \partial x_n} \\ \dfrac{\partial^2 f}{\partial x_2 \partial x_1} & \dfrac{\partial^2 f}{\partial x_2^2} & \cdots & \dfrac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \cdots & \vdots \\ \dfrac{\partial^2 f}{\partial x_n \partial x_1} & \dfrac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

Lema 1.19. *Sea $F : \mathbb{R}^n \to \mathbb{R}$ una función continuamente diferenciable en un abierto convexo $D \subseteq \mathbb{R}^n$, $\boldsymbol{x} \in D$ y $\nabla^2 F$ es Lipschitz continua en $\boldsymbol{x}$ en la vecindad $\mathcal{D}$, usando la norma matricial inducida por la norma vectorial con constante γ. Entonces, para cualquier $\boldsymbol{x} + \boldsymbol{p} \in D$,*

$$||| F(\boldsymbol{x} + \boldsymbol{p}) - F(\boldsymbol{x}) - \nabla^2 F(\boldsymbol{x})\, \boldsymbol{p} ||| \leq \frac{\gamma}{2} \|\boldsymbol{p}\|^2,$$

ver [15].

Se dice que f es diferenciable en un dominio $\mathcal{D}$ si $\nabla f(\boldsymbol{x})$ existe para todo $\boldsymbol{x} \in \mathcal{D}$, y continuamente diferenciable si $\nabla f(\boldsymbol{x})$ es una función continua en $\boldsymbol{x}$. Similarmente, f es dos veces diferenciable sobre $\mathcal{D}$ si $\nabla^2 f(\boldsymbol{x})$ existe para todo $\boldsymbol{x} \in \mathcal{D}$ y es dos veces continuamente diferenciable si $\nabla^2 f(\boldsymbol{x})$ es continua sobre $\mathcal{D}$. Note que cuando f es dos veces continuamente diferenciable, la Hessiana es una matriz simétrica, es decir

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial^2 f}{\partial x_j \partial x_i}, \text{ para todo } i, j = 1, 2, \ldots, n.$$

Cuando f es una función de valor vectorial $f : \mathbb{R}^n \to \mathbb{R}^m$ se define $\nabla f(\boldsymbol{x})$ como una matriz de tamaño $n \times m$ cuya i- ésima columna es $\nabla f_i(\boldsymbol{x})$ que es el gradiente de f_i con respecto a x. Por conveniencia de notación se suele usar la matriz transpuesta de dimensión $m \times n$, esta matriz se conoce como Jacobiana y se denota por $J(\boldsymbol{x})$,

donde el elemento (i, j) de $J(\boldsymbol{x})$ es $\dfrac{\partial f_i(\boldsymbol{x})}{\partial x_j}$ y el vector $\boldsymbol{x}$ depende a su vez de otro vector $\boldsymbol{t}$ ($\boldsymbol{x} = \boldsymbol{x}(\boldsymbol{t})$) extendiendo la definición de la regla de la cadena para función de una variable (12) se tiene que

$$h(t) = f(x(t)),$$

donde

$$\nabla h(t) = \sum_{i=1}^{n} \frac{\partial f}{\partial x_i} \nabla x_i(t) = \nabla x \nabla f(x(t)). \tag{13}$$

Definición 1.20. *(Derivada direccional) La derivada direccional de una función* $f : \mathbb{R}^n \to \mathbb{R}$ *en la dirección* $\boldsymbol{p}$ *esta dado por*

$$D(f(\boldsymbol{x}); \boldsymbol{p}) = \lim_{\varepsilon \to 0} \frac{f(\boldsymbol{x} + \varepsilon \boldsymbol{p}) - f(\boldsymbol{x})}{\varepsilon}.$$

Cuando f *es continuamente diferenciable en una vecindad de* $\boldsymbol{x}$*, se tiene que*

$$D(f(\boldsymbol{x}); \boldsymbol{p}) = \nabla f(\boldsymbol{x})^T \, \boldsymbol{p}.$$

Si se define la función

$$\Phi(\alpha) = f(\boldsymbol{x} + \alpha \, \boldsymbol{p}) = f(y(\alpha)),$$

donde $y(\alpha) = \boldsymbol{x} + \alpha \, \boldsymbol{p}$*, se tiene que*

$$\lim_{\varepsilon \to 0} \frac{f(\boldsymbol{x} + \varepsilon \, \boldsymbol{p}) - f(\boldsymbol{x})}{\varepsilon} = \lim_{\varepsilon \to 0} \frac{\Phi(\varepsilon) - \Phi(0)}{\varepsilon} = \Phi'(0).$$

Aplicando la regla de la cadena (13) en $f(y(\alpha))$, se tiene que

$$\begin{aligned}\Phi^{'}(\alpha) &= \sum_{i=1}^{n} \frac{\partial f(y(\alpha))}{\partial y_i} \nabla y_i(\alpha) \\ &= \sum_{i=1}^{n} \frac{\partial f(y(\alpha))}{\partial y_i}\, \boldsymbol{p_i} = \nabla f(y(\alpha))^T\, \boldsymbol{p} = \nabla f(\boldsymbol{x} + \alpha\, \boldsymbol{p})^T\, \boldsymbol{p}.\end{aligned}$$

Observación 1.21. *Tomando $\alpha = 0$ se tiene que $\Phi^{'}(\alpha) = \nabla f(\boldsymbol{x})^T\, \boldsymbol{p}$ en (14).*

1.5. Órdenes de convergencia y sucesiones.

En los cálculos numéricos, especialmente con las computadoras de alto rendimiento, sucede con frecuencia que la respuesta a un problema no se obtiene de inmediato. Más bien, lo que se genera es una sucesión de respuestas aproximadas cada vez más precisas.

Definición 1.22. *(Sucesión) Sea $\{\boldsymbol{x_k}\}$ una sucesión de puntos en $\mathbb{R}^n$. Decimos que una sucesión converge a algún punto $\boldsymbol{x}$, si para cada $\varepsilon > 0$ existe un $K \in \mathbb{N}$ tal que*

$$||\boldsymbol{x_k} - \boldsymbol{x}|| < \varepsilon, \quad \textit{para todo } k \geq K.$$

Se usará la notación $\lim_{k\to\infty} \boldsymbol{x_k} = \boldsymbol{x}$ para designar la convergencia de una sucesión a algún punto $\boldsymbol{x}$.

Ejemplo 1.6. *la sucesión $x_k = \left(1 - 2^{-k}, \frac{1}{k^2}\right)^T$ converge a $(1, 0)^T$.*

Definición 1.23. *(Punto de acumulación) Dado un conjunto de índices $\mathcal{S} \subset \{1, 2, \ldots\}$ se define una subsucesión de $\{t_k\}$ correspondiente a $\mathcal{S}$ y es denotada por $\{t_k\}_{t\in\mathcal{S}}$. Se dice que $\hat{\boldsymbol{x}} \in \mathbb{R}^n$ es un **Punto de acumulación o punto límite** para $\boldsymbol{x_k}$ si existe un conjunto de índices infinitos $k_1, k_2, \ldots$ tal que la subsucesión $\{x_{k_i}\}_{i=1,2,3,\ldots}$ converge a $\hat{\boldsymbol{x}}$ y se denota por*

$$\lim_{i\to\infty} \boldsymbol{x_{k_i}} = \hat{\boldsymbol{x}}.$$

Proposición 1.24. *Una sucesión es convergente si, y solo si, tiene exactamente un punto límite.*

Definición 1.25. *(Sucesión de Cauchy) Una sucesión es de Cauchy si para cualquier* $\varepsilon > 0$ *existe un entero positivo* K *tal que* $||\boldsymbol{x_k} - \boldsymbol{x_l}|| < \varepsilon$ *para todo índice* $k \geq K$ *y* $l \geq K$*.*

Proposición 1.26. *Una sucesión es convergente si, y solo si, es una sucesión de Cauchy.*

Definición 1.27. *(Órdenes de convergencia) Sea una sucesión* $\{\boldsymbol{x_k}\}$ *en* $\mathbb{R}^n$ *que converge a* $\boldsymbol{x^\star}$*. Se dice que la convergencia es* ***Q- lineal*** *si existe una constante* $r \in (0, 1)$ *tal que*

$$\frac{||\boldsymbol{x_{k+1}} - \boldsymbol{x^\star}||}{||\boldsymbol{x_k} - \boldsymbol{x^\star}||} \leq r, \quad \textit{para todo } k \in \mathbb{N} \textit{ suficientemente grande.}$$

Significa que la distancia a la solución $\boldsymbol{x^\star}$ *disminuye en cada iteración por un factor acotado por la constante 1. El prefijo Q significa cociente, porque este tipo de convergencia se define en términos del cociente de los errores sucesivos.*

Ejemplo 1.7. *La sucesión* $x_k = 1 + \left(\frac{1}{2}\right)^k$ *converge linealmente a 1, con una constante* $r = \frac{1}{2}$*.*

La convergencia es **Q-superlineal** si

$$\lim_{k\to\infty} \frac{||\boldsymbol{x_{k+1}} - \boldsymbol{x^\star}||}{||\boldsymbol{x_k} - \boldsymbol{x^\star}||} = 0, \quad \text{para todo } k \in \mathbb{N} \text{ suficientemente grande.}$$

Ejemplo 1.8. *La sucesión* $x_k = 1 + k^{-k}$ *converge superlinealmente a 1.*

La convergencia es **Q-cuadrática** si

$$\frac{||\boldsymbol{x_{k+1}} - \boldsymbol{x^\star}||}{||\boldsymbol{x_k} - \boldsymbol{x^\star}||^2} \leq M, \quad \text{para todo } k \in \mathbb{N} \text{ suficientemente grande,}$$

donde M es una constante positiva, no necesariamente menor o igual que 1.

Ejemplo 1.9. *La sucesión* $x_k = 1 + \left(\frac{1}{2}\right)^{2^k}$ *converge cuadráticamente.*

1.6. Optimización sin restricciones

Definición 1.28. *(Mínimo global) Un punto* $\boldsymbol{x}^\star \in \mathbb{R}^n$ *es un mínimo global si, y solo si,* $f(\boldsymbol{x}^\star) \leq f(\boldsymbol{x})$ *para todo* $\boldsymbol{x} \in \mathbb{R}^n$.

Definición 1.29. *(Mínimo local) Un punto* $\boldsymbol{x}^\star \in \mathbb{R}^n$ *es un mínimo local si, y solo, si existe una vecindad* $\mathcal{N}$ *de* $\boldsymbol{x}^\star$ *tal que* $f(\boldsymbol{x}^\star) \leq f(\boldsymbol{x})$ *para todo* $\boldsymbol{x} \in \mathcal{N}$.

Definición 1.30. *Un punto* $\boldsymbol{x}^\star$ *es un mínimo local estricto, si existe una vecindad* $\mathcal{N}$ *de* $\boldsymbol{x}^\star$ *tal que* $f(\boldsymbol{x}^\star) < f(\boldsymbol{x})$ *para todo* $\boldsymbol{x} \in \mathcal{N}$ *con* $\boldsymbol{x} \neq \boldsymbol{x}^\star$.

Definición 1.31. *Un punto* $\boldsymbol{x}^\star$ *es un mínimo local aislado, si existe una vecindad* $\mathcal{N}$ *de* $\boldsymbol{x}^\star$ *tal que* $\boldsymbol{x}^\star$ *es el único mínimo local en* $\mathcal{N}$.

Teorema 1.32. *(Teorema de Taylor) Sea* $f : \mathbb{R}^n \to \mathbb{R}$ *una función continuamente diferenciable y* $\boldsymbol{d} \in \mathbb{R}^n$*, entonces se tiene que*

$$f(\boldsymbol{x} + \boldsymbol{d}) = f(\boldsymbol{x}) + \nabla f(\boldsymbol{x} + t\boldsymbol{d})^T \boldsymbol{d},$$

para algún $t \in (0, 1)$. *Si* f *es dos veces continuamente diferenciable, se tiene que*

$$\nabla f(\boldsymbol{x} + \boldsymbol{d}) = \nabla f(\boldsymbol{x}) + \int_0^1 \nabla^2 f(\boldsymbol{x} + t\boldsymbol{d})\boldsymbol{d}\, dt,$$

y que

$$f(\boldsymbol{x} + \boldsymbol{d}) = f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T\, \boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T\, \nabla^2 f(\boldsymbol{x} + t\boldsymbol{d})\boldsymbol{d},$$

para algún $t \in (0, 1)$.

Teorema 1.33. *(Condiciones necesarias de primer orden) Si $\boldsymbol{x}^\star$ es un mínimo local y $f : \mathbb{R}^n \to \mathbb{R}$ es una función continuamente diferenciable en una vecindad de $\boldsymbol{x}^\star$, entonces $\nabla f(\boldsymbol{x}^\star) = 0$.*

Demostración. Supongamos que $\nabla f(\boldsymbol{x}_\star) \neq 0$ y sea $\boldsymbol{d} = -\nabla f(\boldsymbol{x}_\star)$. Note que $\boldsymbol{d}^T \nabla f(\boldsymbol{x}_\star) = -||\nabla f(\boldsymbol{x}_\star)||^2 < 0$, como ∇f es continua en una vecindad de $\boldsymbol{x}_\star$, existe un escalar $t > 0$ tal que

$$\boldsymbol{d}^T \nabla f(\boldsymbol{x}_\star + t\boldsymbol{d}) < 0, \text{para todo } t \in [0, \alpha],$$

por Teorema (1.32), para cualquier $\hat{t} \in (0, \alpha]$ se tiene que

$$f(\boldsymbol{x}_\star + t\boldsymbol{d}) = f(\boldsymbol{x}_\star) + \hat{t}\boldsymbol{d}^T \nabla f(\boldsymbol{x}_\star + \hat{t}\boldsymbol{d}), \text{ para algún } \ t \in (0, \hat{t}),$$

por lo tanto $f(\boldsymbol{x}_\star + \hat{\boldsymbol{t}}\boldsymbol{d}) < f(\boldsymbol{x}_\star)$ para todo $\hat{t} \in (0, \alpha]$, lo que contradice la definición (1.29). □

Teorema 1.34. *(Condiciones necesarias de segundo orden) si $\boldsymbol{x}_\star$ es un mínimo local y $\nabla^2 f$ es continua en una vecindad de $\boldsymbol{x}^\star$, entonces $\nabla f(\boldsymbol{x}_\star) = 0$ y $\nabla^2 f(\boldsymbol{x}_\star)$ es semidefinida positiva.*

Demostración. Supongamos que $\boldsymbol{d}^T \nabla^2 f(\boldsymbol{x}_\star)\boldsymbol{d} < 0$ no es positiva semidefinida con $\boldsymbol{d} \in \mathbb{R}^n$ no nulo, además por Teorema (1.34) se tiene que $\nabla f(\boldsymbol{x}_\star) = 0$ y dado que $\nabla^2 f$ es continua en una vecindad de $\boldsymbol{x}_\star$, entonces existe $\alpha > 0$ tal que $\boldsymbol{d}^T \nabla^2 f(\boldsymbol{x}_\star + t\boldsymbol{d}) < 0$ para todo $t \in [0, \alpha]$, por Teorema (1.32) centrado en $\boldsymbol{x}_\star$, se tiene que para todo $\hat{t} \in (0, \alpha]$ para algún $t \in (0, \hat{t})$

$$f(\boldsymbol{x}_\star + \hat{t}\boldsymbol{d}) = f(\boldsymbol{x}_\star) + t\boldsymbol{d}^T \nabla f(\boldsymbol{x}_\star) + \frac{1}{2}\hat{t}^2 \boldsymbol{d}^T \nabla f(\boldsymbol{x}_\star + t\boldsymbol{d}) < f(\boldsymbol{x}_\star)$$

lo que contradice la definición (1.30). □

Teorema 1.35. *(Condiciones suficientes de segundo orden) Si $\nabla^2 f$ es continua en*

una vecindad de $\boldsymbol{x}^\star$; $\nabla f(\boldsymbol{x}^\star) = 0$ y $\nabla^2 f(\boldsymbol{x}^\star)$ es simétrica definida positiva, entonces $\boldsymbol{x}^\star$ es un mínimo global estricto de f.

Demostración. Como $\nabla^2 f$ es continua y definida positiva en $\boldsymbol{x}_\star$, sea $r > 0$ así que $\nabla^2 f$ es definida positiva para todo $\boldsymbol{x}$ en $\mathcal{B} = \{\boldsymbol{z} \in \mathbb{R}^n : ||\boldsymbol{z} - \boldsymbol{x}_\star|| < r\}$ con $\boldsymbol{d} \in \mathbb{R}^n$ no nulo, entonces $\boldsymbol{x}_\star + \boldsymbol{d} \in \mathcal{B}$ y por Teorema (1.32) se tiene que

$$\begin{aligned} f(\boldsymbol{x}_\star + \boldsymbol{d}) &= f(\boldsymbol{x}_\star) + \boldsymbol{d}^T \nabla f(\boldsymbol{x}_\star) + \frac{1}{2}\boldsymbol{d}^T f(\boldsymbol{z})\boldsymbol{d} \\ &= f(\boldsymbol{x}_\star) + \frac{1}{2}\boldsymbol{d}^T \nabla^2 f(\boldsymbol{z})\boldsymbol{d}, \end{aligned}$$

donde $\boldsymbol{z} = \boldsymbol{x}_\star + t\boldsymbol{d}$ para algún $t \in (0, 1)$. Como $\boldsymbol{z} \in \mathcal{B}$, se tiene que $\boldsymbol{d}^T \nabla^2 f(\boldsymbol{z})\boldsymbol{d} > 0$ y por lo tanto $f(\boldsymbol{x}_\star + \boldsymbol{d}) > f(\boldsymbol{x}_\star)$, así $\boldsymbol{x}_\star$ es un mínimo local estricto (1.30). □

La convexidad juega un papel fundamental en la optimización, aunque el concepto de convexidad tanto de funciones como de conjuntos parezcan dos conceptos ajenos, juntos conceptos garantizan la existencia del máximo o mínimo de la función objetivo.

Definición 1.36. *(Conjunto convexo) El conjunto $A \subset \mathbb{R}^n$ se dice que es convexo, si $\alpha\boldsymbol{x_1} + (1-\alpha)\boldsymbol{x_2} \in A$, para cualesquiera $\boldsymbol{x_1}, \boldsymbol{x_2} \in A$ y $\alpha \in [0, 1]$. Geométricamente, significa que el segmento de recta $[\boldsymbol{x_1}, \boldsymbol{x_2}] := \{\alpha\boldsymbol{x_1} + (1-\alpha)\boldsymbol{x_2} : \alpha \in [0, 1]\}$ está completamente contenido en A, siempre que $\boldsymbol{x_1}, \boldsymbol{x_2} \in A$.*

Definición 1.37. *(Función convexa) Sea A un conjunto convexo no vació de $\mathbb{R}^n$. Una función $f : A \to \mathbb{R}$ se dice que es convexa en A, si para todo los pares $(\boldsymbol{x_1}, \boldsymbol{x_2}) \in A \times A$ y para todo $\alpha \in [0, 1]$, se tiene que*

$$f(\alpha\boldsymbol{x_1} + (1-\alpha)\boldsymbol{x_2}) \leq \alpha f(\boldsymbol{x_1}) + (1-\alpha) f(\boldsymbol{x_2}).$$

Geométricamente, significa que la función aplicada al segmento de recta $[\boldsymbol{x_1}, \boldsymbol{x_2}]$

esta por debajo del segmento de recta $[f(\boldsymbol{x_1}), f(\boldsymbol{x_2})]$. *Si existe* $t > 0$ *tal que*

$$f(\alpha \boldsymbol{x_1} + (1-\alpha)\boldsymbol{x_2}) \leq \alpha f(\boldsymbol{x_1}) + (1-\alpha) f(\boldsymbol{x_2}) - \frac{t\alpha}{2}(1-\alpha)||\boldsymbol{x_1} - \boldsymbol{x_2}||^2$$

para todo $(\boldsymbol{x_1}, \boldsymbol{x_2}) \in A \times A$ *y todo* $\alpha \in (0,1)$*, se dice que* f *es fuertemente convexa en* A.

2. Búsqueda lineal

Un método de búsqueda lineal calcula una dirección de descenso $\boldsymbol{d_k} \in \mathbb{R}^n$ en cada iteración, el cual decide qué tan lejos se moverá a lo largo de dicha dirección. Cada iteración dada por la expresión $\boldsymbol{x_{k+1}} = \boldsymbol{x_k} + \alpha_k \ \boldsymbol{d_k}$, donde $\alpha_k \in \mathbb{R}$ se conoce como la longitud del paso. Así los métodos de búsqueda lineal dependen de la elección $\boldsymbol{d_k}$ de la longitud de paso α_k. La efectividad de los métodos de búsqueda lineal se basan en encontrar la longitud de paso y la dirección de descenso óptimo, que maximiza o minimiza la función objetivo a lo largo de una dirección de búsqueda, lo cual garantiza un descenso suficiente con respecto a la función objetivo.

Definición 2.1. *(Dirección de descenso) Un vector* $\boldsymbol{d} \in \mathbb{R}^n$ *es una dirección de descenso de* f *a partir de* $\boldsymbol{x} \in \mathbb{R}^n$*, si la derivada direccional de* f *en la dirección de* $\boldsymbol{d}$*, es negativa; es decir* $f'(\boldsymbol{x}; \boldsymbol{d}) < 0$ *o también* $\nabla f(\boldsymbol{x})^T \ \boldsymbol{d} < 0$*.*

Ejemplo 2.1. *Sea* $\boldsymbol{d} = -\nabla f(\boldsymbol{x})$

$$\nabla f(\boldsymbol{x})^T \ \boldsymbol{d} = -\nabla f(\boldsymbol{x})^T \ \nabla f(\boldsymbol{x}) = -||\nabla f(\boldsymbol{x})||^2, \tag{14}$$

por lo tanto $\nabla f(\boldsymbol{x})^T \ \boldsymbol{d} \leq 0$*.*

Note que $-\nabla f(\boldsymbol{x})$ es una dirección de descenso de f a partir de $\boldsymbol{x}$ y es conocido como la **Dirección de máximo descenso** no quiere decir que es la única dirección. Para caracterizar la dirección de descenso se tiene que

$$\nabla f(\boldsymbol{x})^T \ \boldsymbol{d} \ < \ 0 \tag{15}$$

$$||\nabla f(\boldsymbol{x})|| \ ||\boldsymbol{d}|| \cos\theta \ < \ 0 \tag{16}$$

$$\cos\theta \ < \ 0, \ \text{así} \ \frac{\pi}{2} < \theta < \frac{3\pi}{2}. \tag{17}$$

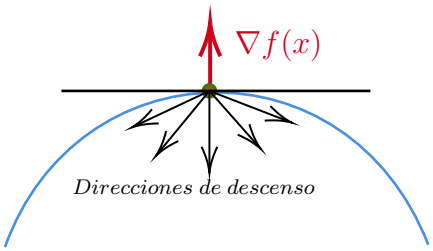

Figura 1. Direcciones de descenso. Hecho por el autor.

Teorema 2.2. *Sea* $f : \mathbb{R}^n \to \mathbb{R}$, $f \in \mathcal{C}^1$, *si existe una dirección de descenso* $\boldsymbol{d} \in \mathbb{R}^n$ *tal que* $\nabla f(\boldsymbol{x})\ \boldsymbol{d} < 0$, *entonces existe* $\alpha > 0$ *tal que* $f(\boldsymbol{x} + \alpha\ \boldsymbol{d}) < f(\boldsymbol{x})$ *para todo* $\boldsymbol{x} \in \mathbb{R}^n$.

Demostración. Supongamos que existe $\boldsymbol{d} \in \mathbb{R}^n$ tal que $\nabla f(\boldsymbol{x})^T\ \boldsymbol{d} < 0$ y se define

$$\Phi(\alpha) := f(\boldsymbol{x} + \alpha\ \boldsymbol{d}), \tag{18}$$

con $\Phi(0) := f(\boldsymbol{x})$ y además $\Phi^{'}(\alpha) = \nabla f(\boldsymbol{x} + \alpha\ \boldsymbol{d})^T \boldsymbol{d}$, por hipótesis se tiene que $\Phi^{'}(0) = \nabla f(\boldsymbol{x})^T\ \boldsymbol{d} < 0$. Note que

$$\begin{aligned} \Phi^{'}(\alpha) &= \lim_{\alpha \to 0} \frac{\Phi(\alpha) - \Phi(0)}{\alpha} \\ &= \lim_{\alpha \to 0^+} \frac{\Phi(\alpha) - \Phi(0)}{\alpha} \end{aligned}$$

para α muy pequeño se tiene que $\Phi^{'}(\alpha)$ y $\Phi(\alpha) - \Phi(0)$ son negativos; es decir $\Phi(\alpha) - \Phi(0) < 0$, así $\Phi(\alpha) < \Phi(0)$, por lo tanto se concluye que $f(\boldsymbol{x} + \alpha\ \boldsymbol{d}) < f(\boldsymbol{x})$. □

Algoritmo 2.0.1. *(Algoritmo básico en la iteración* $\boldsymbol{x_k}$ *en dirección de descenso)*

1. ***Inicialización.*** *Dado el punto* $\boldsymbol{x_k}$.
2. ***Dirección de descenso.*** *Calcular la dirección* $\boldsymbol{d_k}$.
3. ***Descenso.****Se asigna* $\boldsymbol{x_{k+1}} = \boldsymbol{x_k} + \alpha\ \boldsymbol{d_k}$ *para algún* α *que sea aceptable para* $\boldsymbol{x_{k+1}}$.

Buscar el α adecuado para la dirección de descenso en el algoritmo básico en la iteración $\boldsymbol{x_k}$ se le conoce como **Búsqueda lineal (exacta o inexacta)**.

2.1. Búsqueda lineal exacta

Consiste en resolver en cada paso el problema

$$\begin{aligned} &\text{minimizar} \quad \Phi(\alpha), \\ &0 < \alpha \in \mathbb{R} \end{aligned} \tag{19}$$

donde $\Phi : \mathbb{R} \to \mathbb{R}$ está definida por (18), a α se le conoce como la longitud de paso. Teniendo en cuenta el problema, se tiene que resolver $\Phi^{'}(\alpha) = 0$ lo que se traduce a $\nabla f(\boldsymbol{x} + \alpha\ \boldsymbol{d})^T\ \boldsymbol{d} = 0$.
En general resolver el problema de minimizar la función Φ en (20) no es un problema tan fácil.

2.2. Búsqueda lineal inexacta - Condición de Armijo

El método de búsqueda lineal inexacta busca en cada iteración $\boldsymbol{x_{k+1}} = \boldsymbol{x_k} + \alpha_k \boldsymbol{d_k}$ reducir la longitud del paso α_k de forma substancial con respecto a la función objetivo f, tal que el tiempo en encontrar la longitud no sea muy grande. Lo que consiste en resolver en cada iteración el problema

$$\begin{aligned} &\text{minimizar} \quad \Phi(\alpha) = f(\boldsymbol{x_k} + \alpha \boldsymbol{d_k}). \\ &0 < \alpha \in \mathbb{R} \end{aligned} \tag{20}$$

En general, resolver el problema es complejo, lo cual se usan lineas de búsqueda inexactas para identificar cuál es la longitud de paso que tiene una reducción óptima

en f. Una estrategia es usar las condiciones de Wolfe[18] que garantiza una dirección de descenso suficiente con respecto a la función objetivo f, y dada por la siguiente desigualdad conocida como la condición de Armijo:

$$f(\boldsymbol{x_k} + \alpha \boldsymbol{d_k}) \leq f(\boldsymbol{x_k}) + \alpha\lambda\nabla f(\boldsymbol{x_k})^T \boldsymbol{d_k}, \text{ donde } \lambda \in (0,1) \text{ y } \alpha \in \mathbb{R}_+. \tag{21}$$

Note que la desigualdad de Armijo (21) no es suficiente para asegurar que el algoritmo haga un descenso suficiente, ya que la desigualdad será válida para valores de α muy pequeños, así es necesario otra condición que evite pasos muy cortos, la cual es conocida como la condición de curvatura

$$\beta\nabla f(\boldsymbol{x_k})^T \boldsymbol{d_k} \leq \nabla f(\boldsymbol{x_k} + \alpha_k \boldsymbol{d_k})^T \boldsymbol{d_k} \text{ para algún } \beta \in (\lambda, 1),$$

donde $\alpha \in \mathbb{R}_+$. Una longitud de paso puede satisfacer las condiciones de Wolfe sin estar muy cerca a la solución del problema (20), sin embargo se puede modificar la condición de curvatura para que α_k se encuentre en la vecindad del punto estacionario de Φ. Las condiciones fuertes de Wolfe requieren que α_k satisfaga

$$f(\boldsymbol{x_k} + \alpha_k \boldsymbol{d_k}) \leq f(\boldsymbol{x_k}) + \alpha_k\lambda\nabla f(\boldsymbol{x_k})^T \boldsymbol{x_k} \tag{22}$$

$$|\nabla f(\boldsymbol{x_k} + \alpha_k \boldsymbol{d_k})^T \boldsymbol{d_k}| \leq \beta|\nabla f(\boldsymbol{x_k})^T \boldsymbol{d_k}|, \text{con } 0 < \lambda < \beta < 1. \tag{23}$$

Así, la diferencia entre las condiciones de Wolfe, es que no se permite que $\Phi'(\alpha_k)$ sea muy grande. Por lo tanto, se excluyen los puntos que están lejos de alguna vecindad del minimizador de Φ.

El siguiente ejemplo (2.2) utiliza la búsqueda lineal exacta, para definir la longitud de paso adecuada en la función objetivo (24) con dirección de descenso $-\nabla f(\boldsymbol{x_k})$

---

[18] Félix Calderón Solorio. «OPTIMIZACION NO–LINEAL (VER 1.10)». En: (2005).

evaluada en la iteración $\boldsymbol{x_k}$, el cual garantiza la convergencia lineal.

Ejemplo 2.2. *Sea $f : \mathbb{R}^n \to \mathbb{R}$ de clase $\mathcal{C}^1$ y $A \in \mathbb{R}^{n\times n}$ simétrica definida positiva (SPD) con $c \in \mathbb{R}$ se define*

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x} + c. \tag{24}$$

El gradiente de la función (24) es

$$\nabla f(\boldsymbol{x}) = A\boldsymbol{x} - \boldsymbol{b}, \text{ ver } (2.5). \tag{25}$$

Con el objetivo de resolver el problema

$$\begin{aligned} &\textit{minimizar} \quad f(\boldsymbol{x_k} - \alpha\nabla f(\boldsymbol{x_k})), \\ &\alpha > 0 \end{aligned}$$

sea $\Phi(\alpha) = f(\boldsymbol{x_k}) - \alpha\nabla f(\boldsymbol{x_k}) = f(\boldsymbol{x_k} - \alpha\nabla f(\boldsymbol{x_k}))$ y sea $\boldsymbol{g_k} = \nabla f(\boldsymbol{x_k})$. Así, se tiene que

$$\begin{aligned} \Phi(\alpha) &= \frac{1}{2}(\boldsymbol{x_k} - \alpha\boldsymbol{g_k})^T A(\boldsymbol{x_k} - \alpha\boldsymbol{g_k}) - \boldsymbol{b}^T(\boldsymbol{x_k} - \alpha\boldsymbol{g_k}) + c \\ &= \frac{1}{2}(\boldsymbol{x_k}^T - \alpha\boldsymbol{g_k}^T)(A\boldsymbol{x_k} - \alpha A\boldsymbol{g_k}) - \boldsymbol{b}^T\boldsymbol{x_k} + \alpha\boldsymbol{b}^T\boldsymbol{g_k} + c \\ &= \frac{1}{2}\boldsymbol{x_k}^T A\boldsymbol{x_k} - \frac{\alpha}{2}\boldsymbol{x_k}^T A\boldsymbol{g_k} - \frac{\alpha}{2}\boldsymbol{g_k}^T A\boldsymbol{x_k} + \frac{\alpha^2}{2}\boldsymbol{g_k}^T A\boldsymbol{g_k} - \boldsymbol{b}^T\boldsymbol{x_k} + \alpha\boldsymbol{b}^T\boldsymbol{g_k} + c \\ &= \frac{1}{2}\boldsymbol{x_k}^T A\boldsymbol{x_k} - \alpha\boldsymbol{x_k}^T A\boldsymbol{g_k} + \frac{\alpha^2}{2}\boldsymbol{g_k}^T A\boldsymbol{g_k} - \boldsymbol{b}^T\boldsymbol{x_k} + \alpha\boldsymbol{b}^T\boldsymbol{g_k} + c, \end{aligned}$$

donde el gradiente de $\Phi(\alpha)$ es

$$\frac{d\Phi(\alpha)}{d\alpha} = -\boldsymbol{x_k}^T A\boldsymbol{g_k} + \alpha\boldsymbol{g_k}^T + \boldsymbol{b}^T\boldsymbol{g_k} = 0.$$

Luego,

$$\alpha = \frac{\boldsymbol{x_k}^T A\boldsymbol{g_k} - \boldsymbol{b}^T\boldsymbol{g_k}}{\boldsymbol{g_k}^T A\boldsymbol{g_k}} = \frac{(\boldsymbol{x_k}^T A - \boldsymbol{b}^T)\boldsymbol{g_k}}{\boldsymbol{g_k}^T A\boldsymbol{g_k}} = \frac{(A\boldsymbol{x_k} - \boldsymbol{b})^T\boldsymbol{g_k}}{\boldsymbol{g_k}^T A\boldsymbol{g_k}}.$$

Así

$$\alpha = \frac{\boldsymbol{g_k}^T\boldsymbol{g_k}}{\boldsymbol{g_k}^T A\boldsymbol{g_k}}. \tag{26}$$

Veamos la razón de convergencia del método de búsqueda lineal exacta, usando la dirección de Máximo descenso de la función (24).
Se define la norma vectorial con la propiedad $||\boldsymbol{y}||^2 = \boldsymbol{y}^T A\boldsymbol{y}$ tal que, depende de la matriz $A \in \mathbb{R}^{n\times n}$ y con la propiedad de ser simétrica definida positiva. Note que

$$\begin{aligned}
\frac{1}{2}||\boldsymbol{x} - \boldsymbol{x}_\star|| &= \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_\star)^T A(\boldsymbol{x} - \boldsymbol{x}_\star) \\
&= \frac{1}{2}(\boldsymbol{x}^T A\boldsymbol{x} - \boldsymbol{x}^T A\boldsymbol{x}_\star - \boldsymbol{x}_\star{}^T A\boldsymbol{x} + \boldsymbol{x}_\star{}^T A\boldsymbol{x}_\star) \\
&= \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x}_\star - \boldsymbol{x}_\star{}^T A\boldsymbol{x} + \frac{1}{2}\boldsymbol{x}_\star{}^T A\boldsymbol{x}_\star \\
&= \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - \boldsymbol{x}_\star{}^T A\boldsymbol{x} + c + \frac{1}{2}\boldsymbol{x}_\star{}^T A\boldsymbol{x}_\star - c \\
&= \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - (A\boldsymbol{x}_\star)^T\boldsymbol{x} + c - \left(\frac{1}{2}\boldsymbol{x}_\star{}^T A\boldsymbol{x}_\star - \boldsymbol{x}_\star{}^T A\boldsymbol{x}_\star + c\right) \\
&= \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - (A\boldsymbol{x}_\star)^T\boldsymbol{x} + c - \left(\frac{1}{2}\boldsymbol{x}_\star{}^T A\boldsymbol{x}_\star - (A\boldsymbol{x}_\star)^T\boldsymbol{x}_\star + c\right).
\end{aligned}$$

Se sabe que $\nabla f(\boldsymbol{x}) = A\boldsymbol{x} - \boldsymbol{b}$ *y en la solución* $\nabla f(\boldsymbol{x_\star}) = A\boldsymbol{x_\star} - \boldsymbol{b} = 0$*,*

$$\begin{aligned}
\frac{1}{2}||\boldsymbol{x} - \boldsymbol{x_\star}|| &= \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - (A\boldsymbol{x_\star})^T\boldsymbol{x} + c - \left(\frac{1}{2}\boldsymbol{x_\star}^T A\boldsymbol{x_\star} - (A\boldsymbol{x_\star})^T\boldsymbol{x_\star} + c\right) \\
&= \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x} + c - \left(\frac{1}{2}\boldsymbol{x_\star}^T A\boldsymbol{x_\star} - \boldsymbol{b}^T\boldsymbol{x_\star} + c\right) \\
&= f(\boldsymbol{x}) - f(\boldsymbol{x_\star}).
\end{aligned}$$

Definamos $\nabla f(\boldsymbol{x_k}) := \boldsymbol{g_k}$*,* $\boldsymbol{e_k} := \boldsymbol{x_k} - \boldsymbol{x_\star}$ *y* $\boldsymbol{e_{k+1}} := \boldsymbol{x_{k+1}} - \boldsymbol{x_\star}$*. Así se tiene que*

$$\begin{aligned}
\boldsymbol{e_{k+1}} &:= \boldsymbol{x_k} - \alpha\boldsymbol{g_k} - \boldsymbol{x_\star} \\
&:= \boldsymbol{x_k} - \boldsymbol{x_\star} - \alpha\boldsymbol{g_k} \\
&:= \boldsymbol{e_k} - \alpha\boldsymbol{g_k}.
\end{aligned}$$

Tomando α *definido en (2.2), se tiene que*

$$\begin{aligned}
\frac{||\boldsymbol{e_k}||^2 - ||\boldsymbol{e_{k+1}}||^2}{||\boldsymbol{e_k}||^2} &= \frac{\boldsymbol{e_k}^T A\boldsymbol{e_k} - \boldsymbol{e_{k+1}}^T A\boldsymbol{e_{k+1}}}{\boldsymbol{e_k}^T A\boldsymbol{e_k}} = \frac{\boldsymbol{e_k}^T A\boldsymbol{e_k} - (\boldsymbol{e_k} - \alpha\boldsymbol{g_k})^T A(\boldsymbol{e_k} - \alpha\boldsymbol{g_k})}{\boldsymbol{e_k}^T A\boldsymbol{e_k}} \\
&= \frac{\boldsymbol{e_k}^T A\boldsymbol{e_k} - \boldsymbol{e_k}^T A\boldsymbol{e_k} + \alpha\boldsymbol{e_k}^T A\boldsymbol{g_k} + \alpha\boldsymbol{g_k}^T A\boldsymbol{e_k} - \alpha^2\boldsymbol{g_k}^T A\boldsymbol{g_k}}{\boldsymbol{e_k}^T A\boldsymbol{e_k}} \\
&= \frac{2\alpha\boldsymbol{g_k}^T A\boldsymbol{e_k} - \alpha^2\boldsymbol{g_k}^T A\boldsymbol{g_k}}{\boldsymbol{e_k}^T A\boldsymbol{e_k}} \\
&= \frac{2\left(\dfrac{\boldsymbol{g_k}^T\boldsymbol{g_k}}{\boldsymbol{g_k}^T A\boldsymbol{g_k}}\right)\boldsymbol{g_k}^T A\boldsymbol{e_k} - \left(\dfrac{\boldsymbol{g_k}^T\boldsymbol{g_k}}{\boldsymbol{g_k}^T A\boldsymbol{g_k}}\right)^2\boldsymbol{g_k}^T A\boldsymbol{g_k}}{\boldsymbol{e_k}^T A\boldsymbol{e_k}} \\
&= \frac{\dfrac{2(\boldsymbol{g_k}^T\boldsymbol{g_k})(\boldsymbol{g_k}^T A\boldsymbol{e_k})}{\boldsymbol{g_k}^T A\boldsymbol{g_k}} - \dfrac{(\boldsymbol{g_k}^T\boldsymbol{g_k})^2}{\boldsymbol{g_k}^T A\boldsymbol{g_k}}}{\boldsymbol{e_k}^T A\boldsymbol{e_k}}.
\end{aligned}$$

Así, se tiene que

$$\frac{||\boldsymbol{e_k}||^2 - ||\boldsymbol{e_{k+1}}||^2}{||\boldsymbol{e_k}||^2} = \frac{2(\boldsymbol{g_k}^T\boldsymbol{g_k})(\boldsymbol{g_k}^T A\boldsymbol{e_k}) - (\boldsymbol{g_k}^T\boldsymbol{g_k})^2}{(\boldsymbol{g_k}^T A\boldsymbol{g_k})(\boldsymbol{e_k}^T A\boldsymbol{e_k})},$$

como $\boldsymbol{g_k} = A\boldsymbol{x_k} - \boldsymbol{b}$ *en la solución* $A\boldsymbol{x_k} - \boldsymbol{b} = 0$*, entonces* $\boldsymbol{x_\star} = A^{-1}\boldsymbol{b}$*, luego se tiene que*

$$\boldsymbol{g_k} = A\boldsymbol{x_\star} - \boldsymbol{b} = A\boldsymbol{x_k} - AA^{-1}\boldsymbol{b} = A(\boldsymbol{x_k} - A^{-1}\boldsymbol{b}) = A(\boldsymbol{x_k} - \boldsymbol{x_\star}) = A\boldsymbol{e_k},$$

así

$$\boldsymbol{e_k} = A^{-1}\boldsymbol{g_k}.$$

Finalmente

$$\begin{aligned}\frac{||\boldsymbol{e_k}||^2 - ||\boldsymbol{e_{k+1}}||^2}{||\boldsymbol{e_k}||^2} &= \frac{2(\boldsymbol{g_k}^T\boldsymbol{g_k})(\boldsymbol{g_k}^T A\boldsymbol{e_k}) - (\boldsymbol{g_k}^T\boldsymbol{g_k})^2}{(\boldsymbol{g_k}^T A\boldsymbol{g_k})(\boldsymbol{e_k}^T A\boldsymbol{e_k})} \\ &= \frac{(\boldsymbol{g_k}^T\boldsymbol{g_k})}{(\boldsymbol{g_k}^T A\boldsymbol{g_k})(\boldsymbol{g_k}^T A^{-1}\boldsymbol{g_k})}.\end{aligned}$$

Como A *es simétrica definida positiva (1.13), se tiene que los valores propios son reales positivos. Así puedo escoger el máximo y el mínimo de los valores propios* $\alpha_{min}, \alpha_{max}$*; respectivamente. Por Teorema de Kantorovich (2.4), se tiene que*

$$\begin{aligned}\frac{||\boldsymbol{e_k}||^2 - ||\boldsymbol{e_{k+1}}||^2}{||\boldsymbol{e_k}||^2} &\geq \frac{4\alpha_{min}\alpha_{max}}{(\alpha_{min} + \alpha_{max})^2} \\ ||\boldsymbol{e_k}||^2 - ||\boldsymbol{e_{k+1}}||^2 &\geq \frac{4\alpha_{min}\alpha_{max}}{(\alpha_{min} + \alpha_{max})^2}||\boldsymbol{e_k}||^2 \\ ||\boldsymbol{e_{k+1}}||^2 &\leq \left[1 - \frac{4\alpha_{min}\alpha_{max}}{(\alpha_{min} + \alpha_{max})^2}\right]||\boldsymbol{e_k}||^2 \\ &\leq \left[\frac{(\alpha_{min} - \alpha_{max})^2}{(\alpha_{min} + \alpha_{max})^2}\right]||\boldsymbol{e_k}||^2,\end{aligned}$$

se concluye que

$$\frac{||\boldsymbol{e_{k+1}}||}{||\boldsymbol{e_k}||} \leq \frac{\alpha_{min} - \alpha_{max}}{\alpha_{min} + \alpha_{max}}.$$

La convergencia del método de búsqueda lineal exacta de la función (24) *es lineal.*

Definición 2.3. *Una matriz* $A \in \mathbb{R}^{n\times n}$ *es simétrica si, y solo si* $A^T = A$.

Teorema 2.4. *Sea* $A \in \mathbb{R}^{n\times n}$*, simétrica y definida positiva. Para todo* $\boldsymbol{x} \in \mathbb{R}^n$ *no nulo se cumple que*

$$\frac{(\boldsymbol{x^T x})^2}{(\boldsymbol{x}^T A\boldsymbol{x})(\boldsymbol{x}^T A^{-1}\boldsymbol{x})} \geq \frac{4\alpha_{min}\alpha_{max}}{(\alpha_{min} + \alpha_{max})^2},$$

ver [19]*.*

Nota 2.5. *Si* $h : \mathbb{R}^n \to \mathbb{R}$ *es una función cuadrática definida en (27) y* $A \in \mathbb{R}^{n\times n}$ *es una matriz simétrica (2.3), entonces*

$$h(x) := \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{x} + c. \tag{27}$$

Note que $h(\boldsymbol{x})$ *es*

$$\begin{aligned} h(\boldsymbol{x}) &= c + \begin{bmatrix} b_1 & \dots & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \frac{1}{2}\begin{bmatrix} x_1 & \dots & x_n \end{bmatrix} \begin{bmatrix} a_{11} & \dots & a_{1n} \\ a_{21} & \dots & a_{2n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \\ &= c + \sum_{i=1}^{n} b_i x_i + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}x_i x_j. \end{aligned}$$

[19] Javier López. «Optimización multi-objetivo». Tesis doct. Universidad Nacional de La Plata, 2013.

Luego el $\nabla h(\boldsymbol{x})$ *es*

$$\begin{aligned}
\nabla h(\boldsymbol{x}) &= \left[\frac{\partial h(\boldsymbol{x})}{\partial x_s}\right] = \left[\sum_{i=1}^{n} b_i + \sum_{j=1}^{n} A_{sj}x_j + \sum_{i=1}^{n} A_{is}x_i\right] \text{ con } s = 1, \ldots, n. \\
&= \left[\sum_{i=1}^{n} b_i + \sum_{j=1}^{n} A_{sj}x_j\right] \text{ por hipótesis A es simétrica}, \\
&= A\boldsymbol{x} + \boldsymbol{b}.
\end{aligned}$$

3. Búsquedas direccionales

3.1. Método de Máximo Descenso

Sea $f : \mathbb{R}^n \to \mathbb{R}$ una función de clase $\mathcal{C}^1$. La derivada direccional de f en dirección $\boldsymbol{d} \in \mathbb{R}^n$ está dada por

$$Df(\boldsymbol{x}; \boldsymbol{d}) = \nabla f(\boldsymbol{x})^T \boldsymbol{d}.$$

Para obtener la dirección de **Máximo descenso** de la función f en un punto $\boldsymbol{x} \in \mathbb{R}^n$ tal que $\nabla f(\boldsymbol{x}) \neq 0$, se debe resolver el siguiente problema

$$\begin{aligned} &\text{minimizar} \quad \nabla f(\boldsymbol{x})^T \boldsymbol{d}. \\ &\boldsymbol{d} \in \mathbb{R}^n, \quad ||\boldsymbol{d}|| = 1 \end{aligned}$$

Note que si $\boldsymbol{d}$ es una dirección de descenso con $||\boldsymbol{d}|| = 1$, la razón de crecimiento de f en dirección de $\boldsymbol{d}$, a partir de un $\boldsymbol{x}$ en $\mathbb{R}^n$, es $\boldsymbol{d}^T \nabla f(\boldsymbol{x})$. Por la desigualdad de Cauchy-Schwartz se tiene que

$$\boldsymbol{d}^T \nabla f(\boldsymbol{x}) = \langle \boldsymbol{d}, \nabla f(\boldsymbol{x}) \rangle \leq ||\boldsymbol{d}||\, ||\nabla f(\boldsymbol{x})|| = ||\nabla f(\boldsymbol{x})||.$$

La solución de este problema es

$$\boldsymbol{d} = \frac{\nabla f(\boldsymbol{x})}{||\nabla f(\boldsymbol{x})||}.$$

Por lo tanto, se tiene que

$$\boldsymbol{d}^T \nabla f(\boldsymbol{x}) = \frac{(\nabla f(\boldsymbol{x}))^T}{||f(\boldsymbol{x})||} \nabla f(\boldsymbol{x}) = ||\nabla f(\boldsymbol{x})||,$$

es decir se alcanza el máximo.

Note que el máximo decrecimiento se da cuando $\boldsymbol{d} = -\nabla f(\boldsymbol{x})$.

Definición 3.1. *El vector $\boldsymbol{d}$ en $\mathbb{R}^n$ es una dirección de descenso de f, donde f es una función de clase $\mathcal{C}^1$, en el punto $\boldsymbol{x} \in \mathbb{R}^n$ si*

$$\nabla f(\boldsymbol{x})^T \boldsymbol{d} < 0 \text{ si, y solo si, } (-\nabla f(\boldsymbol{x}))^T \boldsymbol{d} > 0.$$

El método de máximo descenso busca definir una sucesión de puntos que tenga la dirección de máximo descenso de la función f.
Suponiendo que se conoce un conjunto de puntos de avance hacia el mínimo $\boldsymbol{x_0}, \boldsymbol{x_1}, \ldots, \boldsymbol{x_k}$, para construir $\boldsymbol{x_{k+1}}$ nos movemos en la dirección $-\nabla f(\boldsymbol{x_k})$ por una cantidad α_k.
Este procedimiento proporciona el algoritmo iterativo

$$\boldsymbol{x_{k+1}} = \boldsymbol{x_k} - \alpha_k \nabla f(\boldsymbol{x_k}), \text{ para cada } k = 0, 1, 2, \ldots$$

La forma general del algoritmo de máximo descenso es
Dado $\boldsymbol{x_k}$ se genera $\boldsymbol{x_{k+1}}$ por medio de

$$\boldsymbol{x_{k+1}} = \boldsymbol{x_k} + \alpha_k \boldsymbol{d_k} \text{ con } k = 0, 1, 2, \ldots$$

donde

- $\boldsymbol{d_k}$ es la dirección de búsqueda.
- α_k es la longitud o tamaño del paso.

Si se cumple $f(\boldsymbol{x_{k+1}}) < f(\boldsymbol{x_k})$, se dice que $\boldsymbol{d_k}$ es una dirección de descenso desde $\boldsymbol{x_k}$.

Algoritmo 3.1.1. *(Método de Máximo descenso)*

1. ***Inicialización.*** *Escoger un valor* $\boldsymbol{x_0}$ *en* $\Omega \in \mathbb{R}^n$.

   Para k =0,1,2,... hasta convergencia hacer lo siguiente:

2. ***Búsqueda de línea.*** *Calcular* $\boldsymbol{d_k} = -\nabla f(\boldsymbol{x_k})$ *y resolver*

$$\alpha_k = \textit{arg} \quad f(\boldsymbol{x_k} + \alpha \boldsymbol{d_k})$$
$$\alpha \geq 0$$

3. ***Descenso.*** *Evaluar* $\boldsymbol{x_{k+1}} = \boldsymbol{x_k} + \alpha_k \boldsymbol{d_k}$.

4. ***Actualización.*** *Hacer* $k = k + 1$ *y volver al paso (2).*

Note que el algoritmo es efectivamente un método de descenso, pues

$$\begin{aligned} f(\boldsymbol{x_{k+1}}) &= f(\boldsymbol{x_{k+1}} + \alpha \boldsymbol{d_k}) = f(\boldsymbol{x_k}) + \alpha_k (\boldsymbol{d_k})^T \nabla f(\boldsymbol{x_k}) + o(\alpha^2) \\ &= f(\boldsymbol{x_k}) - \alpha_k \|\boldsymbol{d_k}\|^2 + o(\alpha_k^2), \text{ pues } \boldsymbol{d_k} = -\nabla f(\boldsymbol{x_k}) \end{aligned}$$

Es decir, $f(\boldsymbol{x_{k+1}}) < f(\boldsymbol{x_k}) = -\boldsymbol{d_k} \neq \boldsymbol{0}$.

Proposición 3.2. *Si* $\{\boldsymbol{x_k}\}_{k=1}^{\infty}$ *es la sucesión de descenso máximo para* $f : \mathbb{R}^n \to \mathbb{R}$*, entonces* $\boldsymbol{x_{k+1}} - \boldsymbol{x_k}$ *es un vector ortogonal a* $\boldsymbol{x_{k+2}} - \boldsymbol{x_{k+1}}$ *para k=0,1,2,...*

Demostración. Como $\boldsymbol{x_{k+1}} = x_k + \alpha_k \boldsymbol{d_k}$ con $\boldsymbol{d_k} = -\nabla f(\boldsymbol{x_k})$, entonces veamos que se cumple la siguiente expresión

$$\langle \boldsymbol{x_{k+2}} - \boldsymbol{x_{k+1}}, \boldsymbol{x_{k+1}} - \boldsymbol{x_k} \rangle = \alpha_{k+1} \alpha_k \langle \boldsymbol{d_{k+1}}, \boldsymbol{d_k} \rangle = 0.$$

Sea $\phi(\alpha) = f(\boldsymbol{x}(\alpha)) = f(\boldsymbol{x_k} + \alpha \boldsymbol{d_k})$. Como α_k es el argumento mínimo de $\phi_k(\alpha)$, entonces

$$0 = \phi^{'}(\alpha_k) = \nabla f(\boldsymbol{x}(\alpha_k))^T \boldsymbol{d_k} = \nabla f(\boldsymbol{x_{k+1}})^T \nabla f(\boldsymbol{x_k}) = (\boldsymbol{d_{k+1}})^T \boldsymbol{d_k}.$$

□

Proposición 3.3. *Si* $\{\boldsymbol{x_k}\}_{k=1}^{\infty}$ *es la sucesión de descenso máximo para* $f : \mathbb{R}^n \to \mathbb{R}$ *y* $\nabla f(\boldsymbol{x_k}) \neq 0$*, entonces* $f(\boldsymbol{x_{k+1}}) < f(\boldsymbol{x_k})$*.*

Demostración. Sea $\boldsymbol{x_{k+1}} = \boldsymbol{x_k} + \alpha_k \boldsymbol{d_k}$ con $\boldsymbol{d_k} = -\nabla f(\boldsymbol{x_k})$.Como

$$\alpha_k = \text{arg } \Phi_k(\alpha) \text{ con } \alpha \geq 0,$$

entonces $\Phi_k(\alpha_k) \leq \Phi_k(\alpha)$, para todo $\alpha \geq 0$.

$$\begin{aligned} \Phi_k^{'}(0) &= \left.\frac{\partial \Phi_k}{\partial \alpha}\right|_{\alpha=0} = (\boldsymbol{d_k})^T \nabla f(\boldsymbol{x_k} + \alpha \boldsymbol{d_k})\Big|_{\alpha=0} = -(\boldsymbol{d_k})^T \boldsymbol{d_k} \\ &= -||\boldsymbol{d_k}||^2 < 0, \text{pues } \boldsymbol{d_k} = -\nabla f(\boldsymbol{x_k}) \neq \boldsymbol{0}. \end{aligned}$$

Por continuidad, debe existir un $\hat{\alpha} > 0$ tal que

$$\Phi_k(\alpha) < \Phi_k(0), \text{ para todo } \alpha \in (0, \hat{\alpha}).$$

Por lo tanto, $f(\boldsymbol{x_{k+1}}) = \Phi_k(\alpha_k) \leq \Phi_k(\alpha) < \Phi_k(0) = f(\boldsymbol{x_k})$ es una dirección de descenso. □

Note que si $\nabla f(\boldsymbol{x_k}) = 0$ para algún k, entonces no se desciende, y además $\boldsymbol{x_\star} = \boldsymbol{x_k}$ es un punto de mínimo local.

Teorema 3.4. *Sea* $f : \mathbb{R}^n \to \mathbb{R}$ *de clase* $\mathcal{C}^1$ *y* $\boldsymbol{x_0} \in \mathbb{R}^n$*. EL vector* $\nabla f(\boldsymbol{x_0})$ *es ortogonal al vector tangente de una curva arbitaria que pasa por* $\boldsymbol{x_0}$ *y además se encuentra sobre el conjunto de curvas de nivel* $\mathcal{S}_c$ *determinado por* $c = f(\boldsymbol{x_0})$*.*

Demostración. Sea $\mathcal{S}_c = \{\boldsymbol{x} \in \mathbb{R}^n : f(\boldsymbol{x}) = c\}$ el conjunto de curvas de nivel (1.17) y $g : (a, b) \to \mathbb{R}^n$ una curva contenida en $\mathcal{S}_c$, con $g(t_0) = \boldsymbol{x_0}$ para algún $t \in (a, b)$. Se tiene que

- $f(g(t)) = c$ ya que $g(t) \in \mathcal{S}_c$ para todo $t \in (a, b)$.
- $\dfrac{\partial}{\partial t} f(g(t)) = 0$, entonces $\nabla f(g(t))^T g'(t) = 0$.
- En particular $\boldsymbol{x_0} = g(t_0)$ y $\nabla f(\boldsymbol{x_0})^T g'(t_0) = 0$.

Así, el plano tangente a $\mathcal{S}_c$ en $\boldsymbol{x_0}$ es

$$\nabla f(\boldsymbol{x_0})^T (\boldsymbol{x} - \boldsymbol{x_0}) = 0, \text{ si } \nabla f(\boldsymbol{x_0}) \neq 0.$$

□

Ejemplo 3.1. *Se va a implementar el método de máximo descenso, para encontrar el mínimo de la función definida como*

$$f(x, y) = 2x^2 + 2xy + 3y^2 - 4x + 8y + 12 \tag{24}$$

Tomando como punto inicial $\boldsymbol{x_0} = (0, 0)^T$. Primero se realizan tres iteraciones y se llegará al mínimo mediante la codificación del método de descenso en el lenguaje de programación MATLAB. Note que el gradiente es

$$\nabla f(x, y) = \begin{bmatrix} 4x + 2y - 4 \\ 2x + 6y + 8 \end{bmatrix}$$

En la iteración 1 se tiene que

- $\boldsymbol{d_0}$= - $\nabla f(0, 0) = (4, -8)^T$.
- $\boldsymbol{x(\alpha)} = \boldsymbol{x_0} + \alpha \boldsymbol{d_0} = (4\alpha, -8\alpha)^T$.
- $\Phi(\alpha) = f(\boldsymbol{x}(\alpha)) = f(4\alpha, -8\alpha) = 160\alpha^2 - 8\alpha + 12$ *tiene un mínimo en* $\alpha_0 = \dfrac{1}{4}$.
- $\boldsymbol{x_1} = \boldsymbol{x_0} + \alpha_0 \boldsymbol{d_0} = (0, 0)^T + \dfrac{1}{4}(4, -8)^T = (1, -2)^T$.

En la iteración 2 se obtiene que

- $\boldsymbol{d_1} = -\nabla f(\boldsymbol{x_1}) = -\nabla f(1,-2) = (4,2)^T$.
- $\boldsymbol{x}(\alpha) = \boldsymbol{x_1} + \alpha \boldsymbol{d_1} = (1+4\alpha, -2+2\alpha)^T$.
- $\Phi(\alpha) = f(\boldsymbol{x}(\alpha)) = f(1+4\alpha,-2+2\alpha) = 60\alpha^2 - 20\alpha + 2$ *tiene un mínimo en* $\alpha_1 = \dfrac{1}{6}$.
- $\boldsymbol{x_2} = \boldsymbol{x_1} + \alpha_1 \boldsymbol{d_1} = (1,-2)^T + \dfrac{1}{6}(4,2)^T = \left(\dfrac{5}{3}, -\dfrac{5}{3}\right)^T$.

Seguidamente, en la iteración 3 se tiene que

- $\boldsymbol{d_2} = -\nabla f(\boldsymbol{x_2}) = -\nabla f\left(\dfrac{5}{3}, -\dfrac{5}{3}\right) = \left(\dfrac{2}{3}, -\dfrac{4}{3}\right)^T$.
- $\boldsymbol{x}(\alpha) = \boldsymbol{x_2} + \alpha \boldsymbol{d_2} = \left(\dfrac{5+2\alpha}{3}, \dfrac{-5-4\alpha}{3}\right)^T$.
- $\Phi(\alpha) = f(\boldsymbol{x}(\alpha)) = \dfrac{40\alpha^2 - 20\alpha + 3}{9}$ *tiene un mínimo en* $\alpha_2 = \dfrac{1}{4}$.
- $\boldsymbol{x_3} = \boldsymbol{x_2} + \alpha_2 \boldsymbol{d_2} = \left(\dfrac{5}{3}, -\frac{5}{3}\right)^T + \dfrac{1}{4}\left(\dfrac{2}{3}, -\dfrac{4}{3}\right)^T = \left(\dfrac{11}{6}, -2\right)^T$.

Note que la función (24) se puede escribir como:

$$f(x,y) = \frac{1}{2}(x,y)\begin{pmatrix} 4 & 2 \\ 2 & 6 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} - (4,-8)\begin{pmatrix} x \\ y \end{pmatrix} + 12,$$

donde el gradiente es (25), la longitud de paso α_k *definida en el ejemplo (2.2) y la norma vectorial (2) se llega al mínimo en la iteración* $\boldsymbol{x}_{10} = (2,-2)$*, con una tolerancia (tol* $= 10^{-6}$*), donde el criterio de parada es* $\|\nabla f(\boldsymbol{x_k})\| <$ *tol y n <* *max, n es el número de iteraciones y max es el valor máximo de iteraciones dado por el usuario. Ver en Tabla 1.*

k	$\boldsymbol{x_k}$	α_k	$\|\|\nabla f(\boldsymbol{x_k})\|\|$	$f(\boldsymbol{x_k})$
0	(0.000, 0.000)	0.250	8.944	12.00000
1	(1.000, -2.000)	0.167	4.472	2.00000
2	(1.667, -1.667)	0.250	1.491	0.33333
3	(1.833, -2.000)	0.167	0.745	0.05556
4	(1.944, -1.944)	0.250	0.248	0.00926
5	(1.972, -2.000)	0.167	0.124	0.00154
6	(1.991, -1.991)	0.250	0.041	0.00026
7	(1.995, -2.000)	0.167	0.021	0.00004
8	(1.998, -1.998)	0.250	0.007	0.00001
9	(1.999, -2.000)	0.167	0.003	0.00000
10	(2.000, -2.000)	0.250	0.001	0.00000
11	(2.000, -2.000)	0.167	0.001	0.00000
12	(2.000, -2.000)	0.250	0.000	0.00000
13	(2.000, -2.000)	0.167	0.000	0.00000
14	(2.000, -2.000)	0.250	0.000	0.00000
15	(2.000, -2.000)	0.167	0.000	0.00000
16	(2.000, -2.000)	0.250	0.000	0.00000
17	(2.000, -2.000)	0.167	0.000	0.00000
18	(2.000, -2.000)	0.250	0.000	0.00000

Tabla 1. Implementación del Método de Máximo Descenso con búsqueda de línea exacta de la función (24). Hecho por el autor.

Note que el Método iterativo de Máximo Descenso

$$f(\boldsymbol{x_0}) = 12 > f(\boldsymbol{x_1}) = 2 > f(\boldsymbol{x_2}) = \frac{1}{3} > f(\boldsymbol{x_3}) = \frac{1}{18}$$

es una dirección de descenso.

Ejemplo 3.2. *En este ejemplo se implementa el Método de Máximo Descenso sobre la función* $f : \mathbb{R}^2 \to \mathbb{R}$ *definida por*

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2 \tag{25}$$

conocida como la función de Rosenbrock, con el objetivo de usar MATLAB para encontrar el mínimo.

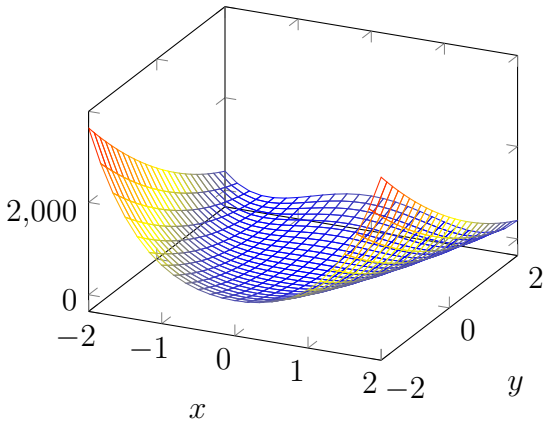

Figura 2. Representación gráfica de la función $f(x, y)$. Hecho por el autor.

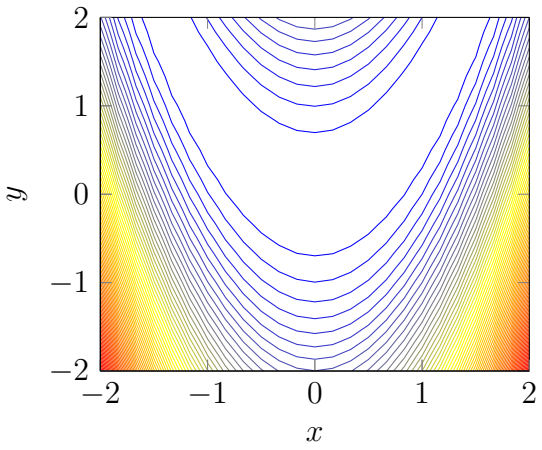

Figura 3. Representación de las curvas de nivel de la función $f(x, y)$. Hecho por el autor.

El gradiente se define como

$$\nabla f(x, y) = \begin{bmatrix} -2(1 - x) - 400x(y - x^2) \\ 200(y - x^2) \end{bmatrix}. \tag{26}$$

Tomando como punto inicial $\boldsymbol{x_0} = (-0.5, 0.5)^T$ y la longitud de paso $\alpha_k = \arg\ \Phi(\alpha)$ con $\alpha > 0$, mediante búsqueda lineal inexacta (2.2), se tienen los resultados. Ver en la Tabla 2.

k	$\boldsymbol{x_k}$	α_k	$\|\|\nabla f(\boldsymbol{x_k})\|\|$	$f(\boldsymbol{x_k})$
1	(-0.500, 0.500)	0.002	68.622	8.50000
2	(-0.684, 0.305)	0.004	57.841	5.47878
3	(-0.590, 0.368)	0.002	4.261	2.56827
4	(-0.596, 0.353)	0.004	3.869	2.54827
5	(-0.581, 0.355)	0.004	3.499	2.52901
6	(-0.584, 0.341)	0.004	3.146	2.50985
7	(-0.572, 0.341)	0.004	2.848	2.49131
8	(-0.573, 0.319)	0.008	5.511	2.48191
9	⋮	⋮	⋮	⋮
12413	(1.000, 1.000)	0.002	0.000	0.00000
12414	(1.000, 1.000)	0.002	0.000	0.00000
12415	(1.000, 1.000)	0.002	0.000	0.00000
12415	(1.000, 1.000)	0.002	0.000	0.00000

Tabla 2. Implementación del Método de Máximo Descenso con búsqueda de linea inexacta de la función (25).

Los resultados del método de Máximo Descenso implementado en MATLAB Tabla 2 tienen una tolerancia (tol $= 10^{-6}$), donde el criterio de parada es $\|\nabla f(\boldsymbol{x_k})\| <$ tol y *n* $<$ *max* , donde *n* es el número de iteraciones, *max* es el valor máximo de iteraciones dado por el usuario, la norma vectorial es (2) y α_k para cada iteración cumple con la condición (2.2) y además, el mínimo se encuentra en la iteración $\boldsymbol{x}_{12413} = (1, 1)^T$.

3.2. Método de Newton

Sea $f : \mathbb{R}^n \to \mathbb{R}$ una función de clase $\mathcal{C}^2$. Un problema de minimización sin restricciones consiste en resolver el siguiente problema

$$\begin{aligned} &\text{minimizar} \quad f(\boldsymbol{x}). \\ &\boldsymbol{x} \in \mathbb{R}^n \end{aligned} \tag{27}$$

Una estrategia para abordar el problema (27) consiste en construir en cada iteración $\boldsymbol{x_k} \in \mathbb{R}^n$ un modelo cuadrático " local" con respecto a la función f alrededor de $\boldsymbol{x_k}$,

$$h(\boldsymbol{x}) = f(\boldsymbol{x_k}) + \nabla f(\boldsymbol{x_k})(\boldsymbol{x} - \boldsymbol{x_k}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x_k})^T \nabla^2 f(\boldsymbol{x_k})^T (\boldsymbol{x} - \boldsymbol{x_k}).$$

Note que la función $h(\boldsymbol{x})$ aproxima a $f(\boldsymbol{x})$ alrededor de $\boldsymbol{x_k}$, así que para resolver el problema (27) se transforma en una sucesión de problemas de la forma

$$\begin{aligned} &\text{minimizar} \quad h(\boldsymbol{x}), \\ &\boldsymbol{x} \in \mathbb{R}^n \end{aligned}$$

lo que es equivalente a resolver un sistema de ecuaciones lineales. Si escogemos $\boldsymbol{x_{k+1}}$ = arg $h(\boldsymbol{x})$ con $\boldsymbol{x} \in \Omega \subset \mathbb{R}^n$, entonces por las condiciones de primer orden (1.34), se tiene que $\boldsymbol{x_k}$ es un mínimo local, por lo tanto $\nabla h(\boldsymbol{x_{k+1}}) = \boldsymbol{0}$; aplicando (2.5) se tiene que

$$\nabla^2 f(\boldsymbol{x_k})(\boldsymbol{x_{k+1}} - \boldsymbol{x_k}) + \nabla f(\boldsymbol{x_{k+1}}) = \boldsymbol{0}.$$

De donde se obtiene el algoritmo de Newton

$$\begin{aligned} \nabla^2 f(\boldsymbol{x_k})\boldsymbol{d_k} &= -\nabla f(\boldsymbol{x_k}) \\ \boldsymbol{x_{k+1}} &= \boldsymbol{x_k} + \boldsymbol{d_k}. \end{aligned}$$

La expresión (28) se conoce como iteración de Newton y está bien definida siempre y cuando la matriz $\nabla^2 f(\boldsymbol{x_k})$ es no singular o definida positiva.
Si $\boldsymbol{d_k} = -(\nabla^2 f(\boldsymbol{x_k}))^{-1}\nabla f(\boldsymbol{x_k})$ y $\alpha_k = 1$ en el método de búsqueda lineal, se tiene la forma general del **Método de Newton**. Si f no es una función cuadrática entonces como $x_\star$= arg $f(\boldsymbol{x})$ con $\boldsymbol{x} \in \Omega$ es un mínimo local, entonces $\nabla f(\boldsymbol{x_\star}) = 0$. Ahora, el problema se convierte en resolver un sistema de ecuaciones no lineales $\nabla f(\boldsymbol{x}) = 0$ ya que f no es cuadrática, aplicando el método de Newton se tiene que

$$\boldsymbol{x_{k+1}} = \boldsymbol{x_k} - \left(\nabla^2 f(\boldsymbol{x_k})\right)^{-1} \nabla f(\boldsymbol{x_k}), \text{ para } k = 0, 1, \ldots$$

Proposición 3.5. *Sea $\{x_k\}_{k=1}^{\infty}$ la sucesión generada por el método de Newton para minimizar la función objetivo $f : \mathbb{R}^n \to \mathbb{R}$. Si $\nabla^2 f(\boldsymbol{x_k}) > \mathbf{0}$ y $\nabla f(\boldsymbol{x_k}) \neq \mathbf{0}$, entonces*

1. $\boldsymbol{d_k} = -(\nabla^2 f(\boldsymbol{x_k}))^{-1}\nabla f(\boldsymbol{x_k}) \neq \mathbf{0}$.
2. $\boldsymbol{d_k}$ *es una dirección de descenso.*

Demostración. Dada la función a minimizar $f : \mathbb{R}^n \to \mathbb{R}$ y un punto $\boldsymbol{x_k}$, su expansión de Taylor centrada en $\boldsymbol{x_k}$ es

$$f(\boldsymbol{x_k}) = f(\boldsymbol{x_k}) + \nabla f(\boldsymbol{x_k})^T(\boldsymbol{x} - \boldsymbol{x_k}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x_k})^T\nabla^2 f(\boldsymbol{x_k})(\boldsymbol{x} - \boldsymbol{x_k}) + o(\|\boldsymbol{x} - \boldsymbol{x_k}\|^2).$$

La función cuadrática que aproxima a $f(\boldsymbol{x})$ cerca de $\boldsymbol{x_k}$ es

$$q(\boldsymbol{x}) = f(\boldsymbol{x_k}) + \nabla f(\boldsymbol{x_k})^T(\boldsymbol{x} - \boldsymbol{x_k}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x_k})^T\nabla^2 f(\boldsymbol{x_k})(\boldsymbol{x} - \boldsymbol{x_k}).$$

Se escoge $\boldsymbol{x_{k+1}} = \text{argmin } q(\boldsymbol{x})$ con $\boldsymbol{x} \in \Omega$, por condiciones de primer orden (1.34) se tiene que

$$\nabla q(\boldsymbol{x_{k+1}}) = \nabla f(\boldsymbol{x_k}) + \nabla^2 f(\boldsymbol{x_k})(\boldsymbol{x_{k+1}} - \boldsymbol{x_k}) = 0.$$

Así, se tiene que

$$\boldsymbol{x_{k+1}} = \boldsymbol{x_k} - (\nabla^2 f(\boldsymbol{x_k}))^{-1} \nabla f(\boldsymbol{x_k}).$$

Por lo tanto

$$\boldsymbol{x_{k+1}} - \boldsymbol{x_k} = \boldsymbol{d_k} = -(\nabla^2 f(\boldsymbol{x_k}))^{-1} \nabla f(\boldsymbol{x_k}).$$

Como $\nabla^2 f(\boldsymbol{x_k}) > 0$ y $\nabla f(\boldsymbol{x_k}) \neq 0$, entonces $\boldsymbol{d_k} \neq 0$. Así, **se ha probado el inciso 1**. Por otro lado, consideremos el algoritmo de Newton $\boldsymbol{x_{k+1}} = \boldsymbol{x_k} + \alpha_k \boldsymbol{d_k}$ con $\boldsymbol{d_k} = -(\nabla^2 f(\boldsymbol{x_k}))^{-1} \nabla f(\boldsymbol{x_k})$ donde $\boldsymbol{d_k}$ es el mínimo $\Phi_k(\alpha) = f(\boldsymbol{x_k} + \alpha \boldsymbol{d_k})$, entonces se mostrará que $\Phi(\alpha) < \Phi(0)$ para algún intervalo $(0, \hat{\alpha})$ con $\Phi_k(\alpha) = f(\boldsymbol{x_k} + \alpha \boldsymbol{d_k})$ y $\Phi_k(0) = f(\boldsymbol{x_k})$, usando la regla de la cadena se tiene que $\Phi_k'(\alpha) = (\boldsymbol{d_k})^T \nabla f(\boldsymbol{x_k} + \alpha \boldsymbol{d_k})$ con $\Phi_k'(0) = (\boldsymbol{d_k})^T \nabla f(\boldsymbol{x_k})$, como $\boldsymbol{d_k} = -(\nabla^2 f(\boldsymbol{x_k}))^{-1} \nabla f(\boldsymbol{x_k})$, entonces $\nabla f(\boldsymbol{x_k}) = -\nabla^2 f(\boldsymbol{x_k}) \boldsymbol{d_k}$ luego $\Phi_k'(0) = -(\boldsymbol{d_k})^T \nabla 2 f(\boldsymbol{x_k}) \boldsymbol{d_k}$ así $\Phi_k'(0) < 0$ ya que $\boldsymbol{d_k} \neq \boldsymbol{0}$ y $\nabla^2 f(\boldsymbol{x_k}) > 0$ por lo tanto Φ es una función decreciente en $\alpha = 0$. Por continuidad existe α_k tal que $\Phi(\alpha) < \Phi(0)$ para todo $\alpha \in (0, \alpha_k)$, es decir

$$f(\boldsymbol{x_k} + \boldsymbol{\alpha d_k}) < f(\boldsymbol{x_k}) \text{ para todo } \alpha \in (0, \hat{\alpha}).$$

En particular $f(\boldsymbol{x_{k+1}}) = f(\boldsymbol{x_k} + \alpha \boldsymbol{d_k}) < f(\boldsymbol{x_k})$, entonces $\boldsymbol{d_k}$ es una dirección de descenso. Así, **se ha probado inciso 2**. □

Algoritmo 3.2.1. *(Método de Newton)*

1. ***Inicialización.*** *Escoger un valor $\boldsymbol{x_0} \in \Omega$, donde $\Omega \subset \mathbb{R}^n$.*

   Para $k = 0, 1, 2 \ldots$ hasta convergencia hacer lo siguiente:

2. *Resolver el sistema*

$$\nabla^2 f(\boldsymbol{x_k}) \boldsymbol{d_k} = -\nabla f(\boldsymbol{x_k}).$$

3. ***Descenso.*** *Evaluar*

$$\boldsymbol{x_{k+1}} = \boldsymbol{x_k} + \boldsymbol{d_k}.$$

4. ***Actualización.*** *Hacer k= k+1 y volver al paso (2).*

El algoritmo del método de Newton en el paso (3) se puede actualizar de la forma $\boldsymbol{x_{k+1}} = \boldsymbol{x_k} + \alpha_k \boldsymbol{d_k}$ y para calcular el α_k se usa búsqueda lineal (2.2) y (2.1).
El siguiente teorema garantiza que bajo ciertas hipótesis la sucesión generada por el algoritmo de Newton, es convergente. Además, con una hipótesis adicional se logra convergencia cuadrática.

Teorema 3.6. *Sea $F : \mathbb{R}^n \to \mathbb{R}$ una función de clase $\mathcal{C}^2$ y un abierto convexo $\mathcal{D} \subseteq \mathbb{R}^n$ (1.36). Supongamos que*

h1) Existe $\boldsymbol{x_\star} \in \mathbb{R}^n$ tal que $\nabla F(\boldsymbol{x_\star}) = 0$.

h2) Existe $\beta > 0$ y $(\nabla^2 F(\boldsymbol{x_\star}))^{-1}$ tal que $|||(\nabla^2 F(\boldsymbol{x_\star}))^{-1}||| \leq \beta$.

h3) Existe $r > 0$ tal que $\nabla^2 F \in Lip_\gamma(B(\boldsymbol{x_\star}; r))$, donde $B(\boldsymbol{x_\star}; r) \subset D$.

Entonces, existe $\varepsilon > 0$ tal que para todo $x_0 \in B(\boldsymbol{x_\star}; \varepsilon)$, la sucesión $\{\boldsymbol{x_n}\}_{n=1}^{\infty}$ generada por

$$\boldsymbol{x_{k+1}} = \boldsymbol{x_k} - (\nabla^2 F(\boldsymbol{x_k}))^{-1} \nabla F(\boldsymbol{x_k}), \text{ para } k = 0, 1, \ldots$$

está bien definida, converge a $\boldsymbol{x_\star}$ y satisface

$$\|\boldsymbol{x_{k+1}} - \boldsymbol{x_\star}\| \leq \beta\gamma \, \|\boldsymbol{x_k} - \boldsymbol{x_\star}\|^2, \text{ para } k = 0, 1, \ldots$$

Demostración. Sea $\varepsilon = \min\{r, \frac{1}{2\beta\gamma}\}$. Se procede por inducción sobre k, que para cada paso se tiene que

$$\|\boldsymbol{x_{k+1}} - \boldsymbol{x_\star}\| \leq \beta\gamma \, \|\boldsymbol{x_k} - \boldsymbol{x_\star}\|^2, \; k = 0, 1, \ldots$$

Sea $\|\boldsymbol{x_0} - \boldsymbol{x_\star}\| \leq \varepsilon$ y como $\nabla^2 F(\boldsymbol{x_0})$ es Lipschitz por (h3) y (h2), se tiene que

$$\begin{aligned} |||(\nabla^2 F(x_\star)^{-1})[\nabla^2 F(\boldsymbol{x_0}) - \nabla^2 F(\boldsymbol{x_\star})]||| &\leq |||(\nabla^2 F(\boldsymbol{x_\star}))^{-1}||| \; |||\nabla^2 F(\boldsymbol{x_0}) - \nabla^2 F(\boldsymbol{x_\star})||| \\ &\leq \gamma \; |||(\nabla^2 F(\boldsymbol{x_\star})^{-1}||| \; ||\boldsymbol{x_0} - \boldsymbol{x_\star}|| \\ &\leq \gamma\beta \; ||\boldsymbol{x_0} - \boldsymbol{x_\star}|| = \gamma\beta \; \left(\frac{1}{2\gamma\beta}\right) = \frac{1}{2}, \end{aligned}$$

por Lema (8) se tiene que

$$\begin{aligned} |||(\nabla^2 F(\boldsymbol{x_0})^{-1}||| &\leq \frac{|||(\nabla^2 F(\boldsymbol{x_\star}))^{-1}|||}{1 - |||(\nabla^2 F(\boldsymbol{x_\star}))^{-1}[\nabla^2 F(\boldsymbol{x_0}) - \nabla^2 F(\boldsymbol{x_\star})] \; |||} \\ &\leq 2 \; |||\nabla^2 F(\boldsymbol{x_\star})||| \\ &\leq 2\beta. \end{aligned} \tag{28}$$

Así, $\nabla^2 F(\boldsymbol{x_0})$ es no singular; es decir está bien definida, y además

$$\begin{aligned} \boldsymbol{x_1} &= \boldsymbol{x_0} - (\nabla^2 F(\boldsymbol{x_0}))^{-1} \; \nabla F(\boldsymbol{x_0}) \\ \boldsymbol{x_1} - \boldsymbol{x_\star} &= \boldsymbol{x_0} - \boldsymbol{x_\star} - (\nabla^2 F(\boldsymbol{x_0}))^{-1} \; \nabla F(\boldsymbol{x_0}) \\ &= \boldsymbol{x_0} - \boldsymbol{x_\star} - (\nabla^2 F(\boldsymbol{x_0})^{-1} \; [\nabla F(\boldsymbol{x_0}) - \nabla F(\boldsymbol{x_\star})], \end{aligned}$$

luego

$$\begin{aligned} \|\boldsymbol{x_1} - \boldsymbol{x_\star}\| &= |||(\nabla^2 F(\boldsymbol{x_0}))^{-1} \; [\nabla F(\boldsymbol{x_\star}) - \nabla F(\boldsymbol{x_0}) - \nabla^2 F(\boldsymbol{x_0})(\boldsymbol{x_\star} - \boldsymbol{x_0})]||| \\ &\leq |||(\nabla^2 F(\boldsymbol{x_0})^{-1})||||||\nabla F(\boldsymbol{x_\star}) - \nabla F(\boldsymbol{x_0}) - \nabla^2 F(\boldsymbol{x_0})(\boldsymbol{x_\star} - \boldsymbol{x_0})|||, \end{aligned} \tag{29}$$

por Lema (1.19) se tiene que

$$|||\nabla F(\boldsymbol{x_\star}) - \nabla F(\boldsymbol{x_0}) - (\nabla^2 F(\boldsymbol{x_0})^{-1}(\boldsymbol{x_\star} - \boldsymbol{x_0})||| \leq \frac{\gamma}{2}\|\boldsymbol{x_\star} - \boldsymbol{x_0}\|^2, \tag{30}$$

por (28) y (30) en (29) se tiene que

$$\begin{aligned} \|\boldsymbol{x_1} - \boldsymbol{x_\star}\| \leq (2\beta)\ \left(\frac{\gamma}{2}\right) \|\boldsymbol{x_\star} - \boldsymbol{x_0}\|^2 &= \beta\gamma\ \|\boldsymbol{x_\star} - \boldsymbol{x_0}\|^2 \\ &\leq \frac{1}{2}\ \|\boldsymbol{x_\star} - \boldsymbol{x_0}\| \leq \frac{\varepsilon}{2} < \varepsilon, \end{aligned} \tag{31}$$

por lo tanto, es válido para el caso $k = 0$. Supongamos que en la hipótesis de inducción se cumple para el caso de $n-1$, es decir que

$$\|\boldsymbol{x_{n-1}} - \boldsymbol{x_\star}\| \leq \|\boldsymbol{x_{n-2}} - \boldsymbol{x_\star}\| \leq \cdots \leq \frac{1}{2}\ \|\boldsymbol{x_0} - \boldsymbol{x_\star}\|,$$

tales que $\nabla^2 F(\boldsymbol{x_{n-1}})$ es no singular. Veamos que se cumple el paso inductivo para todo $n \in \mathbb{N}$. Note que

$$|||(\nabla^2 F(\boldsymbol{x_\star})^{-1})\ [\nabla^2 F(\boldsymbol{x_n}) - \nabla^2 F(\boldsymbol{x_\star})]|||| \leq \gamma\beta\ \|\boldsymbol{x_n} - \boldsymbol{x_\star}\| = \gamma\beta\ \left(\frac{1}{2\gamma\beta}\right) = \frac{1}{2},$$

por Lema (8) $\nabla^2 F(\boldsymbol{x_n})$ se tiene que es no singular; es decir está bien definida, y además

$$\begin{aligned} |||(\nabla^2 F(\boldsymbol{x_n})^{-1}||| &\leq \frac{|||(\nabla^2 F(\boldsymbol{x_\star}))^{-1}|||}{1 - |||(\nabla^2 F(\boldsymbol{x_\star}))^{-1}\ [\nabla^2 F(\boldsymbol{x_n}) - \nabla^2 F(\boldsymbol{x_\star})]\ |||} \\ &\leq 2\beta, \end{aligned} \tag{32}$$

luego

$$\|\boldsymbol{x_n} - \boldsymbol{x_\star}\| \leq |||(\nabla^2 F(\boldsymbol{x_{n-1}}))^{-1}||||||\nabla F(\boldsymbol{x_\star}) - \nabla F(\boldsymbol{x_{n-1}}) - \nabla^2 F(\boldsymbol{x_{n-1}})(\boldsymbol{x_\star} - \boldsymbol{x_{n-1}})|||, \tag{33}$$

por Lema (1.19) se tiene que

$$|||\nabla F(\boldsymbol{x_\star}) - \nabla F(\boldsymbol{x_n}) - (\nabla^2 F(\boldsymbol{x_n})^{-1}(\boldsymbol{x_\star} - \boldsymbol{x_0})||| \leq \frac{\gamma}{2}\ \|\boldsymbol{x_\star} - \boldsymbol{x_n}\|^2, \tag{34}$$

por (32) y (34) en (33) se tiene que

$$\|\boldsymbol{x_n} - \boldsymbol{x_\star}\| \leq (2\beta) \left(\frac{\gamma}{2}\right) \|\boldsymbol{x_\star} - \boldsymbol{x_{n-1}}\|^2 \;\; = \;\; \beta\gamma \, \|\boldsymbol{x_\star} - \boldsymbol{x_{n-1}}\|^2, \tag{35}$$

y por la hipótesis de inducción en (32), se concluye que

$$\|\boldsymbol{x_n} - \boldsymbol{x_\star}\| \leq \|\boldsymbol{x_{n-1}} - \boldsymbol{x_\star}\| \leq \|\boldsymbol{x_{n-2}} - \boldsymbol{x_\star}\| \leq \cdots \leq \frac{1}{2} \, \|\boldsymbol{x_0} - \boldsymbol{x_\star}\|. \tag{36}$$

Así, se tiene que

$$\|\boldsymbol{x_n} - \boldsymbol{x_\star}\| \leq \beta\gamma \, \|\boldsymbol{x_\star} - \boldsymbol{x_{n-1}}\|^2.$$

Por lo tanto, se cumple el paso inductivo para todo $n \in \mathbb{N}$. □

En la demostración (3.2) se usa la norma vectorial $||\cdot||_2$ y la norma matricial inducida por la norma $|| \cdot ||_2$., la cual es la norma de Frobenius. Note que el algoritmo de Newton funciona si se encuentra cerca a la solución, por lo tanto se dice que tiene un comportamiento local.

Proposición 3.7. *El método de Newton converge en un solo paso al mínimo de una función cuadrática* $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T Q\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x} + c$ *con* $Q^T = Q > 0$ *independiente del punto inicial* $\boldsymbol{x_0}$.

Demostración. Como f es cuadrática se tiene que $\nabla^2 f(\boldsymbol{x}) = Q$, es decir, es constante y $\nabla f(\boldsymbol{x_0}) = Q\boldsymbol{x_0} - b$. Sea $\boldsymbol{x_0}$ el punto inicial, entonces por el Algoritmo de Newton se tiene que

$$\begin{aligned}
\boldsymbol{x_1} &= \boldsymbol{x_0} - (\nabla^2 f(\boldsymbol{x_0}))^{-1} \nabla f(\boldsymbol{x_0}) \\
&= \boldsymbol{x_0} - (Q)^{-1}(Q\boldsymbol{x_0} - b) \\
&= \boldsymbol{x_0} - Q^{-1}Q\boldsymbol{x_0} + Q^{-1}b \\
&= Q^{-1}b = \boldsymbol{x_\star}.
\end{aligned}$$

Por lo tanto, el método de Newton converge en una iteración.

Si se considera $\boldsymbol{x_{k+1}} = \boldsymbol{x_k} + \alpha \boldsymbol{d_k}$ con $\boldsymbol{d_k} = -(\nabla^2 f(\boldsymbol{x_k}))^{-1} \nabla f(\boldsymbol{x_k})$, el algoritmo se reduce al método de Newton donde α_k es el mínimo de $\Phi_k(\alpha) = f(\boldsymbol{x_k} + \alpha \boldsymbol{d_k})$ es decir $\alpha_k = 1$ para todo $k \in \{0, 1, \ldots, n\}$ y por lo tanto el método converge también en un solo paso. Se escoge $\boldsymbol{x_0} \in \Omega \subset \mathbb{R}^n$, para resolver $\boldsymbol{d_k}$ con el sistema $\nabla^2 f(\boldsymbol{x_k}) \boldsymbol{d_k} = -\nabla f(\boldsymbol{x_k})$ como la función $f(\boldsymbol{x})$ es cuadrática la matriz Hessiana es constante, es decir $\nabla^2 f(\boldsymbol{x_\star}) = Q$, entonces $-\nabla f(\boldsymbol{x_k}) = \nabla^2 f(\boldsymbol{x_k}) \boldsymbol{d_k} = Q \boldsymbol{d_k}$, además $Q^T = Q > 0$, entonces Q es invertible luego $\boldsymbol{d_k} = -Q^{1-} \nabla f(\boldsymbol{x_k})$. Se calcula $\alpha_k = \arg \Phi(\alpha) = f(\boldsymbol{x_k} + \alpha \boldsymbol{d_k})$ como el mínimo α_k satisface que $\Phi(\alpha) = 0$, luego

$$\begin{aligned} \Phi_k(\alpha) &= (\boldsymbol{d})^T \nabla f(\boldsymbol{x_k} + \alpha \boldsymbol{d_k}) = (\boldsymbol{d_k})^T (Q(\boldsymbol{x_k} + \alpha \boldsymbol{d_k}) - b \\ &= (\boldsymbol{d_k})^T (Q \boldsymbol{x_k} - b + \alpha Q \boldsymbol{d_k} = (\boldsymbol{d_k})^T \nabla f(\boldsymbol{x_k}) - \alpha (\boldsymbol{d_k})^T Q \boldsymbol{d_k}, \end{aligned}$$

despejando α e igualando a cero, se obtiene α_k

$$\begin{aligned} \alpha_k &= \frac{-(\boldsymbol{d_k})^T \nabla f(\boldsymbol{x_k})}{(\boldsymbol{d_k})^T Q \boldsymbol{d_k}} = \frac{-(-Q^{-1} \nabla f(\boldsymbol{x_k}))^T \nabla f(\boldsymbol{x_k})}{(-Q^{-1} \nabla f(\boldsymbol{x_k}))^T Q (-Q^{-1} \nabla f(\boldsymbol{x_k})} \\ &= \frac{(Q^{-1} \nabla f(\boldsymbol{x_k}))^T \nabla f(\boldsymbol{x_k})}{(Q^{-1} \nabla f(\boldsymbol{x_k}))^T \nabla f(\boldsymbol{x_k})} = 1. \end{aligned}$$

Evaluando $\boldsymbol{x_{k+1}} = \boldsymbol{x_k} + \alpha \boldsymbol{d_k} = \boldsymbol{x_k} - Q^{-1} \nabla f(\boldsymbol{x_k}) = \boldsymbol{x_k} - (\nabla^2 f(\boldsymbol{x_k}))^{-1} \nabla f(\boldsymbol{x_k})$. □

Proposición 3.8. *Si $\nabla^2 f(\boldsymbol{x_k})$ es una matriz positiva definida; es decir, $\nabla^2 f(\boldsymbol{x_k}) = \nabla^2 f(\boldsymbol{x_k})^T > 0$, entonces $\boldsymbol{x_\star} = \nabla^2 f(\boldsymbol{x_k})^{-1} \nabla f(\boldsymbol{x_k})$ es un mínimo global estricto de la función cuadrática $h(\boldsymbol{x})$.*

Demostración. Sea $\boldsymbol{x_\star}$ una solución del sistema $\nabla^2 f(\boldsymbol{x}) \boldsymbol{d} = \nabla f(\boldsymbol{x})$; es decir $\nabla f(\boldsymbol{x_\star}) =$

$\mathbf{0}$, entonces

$$\begin{aligned} h(\boldsymbol{x}) - h(\boldsymbol{x}_\star) &= \nabla f(\boldsymbol{x}_\star)^T(\boldsymbol{x} - \boldsymbol{x}_\star) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_\star)^T \nabla^2 f(\boldsymbol{x}_\star)(\boldsymbol{x} - \boldsymbol{x}_\star) \\ &= \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_\star)^T \nabla^2 f(\boldsymbol{x}_\star)(\boldsymbol{x} - \boldsymbol{x}_\star) > 0, \text{ para todo } \boldsymbol{x} \neq \boldsymbol{x}_\star. \end{aligned}$$

Note que $\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_\star)^T \nabla^2 f(\boldsymbol{x}_\star)(\boldsymbol{x} - \boldsymbol{x}_\star) > 0$ debido a que $\nabla^2 f(\boldsymbol{x})$ es positiva definida. Por lo tanto, se concluye que

$$h(\boldsymbol{x}) > h(\boldsymbol{x}_\star), \text{ para todo } \boldsymbol{x} \neq \boldsymbol{x}_\star.$$

□

Ejemplo 3.3. *En este ejemplo se usa el método iterativo de Newton para encontrar el mínimo global en la función $h : \mathbb{R}^2 \to \mathbb{R}$, donde h es de clase C^2 y dado un punto inicial $\boldsymbol{x_0} = [0, 0]$, se tiene que*

$$h(x, y) = x^2 - xy + y^2 - 3y, \tag{37}$$

donde su gradiente es

$$\nabla h(x, y) = \begin{bmatrix} 2x - y \\ -x + 2y - 3 \end{bmatrix},$$

y su Hessiana es

$$\nabla^2 h(x, y) = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix},$$

usando el método de Newton se encontrará el mínimo global de la función (37); es

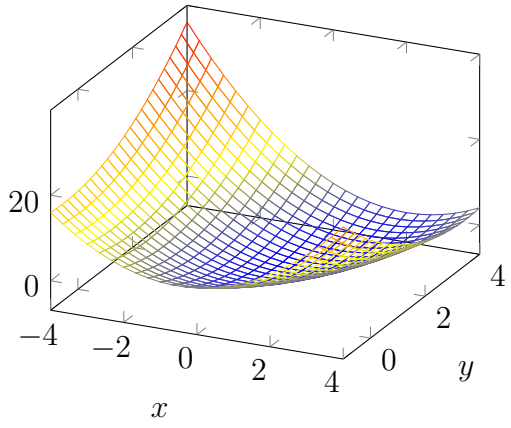

Figura 4. Representación gráfica de la función (37). Hecho por el autor.

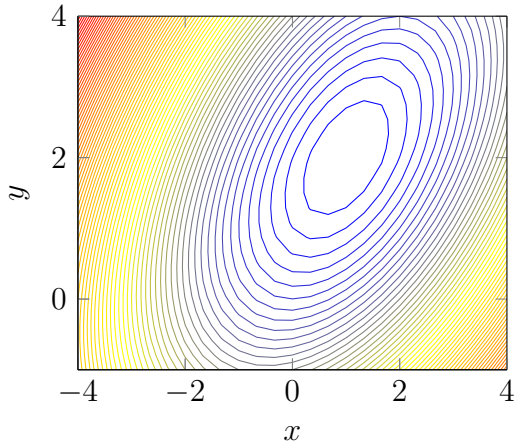

Figura 5. Representación de las curvas de nivel de la función (37). Hecho por el autor.

decir

$$
\begin{aligned}
\boldsymbol{x_1} &= \boldsymbol{x_0} - (\nabla^2 h(\boldsymbol{x_0}))^{-1} \nabla h(\boldsymbol{x_0}), \\
&= \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 2 \cdot 0 - 0 \\ 0 + 2 \cdot 0 - 3 \end{bmatrix}, \\
&= \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \begin{bmatrix} \frac{2}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{2}{3} \end{bmatrix} \begin{bmatrix} 0 \\ -3 \end{bmatrix}, \\
&= \begin{bmatrix} 1 \\ 2 \end{bmatrix}.
\end{aligned}
$$

Así, en un solo paso (3.7) se llega al óptimo de la función (37) y es un mínimo global (3.8).

Usando el método de Newton al aproximar una función cuadrática se llega al óptimo en una iteración y además se puede comenzar en cualquier parte del dominio de la función (3.7).

Ejemplo 3.4. *En este ejemplo se implementa el método de Newton, se considera la función (25) del ejemplo (3.2) y su respectiva Hessiana se define como*

$$\nabla^2 f(x,y) = \begin{bmatrix} 2 - 400y + 1200x^2 & -400x \\ -400x & 200 \end{bmatrix},$$

tomando como punto inicial $\boldsymbol{x}_0 = (-.05, 0.5)$ *y longitud de paso* $\alpha_k = 1$*, se tienen los resultados. Ver tabla 3.*

k	$\boldsymbol{x_k}$	α_k	$\|\|\nabla f(\boldsymbol{x_k})\|\|$	$f(\boldsymbol{x_k})$
0	(-0.500, 0.500)	1	68.622	8.5
1	(-0.530, 0.280)	1	3.265	2.342861
2	(0.758, -1.086)	1	603.344	276.1440
3	(0.759, 0.576)	1	0.481	0.058016
4	(0.999, 0.941)	1	25.939	0.336449
5	(0.999, 0.999)	1	0.000	0.299039
6	(0.999, 0.999)	1	0.000	2.16×10^{-9}

Tabla 3. Implementación del Método de Newton de la función del ejemplo (3.2). Hecho por el autor.

Los resultados del método iterativo de Newton implementado en MATLAB, en la Tabla 3 tienen una tolerancia de (tol $= 10^{-6}$*), donde su criterio de parada se define por* $\|\nabla f(\boldsymbol{x_k})\| < tol\ ||\nabla f(\boldsymbol{x_0})||$ *y la norma vectorial es (2) con la longitud de paso* $\alpha_k = 1$ *y además, el mínimo se encuentra en la iteración* $\boldsymbol{x}_6 = (1, 1)$*.*
La siguiente tabla se implementa el método iterativo de Newton, en la función del ejemplo (3.2) con búsqueda lineal inexacta, se tienen los resultados, ver Tabla 4.

k	$\boldsymbol{x_k}$	α_k	$\|\|\nabla f(\boldsymbol{x_k})\|\|$	$f(\boldsymbol{x_k})$
0	(-0.500, 0.500)	2.3420	2.6918	8.5
1	(-0.5302, 0.2830)	1.9145	10.118	2.34207
2	(-0.3388, 0.0799)	1.3884	9.1964	1.9145
3	(-0.1023, -0.0311)	0.7095	5.8818	1.3885
4	(0.2094, 0.0147)	0.3728	6.1028	0.0092
5	(0.4445, 0.1722)	0.1265	4.9188	0.7095
6	(0.6822, 0.4494)	0.0295	3.6865	0.3728
7	(0.8593, 0.7286)	5.27×10^{-6}	0.08185	0.1263
8	(0.9986, -0.9979)	9.19×10^{-10}	0.00008	0.0295
9	(0.9999,0.9999)	1.034×10^{-19}	1.154×10^{-8}	1.28×10^{-9}

Tabla 4. Implementación del Método de Newton con búsqueda de linea inexacta de la función del ejemplo (3.2). Hecho por el autor.

Los resultados del algoritmo del Método de Newton implementado en MATLAB, en la tabla 4 tienen una tolerancia de (tol $= 10^{-6}$*), donde su criterio de parada se define por* $\|\nabla f(\boldsymbol{x_k})\| < tol\, ||\nabla f(\boldsymbol{x_0})||$*, la norma vectorial es (2) y* α_k *en cada iteración cumple con la condición (2.2) y además, el mínimo se alcanza en la iteración* $\boldsymbol{x}_9 = (1, 1)$.

3.3. Métodos Cuasi-Newton

El primer método Cuasi-Newton fue creado a mediados de 1950 por William C. Davidon, mas tarde Fletcher y Powell[20] demostraron que el algoritmo creado por Davidon era mucho más rápido y confiable que los ya existentes. Dicho método

[20] Acevedo Vázquez Julio Andrés. «Implementación De Los Métodos Cuasi-Newton». Tesis doct. BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA, 2019.

consiste en considerar una aproximación cuadrática de la función objetivo (27) en la iteración $\boldsymbol{x_k}$ y está dada por

$$h_k(\boldsymbol{x}) = f(\boldsymbol{x_k}) + \nabla(\boldsymbol{x_k})^T(\boldsymbol{x} - \boldsymbol{x_k}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x_k})^T H_k(\boldsymbol{x} - \boldsymbol{x_k}),$$

dada H_k, se busca construir una matriz simétrica H_{k+1} que aproxime a la inversa de la Hessiana en $\boldsymbol{x_{k+1}}$, que satisface la condición de la secante.
El método general Cuasi-Newton consiste en

1 Dado $\boldsymbol{x_0} \in \mathbb{R}^n$, en cada paso k hacer lo siguiente.

1.1 Dado H_k, con $\boldsymbol{d_k}$ tal que $d_k = -H_k \nabla f(\boldsymbol{x_k})$.

1.2 Definir $\boldsymbol{x_{k+1}} = \boldsymbol{x_k} + \alpha_k \boldsymbol{d_k}$ mediante dirección de descenso. Si $\nabla f(\boldsymbol{x_{k+1}}) = 0$ parar. Si no, calcular una nueva matriz H_{k+1} y continuar el proceso.

La nueva iteración $\boldsymbol{x_{k+1}}$ es

$$h_{k+1}(\boldsymbol{x}) = f(\boldsymbol{x_{k+1}}) + \nabla f(\boldsymbol{x_{k+1}})^T(\boldsymbol{x} - \boldsymbol{x_{k+1}}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x_{k+1}})^T H_{k+1}(\boldsymbol{x} - \boldsymbol{x_{k+1}}).$$

Luego, se necesita que la nueva matriz cumpla con $h_{k+1}(\boldsymbol{x})$ y $f(\boldsymbol{x})$ tal que tengan gradientes coincidentes en $\boldsymbol{x_k}$ y $\boldsymbol{x_{k+1}}$, es decir

$$\nabla h_{k+1}(\boldsymbol{x}) = \nabla f(\boldsymbol{x_{k+1}}) + H_{k+1}(\boldsymbol{x} - \boldsymbol{x_{k+1}}),$$

y se requiere que $\nabla h_{k+1}(\boldsymbol{x_k}) = \nabla f(\boldsymbol{x_k})$ se verifique la condición de la secante; es decir

$$H_{k+1}(\boldsymbol{x_k} - \boldsymbol{x_{k+1}}) = \nabla f(\boldsymbol{x_k}) - \nabla f(\boldsymbol{x_{k+1}}).$$

Se define $s_k = \boldsymbol{x_{k+1}} + \boldsymbol{x_k}$ y $y_k = \nabla f(\boldsymbol{x_{k+1}}) - \nabla f(\boldsymbol{x_k})$, la condición secante se escribe como

$$H_{k+1} s_k = y_k.$$

Al multiplicar la condición secante por s_k y usando el Teorema del Valor Medio[21] se tiene que

$$s_k^T H_{k+1} s_k = s_k^T(\nabla f(\boldsymbol{x_{k+1}}) - \nabla f(\boldsymbol{x_k})) = s_k^T \nabla^2 f(\hat{\boldsymbol{x}}) s_k,$$

para un $\hat{\boldsymbol{x}}$ punto intermedio entre $\boldsymbol{x_k}$ y $\boldsymbol{x_{k+1}}$. Así, la matriz H_{k+1} que cumple con la condición secante se tiene la misma curvatura de $\nabla^2 f(\hat{\boldsymbol{x}})$ en un punto intermedio a lo largo del segmento que une $\boldsymbol{x_{k+1}}$ con $\boldsymbol{x_k}$ y además la matriz dada cumple la condición $H_k \approx (\nabla^2 f(\boldsymbol{x_k}))^{-1}$.

Note que la desigualdad $0 < s_k^T B_{k+1} s_k = s_k^T y_k$ es conocida como la condición de curvatura.

Existe una condición dual de la secante que consiste en dada una matriz H_k, queremos construir una matriz simétrica H_{k+1} que aproxime a la inversa de la matriz $(\nabla^2 f(\boldsymbol{x_{k+1}}))^{-1}$, que satisfaga la condición secante

$$H_{k+1} y_k = s_k.$$

La alternativa a resolver dicho problema es dada una matriz B_k, se busca construir un matriz simétrica B_{k+1} que aproxime la matriz $\nabla^2 f(\boldsymbol{x_{k+1}})$, que satisfaga la condición dual secante

$$B_{k+1} s_k = y_k,$$

una vez aproximada la matriz se construye la inversa utilizando fórmulas de inversión, así se tienen dos opciones; es decir, se construye la matriz aproximando a la inversa de la Hessiana o se aproxima a la matriz Hessiana y por medio de fórmulas de inversión se obtiene la inversa de la matiz Hessiana dichas condiciones se

---

[21] Luis Enrique Ruíz Hernández. «Teorema del Valor Medio para Derivadas en RN en Puntos Condicionados». En: (1990).

conocen como duales o complementarias de la secante.

Para encontrar una matriz que cumpla la condición secante hay que resolver un sistema de n ecuaciones con $n(n+1)/2$ incógnitas (las entradas de la matriz H que es simétrica). Luego, hay infinitas matrices que cumplen dicha condición.

Note que si $H_k = I$ se obtiene el método de Máximo descenso, si $H_k = \nabla^2 f(\boldsymbol{x_k})$ se obtiene el método de Newton. Así que con una matriz H_k razonable, se obtiene un método intermedio entre el método del Máximo descenso y el método de Newton.

3.4. Método de Actualizaciones de rango uno ó SR1

Dada la matriz B_k para actualizar B_{k+1} de rango uno, se encuentra un vector no nulo $\boldsymbol{z} \in \mathbb{R}^n$ tal que

$$B_{k+1} \;=\; B_k + \rho \boldsymbol{z}\boldsymbol{z}^{\boldsymbol{T}}, \text{ en donde } \boldsymbol{z}\boldsymbol{z}^{\boldsymbol{T}} \text{ es el producto externo.}$$

Obteniendo una matriz simétrica de rango 1, debido a que cada columna es múltiplo de $\boldsymbol{z}$ y la constante ρ se puede considerar como un factor de normalización para obtener la condición dual secante y además, se reduce a resolver un sistema de n ecuaciones con $n+1$ incógnitas.

Teorema 3.9. *Toda matriz simétrica $A \in \mathbb{R}^{n\times n}$ es diagonalizable y se tiene que*

$$A = U \wedge U^T = \lambda_1 u_1 u_1^T + \cdots + \lambda_n u_n u_n^T,$$

donde $\wedge = diag(\lambda_i)$ matriz diagonal con los valores propios reales, $U = [u_1|u_2|\cdots|u_n]$ matriz ortogonal con los vectores propios ortonormales $u_i^T u_j = \delta_{ij}$, ver [22].

---

[22] Alejandra C Zaia. «Algebra: Trabajo Práctico. Unidad Temática Nº 4: Diagonalización de matrices». En: (2018).

Así, es razonable actualizar en cada iteración la aproximación de la matriz Hessiana con una matriz de rango uno.

El problema de encontrar $\rho \in \mathbb{R}$ y $\boldsymbol{z} \in \mathbb{R}^n$, que satisface

$$\begin{aligned}(B_k + \rho \boldsymbol{z}\boldsymbol{z}^{\boldsymbol{T}})(\boldsymbol{x_{k+1}} - \boldsymbol{x_k}) &= \nabla f(\boldsymbol{x_{k+1}}) - \nabla f(\boldsymbol{x_k}) \\ (B_k + \rho \boldsymbol{z}\boldsymbol{z}^{\boldsymbol{T}})s_k &= y_k \\ \rho \boldsymbol{z}\boldsymbol{z}^{\boldsymbol{T}} s_k &= y_k - B_k s_k, \end{aligned} \tag{38}$$

multiplicando (38) por s_k^T y $(\rho \boldsymbol{z}\boldsymbol{z}^{\boldsymbol{T}} s_k)^T$,$(y_k - B_k s_k)^T$ se tienen las siguientes expresiones

$$\rho(\boldsymbol{z}^{\boldsymbol{T}} \boldsymbol{s_k})^2 = s_k^T(y_k - B_k s_k) \tag{39}$$

$$\rho^2(\boldsymbol{z}^{\boldsymbol{T}} \boldsymbol{s_k})^2 \boldsymbol{z}\boldsymbol{z}^{\boldsymbol{T}} = (y_k - B_k s_k)(y_k - B_k s_k)^T, \tag{40}$$

así, reemplazando (39) en (40) se tiene que

$$\rho \boldsymbol{z}\boldsymbol{z}^{\boldsymbol{T}} = \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{\rho(\boldsymbol{z}^{\boldsymbol{T}} s_k)^2} = \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{s_k^T(y_k - B_k s_k)},$$

y sabemos que $B_{k+1} = B_k + \rho \boldsymbol{z}\boldsymbol{z}^{\boldsymbol{T}}$ se tiene que

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{s_k^T(y_k - B_k s_k)}. \tag{41}$$

A partir de esta expresión se puede obtener la actualización de H_k utilizando la relación de dualidad o complementariedad de la secante.

Se intercambia el B_k con H_k, y y_k con s_k se obtiene la expresión

$$H_{k+1} = H_k + \frac{(s_k - H_k y_k)(s_k - H_k y_k)^T}{y_k^T(s_k - H_k y_k)}. \tag{42}$$

Algoritmo 3.4.1. *(Algoritmo Cuasi-Newton con actualización de rango 1)*

1. *Hacer* $k = 0$ *y escoger* $\boldsymbol{x_0} \in \mathbb{R}^n$, $H_0 = H_0^T \in \mathbb{R}^{n \times n}$.

   Descenso. Para $k \geq 0$ *hacer*

2. $\boldsymbol{d_k} = -H_k \nabla f(\boldsymbol{x_k})$.

3. $\alpha_k =$ *arg* $f(\boldsymbol{x_k} + \alpha \boldsymbol{d_k})$.

4. $\boldsymbol{x_{k+1}} = \boldsymbol{x_k} + \alpha_k \boldsymbol{d_k}$.

5. $y_{k+1} = \nabla f(\boldsymbol{x_{k+1}})$.

   Actualización*. Si* $\|y_{k+1}\| \leq$ *Tol* $\|y_0\|$ *parar y salir con* $\boldsymbol{x_\star} = \boldsymbol{x_{k+1}}$. *Sino, continuar.*

6. $\Delta \boldsymbol{x_k} = \alpha_k \boldsymbol{d_k}$, $\Delta y_k = \nabla f(\boldsymbol{x_{k+1}}) - \nabla f(\boldsymbol{x_k})$.

7. $H_{k+1} = H_k + \dfrac{(\Delta \boldsymbol{x_k} - H_k \Delta \boldsymbol{y_k})(\Delta \boldsymbol{x_k} - H_k \Delta \boldsymbol{y_k})^T}{(\Delta \boldsymbol{y_k})^T (\Delta \boldsymbol{x_k} - H_k \Delta \boldsymbol{y_k})}$.

8. *Hacer* $k = k + 1$ *y volver a (2).*

El siguiente teorema nos garantiza una forma de invertir la matriz Hessiana.

Teorema 3.10. *(Inversión de Sherman-Morrison) Si* A *es una matriz no singular y* $1 + \boldsymbol{q^T} A^{-1} \boldsymbol{p} \neq 0$, *con* $\boldsymbol{p}, \boldsymbol{q}$ *vectores no nulos, entonces*

$$(A + \boldsymbol{p}\boldsymbol{q^T})^{-1} = A^{-1} - \frac{(A^{-1}\boldsymbol{p})(\boldsymbol{q^T} A^{-1})}{1 + \boldsymbol{q^T} A^{-1} \boldsymbol{p}}. \tag{43}$$

Si se aplica la expresión (43) para invertir la matriz (41). Se toma

$$A = b_k, A^{-1} = H_k,$$

matrices simétricas, haciendo $\boldsymbol{q} = \rho \boldsymbol{p}$, se tiene que

$$H_{k+1} = H_k - \rho \frac{(H_k \boldsymbol{p})(H_k \boldsymbol{p})^T}{1 + \rho\, \boldsymbol{p}^T H_k \boldsymbol{p}}, \text{ con } \rho = \frac{1}{s_k^T (y_k - B_k s_k)} \text{ y } \boldsymbol{p} = y_k - B_k s_k.$$

Nota 3.11. *Es un poco ineficiente la inversión de Sherman-Morrison, ya que al actualizar H_{k+1} se necesita conocer H_k y B_k, así se tendría que estar actualizando las dos expresiones. En la práctica se usa la expresión (42).*

3.5. Método de Actualizaciones de rango dos ó DFP (Davidon-Fletcher-Powell)

En este método se incorpora la condición de positividad en la actualización (3.4.1) de la matriz H_{k+1}, además de la condición secante. Conocidos los vectores $\boldsymbol{y_k}, \boldsymbol{s_k}$ y la matriz simétrica H_k, se deben encontrar escalares ρ_1, ρ_2 y vectores $\boldsymbol{z_1}, \boldsymbol{z_2} \in \mathbb{R}^n$ tales que

$$H_{k+1} = H_k + \rho_1 \boldsymbol{z_1} \boldsymbol{z_1^T} + \rho_2 \boldsymbol{z_2} \boldsymbol{z_2^T}, \tag{44}$$

satisfacen las dos condiciones

- $H_{k+1}\boldsymbol{y_k} = \boldsymbol{s_k}$.
- $\boldsymbol{y_k}^T H_{k+1} \boldsymbol{y_k} > 0$, es decir $\boldsymbol{y_k^T s_k} > 0$.

Note que ahora se tienen $2n$ condiciones, lo cual se puede usar una actualización de rango dos.
Sustituyendo en H_{k+1} y desarrollando, se tiene que

$$(\rho_1 \boldsymbol{z_1} \boldsymbol{z_1^T} + \rho_2 \boldsymbol{z_2} \boldsymbol{z_2^T}) \boldsymbol{y_k} = \boldsymbol{s_k} - H_k \boldsymbol{y_k} \tag{45}$$

$$\boldsymbol{y_k}^T H_k \boldsymbol{y_k} + \rho_1 (\boldsymbol{y_k^T z_1})^2 + \rho_2 (\boldsymbol{y_k^T z_2})^2 = \boldsymbol{y_k^T s_k} > 0. \tag{46}$$

En (46) se hace el producto exterior consigo mismo en cada lado, es decir

$$((\rho_1 \boldsymbol{z_1} \boldsymbol{z_1^T} + \rho_2 \boldsymbol{z_2} \boldsymbol{z_2^T}) \boldsymbol{y_k})((\rho_1 \boldsymbol{z_1} \boldsymbol{z_1^T} + \rho_2 \boldsymbol{z_2} \boldsymbol{z_2^T}) \boldsymbol{y_k}) = (\boldsymbol{s_k} - H_k \boldsymbol{y_k})(\boldsymbol{s_k} - H_k \boldsymbol{y_k}),$$

así, se tiene que

$$\rho_1^2(\boldsymbol{y_k^T z_1})^2 \boldsymbol{z_1 z_1^T} + \rho_1\rho_2(\boldsymbol{y_k^T z_1})(\boldsymbol{y_k^T z_2})\boldsymbol{z_1 z_2^T} + \rho_1\rho_2(\boldsymbol{y_k^T z_2})(\boldsymbol{y_k^T z_1})\boldsymbol{z_2 z_1^T} + \rho_2^2(\boldsymbol{y_k^T z_2})^2 \boldsymbol{z_2 z_2^T}$$
$$= \boldsymbol{s_k s_k^T} - \boldsymbol{s_k}(H_k\boldsymbol{y_k})^T - (H_k\boldsymbol{y_k})\boldsymbol{s_k^T} + (H_k\boldsymbol{y_k})(H_k\boldsymbol{y_k})^T.$$

Note que la anterior expresión tiene cierta simetría, así haciendo sus debidas correspondencias se tiene que

$$\begin{aligned}
\rho_1^2(\boldsymbol{y_k^T z_1})^2 \boldsymbol{z_1 z_1^T} &= \boldsymbol{s_k s_k^T} \\
\rho_1\rho_2(\boldsymbol{y_k^T z_1})(\boldsymbol{y_k^T z_2})\boldsymbol{z_1 z_2^T} &= -\boldsymbol{s_k}(H_k\boldsymbol{y_k})^T \\
\rho_1\rho_2(\boldsymbol{y_k^T z_2})(\boldsymbol{y_k^T z_1})\boldsymbol{z_2 z_1^T} &= -(H_k\boldsymbol{y_k})\boldsymbol{s_k^T} \\
\rho_2^2(\boldsymbol{y_k^T z_2})^2 \boldsymbol{z_2 z_2^T} &= (H_k\boldsymbol{y_k})(H_k\boldsymbol{y_k})^T,
\end{aligned}$$

tomando $\boldsymbol{z_1} = \boldsymbol{s_k}$ y $\boldsymbol{z_2} = H_k\boldsymbol{y_k}$, entonces se debe satisfacer

$$\begin{aligned}
\rho_1^2(\boldsymbol{y_k^T s_k})^2 &= 1 \\
\rho_1\rho_2(\boldsymbol{y_k^T s_k})(\boldsymbol{y_k^T} H_k\boldsymbol{y_k}) &= -1 \\
\rho_1\rho_2(\boldsymbol{y_k^T} H_k\boldsymbol{y_k})(\boldsymbol{y_k^T s_k}) &= -1 \\
\rho_2^2(\boldsymbol{y_k^T} H_k\boldsymbol{y_k})^2 &= 1.
\end{aligned}$$

En las anteriores expresiones y teniendo en cuenta la condición de positividad (46) se tiene que

$$\rho_1 = \frac{1}{\boldsymbol{s_k^T y_k}} \text{ y } \rho_2 = -\frac{1}{\boldsymbol{y_k}^T H_k\boldsymbol{y_k}}.$$

En (44) se tiene la expresión

$$H_{k+1} = H_k + \frac{\boldsymbol{s_k s_k^T}}{\boldsymbol{s_k^T y_k}} - \frac{H_k\boldsymbol{y_k}(H_k\boldsymbol{y_k})^T}{\boldsymbol{y_k}^T H_k\boldsymbol{y_k}}. \tag{47}$$

Algoritmo 3.5.1. *(Algoritmo Cuasi-Newton DFP)*

1. Hacer $k = 0$ y escoger $\boldsymbol{x_0} \in \mathbb{R}^n$, $H_0 = H_0^T \in \mathbb{R}^{n\times n}$.

   Descenso. Para $k \geq 0$ hacer

2. $\boldsymbol{d_k} = -H_k \nabla f(\boldsymbol{x_k})$.

3. $\alpha_k = \arg\ f(\boldsymbol{x_k} + \alpha \boldsymbol{d_k})$.

4. $\boldsymbol{x_{k+1}} = \boldsymbol{x_k} + \alpha_k \boldsymbol{d_k}$.

5. $g_{k+1} = \nabla f(\boldsymbol{x_{k+1}})$.

   Actualización. Si $\|g_{k+1}\| \leq$ Tol $\|g_0\|$ parar y salir con $\boldsymbol{x_\star} = \boldsymbol{x_{k+1}}$. Sino, continuar.

6. $\Delta \boldsymbol{x_k} = \alpha_k \boldsymbol{d_k}$, $\Delta y_k = \nabla f(\boldsymbol{x_{k+1}}) - \nabla f(\boldsymbol{x_k})$.

7. $H_{k+1} = H_k + \dfrac{(\Delta \boldsymbol{x_k})(\Delta \boldsymbol{x_k})^T}{(\Delta \boldsymbol{x_k})^T(\Delta \boldsymbol{y_k})} - \dfrac{(H_k \Delta \boldsymbol{y_k})(H_k \Delta \boldsymbol{y_k})^T}{(\Delta \boldsymbol{y_k})^T (H_k \Delta \boldsymbol{y_k})}$.

8. Hacer $k = k + 1$ y volver a (2).

Proposición 3.12. *Si $\boldsymbol{s_k}^T \boldsymbol{y_k} > 0$ para todo $k \in \mathbb{Z}_+$, entonces el método (47) preserva matrices definidas positivas H_k.*

Demostración. Note que para el paso base $k = 0$ se cumple $\boldsymbol{z}^T H_0 \boldsymbol{z} > 0$, supongamos que se cumple para $k \geq 1$, como H_k es simétrica por la factorización de Cholesky se tiene que $H_k = LL^T$ y sea $\boldsymbol{z} \in \mathbb{R}^n$ no nulo, entonces

$$\begin{aligned}
\boldsymbol{z}^T \left(H_k + \frac{\boldsymbol{s_k}\boldsymbol{s_k}^T}{\boldsymbol{s_k}^T y_k} - \frac{H_k \boldsymbol{y_k}\boldsymbol{y_k}^T H_k}{\boldsymbol{y_k}^T H_k \boldsymbol{y_k}} \right) \boldsymbol{z} &= a^T a - \frac{(a^T b)(b^T a)}{b^T b} + \frac{(\boldsymbol{z}^T \boldsymbol{s_k})(\boldsymbol{s_k}^T \boldsymbol{z})}{\boldsymbol{s_k}^T \boldsymbol{y_k}} \\
&= ||a||^2 (1 - cos^2(\theta)) + \frac{||\boldsymbol{z}^T \boldsymbol{s_k}||^2}{\boldsymbol{s_k}^T \boldsymbol{y_k}} \\
&= ||a||^2 sen^2(\theta) + \frac{||\boldsymbol{z}^T \boldsymbol{s_k}||^2}{\boldsymbol{s_k}^T \boldsymbol{y_k}} \geq 0,
\end{aligned}$$

donde $a = L^T \boldsymbol{z}$ y $b = L^T \boldsymbol{y_k}$, así por inducción es fácil ver que las nuevas aproximaciones son simétricas positivas definidas y satisfacen la condición secante. □

3.6. Método de actualización de rango dos o BFGS (Broyden-Fletcher-Goldfarb-Shanno)

El método de actualización de rango dos DFP (3.5) es efectivo, pero fue superado por otro método llamado BFGS. En 1970, los investigadores Broyden, Fletcher, Goldfarb y Shano[23] sugirieron una expresión de actualización de grado dos.
Usando la expresión dual o complementaria en (47) se tiene que

$$B_{k+1}^{DFP} = B_k + \frac{\boldsymbol{y_k}\boldsymbol{y_k^T}}{\boldsymbol{y_k^T}\boldsymbol{s_k}} - \frac{B_k \boldsymbol{s_k}(B_k \boldsymbol{s_k})^T}{\boldsymbol{s_k}^T B_k \boldsymbol{s_k}}. \tag{48}$$

Aplicando la formula de Sherman-Morrison-Woodbury (3.15) se tiene que

$$\begin{aligned} H_{k+1}^{BFGS} = (B_{k+1}^{DFP})^{-1} &= \left(B_k + \frac{\boldsymbol{y_k}\boldsymbol{y_k^T}}{\boldsymbol{y_k^T}\boldsymbol{s_k}} - \frac{B_k \boldsymbol{s_k}(B_k \boldsymbol{s_k})^T}{\boldsymbol{s_k}^T B_k \boldsymbol{s_k}} \right)^{-1} \\ &= \left(B_k - \frac{B_k \boldsymbol{s_k}(B_k \boldsymbol{s_k})^T}{\boldsymbol{s_k}^T B_k \boldsymbol{s_k}} + \frac{\boldsymbol{y_k}\boldsymbol{y_k^T}}{\boldsymbol{y_k^T}\boldsymbol{s_k}} \right)^{-1} \\ &= \left(B_k + (B_k \boldsymbol{s_k} \;\; \boldsymbol{y_k}) \begin{pmatrix} -\dfrac{1}{\boldsymbol{s_k}^T B_k \boldsymbol{s_k}} & 0 \\ 0 & \dfrac{1}{\boldsymbol{y_k}^T \boldsymbol{s_k}} \end{pmatrix} \begin{pmatrix} \boldsymbol{s_k^T} B_k^T \\ \boldsymbol{y_k^T} \end{pmatrix} \right)^{-1}, \end{aligned}$$

tomando $\boldsymbol{u} = (B_k \boldsymbol{s_k} \;\; \boldsymbol{y_k})$, $C = \begin{pmatrix} -\dfrac{1}{\boldsymbol{s_k}^T B_k \boldsymbol{s_k}} & 0 \\ 0 & \dfrac{1}{\boldsymbol{y_k}^T \boldsymbol{s_k}} \end{pmatrix}$ y $\boldsymbol{v} = (\boldsymbol{s_k}^T B_k^T \;\; \boldsymbol{y_k}^T)^T$, se tiene

---

[23] Luz Marina Rondón Poveda. «Comparación de la eficiencia del método de optimización BFGS en C, OX y R para un modelo de regresión no lineal.» En: *Comunicaciones en Estadística* 2.2 (2009), págs. 175-188.

que

$$
\begin{aligned}
H_{k+1}^{BFGS} &= B_k^{-1} - B_k^{-1}\boldsymbol{u}\left(C^{-1} + \boldsymbol{v}B_k^{-1}\boldsymbol{u}\right)^{-1}\boldsymbol{v}B_k^{-1} \\
&= B_k^{-1} - (\boldsymbol{s_k} \;\; B_k^{-1}\boldsymbol{y_k}) \begin{pmatrix} 0 & \boldsymbol{s_k}^T\boldsymbol{y_k} \\ \boldsymbol{y_k}^T\boldsymbol{s_k} & \boldsymbol{y_k}^T\boldsymbol{s_k} + \boldsymbol{y_k}^T B_k^{-1}\boldsymbol{y_k} \end{pmatrix}^{-1} \begin{pmatrix} \boldsymbol{s_k}^T \\ \boldsymbol{y_k}^T B_k^{-1} \end{pmatrix} \\
&= B_k^{-1} - (\boldsymbol{s_k} \;\; B_k^{-1}\boldsymbol{y_k}) \begin{pmatrix} -\left(\dfrac{\boldsymbol{y_k}^T\boldsymbol{s_k} + \boldsymbol{y_k}^T B_k^{-1}\boldsymbol{y_k}}{\boldsymbol{s_k}^T\boldsymbol{y_k}\boldsymbol{y_k}^T\boldsymbol{s_k}}\right) & \dfrac{\boldsymbol{s_k}^T\boldsymbol{y_k}}{\boldsymbol{s_k}^T\boldsymbol{y_k}\boldsymbol{y_k}^T\boldsymbol{s_k}} \\ \dfrac{\boldsymbol{y_k}^T\boldsymbol{s_k}}{\boldsymbol{s_k}^T\boldsymbol{y_k}\boldsymbol{y_k}^T\boldsymbol{s_k}} & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{s_k}^T \\ \boldsymbol{y_k}^T B_k^{-1} \end{pmatrix} \\
&= B_k^{-1} - \left(-\frac{\boldsymbol{s_k}\boldsymbol{y_k}^T\boldsymbol{s_k} + B_k^{-1}\boldsymbol{y_k}}{\boldsymbol{s_k}^T\boldsymbol{y_k}} + \frac{B_k^{-1}\boldsymbol{y_k}}{\boldsymbol{s_k}^T\boldsymbol{y_k}} \quad \frac{\boldsymbol{s_k}}{\boldsymbol{s_k}^T\boldsymbol{y_k}} \right) \begin{pmatrix} \boldsymbol{s_k}^T \\ \boldsymbol{y_k}^T B_k^{-1} \end{pmatrix} \\
&= B_k^{-1} + \left(\frac{\boldsymbol{s_k} + B_k^{-1}\boldsymbol{y_k}}{\boldsymbol{s_k}^T\boldsymbol{y_k}}\right)\boldsymbol{s_k} - \left(\frac{B_k^{-1}\boldsymbol{y_k}}{\boldsymbol{s_k}^T\boldsymbol{y_k}}\right)\boldsymbol{s_k}^T - \frac{\left((B_k^{-1}\boldsymbol{y_k})\boldsymbol{s_k}^T\right)^T}{\boldsymbol{s_k}^T\boldsymbol{y_k}} \\
&= B_k^{-1} + \left(1 + \frac{\boldsymbol{y_k}^T B_k^{-1}\boldsymbol{y_k}}{\boldsymbol{s_k}^T\boldsymbol{y_k}}\right)\frac{\boldsymbol{s_k}\boldsymbol{s_k}^T}{\boldsymbol{s_k}^T\boldsymbol{y_k}} - \frac{(B_k^{-1}\boldsymbol{y_k})\boldsymbol{s_k}^T + \left((B_k^{-1}\boldsymbol{y_k})\boldsymbol{s_k}^T\right)^T}{\boldsymbol{s_k}^T\boldsymbol{y_k}}.
\end{aligned}
$$

La inversa de B_k es denotada por $B_k^{-1} = H_k$ se tiene que

$$
\begin{aligned}
H_{k+1}^{BFGS} &= (B_{k+1}^{DFP})^{-1} = H_k + \left(1 + \frac{\boldsymbol{y_k}^T H_k\boldsymbol{y_k}}{\boldsymbol{s_k}^T\boldsymbol{y_k}}\right)\frac{\boldsymbol{s_k}\boldsymbol{s_k}^T}{\boldsymbol{s_k}^T\boldsymbol{y_k}} - \frac{(H_k\boldsymbol{y_k})\boldsymbol{s_k}^T + \left((H_k\boldsymbol{y_k})\boldsymbol{s_k}^T\right)^T}{\boldsymbol{s_k}^T\boldsymbol{y_k}} \\
&= (I - \rho\boldsymbol{s_k}\boldsymbol{y_k}^T)H_k(I - \rho\boldsymbol{y_k}\boldsymbol{s_k}^T) + \rho\boldsymbol{s_k}\boldsymbol{s_k}^T \text{ donde } \rho = \frac{1}{\boldsymbol{y_k}^T\boldsymbol{s_k}}. \qquad (49)
\end{aligned}
$$

Note que (48) y (49) son inversas entre si; es decir $B_{k+1}H_{k+1} = I$. Además $B_{k+1}\boldsymbol{s_k} =$

$\boldsymbol{y_k}$ es la condición de la secante

$$\begin{aligned} B_{k+1}H_{k+1} &= B_{k+1}\left(\left(I-\rho_k \boldsymbol{s_k}\boldsymbol{y_k}^T\right)H_k\left(I-\rho_k \boldsymbol{y_k}\boldsymbol{s_k}^T\right)+\rho_k \boldsymbol{s_k}\boldsymbol{s_k}^T\right) \\ &= \left(B_{k+1}-\rho_k \boldsymbol{y_k}\boldsymbol{y_k}^T\right)H_k\left(I-\rho_k \boldsymbol{y_k}\boldsymbol{s_k}^T\right)+\rho_k \boldsymbol{y_k}\boldsymbol{s_k}^T \\ &= \left(B_k-\frac{B_k \boldsymbol{s_k}\boldsymbol{s_k}^T B_k}{\boldsymbol{s_k}^T B_k \boldsymbol{s_k}}\right)H_k\left(I-\rho_k \boldsymbol{y_k}\boldsymbol{s_k}^T\right)+\rho_k \boldsymbol{y_k}\boldsymbol{s_k}^T \\ &= \left(I-\frac{B_k \boldsymbol{s_k}\boldsymbol{s_k}^T}{\boldsymbol{s_k}^T B_k \boldsymbol{s_k}}\right)\left(I-\rho_k \boldsymbol{y_k}\boldsymbol{s_k}^T\right)+\rho_k \boldsymbol{y_k}\boldsymbol{s_k}^T \\ &= I-\rho_k \boldsymbol{y_k}\boldsymbol{s_k}^T-\frac{B_k \boldsymbol{s_k}\boldsymbol{s_k}^T}{\boldsymbol{s_k}^T B_k \boldsymbol{s_k}}+\rho_k\frac{B_k \boldsymbol{s_k}\boldsymbol{s_k}^T \boldsymbol{y_k}\boldsymbol{s_k}^T}{\boldsymbol{s_k}^T B_k \boldsymbol{s_k}}+\rho_k \boldsymbol{y_k}\boldsymbol{s_k}^T = I. \end{aligned}$$

Note que

$$\rho_k\frac{B_k \boldsymbol{s_k}\boldsymbol{s_k}^T}{\boldsymbol{s_k}^T B_k \boldsymbol{s_k}}\boldsymbol{y_k}\boldsymbol{s_k}^T = \frac{1}{\boldsymbol{y_k}^T \boldsymbol{s_k}}\frac{B_k \boldsymbol{s_k}\boldsymbol{s_k}^T(\boldsymbol{y_k}\boldsymbol{s_k}^T)}{\boldsymbol{s_k}^T B_k \boldsymbol{s_k}} = \frac{B_k \boldsymbol{s_k}\boldsymbol{s_k}^T}{\boldsymbol{s_k}^T B_k \boldsymbol{s_k}}.$$

Algoritmo 3.6.1. *(Algoritmo Cuasi-Newton BFGS)*

1. *Hacer* $k=0$ *y escoger* $\boldsymbol{x_0}\in\mathbb{R}^n$, $H_0=H_0^T\in\mathbb{R}^{n\times n}$.

   Descenso. Para $k\geq 0$ *hacer*

2. $\boldsymbol{d_k}=-H_k\nabla f(\boldsymbol{x_k})$.

3. $\alpha_k=$ *arg* $f(\boldsymbol{x_k}+\alpha \boldsymbol{d_k})$.

4. $\boldsymbol{x_{k+1}}=\boldsymbol{x_k}+\alpha_k \boldsymbol{d_k}$.

5. $g_{k+1}=\nabla f(\boldsymbol{x_{k+1}})$.

   Actualización. *Si* $\|g_{k+1}\|\leq$ *Tol* $\|g_0\|$ *parar y salir con* $\boldsymbol{x_\star}=\boldsymbol{x_{k+1}}$. *Sino, continuar.*

6. $\Delta \boldsymbol{x_k}=\alpha_k \boldsymbol{d_k}$, $\Delta y_k=\nabla f(\boldsymbol{x_{k+1}})-\nabla f(\boldsymbol{x_k})$.

7. $H_{k+1}=\left(I-\rho \boldsymbol{s_k}\boldsymbol{y_k}^T\right)H_k\left(I-\rho \boldsymbol{y_k}\boldsymbol{s_k}^T\right)+\rho \boldsymbol{s_k}\boldsymbol{s_k}^T$.

8. Hacer $k = k + 1$ *y volver a (2).*

El siguiente Teorema y Lema garantizan que el método BFGS está bien definido.

Teorema 3.13. *Sea* $A \in L(\mathbb{R}^{n\times n})$ *conjunto de matrices simétricas no singulares, y el conjunto* $\boldsymbol{y_k} = A\boldsymbol{s_k}$ *para* $0 \leq k \leq m$ *donde* $k \in \{s_0, s_1, \ldots, s_m\} \in$ *Span*$(A) = \{\boldsymbol{s} \in \mathbb{R}^n : \boldsymbol{s} = \sum_{i=1}^{m} s_i x_i,$ *con* $x_0, x_1 \ldots, x_m \in \mathbb{R}\}$*. Sea* H_0 *simétrica y para cada* $k = 0, 1, \ldots, m$ *genera las matrices*

$$H_{k+1} = H_k + \frac{(\boldsymbol{s_k} - H_k\boldsymbol{y_k})(\boldsymbol{s_k} - H_k\boldsymbol{y_k})^T}{(\boldsymbol{s_k} - H_k\boldsymbol{y_k})^T\boldsymbol{y_k}},\ \textit{con}\ (\boldsymbol{s_k} - H_k\boldsymbol{y_k})^T\boldsymbol{y_k} \neq 0, \tag{50}$$

entonces $H_{m+1} = A^{-1}$*, ver* [24]*.*

Note que la expresión (50) es la actualización del método SR1 (42).

Lema 3.14. *Sea* $B \in L(\mathbb{R}^{n\times n})$ *el conjunto de matrices simétricas no singulares y sea* $c, s, y \in \mathbb{R}^n$ *con* $\boldsymbol{c}^T\boldsymbol{s} \neq 0$*. Si la sucesión* $\{C_k\}$ *definida por*

$$\begin{aligned} C_{2k+1} &= C_{2k} + \frac{(\boldsymbol{y} - C_{2k}\boldsymbol{s})\boldsymbol{c}^T}{\boldsymbol{c}^T\boldsymbol{s}} \\ C_{2k+2} &= \frac{C_{2k+1}C_{2k+1}^T}{2},\ \textit{para}\ k = 0, 1, \ldots \end{aligned}$$

con $C_0 = B$*, entonces la sucesión* $\{C_k\}$ *converge a*

$$\hat{B} = B + \frac{(\boldsymbol{y} - B\boldsymbol{s})\boldsymbol{c}^T + \boldsymbol{c}(\boldsymbol{y} - B\boldsymbol{s})^T}{\boldsymbol{c}^T\boldsymbol{s}} - \frac{(\boldsymbol{y} - B\boldsymbol{s})^T\boldsymbol{s}\ \boldsymbol{c}\boldsymbol{c}^T}{(\boldsymbol{c}^T\boldsymbol{s})^2}, \tag{51}$$

ver [24]*.*

Note que la expresión (51) satisface la condición secante. Si B es simétrica implica

---

[24] John E Dennis Jr y Jorge J Moré. «Quasi-Newton methods, motivation and theory». En: *SIAM review* 19.1 (1977), págs. 46-89.

que $\hat{B}$ es simétrica y además se tiene que la expresión (51), si $\boldsymbol{c} = \boldsymbol{y_k}$

$$\begin{aligned} H_{k+1} &= H_k + \frac{(\boldsymbol{y_k} H_k \boldsymbol{s_k})\boldsymbol{y_k}^T + \boldsymbol{y_k}(\boldsymbol{y_k} - H_k \boldsymbol{s_k})^T}{\boldsymbol{y_k}^T \boldsymbol{s_k}} - \frac{(\boldsymbol{y_k} - H_k \boldsymbol{s_k})^T \boldsymbol{s_k}\ \boldsymbol{y_k}\boldsymbol{y_k}^T}{(\boldsymbol{y_k}^T \boldsymbol{s_k})^2} \\ &= \left(I - \frac{\boldsymbol{y}\boldsymbol{s_k}^T}{\boldsymbol{y_k}^T \boldsymbol{s_k}}\right) H_k \left(I - \frac{\boldsymbol{s_k}\boldsymbol{y_k}^T}{\boldsymbol{y_k}^T \boldsymbol{s_k}}\right) + \frac{\boldsymbol{y}\boldsymbol{y_k}^T}{\boldsymbol{y_k}^T \boldsymbol{s_k}}. \end{aligned}$$

Se obtiene el método BFGS (49). El siguiente Teorema garantiza la invertibilidad de matrices no singulares para deducir el método BFGS.

Teorema 3.15. *(Shermann-Morrison-Woodbury)*
Si A, U, C, V son matrices de tamaños $n \times n, k \times k, n \times k, k \times n$ respectivamente con A y C no singulares, entonces

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}.$$

Demostración. Sea $M = A + UCV$, entonces se tiene que

$$\begin{aligned} MM^{-1} &= (A + UCV)(A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1} \\ &= I - U(C^{-1} + VA^{-1}U)^{-1}VA^{-1} + UCVA^{-1} \\ &\qquad - UCVA^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1} \\ &= (I + UCVA^{-1}) - (U(C^{-1} + VA^{-1}U)^{-1}VA^{-1} \\ &\qquad + UCVA^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}) \\ &= (I + UCVA^{-1}) - (U + UCVA^{-1})(C^{-1} + VA^{-1}U)^{-1}VA^{-1} \\ &= I + UCVA^{-1} - UC(C^{-1} + VA^{-1}U)(C^{-1} + VA^{-1}U)^{-1}VA^{-1} \\ &= I + UCVA^{-1} - UCIVA^{-1} = I \end{aligned}$$

□

Note que la actualización del método BFGS preserva la propiedad de ser positiva

definida. Sea $\boldsymbol{z} \in \mathbb{R}^n$ no nulo tal que

$$\boldsymbol{z}^T(B_{k+1})^{-1}\boldsymbol{z} = (I - \rho \boldsymbol{s_k}\boldsymbol{y_k}^T)H_k(I - \rho \boldsymbol{y_k}\boldsymbol{s_k}^T) + \rho \boldsymbol{s_k}\boldsymbol{s_k}^T \text{ donde } \rho = \frac{1}{\boldsymbol{s_k}^T\boldsymbol{y_k}}.$$

H_{k+1} es definida positiva en (49), siempre que H_k sea definida positiva. Para cada vector no nulo $\boldsymbol{z} \in \mathbb{R}^n$, se tiene que

$$\boldsymbol{z}^T H_{k+1}\boldsymbol{z} = \boldsymbol{w}^T H_k \boldsymbol{w} + \rho_k(\boldsymbol{z}^T \boldsymbol{s_k})^2 \geq 0,$$

donde $w = \boldsymbol{z} - \rho_k \boldsymbol{y_k}(\boldsymbol{s_k}^T\boldsymbol{z})$. Note que si $\boldsymbol{s_k}^T\boldsymbol{z} = 0$, entonces $w = \boldsymbol{z} \neq 0$ por lo tanto H_{k+1} es definida positiva, y si además se cumple que $\boldsymbol{s_k}^T\boldsymbol{y_k} > 0$, entonces las nuevas aproximaciones son simétricas positivo definidas y satisfacen la condición secante.

Ejemplo 3.5. *En el siguiente ejemplo se implementa el método BFGS, se considera la función (25) del ejemplo (3.2). Tomando como punto inicial $\boldsymbol{x}_0 = (-0.5, 0.5)$, la longitud de paso α_k usando búsqueda lineal inexacta (2.2), se tienen los resultados, ver Tabla 5.*

k	$\boldsymbol{x_k}$	α_k	$\|\|\nabla f(\boldsymbol{x_k})\|\|$	$f(\boldsymbol{x_k})$
1	(-0.500, 0.500)	0.004	68.622	8.50000
2	(-0.684, 0.3057)	0.062	57.841	5.47878
3	(-0.394, -0.017)	1.000	45.548	4.89834
4	(-0.497, 0.280)	1.000	2 7.536	2.34988
5	(-0.455, 0.207)	1.000	2.943	2.11815
6	(-0.372, 0.108)	1.000	9.568	1.97757
7	(-0.329, 0.077)	1.000	9.157	1.86186
⋮	⋮	⋮	⋮	⋮
30	(0.999, 0.999)	1.000	0.060	0.00000
31	(1.000, 1.000)	1.000	0.019	0.00000
32	(1.000, 1.000)	1.000	0.000	0.00000

Tabla 5. Implementación del Método BFGS con búsqueda de línea inexacta de la función (25) del ejemplo (3.2). Hecho por el autor.

Los resultados del algoritmo del Método BFGS implementado en (MATLAB) en la tabla 5 tienen una tolerancia (tol $= 10^{-6}$), donde su criterio de parada se define por $\|\nabla f(\boldsymbol{x_k})\| <$ tol $||\nabla f(\boldsymbol{x_0})||$, la norma vectorial es (2) y α_k en cada iteración cumple con la condición (2.2) y además, el mínimo se alcanza en la iteración $\boldsymbol{x}_{32} = (1, 1)$.
El siguiente Teorema garantiza la convergencia del método BFGS-SR1-DFP bajo ciertas hipótesis vistas en la convergencia del método iterativo de Newton.

Teorema 3.16. *Sea $\mathcal{D} \subset \mathbb{R}^n$ un conjunto abierto convexo, $R : \mathcal{D} \to \mathbb{R}^m$ una función de clase $\mathcal{C}^2$, f definida en (53) y $\{H_k\}$ una sucesión de actualizaciones secantes (SR1-DFP-BFGS) de $\nabla^2 f$. Si se cumplen las condiciones $(h1)$, $(h2)$ y $(h3)$ del Teorema (3.6). Entonces existen $\varepsilon, \delta > 0$ tales que si $||\boldsymbol{x_0} - \boldsymbol{x_\star}|| < \varepsilon$ y $|||H_0 - \nabla^2 f(\boldsymbol{x_\star})||| <$*

δ, la sucesión generada por

$$\boldsymbol{x_{k+1}} = \boldsymbol{x_k} - (H_k)^{-1}\nabla f(\boldsymbol{x_k}),$$

está bien definida y converge q-superlinealmente (14) a $\boldsymbol{x_\star}$, ver [15].

Note que la norma matricial del Teorema (3.16) es la norma de Frobenius (6) y la norma vectorial es (2).

Nota 3.17. *El Teorema (3.16) es válido para la familia de las actualizaciones Broyden convexa; es decir que H_k sea simétrica y que $\boldsymbol{s_k}^T\boldsymbol{y_k} > 0$ en cada actualización.*

Para finalizar esta sección, se presentan dos Teoremas de los cuales, el primero garantiza condiciones suficientes para controlar el error de las aproximaciones de la matriz Hessiana en los métodos Cuasi Newton y el segundo garantiza la convergencia de la sucesión $\boldsymbol{x_{k+1}} = \boldsymbol{x_k} - (H_k)^{-1}\nabla f(\boldsymbol{x_k})$; es decir que los Teoremas nos garantizan que, si la sucesión de matrices H_k no converge $\nabla^2 f(\boldsymbol{x_\star})$, y si la perturbación del error está lo suficientemente controlada, entonces la sucesión $\boldsymbol{x_k}$ converge a $\boldsymbol{x_\star}$.

Teorema 3.18. *Sea $\mathcal{D} \subset \mathbb{R}^n$ un conjunto abierto y convexo, $\boldsymbol{x_0}, \boldsymbol{x_1} \in \mathcal{D}$ con $\boldsymbol{x_0} \neq \boldsymbol{x_\star}$. Sea $f : \mathbb{R}^n \to \mathbb{R}$, $\nabla^2 f(\boldsymbol{x}) \in Lip_\gamma(\mathcal{D})$ y $\{H_k\}$ definida por alguna actualización (SR1-DFP-BFGS) secante de la clase convexa a la matriz Hessiana. Entonces se tiene que*

$$|||H_{k+1} - \nabla^2 f(\boldsymbol{x_{k+1}})|| \leq |||H_k - \nabla^2 f(\boldsymbol{x_k})||| + \frac{3\gamma}{2}||\boldsymbol{x_{k+1}} - \boldsymbol{x_k}||,$$

donde la norma matricial es la norma de Frobenius que es inducida por la norma vectorial euclídea. Además, si $\boldsymbol{x_\star} \in \mathcal{D}$, es la solución del problema, y $\nabla^2 f(\boldsymbol{x})$ satisface la condición Lipschitz débil, se tiene que

$$|||H_{k+1} - \nabla^2 f(\boldsymbol{x_\star})||| \leq |||H_k - \nabla^2 f(\boldsymbol{x_\star})||| + \frac{\gamma}{2}\left(||\boldsymbol{x_{k+1}} - \boldsymbol{x_\star}|| + ||\boldsymbol{x_k} - \boldsymbol{x_\star}||\right),$$

ver [15].

El siguiente Teorema con ciertas hipótesis garantiza que al utilizar los métodos Cuasi Newton se pueden resolver problemas de Mínimos Cuadrados No Lineales.

Teorema 3.19. *Sea $\mathcal{D} \subset \mathbb{R}^n$ un conjunto abierto y convexo, $R : \mathcal{D} \to \mathbb{R}^m$ una función de clase $\mathcal{C}^2$, $f(\boldsymbol{x}) = \frac{1}{2} R(\boldsymbol{x})^T R(\boldsymbol{x})$ y $\{H_k\}$ una sucesión de actualizaciones secantes de $\nabla^2 f(\boldsymbol{x})$. Supongamos que se cumplen las condiciones (h1), (h2) y (h3) del Teorema (3.6). Entonces, existen $\varepsilon, \delta > 0$ tales que si $||\boldsymbol{x_0} - \boldsymbol{x_\star}|| < \varepsilon$ y $|||H_0 - \nabla^2 f(\boldsymbol{x_\star})|| < \delta$, la sucesión generada por $\boldsymbol{x_{k+1}} = \boldsymbol{x_k} - (H_k)^{-1} \nabla f(\boldsymbol{x_k})$ está bien definida y converge q-superlinealmente a $\boldsymbol{x_\star}$. Si en lugar de (h2), $\nabla^2 f(\boldsymbol{x})$ cumple la condición de Lipschitz débil, $\{x_k\}$ está bien definida y converge al menos q-linealmente a $\boldsymbol{x_\star}$, ver* [15].

Note que los teoremas (3.18) y (3.19) son usados en el problema de Mínimos Cuadrados No Lineales (4) usando los métodos Cuasi-Newton.

4. Problema de Mínimos Cuadrados No Lineales (MCNL)

Dada una función R no lineal, $R \in \mathcal{C}^2$ definida por

$$\begin{aligned} R: \ \mathbb{R}^n &\to \mathbb{R}^m, \ \ m \geq n, \\ \boldsymbol{x} &\to R(\boldsymbol{x}) = \begin{bmatrix} r_1(\boldsymbol{x}) \\ r_2(\boldsymbol{x}) \\ \vdots \\ r_m(\boldsymbol{x}) \end{bmatrix}. \end{aligned} \tag{52}$$

El problema de **Mínimos Cuadrados No Lineales (MCNL)** consiste en resolver el siguiente problema de minimización sin restricciones

$$\begin{aligned} &\text{minimizar} \quad f(\boldsymbol{x}) = \tfrac{1}{2} R(\boldsymbol{x})^T R(\boldsymbol{x}). \\ &\boldsymbol{x} \in \mathbb{R}^n \end{aligned} \tag{53}$$

Note que

$$f(\boldsymbol{x}) = \frac{1}{2} R(\boldsymbol{x})^T R(\boldsymbol{x}) = \frac{1}{2} \sum_{i=1}^{m} r_i(\boldsymbol{x})^2 = \frac{1}{2} ||R(\boldsymbol{x})||_2^2.$$

En algunas ocasiones, el sistema de ecuaciones no lineales

$$r_i(\boldsymbol{x}) = 0, \text{ para } i = 1, \ldots, m,$$

es sobredeterminado y por lo general, no tiene solución. Para solucionar este problema se busca un vector $\boldsymbol{x}$ tal que

$$r_i(\boldsymbol{x}) \approx 0, \text{ para } i = 1, \ldots, m,$$

note que el vector gradiente de $f(\boldsymbol{x})$ está definido por

$$
\begin{aligned}
[\nabla f(\boldsymbol{x})]_j = \left[\frac{\partial f(\boldsymbol{x})}{\partial x_j}\right] &= \begin{bmatrix} r_1(\boldsymbol{x})\dfrac{\partial r_1}{\partial x_1} & + & r_2(\boldsymbol{x})\dfrac{\partial r_2}{\partial x_1} & + & \dots & + & r_n(\boldsymbol{x})\dfrac{\partial r_n}{\partial x_1} \\ r_1(\boldsymbol{x})\dfrac{\partial r_1}{\partial x_2} & + & r_2(\boldsymbol{x})\dfrac{\partial r_2}{\partial x_2} & + & \dots & + & r_n(\boldsymbol{x})\dfrac{\partial r_n}{\partial x_2} \\ \vdots & & \vdots & & \vdots & & \vdots \\ r_1(\boldsymbol{x})\dfrac{\partial r_1}{\partial x_n} & + & r_2(\boldsymbol{x})\dfrac{\partial r_2}{\partial x_n} & + & \dots & + & r_n(\boldsymbol{x})\dfrac{\partial r_n}{\partial x_n} \end{bmatrix} \\
&= \begin{bmatrix} \dfrac{\partial r_1}{\partial x_1} & \dfrac{\partial r_2}{\partial x_1} & \dfrac{\partial r_3}{\partial x_1} & \dots & \dfrac{\partial r_n}{\partial x_1} \\ \dfrac{\partial r_1}{\partial x_2} & \dfrac{\partial r_2}{\partial x_2} & \dfrac{\partial r_2}{\partial x_2} & \dots & \dfrac{\partial r_3}{\partial x_2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial r_1}{\partial x_n} & \dfrac{\partial r_2}{\partial x_n} & \dfrac{\partial r_3}{\partial x_n} & \dots & \dfrac{\partial r_n}{\partial x_n} \end{bmatrix} \begin{bmatrix} r_1(\boldsymbol{x}) \\ r_2(\boldsymbol{x}) \\ r_3(\boldsymbol{x}) \\ \vdots \\ r_n(\boldsymbol{x}) \end{bmatrix} \\
&= \sum_{i=1}^{m} r_i(\boldsymbol{x})\frac{\partial r_i}{\partial x_j}(\boldsymbol{x}) \text{ con } j = 1, \dots, n \qquad (54)
\end{aligned}
$$

donde $J(\boldsymbol{x})^T \in \mathbb{R}^{n\times m}$ es la matriz Jacobiana de R en $\boldsymbol{x} \in \mathbb{R}^n$ dada por

$$
J(\boldsymbol{x}) = \left[\frac{\partial r_j}{\partial x_i}\right] = \begin{bmatrix} \nabla r_1(\boldsymbol{x})^T \\ \nabla r_2(\boldsymbol{x})^T \\ \vdots \\ \nabla r_n(\boldsymbol{x})^T \end{bmatrix} \text{ con } i = 1, \dots, n \text{ y } j = 1, \dots, m,
$$

así, se tiene que

$$
\nabla f(\boldsymbol{x}) = J(\boldsymbol{x})^T R(\boldsymbol{x}), \qquad (55)
$$

y, la Hessiana esta definida por

$$\nabla^2 f(\boldsymbol{x}) = \left[\frac{\partial^2 f(\boldsymbol{x})}{\partial x_i \partial x_j}\right] = \begin{bmatrix} \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_1} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_2} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_3} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2 \partial x_1} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2 \partial x_2} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2 \partial x_3} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\boldsymbol{x})}{\partial x_n \partial x_1} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_n \partial x_2} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_n \partial x_3} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_n \partial x_n} \end{bmatrix}$$

$$= J(\boldsymbol{x})^T J(\boldsymbol{x}) + S(\boldsymbol{x}),$$

donde

$$\frac{\partial^2 f(\boldsymbol{x})}{\partial x_i \partial x_j} = \sum_{k=1}^{m} \frac{\partial r_k(\boldsymbol{x})}{\partial x_i} \frac{\partial r_k(\boldsymbol{x})}{\partial x_j} + \sum_{k=1}^{m} r_k(\boldsymbol{x}) \frac{\partial^2 r_k(\boldsymbol{x})}{\partial x_i \partial x_j}$$

$$S(\boldsymbol{x}) = \sum_{k=1}^{m} r_k(\boldsymbol{x}) \frac{\partial^2 r_k(\boldsymbol{x})}{\partial x_i \partial x_j}.$$

Por lo tanto se tiene que

$$\nabla^2 f(\boldsymbol{x}) = J(\boldsymbol{x})^T J(\boldsymbol{x}) + \sum_{k=1}^{m} r_k(\boldsymbol{x}) \frac{\partial^2 r_k(\boldsymbol{x})}{\partial x_i \partial x_j} \text{ para todo } i, j = 1, \ldots, n. \tag{56}$$

Note que la matriz $J(\boldsymbol{x})$ tiene solamente información de primer orden, es decir; las primeras derivadas parciales y $S(\boldsymbol{x})$ tiene información de segundo orden (es una combinación lineal de matrices Hessianas). La estructura de la matriz Hessiana de la función f se utiliza en los métodos más usados para resolver el problema de minimización (53). Es importante mencionar que la información de primer orden es fácil de calcular numéricamente, mientras que las de segundo orden son numéricamente difíciles de calcular, ya que involucra el cálculo de n Hessianos, lo que implica un alto costo computacional.

Si se considera la función R como lineal; es decir $R(\boldsymbol{x}) = A\boldsymbol{x} + \boldsymbol{b}$, donde $A \in \mathbb{R}^{m\times n}$ y $b \in \mathbb{R}^n$, entonces el problema (53) se convierte en un problema de mínimos cuadrados lineales, lo que es equivalente a resolver el problema

$$\begin{aligned} &\text{minimizar} \quad h(\boldsymbol{x}) = \frac{1}{2}||A\boldsymbol{x} + \boldsymbol{b}||^2. \\ &\boldsymbol{x} \in \mathbb{R}^n \end{aligned} \tag{57}$$

Como la función h es diferenciable en $\mathbb{R}^n$, entonces para cualquier minimizador el gradiente $\nabla h(\boldsymbol{x})$ es nulo (condiciones necesarias de primer orden) (1.34), y como $h(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T A^T A\boldsymbol{x} + \boldsymbol{b}^T A\boldsymbol{x} + \frac{1}{2}\boldsymbol{b}^T\boldsymbol{b}$ es una función cuadrática y su gradiente es $\nabla h(\boldsymbol{x}) = A^T A\boldsymbol{x} + A^T\boldsymbol{b}$ se resuelve el sistema de ecuaciones lineales

$$\nabla h(\boldsymbol{x}) = 0.$$

Si A es una matriz de rango completo y

$$A^T A\boldsymbol{x} = -A^T\boldsymbol{b},$$

usando el método de Newton (3.2) se resuelve el sistema de ecuaciones lineales (58) en una iteración (3.7) y, además, la solución teórica es

$$\boldsymbol{x} = -(A^T A)^{-1}A^T\boldsymbol{b}. \tag{58}$$

Note que la matriz $A^T A$ es simétrica y será definida positiva, si A es de rango completo; es decir si la matriz $A \in \mathbb{R}^{n\times m}$ y $n < m$, entonces tiene n columnas o filas linealmente independientes.

Nota 4.1. *La calidad de la solución de* (58) *dependerá del método utilizado para resolver el sistema de ecuaciones lineales, usualmente se utilizan dos métodos para*

solucionarlos

- ***Método de Choleski.*** *Se factoriza $A^TA = LL^T$, donde L es una matriz triangular inferior invertible. Se resuelve $LL^T\boldsymbol{x} = \boldsymbol{y}$ en dos pasos: $L\boldsymbol{z} = \boldsymbol{y}$, y después $L^T\boldsymbol{x} = \boldsymbol{z}$.*

- ***Método QR.*** *Consiste en la factorización $A = QR$, con Q matriz ortogonal y R triangular superior. Se resuelve $R\boldsymbol{x} = Q^T\boldsymbol{y}$.*

Para más detalle de los métodos para solucionar sistemas de ecuaciones lineales de forma computacional, ver[25]*.*

4.1. Métodos Cuasi-Newton estructurados

Para aprovechar la estructura de la ecuación (56), se han creado unos métodos Cuasi Newton que únicamente aproximan la matriz $S(\boldsymbol{x})$, creando aproximaciones de la matriz Hessiana de buena calidad, lo cual numéricamente tienen un comportamiento muy deseado, a continuación se presentan dos algoritmos que describen esta estrategia.

4.1.1. Método Gauss-Newton

La idea principal del método de Gauss-Newton es aproximar el problema de Mínimos Cuadrados No Lineales **(MCNL)** a un problema de Mínimos Cuadrados Lineales **(MCL)** alrededor de la iteración actual. Sea h_k la aproximación lineal de la función $R(\boldsymbol{x})$ alrededor de $\boldsymbol{x_k}$ dada por

$$R(\boldsymbol{x}) \approx h_k(\boldsymbol{x}) = R(\boldsymbol{x_k}) + J(\boldsymbol{x_k})(\boldsymbol{x} - \boldsymbol{x_\star}).$$

---

[25] Pablo Castañeda. «Matemática Computacional». En: ().

Donde $h_k : \mathbb{R}^n \to \mathbb{R}^m$ con $m > n$, y resolver el problema $h_k(\boldsymbol{x}) \approx 0$ como el problema de Mínimos Cuadrados Lineales (**MCL**)

$$\begin{aligned} \text{minimizar} \quad & \frac{1}{2}||R(\boldsymbol{x_k}) + J(\boldsymbol{x_k})(\boldsymbol{x} - \boldsymbol{x_k})||^2. \\ & \boldsymbol{x} \in \mathbb{R}^n \end{aligned} \tag{59}$$

El cual es equivalente a (57). Si la matriz $J(\boldsymbol{x_k})$ es de rango completo la solución está dada por (58), luego la solución al problema de mínimos cuadrados lineales es $-(J(\boldsymbol{x_k})^T J(\boldsymbol{x_k}))^{-1} J(\boldsymbol{x_k}) = \boldsymbol{x}$ siempre y cuando la matriz $J(\boldsymbol{x_k})$ sea de rango completo. Usando el método de Newton (3.2) la nueva aproximación a la solución está dado por

$$\begin{aligned} J(\boldsymbol{x_k})^T J(\boldsymbol{x_k}) s_{GN} &= -J(\boldsymbol{x_k}) R(\boldsymbol{x_k}), \\ \boldsymbol{x_{k+1}} &= \boldsymbol{x_k} + s_{GN}. \end{aligned}$$

Note que s_{GN} se conoce como el paso de Gauss-Newton. Sin embargo puede ser más practico resolver el problema (59) por los métodos mencionados en (4.1).
Dado que la matriz $J(\boldsymbol{x_k})$ es de rango completo, se tiene que $J(\boldsymbol{x_k})^T J(\boldsymbol{x_k})$ es no singular y el paso de Gauss-Newton s_{GN} esta bien definido, entonces $J(\boldsymbol{x_k})^T J(\boldsymbol{x_k})$ es definida positiva; es decir que

$$\begin{aligned} \nabla f(\boldsymbol{x_k}) s_{GN} &= (J(\boldsymbol{x_k})^T R(\boldsymbol{x_k}))^T (-(J(\boldsymbol{x_k})^T J(\boldsymbol{x_k}))^{-1} (J(\boldsymbol{x_k})^T R(\boldsymbol{x_k})) \\ &= -(R(\boldsymbol{x_k})^T J(\boldsymbol{x_k}))^T (J(\boldsymbol{x_k})^T J(\boldsymbol{x_k})) (J(\boldsymbol{x_k})^T R(\boldsymbol{x_k})) < 0, \end{aligned}$$

así, $\nabla f(\boldsymbol{x_k}) s_{GN} < 0$ es una dirección de descenso (3.1) concluyendo que el método de Gauss-Newton está bien definido.

Nota 4.2. *El método de Newton aproxima a $S(\boldsymbol{x})$, la segunda parte de la matriz*

Hessiana de f por cero, es decir si todos los $\boldsymbol{r}(\boldsymbol{x}) \to 0$, se tiene que $\nabla^2 f(\boldsymbol{x}) \approx J(\boldsymbol{x})^T J(\boldsymbol{x})$, lo cual se obtiene el método de Gauss-Newton.

Algoritmo 4.1.1. *(Algoritmo de Gauss-Newton)*

1. ***Inicialización.*** *Escoger un valor $\boldsymbol{x_0} \in \Omega \subset \mathbb{R}^n$ para $k = 0, 1, \ldots$ hasta convergencia hacer lo siguiente:*

2. ***Dirección.***

   Calcular $\boldsymbol{d_k}$, solución del sistema lineal $A\boldsymbol{d} = \boldsymbol{b}$, donde $A = J(\boldsymbol{x_k})^T J(\boldsymbol{x_k})$ y $\boldsymbol{b} = -J(\boldsymbol{x_k})^T R(\boldsymbol{x_k})$.

3. ***Descenso.*** *Evaluar $\boldsymbol{x_{k+1}} = \boldsymbol{x_k} + \boldsymbol{d_k}$.*

4. ***Actualización.*** *Hacer $k = k + 1$ y volver a (2).*

Nota 4.3. *Se puede agregar búsqueda lineal en el paso (3) como $\alpha_k = \arg\min_{\alpha} f(\boldsymbol{x_k} + \alpha \boldsymbol{d_k})$, y actualizar en la forma $\boldsymbol{x_{k+1}} = \boldsymbol{x_k} + \alpha_k \boldsymbol{d_k}$.*

El siguiente teorema, permite hacer un análisis de convergencia del método de Gauss-Newton según la linealidad y diferentes magnitudes de la función residuo.

Teorema 4.4. *Sean $\mathcal{D} \subset \mathbb{R}^n$ abierto y convexo, la función $R : \mathbb{R}^n \to \mathbb{R}^m$ y $f(\boldsymbol{x}) = \frac{1}{2} R(\boldsymbol{x})^T R(\boldsymbol{x})$ de clase $\mathcal{C}^2$. Supongamos que $J(\boldsymbol{x}) \in Lip_\gamma(\mathcal{D})$ con $|||J(\boldsymbol{x})^{-1}||| \leq \alpha$ para todo $\boldsymbol{x} \in \mathcal{D}$, existen $\boldsymbol{x_\star} \in \mathcal{D}$ y $\lambda, \sigma \geq 0$, tales que $J(\boldsymbol{x_\star})^T R(\boldsymbol{x_\star}) = 0$, donde λ es el valor propio más pequeño de $J(\boldsymbol{x_\star})^T J(\boldsymbol{x_\star})$ tal que*

$$|||(J(\boldsymbol{x}) - J(\boldsymbol{x_\star}))^T R(\boldsymbol{x_\star})||| \leq \sigma ||\boldsymbol{x} - \boldsymbol{x_\star}||,$$

para todo $\boldsymbol{x} \in \mathcal{D}$. Si $\sigma < \lambda$, entonces para cualquier $c \in \left(1, \frac{\lambda}{\sigma}\right)$, existe $\varepsilon > 0$ tal que para todo $\boldsymbol{x_0} \in B(\boldsymbol{x_\star}; \varepsilon)$ la sucesión generada por el método de Gauss-Newton

$$\boldsymbol{x_{k+1}} = \boldsymbol{x_k} - (J(\boldsymbol{x_k})^T J(\boldsymbol{x_k}))^{-1} J(\boldsymbol{x_k})^T R(\boldsymbol{x_k}),$$

está bien definida, converge a $\boldsymbol{x_\star}$ y satisface las desigualdades

$$\begin{aligned} ||\boldsymbol{x_{k+1}} - \boldsymbol{x_\star}|| &\leq \frac{c\sigma}{\lambda}||\boldsymbol{x_k} - \boldsymbol{x_\star}|| + \frac{c\alpha\gamma}{2\lambda}||\boldsymbol{x_k} - \boldsymbol{x_\star}||^2, \\ ||\boldsymbol{x_{k+1}} - \boldsymbol{x_\star}|| &\leq \frac{c\sigma + \lambda}{2\lambda}||\boldsymbol{x_k} - \boldsymbol{x_\star}|| < ||\boldsymbol{x_k} - \boldsymbol{x_\star}||, \end{aligned}$$

ver [15].

Proposición 4.5. *Si se satisface las condiciones del teorema (4.4) y si $R(\boldsymbol{x_\star}) = 0$, entonces existe $\varepsilon > 0$ tal que para todo $\boldsymbol{x_0} \in B(\boldsymbol{x_\star}; \varepsilon)$, la sucesión $\{\boldsymbol{x_k}\}$ generada por el método de Gauss-newton está bien definida y converge q-cuadráticamente a $\boldsymbol{x_\star}$.*

Note que la expresión (60) quiere decir que la convergencia es q-cuadrática y además, la norma la norma vectorial es $||\cdot||_2$ y la norma de Frobenius (matricial) es la norma inducida por la norma vectorial.

4.1.2. Método Levenberg-Marquardt

Note que las matriz $J(\boldsymbol{x_k})$ en (60) es de rango completo, lo que implica $J(\boldsymbol{x_k})^T J(\boldsymbol{x_k})$ es simétrica definida positiva y además, que $(J(\boldsymbol{x_k})^T J(\boldsymbol{x_k}))^{-1}$ existe. Pudiera ocurrir que el descenso sea muy pequeño cuando el menor valor propio de la matriz $J(\boldsymbol{x_k})^T J(\boldsymbol{x_k})$ sea muy cercano a cero. En estos casos se introduce el método de Levenberg-Marquadt para obtener $\boldsymbol{d_k}$ resolviendo el sistema lineal

$$(J(\boldsymbol{x_k})^T J(\boldsymbol{x_k}) + \mu_k I)\boldsymbol{d_k} = -J(\boldsymbol{x_k})^T R(\boldsymbol{x_k}), \text{con } \mu_k \geq 0 \text{ e } I \text{ es la matriz identidad,}$$

y por lo tanto

$$\boldsymbol{x_{k+1}} = \boldsymbol{x_k} - (J(\boldsymbol{x_k})^T J(\boldsymbol{x_k}) + \mu_k I)^{-1} J(\boldsymbol{x_k})^T R(\boldsymbol{x_k}).$$

Note que los valores propios de $J(\boldsymbol{x_k})^T J(\boldsymbol{x_k}) + \mu I$ son $\lambda_1 + \mu, \ldots, \lambda_n + \mu$, si se toma μ tal que $\lambda_1 + \mu, \ldots, \lambda_n + \mu > 0$, esta perturbación en la matriz garantiza que la aproximación de la Hessiana sea definida positiva.

Nota 4.6.

- *Cuando $\mu_k \to 0^+$, el método (4.1.2) se acerca al método de Gauss-Newton.*
- *Cuando $\mu_k \to \infty$, el método (4.1.2) se acerca al método de Máximo Descenso, ya que $\nabla f(\boldsymbol{x_k}) = J(\boldsymbol{x_k})^T R(\boldsymbol{x_k})$.*

Algoritmo 4.1.2. *(Algoritmo de Levenberg-Marquardt)*

1. ***Inicialización.*** *Escoger un valor $\boldsymbol{x_0} \in \Omega \subset \mathbb{R}^n$ para $k = 0, 1, \ldots$ hasta convergencia hacer lo siguiente:*
2. ***Dirección.***

   Calcular $\boldsymbol{d_k}$, solución del sistema lineal $A\boldsymbol{d} = \boldsymbol{b}$, donde $A = J(\boldsymbol{x_k})^T J(\boldsymbol{x_k}) + \mu_k I$ y $\boldsymbol{b} = -J(\boldsymbol{x_k})^T R(\boldsymbol{x_k})$.
3. ***Descenso.*** *Evaluar $\boldsymbol{x_{k+1}} = \boldsymbol{x_k} + \boldsymbol{d_k}$.*
4. ***Actualización.*** *Hacer $k = k + 1$ y volver a (2).*

El siguiente Teorema garantiza la convergencia del método de Levenberg-Marquardt bajo ciertas hipótesis.

Teorema 4.7. *Si se satisface las condiciones del Teorema (4.4) y sea la sucesión $\{x_k\}$ de números reales no negativos acotada por $b > 0$. Si $\sigma < \lambda$, entonces para algún $c \in \left(1, \dfrac{\lambda + b}{\sigma + b}\right)$ real, existe $\varepsilon > 0$ tal que para todo $\boldsymbol{x_0} \in B(\boldsymbol{x_\star}; \varepsilon)$ la sucesión generada por el método de Levenberg-Marquardt*

$$\boldsymbol{x_{k+1}} = \boldsymbol{x_k} - (J(\boldsymbol{x_k})^T J(\boldsymbol{x_k}) + \mu_k I)^{-1} J(\boldsymbol{x_k})^T R(\boldsymbol{x_k}),$$

está bien definido y se tiene que

$$||\boldsymbol{x_{k+1}} - \boldsymbol{x_\star}|| \leq \frac{c(\sigma+b)}{\lambda+b}||\boldsymbol{x_k} - \boldsymbol{x_\star}|| + \frac{c\alpha\gamma}{2(\lambda+b)}||\boldsymbol{x_k} - \boldsymbol{x_\star}||^2,$$

$$||\boldsymbol{x_{k+1}} - \boldsymbol{x_\star}|| \leq \frac{c(\sigma+b)+(\lambda+b)}{2(\lambda+b)}||\boldsymbol{x_k} - \boldsymbol{x_\star}|| < ||\boldsymbol{x_k} - \boldsymbol{x_\star}||.$$

Si $R(\boldsymbol{x_\star}) = 0$ y $\mu_k = O(||J(\boldsymbol{x_k})^T R(\boldsymbol{x_k})||)$, entonces la sucesión $\{x_k\}$ converge q-cuadráticamente (14) a $\boldsymbol{x_\star}$, ver[15].
Note que la norma usada en el Teorema (4.7) es la norma $||\cdot||_2$.

4.2. Método secante estructurado

Los métodos secantes (SR1-DFP-BFGS) descritos en el Capítulo 3, aproximan toda la matriz Hessiana sin aprovechar la estructura de (56); es decir, no se aprovechan los cálculos realizados del Jacobiano. Considerando la estructura dada por (56) del problema de minimización (53) se han diseñado métodos que se les conoce como **estructurados** y aprovechan dicha estructura. A continuación trataremos el método secante estructurado BFGS creado por Héctor Jairo Martínez en su tesis doctoral [26]. Resaltamos de antemano los resultados computacionales que se lograron con esta alternativa.

4.2.1. Método secante estructurado BFGS (Broyden-Fletcher-Golfarb-Shano)

Supongamos que la matriz hessiana de un problema de minimización se puede escribir de la forma

$$\nabla^2 f(\boldsymbol{x_k}) = C(\boldsymbol{x_k}) + S(\boldsymbol{x_k}), \text{ con } C(\boldsymbol{x_k}) \neq 0.$$

---

[26] JE Dennis, Héctor J Martinez y Richard A Tapia. «Convergence theory for the structured BFGS secant method with an application to nonlinear least squares». En: *Journal of Optimization Theory and Applications* 61 (1989), págs. 161-178.

El método secante estructurado consiste en hacer una aproximación de $S(\boldsymbol{x_k})$ conservando su estructura (simétrica y definida positiva). Definamos $H_k \approx \nabla^2 f(\boldsymbol{x_k})$.
El proceso iterativo del método secante estructurado BFGS se enfoca principalmente en calcular la matriz H_{k+1} que será utilizada en la siguiente iteración, por supuesto, este cálculo es posterior al cálculo de $\boldsymbol{s}_k$ y $\boldsymbol{x}_{k+1}$. Y para esto se procede de la siguiente forma:

$$\begin{aligned} A_{k+1} &= A_k + \triangle(\boldsymbol{s_k}, \boldsymbol{y_k}^{\#}, A_k, \boldsymbol{v_k}), \\ H_{k+1} &= C(\boldsymbol{x_k}) + A_{k+1}, \end{aligned}$$

donde la corrección de actualización secante $\triangle = \triangle(\boldsymbol{s_k}, \boldsymbol{y_k}^{\#}, A_k, \boldsymbol{v_k})$ está definida por

$$\begin{aligned} \triangle &= \frac{(\boldsymbol{y_k}^{\#} - A_k\boldsymbol{s_k})\boldsymbol{v_k}^T + \boldsymbol{v_k}(\boldsymbol{y_k}^{\#} - A_k\boldsymbol{s_k})^T}{\boldsymbol{v_k}^T s_k} - \frac{(\boldsymbol{y_k}^{\#} - A_k\boldsymbol{s_k})^T\boldsymbol{s_k}\boldsymbol{v_k}\boldsymbol{v_k}^T}{(\boldsymbol{v_k}^T\boldsymbol{s_k})^2} \qquad (60) \\ &= \left(I - \frac{\boldsymbol{v_k}\boldsymbol{s_k}^T}{\boldsymbol{v_k}^T\boldsymbol{s_k}}\right) A_k \left(I - \frac{\boldsymbol{s_k}\boldsymbol{v_k}^T}{\boldsymbol{v_k}^T\boldsymbol{s_k}}\right) + \frac{\boldsymbol{y_k}^{\#}\boldsymbol{v_k}^T + \boldsymbol{v_k}\boldsymbol{y_k}^{\#\boldsymbol{T}}}{\boldsymbol{v_k}^T\boldsymbol{s_k}} - \frac{\boldsymbol{s_k}^T\boldsymbol{y_k}^{\#}\boldsymbol{v_k}\boldsymbol{v_k}^T}{(\boldsymbol{v_k}^T\boldsymbol{s_k})^2}. \end{aligned}$$

Donde $\boldsymbol{s_k}$ es la solución del sistema $H_k\boldsymbol{s_k} = -\nabla f(\boldsymbol{x_k})$, $A_k \approx S(\boldsymbol{x_k})$, los vectores $\boldsymbol{y_k} \approx \nabla^2 f(\boldsymbol{x_k})\boldsymbol{s_k}$, $\boldsymbol{y_k}^{\#} \approx S(\boldsymbol{x_k})\boldsymbol{s_k}$.
Escogiendo el vector $\boldsymbol{v_k} \in \mathbb{R}^n$ de la siguiente forma: $\boldsymbol{v_k} = \boldsymbol{y_k} + \left(\frac{\boldsymbol{y_k}^T\boldsymbol{s_k}}{\boldsymbol{s_k}^T H_k\boldsymbol{s_k}}\right)^{\frac{1}{2}} H_k\boldsymbol{s_k}$ en la corrección de actualización secante (60) se obtiene el método secante estructurado BFGS. Note que A_{k+1} y H_{k+1} cumple con la condición secante con respecto a la matriz que aproxime

$$A_{k+1}\boldsymbol{s_k} = \boldsymbol{y}_k^{\#} \quad H_{k+1}\boldsymbol{s_k} = \boldsymbol{y_k},$$

con respecto a los anteriores razonamientos se tiene el algoritmo

Algoritmo 4.2.1. *(Algoritmo secante estructurado BFGS)*

1. ***Inicialización.*** *Escoger un valor* $\boldsymbol{x_0} \in \mathbb{R}^n$ *,* $A_0 \in \mathbb{R}^{n\times n} \approx S(\boldsymbol{x_\star})$ *para* $k = 0, 1, \ldots$ *hasta convergencia hacer lo siguiente:*

2. ***Dirección.***

   Resolver el sistema lineal $H_k\boldsymbol{s_k} = -\nabla f(\boldsymbol{x_k})$*, donde* $H_k = C(\boldsymbol{x_k}) + A_k$*.*

3. ***Descenso.*** *Evaluar* $\boldsymbol{x_{k+1}} = \boldsymbol{x_k} + \boldsymbol{s_k}$*.*

   Calcular: $\boldsymbol{y_k}^{\#} \approx S(\boldsymbol{x_k})\boldsymbol{s_k}$ *y* $\boldsymbol{y_k} \approx \nabla^2 f(\boldsymbol{x_k})\boldsymbol{s_k}$*.*

   $$\boldsymbol{v_k} = \boldsymbol{y_k} + \left(\frac{\boldsymbol{y_k}^T\boldsymbol{s_k}}{\boldsymbol{s_k}^T H_k \boldsymbol{s_k}}\right)^{\frac{1}{2}} H_k\boldsymbol{s_k}$$

   $$\triangle = \frac{(\boldsymbol{y_k}^{\#} - A_k\boldsymbol{s_k})\boldsymbol{v_k}^T + \boldsymbol{v_k}(\boldsymbol{y_k}^{\#} - A_k\boldsymbol{s_k})^T}{\boldsymbol{v_k}^T s_k} - \frac{(\boldsymbol{y_k}^{\#} - A_k\boldsymbol{s_k})^T\boldsymbol{s_k}\boldsymbol{v_k}\boldsymbol{v_k}^T}{(\boldsymbol{v_k}^T\boldsymbol{s_k})^2}$$

   $$A_{k+1} = A_k + \triangle$$

   $$H_{k+1} = C(\boldsymbol{x_{k+1}}) + A_{k+1}$$

4. ***Actualización.*** *Hacer* $k = k + 1$ *y volver a (3).*

En la siguiente parte, se analizará el método secante estructurado para el problema de **Mínimos Cuadrados No Lineales**.

4.2.2. Método secante estructurado BFGS y el Problema de Mínimos Cuadrados No Lineales

Para el problema de Mínimos Cuadrados No Lineales (53) se tiene que

$$\begin{aligned} \nabla f(\boldsymbol{x}) &= J(\boldsymbol{x})^T R(\boldsymbol{x}) \\ \nabla^2 f(\boldsymbol{x}) &= C(\boldsymbol{x}) + S(\boldsymbol{x}). \end{aligned}$$

En donde $C(\boldsymbol{x}) = J(\boldsymbol{x})^T J(\boldsymbol{x})$ y $S(\boldsymbol{x}) = \sum_{i=1}^m r_i(\boldsymbol{x})\nabla^2 r_i(\boldsymbol{x})$ con $C(\boldsymbol{x}) \neq 0$. El siguiente algoritmo se enfoca en minimizar el problema (**MCNL**) aprovechando la estructura del problema.

Algoritmo 4.2.2. *(Algoritmo secante estructurado BFGS)*

1. ***Inicialización.*** *Escoger un valor* $\boldsymbol{x_0} \in \mathbb{R}^n$ *,* $A_0 \in \mathbb{R}^{n\times m} \approx S(\boldsymbol{x_\star})$ *para* $k = 0, 1, \ldots$ *hasta convergencia hacer lo siguiente:*

   Resolver el sistema lineal $H_k \boldsymbol{s_k} = -J(\boldsymbol{x_k})^T R(\boldsymbol{x_k})$*, donde* $H_k = C(\boldsymbol{x_k}) + A_k$*.*

2. ***Descenso.*** *Evaluar* $\boldsymbol{x_{k+1}} = \boldsymbol{x_k} + \boldsymbol{s_k}$*.*

   Calcular:

   $$\boldsymbol{y_k}^{\#} = (J(\boldsymbol{x_{k+1}}) - J(\boldsymbol{x_k}))^T R(\boldsymbol{x_{k+1}})$$

   $$\boldsymbol{y_k} = \boldsymbol{y}^{\#} + J(\boldsymbol{x_{k+1}})^T J(\boldsymbol{x_{k+1}}) \boldsymbol{s_k}$$

   $$\boldsymbol{v_k} = \boldsymbol{y_k} + \left(\frac{\boldsymbol{y_k}^T \boldsymbol{s_k}}{\boldsymbol{s_k}^T H_k \boldsymbol{s_k}} \right)^{\frac{1}{2}} H_k \boldsymbol{s_k}$$

   $$\triangle = \frac{(\boldsymbol{y_k}^{\#} - A_k \boldsymbol{s_k}) \boldsymbol{v_k}^T + \boldsymbol{v_k} (\boldsymbol{y_k}^{\#} - A_k \boldsymbol{s_k})^T}{\boldsymbol{v_k}^T s_k} - \frac{(\boldsymbol{y_k}^{\#} - A_k \boldsymbol{s_k})^T \boldsymbol{s_k} \boldsymbol{v_k} \boldsymbol{v_k}^T}{(\boldsymbol{v_k}^T \boldsymbol{s_k})^2}$$

   $$A_{k+1} = A_k + \triangle$$

   $$H_{k+1} = C(\boldsymbol{x_{k+1}}) + A_{k+1}$$

3. ***Actualización.*** *Hacer* $k = k + 1$ *y volver a (2).*

Note que $A_k \approx S(\boldsymbol{x_k})$ y $\triangle = \triangle(\boldsymbol{s_k}, \boldsymbol{y}_k^{\#}, A_k, \boldsymbol{v_k})$ es conocida como la corrección de actualización secante.
El siguiente teorema nos da condiciones suficientes para garantizar la convergencia q-superlineal del método secante estructurado BFGS.

Teorema 4.8. *Sea* $\mathcal{D} \subset \mathbb{R}^n$ *un conjunto abierto y convexo,* $R : \mathcal{D} \to \mathbb{R}^m$ *sea* f *definida en (53) una función de clase* $\mathcal{C}^2$*. Supongamos que*

1. *Existe un* $\boldsymbol{x_\star}$ *tal que* $\nabla f(\boldsymbol{x_\star})$*.*

2. *Existe* $r > 0$ *tal que* $\nabla^2 f(\boldsymbol{x}) \in Lip_\gamma(B(\boldsymbol{x_\star; r}))$ *y* $C \in Lip_\gamma C(\mathcal{D})$*.*

3. *La matriz* $\nabla^2 f(\boldsymbol{x_\star})$ *es definida positiva.*

Sea $\boldsymbol{s} = \boldsymbol{x_1} - \boldsymbol{x_2}$. Considere los vectores $\boldsymbol{y} \approx \nabla^2 f(\boldsymbol{x_\star})\boldsymbol{s}$ y $\boldsymbol{y}^{\#} \approx S(\boldsymbol{x_\star})\boldsymbol{s}$, tales que $\boldsymbol{y} - \boldsymbol{y}^{\#} = C(\boldsymbol{v})\boldsymbol{s}$ para algún $\boldsymbol{v} \in seg(x_1, x_2)$ y $\boldsymbol{y}^{\#}$ satisface

$$||\boldsymbol{y}^{\#} - S(\boldsymbol{x_\star})\boldsymbol{s}|| \leq K_2\sigma(\boldsymbol{x_1}, \boldsymbol{x_2})||\boldsymbol{s}||,$$

para algún $K_2 > 0$, $\sigma(x_1, x_2) = max\{||\boldsymbol{x_1} - \boldsymbol{x_\star}||, ||\boldsymbol{x_2} - \boldsymbol{x_\star}||\}$ con $\boldsymbol{x_1}, \boldsymbol{x_2} \in \mathcal{D}$. Entonces existen constantes positivas ε, δ tal que, para $\boldsymbol{x_0} \in \mathbb{R}^n$ y $A_0 \in \mathbb{R}^{n\times n}$ simétrica tal que satisfacen $||\boldsymbol{x_0} - \boldsymbol{x_\star}|| \leq \varepsilon$ y $||A_0 - S(\boldsymbol{x_\star})|| \leq \delta$, la sucesión $\{x_k\}$ generada por el método secante estructurado BFGS para el problema (53) converge q-superlinealmente a $\boldsymbol{x_\star}$, ver [15] *.*

5. Redes neuronales profundas y los mecanismos de aprendizaje

En este capítulo hablaremos de la implementación matemática del algoritmo *feed-forward* o propagación hacia adelante de una red neuronal profunda. También se introduce la implementación matemática del algoritmo *backpropagation* o retropropagación, calculando las expresiones (65), (67) y (68) [27]. Después se presenta el algoritmo de optimización máximo descenso [28], el cual se encarga de minimizar la función objetivo mediante los errores de los datos reales y los valores predichos por la red [29]. Finalmente se presenta una ilustración (Ejemplo 5.1) de una red neuronal profunda, realizando el proceso de aprendizaje.

5.1. Red neuronal artificial

Una neurona artificial es un dispositivo simple de cálculo que, a partir de un vector de entrada procedente del exterior, proporciona una única respuesta o salida. En el contexto de las matemáticas es un problema de regresión lineal múltiple, que es regularizado por una función de activación, la cual produce un valor de salida donde el vector de entrada $\boldsymbol{x_i}$ para todo $i \in \{1, 2, \ldots, n\}$ representa el conjunto de datos, el vector $\boldsymbol{w_i}$ para todo $i \in \{1, 2 \ldots n\}$ representa los pesos, la función $\Sigma : \mathbb{R}^n \to \mathbb{R}$ representa las combinaciones lineales de los pesos con el valor de los

---

[27] Robert Hecht-Nielsen. «Theory of the backpropagation neural network». En: *Neural networks for perception*. Elsevier, 1992, págs. 65-93.

[28] Marcin Andrychowicz et al. «Learning to learn by gradient descent by gradient descent». En: *Advances in neural information processing systems* 29 (2016).

[29] Sepp Hochreiter, A Steven Younger y Peter R Conwell. «Learning to learn using gradient descent». En: *Artificial Neural Networks—ICANN 2001: International Conference Vienna, Austria, August 21–25, 2001 Proceedings 11*. Springer. 2001, págs. 87-94.

respectivos datos, y la función $\sigma : \mathbb{R} \to \mathbb{R}$ representa la función de activación, que se encarga de regularizar el valor de la función Σ y se obtiene un valor de salida $\hat{y}$. La representación de una neurona se puede ver en la **Figura 6**.

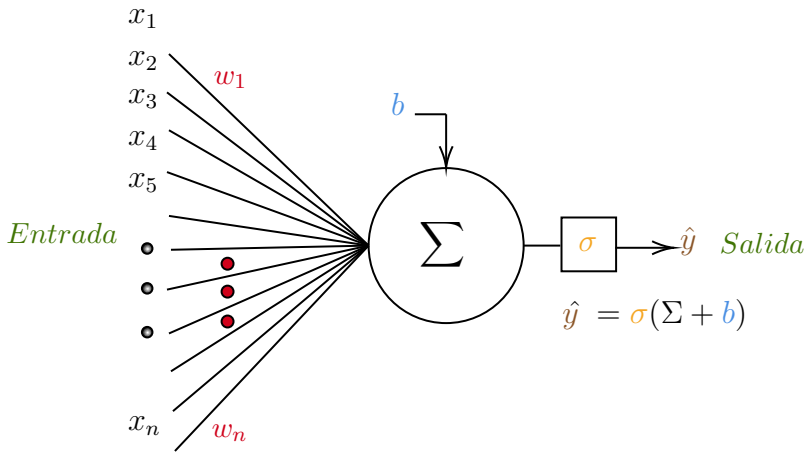

Figura 6. Representación de una neurona artificial. Hecho por el autor.

Las redes neuronales son modelos creados al ordenar operaciones matemáticas siguiendo una determinada estructura. La forma más común de representar la estructura de una red es por medio de capas (*layers*), formadas a su vez por neuronas. En esta sección se describe las redes neuronales profundas, con base en [30] [31] [27]. Cada neurona realiza una operación sencilla y está conectada a las neuronas de la capa anterior mediante pesos. La primera capa de la red neuronal se conoce como la capa de entrada o *Input layer*, la cual se encarga de recibir los datos de entrenamiento. La capa intermedia o *Hidden layer*, recibe los valores de la capa de entrada, ponderados por los pesos (nodos). La última capa, llamada *output layer*, combina los valores que salen de la capa intermedia para generar la predicción.

---

[30] Massimo Buscema. «Back propagation neural networks». En: *Substance use & misuse* 33.2 (1998), págs. 233-270.

[31] Jacques De Villiers y Etienne Barnard. «Backpropagation neural nets with one and two hidden layers». En: *IEEE transactions on neural networks* 4.1 (1993), págs. 136-141.

Cada neurona de la capa de entrada representa el valor de uno de los datos de entrenamiento. Los nodos representan los coeficientes de las combinaciones lineales de un modelo de regresión lineal. En términos de redes neuronales estas combinaciones están ponderadas por pesos, y la neurona de salida representa el valor de la predicción del modelo. Las funciones de activación (*Rectified linear unit*, *sigmoide*, *tanh*, *etc*.)[32], controlan en gran medida la información que se propaga desde una capa a la siguiente (*feed forward* o propagación hacia adelante) y la función costo o *loss function* es la encargada de cuantificar la distancia entre el dato real y el valor predicho por la red, en otras palabras, mide cuanto se equivoca la red al realizar las predicciones. El modelo de una red neuronal con una única capa (*single-layer*), aunque aportó un gran avance en el campo del *machine learning*, solo es capaz de aprender patrones sencillos. Para superar esta limitación, combinando múltiples capas ocultas, la red puede aprender relaciones mucho más complejas entre los valores predichos y los valores dados [33]. A esta estructura se le conoce como *multi-layer*, y puede considerarse como un modelo de *deep learning*. La estructura *multi-layer* consta de varias capas de neuronas ocultas, cada neurona está conectada a todas las neuronas de la capa anterior y a las de la capa posterior, aunque no estrictamente necesario, todas las neuronas que forman parte de una misma capa suelen emplear la misma función de activación [32]. Combinando múltiples capas ocultas y funciones de activación no lineales, los modelos de redes pueden aprender múltiples patrones. De hecho, está demostrado que, con suficientes neuronas, una red neuronal con una capa oculta puede aproximarse uniformemente a cualquier función continua en un conjunto compacto, si la función de activación no es un polinomio

---

[32] Sagar Sharma, Simone Sharma y Anidhya Athaiya. «Activation functions in neural networks». En: *Towards Data Sci* 6.12 (2017), págs. 310-316.

[33] Yann LeCun, Yoshua Bengio y Geoffrey Hinton. «Deep learning». En: *nature* 521.7553 (2015), págs. 436-444.

(en particular, las siguientes funciones: *tanh, Rectified linear unit, sine, cosine, etc*) [32]. Así una red neuronal con una capa profunda es un aproximador universal para cualquier función (**Teorema de aproximación universal**) [34]. En particular una red profunda es una red *multi-layer*. Para escribir el algoritmo de aprendizaje de una red profunda se usarán las siguientes notaciones:

Para representar el número total de capas de una red profunda se usará la letra L. El conjunto de datos de entrenamiento que se evalúan en la red profunda se representa por $\left\{\left(\boldsymbol{x^i}, y^i\right)\right\}_{i=1}^{n} \in \mathbb{R}^P \times \mathbb{R}$. Para algún caso de entrenamiento el vector $\boldsymbol{x^i} = (x_1, \cdots, x_P)$ representa las entradas a evaluar en la red profunda. El vector $\boldsymbol{h_j^{(l)}}$ es el valor que toma la unidad j de la capa l, para $j \in \{0, \ldots, m_l\}$ donde m_l es el número de unidades de la capa l. Por definición se tiene que $h_0^{(l)} = 1$ para todo $l \in \{1, \ldots, L-1\}$. El valor $\boldsymbol{h_j^{(l)}} = x_j$ para todo $j \in \{1, 2, \ldots, P\}$ representa los valores de los datos de entrada. Para $l \in \{1, \ldots, L\}$ la matriz de pesos que se denota por $w_{j,k}^{(l)}$ es el peso de la salida $h_k^{(l)}$ de la capa l en la entrada $h_k^{(l+1)}$ de la capa $l+1$ con $j \in \{1, \ldots, m_l\}$ donde m_l representa el número de unidades de la capa l. La matriz de sesgos o *bias* se denota por $w_{j,0}^{(l)}$ para todo $l \in \{1, \ldots L\}$ y $j \in \{1, \ldots m_l\}$ donde m_l representa el número de unidades de la capa l. La notación m_i con $i \in \{1, \ldots, L\}$ representa el número de neuronas o unidades en su respectiva capa y Φ representa la función de pérdida o costo encargada de calcular los errores entre los datos de entrenamiento y las predicciones.

---

[34] Felix Voigtlaender. «The universal approximation theorem for complex-valued neural networks». En: *Applied and Computational Harmonic Analysis* 64 (2023), págs. 33-61.

5.2. Propagación hacia adelante o *feed-forward*

Para implementar la propagación o *feed-forward* de la red profunda de la **Figura 7,** note que el conjunto de parámetros o pesos de cada capa se representa por $W^{(l)} \in \mathbb{R}^{m_{l+1}} \times \mathbb{R}^{m_l}$. La red completa se caracteriza por el número total de capas, nodos de las capa ocultas y las matrices de pesos en sus respectivas capas ocultas, las cuales se representan como $W^{(1)}, W^{(2)}, \ldots, W^{(L-1)}$. Adicionalmente para facilitar la notación, se escribe de forma vectorial la siguiente expresión

$$\boldsymbol{h^{(l)}} = \left[h_0^{(l)}, h_1^{(l)}, \ldots, h_{m_l}^{(l)}\right]^T$$

la cual representa las salidas calculadas en sus respectivas capas. Cuando se implementa la propagación hacia adelante de la red profunda, se agregan entradas adicionales en cada capa, representadas por $h_0^{(l)}$, $l \in \{1, 2, \cdots, L-1\}$, donde $h_0^{(l)} = 1$. Para facilitar las operaciones matriciales se agrega una columna a la matriz de pesos W que representan los sesgos en sus respectivas capas.

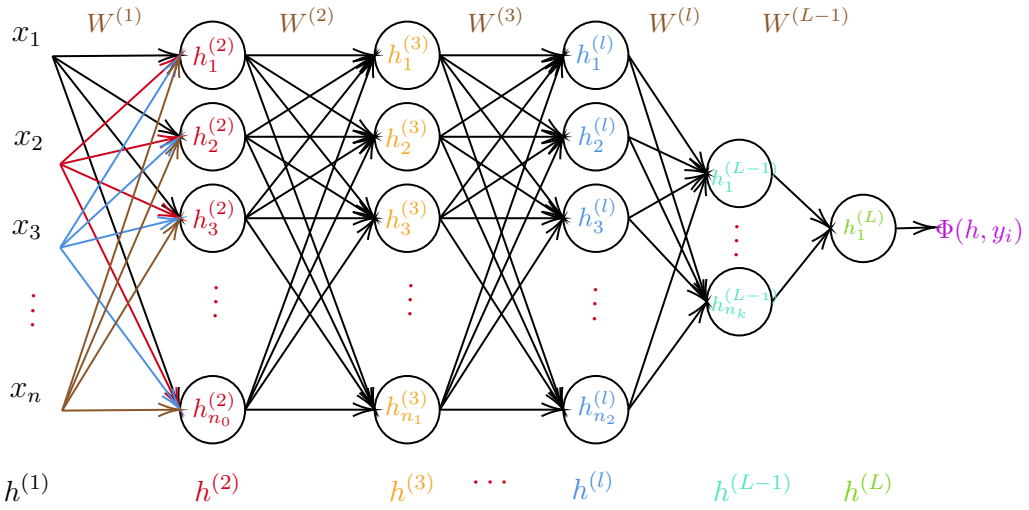

Figura 7. Representación del *feed-forward* de una red neuronal profunda. Hecho por el autor.

En la primera capa por definición los datos de entrada se representan de forma

vectorial como

$$\boldsymbol{h^{(1)}} = [1, x^T]^T.$$

En la segunda capa los valores de salida dependen del resultado de las operaciones vectoriales entre la matriz de pesos $W^{(1)}$ con respecto a los datos de entrada $\boldsymbol{h^{(1)}}$ de la primera capa, los cuales son regularizadas por la función de activación g mediante la siguiente expresión

$$\boldsymbol{h^{(2)}} = g_{\otimes}\left(W^{(1)}\boldsymbol{h^{(1)}}\right).$$

Observe que se emplea la notación $g_{\otimes}$, usando un $\otimes$ después del identificador de la función, para mostrar que la evaluación es realizada componente a componente, es decir, si $z = [z_1, z_2, \ldots, z_m]^T$, entonces

$$g_{\otimes}(z) = \begin{bmatrix} g(z_1) \\ g(z_2) \\ \vdots \\ g(z_m) \end{bmatrix}. \tag{61}$$

Generalizando la expresión para las salidas de cualquier capa de la red profunda de la Figura 7. Se tiene que para cualquier capa arbitraria l donde $l \in \{1, \ldots, L-1\}$, depende de los valores obtenidos por la regularización de la función de activación g y de las operaciones vectoriales dadas por la capa $l-1$ mediante la expresión

$$\boldsymbol{h^{(l)}} = g_{\otimes}\left(W^{(l-1)}\boldsymbol{h^{(l-1)}}\right).$$

Finalmente, para la capa L los datos de salida dependen de los resultados de las operaciones vectoriales entre los pesos y las salidas de la capa $L-1$. Además la función de activación de la última capa, se escoge dependiendo del contexto del problema y no necesariamente es la misma función de activación g. La expresión

para obtener los valores de salida en la última capa es

$$\boldsymbol{h} = g_{\otimes}\left(W^{(L-1)}\boldsymbol{h}^{(\boldsymbol{L-1})}\right).$$

El conjunto de pasos en el que inicialmente se introducen los datos en la red profunda hasta que la red muestra una predicción o dato de salida, y es evaluado en la función de perdida Φ encargada de medir los errores de los datos reales con respecto a las predicciones, se conoce como el algoritmo de propagación hacia adelante o *feed-forward*.

5.3. Retropropagación o *backpropagation*

Para implementar el algoritmo de retropropagación (*backpropagation*) en la red profunda de la **Figura 7,** se hace uso de la regla de la cadena (*chain rule*) para funciones de varias variables con el objetivo de calcular el gradiente [35].

El gradiente se encarga de cuantificar la importancia que tiene cada peso y *bias* con respecto a sus predicciones. En el caso de las redes neuronales profundas, la derivada parcial del error con respecto a un parámetro (peso o *bias*) mide cual es responsabilidad que ha tenido al asignarle un error. Gracias a esto, se puede identificar que pesos de la red hay que modificar para mejorarla.

Mediante el algoritmo del *backpropagation* se calculan los gradientes y además se deduce una expresión recursiva que se encarga de propagar los errores de la red profunda mediante los gradientes desde el final de la red (capa de salida) hasta el principio.

Para derivar con respecto a $w_{j,k}^{(l)}$ se resalta la dependencia explicita de Φ de las salidas de la capa $l+1$ escribiendo $\Phi = \Phi\left(h_0^{(l+1)}, h_1^{(l+1)}, \cdots, h_{m_{l+1}}^{(l+1)}\right)$ y por definición $h_0^{(l)} =$

---

[35] Chrisantha Fernando et al. «Pathnet: Evolution channels gradient descent in super neural networks». En: *arXiv preprint arXiv:1701.08734* (2017).

1 para todo l con $l \in \{1, \ldots, L-1\}$. Por lo tanto se tiene $\Phi = \Phi\left(h_1^{(l+1)}, \ldots, h_{m_{l+1}}^{(l+1)}\right)$. Se va a derivar con respecto a la capa $l+1$, usando la regla de la cadena para funciones de varias variables, así se tiene que

$$\frac{\partial \Phi}{\partial w_{j,k}^{(l)}} = \sum_{t=1}^{m_{l+1}} \frac{\partial \Phi}{\partial h_t^{(l+1)}} \frac{\partial h_t^{(l+1)}}{\partial w_{j,k}^{(l)}}. \tag{62}$$

Si observamos donde aparece $w_{j,k}^{(l)}$ en la gráfica de la red, ver **Figura 7**. Se puede concluir que $\frac{\partial h_t^{(l+1)}}{\partial w_{j,k}^{(l)}} = 0$ cuando $t \neq j$ (pues $h_t^{(l+1)}$ no depende de $w_{j,k}^{(l)}$) entonces para toda $j \in \{1, \cdots, m_{l+1}\}$ y $k \in \{0, 1, \cdots, m_l\}$ se tiene que

$$\frac{\partial \Phi}{\partial w_{j,k}^{(l)}} = \frac{\partial \Phi}{\partial h_j^{(l+1)}} \frac{\partial h_j^{(l+1)}}{\partial w_{j,k}^{(l)}}. \tag{63}$$

Adicionalmente, como

$$h_j^{(l+1)} = g_{\otimes}\left(z_j^{(l+1)}\right)$$

y los $h_k^{(l)}$ no dependen de $w_{j,k}^{(l)}$, por medio de la regla de la cadena, se tiene que

$$\frac{\partial h_j^{(l+1)}}{\partial w_{j,k}^{(l)}} = g'_{\otimes}\left(z_j^{(l+1)}\right) h_k^{(l)}. \tag{64}$$

Esta última expresión solo requiere de la derivada de Φ con respecto a h y los valores obtenidos por el *feed-forward*. Se buscará una expresión recursiva, solo nos queda calcular las derivadas parciales $\frac{\partial \Phi}{\partial h_j^{(l)}}$ con $j \in \{1, \cdots, m_l\}$, obteniendo una recursividad (retropropagación) para esta cantidad. Aplicando nuevamente la regla de la cadena con respecto a la capa l se obtiene la siguiente expresión recursiva

$$\frac{\partial \Phi}{\partial h_j^{(l)}} = \sum_{s=1}^{m_{l+1}} \frac{\partial \Phi}{\partial h_s^{(l+1)}} \frac{\partial h_s^{(l+1)}}{\partial h_j^{(l)}}, \tag{65}$$

la cual se puede entender a partir del siguiente diagrama de la **Figura 8**.

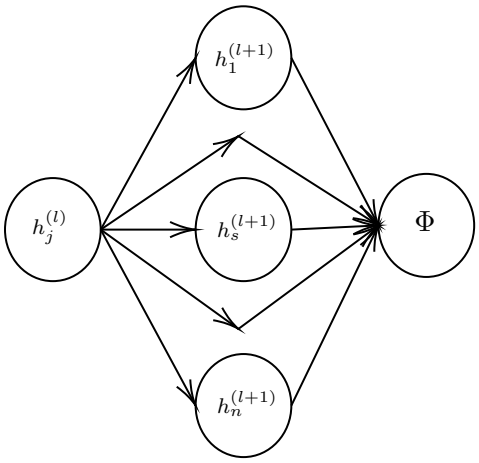

Figura 8. Representación de los valores de h en sus respectivas capas. Hecho por el autor.

Observe que la ecuación empieza en $s = 1$, no en $s = 0$, pues $h_0^{(l+1)}$ no depende de $h_k^{(l+1)}$, en este caso los elementos de la suma no necesariamente se anulan, como $h_s^{(l+1)} = g\left(z_s^{(l+1)}\right)$ considere la siguiente derivada

$$\frac{\partial h_s^{(l+1)}}{\partial h_j^{(l)}} = g'_{\otimes}\left(z_s^{(l+1)}\right) w_{s,j}^{(l)}. \tag{66}$$

De modo que

$$\frac{\partial \Phi}{\partial h_j^{(l)}} = \sum_{s=1}^{m_{l+1}} \frac{\partial \Phi}{\partial h_s^{(l+1)}} g'_{\otimes}\left(z_s^{(l+1)}\right) w_{s,j}^{(l)}. \tag{67}$$

Observe que la expresión nos da una expresión recursiva para las derivadas parciales que nos hace falta calcular. Ahora se va a simplificar la expresión (67) obteniendo las siguiente recursividad. Definamos

$$\delta_s^{(l+1)} = \frac{\partial \Phi}{\partial h_s^{(l+1)}} g'_{\otimes}\left(z_s^{(l+1)}\right). \tag{68}$$

De manera que la expresión recursiva es

$$\frac{\partial \Phi}{\partial h_j^{(l)}} = \sum_{s=1}^{m_{l+1}} \delta_s^{(l+1)} w_{s,j}^{(l)}. \tag{69}$$

Para todo $l \in \{2, 3, \cdots, L-1\}$ por medio de la expresión (66) se obtiene la siguiente expresión recursiva

$$\delta_j^{(l)} = \left(\sum_{s=1}^{m_{l+1}} \delta_s^{(l+1)} w_{s,j}^{(l+1)} \right) g'_{\otimes} \left(z_j^{(l)} \right). \tag{70}$$

Para $j \in \{1, 2, \cdots, m_l\}$. Finalmente usando (62) y (65), se obtiene que

$$\frac{\partial \Phi}{\partial w_{j,k}^{(l)}} = \delta_j^{(l+1)} h_k^{(l)}. \tag{71}$$

Usando (65), (67) y (68) ya podemos hacer la retropropagación o *backpropagation* sobre cada caso de entrenamiento, solo falta obtener el gradiente sobre la muestra de entrenamiento para ajustar sus parámetros. Además la ecuación (67) se puede ver de forma vectorial. Sea $W_*^{(l+1)}$ la matriz de pesos sin la columna correspondiente a los *bias*. Tenemos

$$\delta^{(l+1)} = \frac{\partial \Phi}{\partial h^{(l+1)}} g'_{\otimes} \left(z^{(l+1)} \right), \tag{72}$$

y

$$\delta^{(l)} = \left(W_*^{(l)} \right)^T \delta^{(l+1)} \odot g'_{\otimes} \left(z^{(l)} \right), \tag{73}$$

donde $\bigodot$ denota el producto *Hadamard* (componente a componente), con esto finalizamos los cálculos necesarios. Lo interesante es que $\delta^{(l)}$ con $l \in \{2 \ldots, L-1\}$ se calcula de manera recursiva. Para más detalles, ver [31] [30]. El siguiente paso es necesario, en el que se determina cómo y cuánto hay que modificar los correspondientes parámetros.

5.4. Algoritmo de aprendizaje en el entrenamiento de una red neuronal

El proceso de entrenamiento de una red neuronal profunda consiste en ajustar el valor de los pesos y *bias* de tal forma que las predicciones que se generen, tengan el menor error posible. Gracias a esto, el modelo es capaz de identificar qué predicciones son de mayor importancia, y además saber cómo están relacionados entre ellos y su estimación [30]. También es indispensable escoger la función costo u objetivo a optimizar, para el caso de estudio, se usa la función conocida como el error cuadrático medio, que se define como $f : \mathbb{R}^n \to \mathbb{R}$

$$f(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{74}$$

con la finalidad de minimizar la función $f(\boldsymbol{w})$. Esto se traduce a resolver un problema de **Mínimos Cuadrados No Lineales**.

A continuación se muestra una idea intuitiva de cómo se entrena una red neuronal profunda. La idea intuitiva de cómo entrenar una red profunda es la siguiente:

- Iniciar la red con valores aleatorios, con respecto a los pesos y *bias*.
- Calcular el error que comete la red al hacer sus predicciones para cada observación de entrenamiento y promediar los errores de todas las observaciones.
- Identificar la responsabilidad que ha tenido cada peso y *bias* en el error de la predicción.
- Repetir los pasos necesarios hasta que la red sea suficientemente buena.

La implementación de esta idea ha requerido la combinación de múltiples métodos matemáticos, en efecto, el algoritmo de retropropagación o *backpropagation* y la optimización por medio del máximo descenso (*gradient descent*) [28]. El algoritmo del máximo descenso (*gradient descent*) es un algoritmo de optimización que permite

minimizar una función, haciendo actualizaciones de sus parámetros en la dirección opuesta del gradiente. Aplicado a las redes neuronales, el máximo descenso permite ir actualizando los pesos y *bias* del modelo para reducir su error. El algoritmo se puede ver en la **Figura 9**. Aquí, el hiperparámetro α es conocido como la tasa de aprendizaje.

$Algoritmo$

$1 : Inicializar\ w_{j,k}^{(l)} \in \mathbb{R}^{m_{l+1}} \times \mathbb{R}^{m_l}$
$2 : Inicializar\ k\ \leftarrow 0$
$3 : Repetir$
$4 : Para\ todo\ j \in \{1, \ldots, m_l\}\ \ w_{j,\,k+1}^{(l)} \leftarrow w_{j,\,k}^{(l)} - \alpha \dfrac{\partial \Phi\left(w^{(l)}\right)}{\partial w_{j,\,k}^{(l)}}\ con\ w^{(l)} = w_k^{(l)}$
$5 :\ \ k\ \leftarrow\ k+1$
$6 :\ \ Hasta\ la\ convergencia$

Figura 9. Algoritmo de máximo descenso. Inspirada en[36].

La tasa de aprendizaje (α) establece cuál es la rapidez de cambio en los parámetros del modelo, a medida que se optimiza (aprende). Este hiperparámetro es uno de los más complicados de establecer, ya que depende de los datos e interacciona con el resto de los hiperparámetros. Si la tasa de aprendizaje es muy grande, el proceso de optimización puede ir saltando de una región a otra sin que el modelo sea capaz de aprender, es decir la optimización no converge. Si por el contrario, la tasa de aprendizaje es muy pequeña, el proceso de entrenamiento puede tardar demasiado y no llegar a completarse. Para mas detalles ver [37].

Existen dos maneras de entrenar una red neuronal profunda o totalmente conectada, una conocida como *aprendizaje por lotes o aprendizaje batch* el modelo se entrena con todos los datos disponibles, lo que se hace antes de realizar ningún tipo de predicción. Si los datos de entrenamiento cambian; es decir si se usan nuevos

[37] Matthew D Zeiler. «Adadelta: an adaptive learning rate method». En: *arXiv preprint arXiv:1212.5701* (2012).

datos en el modelo deberá ser reentrenado desde el comienzo. El *aprendizaje en linea o serie* los pesos de un modelo se actualizan de manera incremental después de la presentación de cada patrón de entrenamiento, en lugar de actualizar los pesos solo después de haber pasado por todo el conjunto de datos de entrenamiento, como suele ocurrir en el *aprendizaje por lotes*. En este trabajo nuestro interés es el entrenamiento por lotes, donde a partir de un conjunto de p datos, de los cuales se conoce la "salida", la red es entrenada una sola vez, para posteriormente realizar interpolaciones o extrapolaciones con otros datos.

Ejemplo 5.1. *A continuación se ilustra el algoritmo del feed-forward y el backpropagation en una red neuronal profunda. En particular se usará la red representada en la* ***Figura 10.***

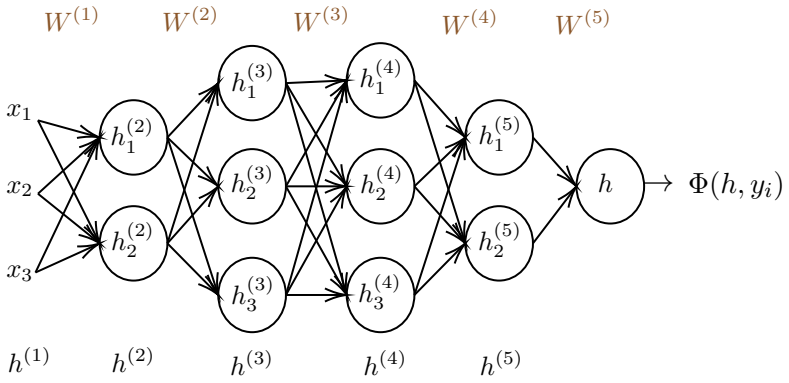

Figura 10. Representación de una red neuronal profunda. Hecho por el autor.

La red neuronal representada en la ***Figura 10*** *tiene 6 capas (una capa de entrada, 4 capas ocultas y una de salida). Primero se debe comenzar implementando el feed-forward que solo devuelve el costo o pérdida sin regularización de los parámetros, donde $h^{(l)}$ se calcula tal como se definió en la* ***Figura 7*** *y además g representa la función de activación del modelo. El algoritmo del feed-forward calcula*

$$\boldsymbol{h^{(l)}} = g\left(W^{(l)}\boldsymbol{h^{(l-1)}}\right) \text{ con } l \in \{1, \ldots, L\}$$

para cada elemento correspondiente a los datos de entrenamiento $\left\{\left(\boldsymbol{x^i}, y^i\right)\right\}_{i=1}^{n} \in \mathbb{R}^3 \times \mathbb{R}$ *y hace sus respectivas operaciones vectoriales con la regularización de la función de activación* g.

El vector de los datos de entrada es

$$\boldsymbol{h^{(1)}} = \begin{bmatrix} 1 & x_1 & x_2 & x_3 \end{bmatrix}^T.$$

La matriz de los pesos que conectan los datos de entrada de la primera capa con las neuronas de la segunda capa es

$$W^{(1)} = \begin{bmatrix} w_{10}^{(1)} & w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{20}^{(1)} & w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \end{bmatrix}. \tag{75}$$

El vector que representa la entrada de la segunda capa es $z^{(2)} = [z_1^{(2)}, z_2^{(2)}]^T$*, donde* $z_i^{(2)}$ *es la entrada de la neurona* i*,* $i \in \{1, 2\}$*. Este vector* $z^{(2)}$ *viene dado por,*

$$z^{(2)} = W^{(1)}\boldsymbol{h^{(1)}} = \begin{bmatrix} w_{10}^{(1)} & w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{20}^{(1)} & w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \end{bmatrix} \begin{bmatrix} 1 \\ h_1^{(1)} \\ h_2^{(1)} \\ h_3^{(1)} \end{bmatrix}. \tag{76}$$

El siguiente paso es aplicar la función de activación de cada neurona para obtener las salidas u output de las neuronas de la segunda capa. Obtenemos el vector $\boldsymbol{h^{(2)}} = [h_1^{(2)}, h_2^{(2)}]^T$ *que representa la salida de la segunda capa, donde* $h_i^{(2)}$ *representa la salida de la neurona* i*,* $i = \{1, 2\}$*, por medio de la siguiente expresión*

$$\boldsymbol{h^{(2)}} = \begin{bmatrix} h_1^{(2)} \\ h_2^{(2)} \end{bmatrix} = \begin{bmatrix} g\left(z_1^{(2)}\right) \\ g\left(z_2^{(2)}\right) \end{bmatrix} = g_{\otimes}(\boldsymbol{h^{(2)}}). \tag{77}$$

La matriz de los pesos que conectan los valores de salida de la primera capa con las neuronas de la segunda capa es

$$W^{(2)} = \begin{bmatrix} w_{10}^{(2)} & w_{11}^{(2)} & w_{12}^{(2)} & w_{13}^{(2)} \\ w_{20}^{(2)} & w_{21}^{(2)} & w_{22}^{(2)} & w_{23}^{(2)} \\ w_{30}^{(2)} & w_{31}^{(2)} & w_{32}^{(2)} & w_{33}^{(2)} \end{bmatrix}. \tag{78}$$

El vector que representa la entrada de la tercera capa es $z^{(3)} = [z_1^{(3)}, z_2^{(3)}, z_3^{(3)}]^T$*, donde* $z_i^{(3)}$ *es la entrada de la neurona* i*,* $i \in \{1, 2, 3\}$*, por medio de la siguiente expresión*

$$z^{(3)} = W^{(2)}\boldsymbol{h^{(2)}} = \begin{bmatrix} w_{10}^{(2)} & w_{11}^{(2)} & w_{12}^{(2)} \\ w_{20}^{(2)} & w_{21}^{(2)} & w_{22}^{(2)} \\ w_{30}^{(2)} & w_{31}^{(2)} & w_{32}^{(2)} \end{bmatrix} \begin{bmatrix} 1 \\ h_1^{(2)} \\ h_2^{(2)} \end{bmatrix}. \tag{79}$$

El siguiente paso es aplicar la función de activación a cada neurona para obtener las salidas u output de las neuronas de la tercera capa. Obtenemos el vector $\boldsymbol{h^{(3)}} = [h_1^{(3)}, h_2^{(3)}, h_3^{(3)}]^T$ *que representa la salida de la tercera capa, donde* $h_i^{(3)}$ *representa la salida de la neurona* i*,* $i \in \{1, 2, 3\}$*, por medio de la siguiente expresión*

$$\boldsymbol{h^{(3)}} = \begin{bmatrix} h_1^{(3)} \\ h_2^{(3)} \\ h_3^{(3)} \end{bmatrix} = \begin{bmatrix} g(z_1^{(3)}) \\ g(z_2^{(3)}) \\ g(z_3^{(3)}) \end{bmatrix} = g_{\otimes}(\boldsymbol{h^{(3)}}). \tag{80}$$

La matriz de pesos que conectan los valores de salida de la segunda capa con las neuronas de las tercera capa es

$$W^{(3)} = \begin{bmatrix} w_{10}^{(3)} & w_{11}^{(3)} & w_{12}^{(3)} & w_{13}^{(3)} \\ w_{20}^{(3)} & w_{21}^{(3)} & w_{22}^{(3)} & w_{23}^{(3)} \\ w_{30}^{(3)} & w_{31}^{(3)} & w_{32}^{(3)} & w_{33}^{(3)} \end{bmatrix}. \tag{81}$$

El vector que representa la entrada de la cuarta capa es $z^{(4)} = [z_1^{(4)}, z_2^{(4)}, z_3^{(4)}]^T$*, donde* $z_i^{(4)}$ *es la entrada de la neurona* i*,* $i \in \{1, 2, 3\}$*, por medio de la siguiente expresión*

$$z^{(4)} = W^{(3)}\boldsymbol{h^{(3)}} = \begin{bmatrix} w_{10}^{(3)} & w_{11}^{(3)} & w_{12}^{(3)} & w_{13}^{(3)} \\ w_{20}^{(3)} & w_{21}^{(3)} & w_{22}^{(3)} & w_{23}^{(3)} \\ w_{30}^{(3)} & w_{31}^{(3)} & w_{32}^{(3)} & w_{33}^{(3)} \end{bmatrix} \begin{bmatrix} 1 \\ h_1^{(3)} \\ h_2^{(3)} \\ h_3^{(3)} \end{bmatrix}. \tag{82}$$

El siguiente paso es aplicar la función de activación a cada neurona para obtener las salidas u output de las neuronas de la cuarta capa. Obtenemos el vector $\boldsymbol{h^{(4)}} = [h_1^{(4)}, h_2^{(4)}, h_3^{(4)}]^T$ *que representa la salida de la cuarta capa, donde* $h_i^{(4)}$ *representa la salida de la neurona* i*,* $i \in \{1, 2, 3\}$*, por medio de la siguiente expresión*

$$\boldsymbol{h^{(4)}} = \begin{bmatrix} h_1^{(4)} \\ h_2^{(4)} \\ h_3^{(4)} \end{bmatrix} = \begin{bmatrix} g(z_1^{(4)}) \\ g(z_2^{(4)}) \\ g(z_3^{(4)}) \end{bmatrix} = g_{\otimes}(\boldsymbol{h^{(4)}}). \tag{83}$$

La matriz de pesos que conectan los valores de salida de la cuarta capa con las neuronas de la quinta capa es

$$W^{(4)} = \begin{bmatrix} w_{10}^{(4)} & w_{11}^{(4)} & w_{12}^{(4)} & w_{13}^{(4)} \\ w_{20}^{(4)} & w_{21}^{(4)} & w_{22}^{(4)} & w_{23}^{(4)} \end{bmatrix}. \tag{84}$$

El vector que representa la entrada de la quinta capa es $z^{(5)} = [z_1^{(5)}, z_2^{(5)}]^T$*, donde* $z_i^{(5)}$ *es la entrada de la neurona* i*,* $i \in \{1, 2\}$*, por medio de la siguiente expresión*

$$z^{(5)} = W^{4)}\boldsymbol{h^{(4)}} = \begin{bmatrix} w_{10}^{(4)} & w_{11}^{(4)} & w_{12}^{(4)} & w_{13}^{(4)} \\ w_{20}^{(4)} & w_{21}^{(4)} & w_{22}^{(4)} & w_{23}^{(4)} \end{bmatrix} \begin{bmatrix} 1 \\ h_1^{(4)} \\ h_2^{(4)} \\ h_3^{(4)} \end{bmatrix}. \tag{85}$$

El siguiente paso es aplicar la función de activación a cada neurona para obtener las salidas u output de las neuronas de la quinta capa. Obtenemos el vector $\boldsymbol{h}^{(5)} = [h_1^{(5)}, h_2^{(5)}]^T$ *que representa la salida de la quinta capa, donde* $h_i^{(5)}$ *representa la salida de la neurona* i*,* $i \in \{1, 2\}$*, por medio de la siguiente expresión*

$$\boldsymbol{h}^{(5)} = \begin{bmatrix} h_1^{(5)} \\ h_2^{(5)} \end{bmatrix} = \begin{bmatrix} g(z_1^{(5)}) \\ g(z_2^{(5)}) \end{bmatrix} = g_{\otimes}(\boldsymbol{h}^{(5)}). \tag{86}$$

La matriz de pesos que conectan los valores de salida de la quinta capa con las neuronas de la sexta capa es

$$W^{(5)} = \begin{bmatrix} w_{10}^{(5)} & w_{11}^{(5)} & w_{12}^{(5)} & w_{13}^{(5)} \end{bmatrix}. \tag{87}$$

Finalmente, el escalar $z^{(6)}$ *representa la salida de la última capa por medio de la siguiente expresión*

$$z^{(6)} = W^{5)}\boldsymbol{h}^{(5)} = \begin{bmatrix} w_{10}^{(5)} & w_{11}^{(5)} & w_{12}^{(5)} & w_{13}^{(5)} \end{bmatrix} \begin{bmatrix} 1 \\ h_1^{(5)} \\ h_2^{(5)} \end{bmatrix}. \tag{88}$$

Observe que el escalar $z^{(6)}$ *es la predicción de la red profunda, la cual será cuantificada por medio de la función de pérdida* Φ *con respecto a los datos reales. La sucesión de pasos implementados por la red convencional se conoce como feed-forward o propagación hacia adelante. Ahora se implementará el algoritmo de retropropagación o backpropagation sobre la red convencional, usando las tres expresiones recursivas* (65)*,* (67) *y* (68) *previamente calculados, y suponiendo que las función costo o pérdida* Φ *y las dos funciones de activación son diferenciables, se tiene que Primero se calcula el paso base para desarrollar el proceso iterativo de retropropagación, el cual es representado por*

$$\frac{\partial \Phi}{\partial w_{1,k}^{(5)}} = \delta_1^{(6)} \boldsymbol{h^{(5)}}. \tag{89}$$

donde

$$\delta_1^{(6)} = \frac{\partial \Phi}{\partial h_k^{(5)}}, \text{ con } k \in \{0,1,2\} \tag{90}$$

Después se calculan los gradientes, los cuales están implícitos en la expresión $\delta^{(5)}$ *y que dependen del paso base, mediante la expresión*

$$\frac{\partial \Phi}{\partial w_{j,k}^{(4)}} = \delta_j^{(5)} h_k^{(4)}. \tag{91}$$

Donde

$$\delta_j^{(5)} = \left(\sum_{s=1}^{m_6} \delta_s^{(6)} w_{sj}^{(5)}) \right) \left(g'(z_j^{(5)}) \right), \text{ con } k \in \{0,1,2,3\} \text{ y } j \in \{1,2\}. \tag{92}$$

En el siguiente paso, se vuelve a calcular los gradientes los cuales están implícitos en la expresión $\delta^{(5)}$ *que depende del paso base* $\delta^{(6)}$*, mediante la expresión*

$$\frac{\partial \Phi}{\partial w_{j,k}^{(3)}} = \delta_j^{(4)} h_k^{(3)}. \tag{93}$$

Donde

$$\delta_j^{(4)} = \left(\sum_{s=1}^{m_5} \delta_s^{(5)} w_{sj}^{(4)}) \right) \left(g'(z_j^{(4)}) \right), \text{ con } k \in \{0,1,2,3\} \text{ y } j \in \{1,2,3\}. \tag{94}$$

Seguidamente, se vuelve a calcular los gradientes los cuales están implícitos en la expresión $\delta^{(4)}$ *que depende de* $\delta^{(5)}$ *y a su vez depende del paso base, mediante la expresión*

$$\frac{\partial \Phi}{\partial w_{j,k}^{(2)}} = \delta_j^{(3)} h_k^{(2)}. \tag{95}$$

Donde

$$\delta_j^{(3)} = \left(\sum_{s=1}^{m_4} \delta_s^{(4)} w_{sj}^{(3)}) \right) \left(g'(z_j^{(3)}) \right), \text{ con } k \in \{0,1,2\} \text{ y } j \in \{1,2,3\}. \tag{96}$$

Finalmente se calculan los gradientes correspondientes a todas las respectivas capas que conforman la red hasta llegar al inicio, donde se evaluaron los datos de entrada para entrenar la red, así la última expresión para calcular los gradientes de la red profunda es

$$\frac{\partial \Phi}{\partial w_{j,k}^{(1)}} = \delta_j^{(2)} h_k^{(1)}. \tag{97}$$

donde

$$\delta_j^{(2)} = \left(\sum_{s=1}^{m_3} \delta_s^{(3)} w_{sj}^{(2)}) \right) \left(g'(z_j^{(2)}) \right), \text{ con } k \in \{0,1,2,3\} \text{ y } j \in \{1,2\}. \tag{98}$$

Después de haber implementado el algoritmo del backpropagation, se ejecutará la actualización del gradiente, implementando el algoritmo del máximo descenso para ejecutar la actualización de los parámetros pre-entrenados, mediante la siguiente expresión

$$\widehat{w}_{jk}^{(l)} = w_{jk}^{(l)} - \alpha \frac{\partial \Phi}{\partial w_{jk}^{(l)}}, \text{ con } j \in \{1, \ldots, m_l\} \text{ y } k \in \{0, 1, \ldots m_{l-1}\}. \tag{99}$$

Donde $\widehat{w}_{jk}^{(l)}$ *son los parámetros actualizados,* $w_{jk}^{(l)}$ *son los parámetros pre-entrenados y* α *es el ratio de aprendizaje.*

Otra forma de ver el algoritmo del máximo descenso, es utilizando la expresión recursiva (10) *y* (11) *de forma vectorial y definiendo la operación producto exterior o tensorial, de la siguiente forma*

$$\delta^{(l+1)} \otimes \boldsymbol{h^{(l)}} := \delta^{(l+1)} (\boldsymbol{h^{(l)}})^T, \text{ para todo } l \in \{1, \ldots, L\}. \tag{100}$$

Donde $\otimes$ denota el producto exterior o tensorial, y además el algoritmo del máximo descenso queda expresado mediante la operación tensorial, expresándose de la siguiente forma

$$\widehat{W}^{(1)} = W^{(1)} - \alpha \begin{bmatrix} \delta_1^{(2)} h_1^{(1)} & \delta_2^{(2)} h_1^{(1)} \\ \delta_1^{(2)} h_2^{(1)} & \delta_2^{(2)} h_2^{(1)} \\ \delta_1^{(2)} h_3^{(1)} & \delta_2^{(2)} h_3^{(1)} \end{bmatrix} \tag{101}$$

y

$$\widehat{w}^{(l+1)} = w^{(l+1)} - \alpha(\delta^{(l+1)} \otimes \boldsymbol{h^{(l)}}), \text{ para todo } l \in \{0, 1, \ldots, L-1\}. \tag{102}$$

Para más detalle de la expresión (99), (100) y (101), véase en [27].

Nota 5.1. *Definir la arquitectura idónea para una red neuronal implica definir la cantidad de neuronas y capas; respectivamente, debido a la dependencia de los datos y su proceso de validación. la implementación se hace en un lenguaje de programación (Python, Matlab, R-studio) usando un algoritmo de minimización que nos permite minimizar el error de la función costo y además, se usa una métrica con respecto al tipo de problema que nos generan los datos; es decir, si es un problema de regresión lineal (simple o múltiple), la métrica (Acurracy) depende de la función costo (Mean square error, Mean absolute error) con sus respectivas validaciones, con el objetivo de minimizar su error. Si el problema es de clasificación (regresión logística binaria), la métrica depende de la función costo (Binary Cross Entropy) y la matriz de confusión o matriz de error que es una herramienta que permite visualizar el desempeño del algoritmo, con respecto al tipo de aciertos y errores que está teniendo el modelo en el proceso de aprendizaje con el objetivo de minimizar su error en las predicciones.*

	Positivos	**Negativos**
Positivos	Verdaderos Positivos	Falsos Negativos
Negativos	Falsos Positivos	Verdaderos Negativos

Tabla 6. Matriz de confusión. Hecho por el autor.

Observación 5.2. *Las métricas en la matriz de confusión* [38] *son:*

- *Exactitud (Acurracy): muestra la distancia entre los datos arrojados por la matriz con respecto a los datos reales.*

$$Acurracy = \frac{TP+TN}{TP+TN+FP+FN}.$$

- *Precisión (Precision): toma la exactitud de los datos correctos y los compara con el total de datos arrojados*

$$Precision = \frac{TP}{TP+FP}.$$

- *Sensibilidad (Recall): determina los aciertos con respecto a los casos que presentan ciertas características que son determinados por el programador como positivos.*

$$Recall = \frac{TP}{TP+FN}.$$

- *Especificidad (Specificity): expresa la cantidad de aciertos que el algoritmo*

[38] Cynthia Lorena Corso. «Aplicación de algoritmos de clasificación supervisada usando Weka». En: *Córdoba: Universidad Tecnológica Nacional, Facultad Regional Córdoba* (2009).

tiene con respecto a los casos negativos

$$\textit{Specificity} = \frac{\textit{TN}}{\textit{TN+FP}}.$$

Observación 5.3. *Notaciones*

- *TP: Verdaderos positivos.*
- *TN : Verdaderos negativos.*
- *FP: Falsos positivos.*
- *FN: Falsos negativos.*

Nota 5.4. *Determinar la arquitectura de una red neuronal depende de la cantidad de neuronas y capas con respeto al modelo, además depende de la métrica y, para esto se necesita realizar varios experimentos para encontrar la mejor distribución de las capas y neuronas con la finalidad de obtener la mejor predicción.*

5.5. Mínimos Cuadrados No Lineales y Redes neuronales profundas

Con el objetivo de implementar el algoritmo (4.2.2) en el proceso de aprendizaje de las redes neuronales, en donde esta inmersa la función error f (104) la cual se encarga de proporcionar el error que comete la red, con respecto a sus predicciones en cada iteración; es decir los valores obtenidos por los datos con respecto a los valores predichos por la red en el proceso de aprendizaje, se tiene el siguiente problema

$$\begin{aligned} &\text{minimizar} \quad f(\boldsymbol{w}), \\ &\boldsymbol{w} \in \mathbb{R}^n \end{aligned} \tag{103}$$

en donde

$$\begin{aligned} f: \ \mathbb{R}^n &\to \mathbb{R}, \\ \boldsymbol{w} &\to f(\boldsymbol{w}) \end{aligned} \tag{104}$$

representa la función error que es una combinación lineal de los pesos $\boldsymbol{w}$ y sus respectivos *bias*, definida en (74) conocida como el error cuadrático medio

$$f(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \hat{y}_i(\boldsymbol{w}))^2,$$

donde y_i, $\hat{y}_i$ respectivamente son los valores obtenidos por los datos y los valores predichos por la red. Así, el proceso de aprendizaje de una red neuronal consiste en encontrar un vector $\boldsymbol{w} \in \mathbb{R}^q$ que minimice la función (104). El valor de q depende de la cantidad de neuronas en cada capas de la red. Note que (103) y f definida en (104), es un problema de **Mínimos Cuadrado No Lineales** (4).

En el caso donde los vectores de entrada de la red o de características corresponden a $\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^p \in \mathbb{R}^n$ con sus respectivos valores obtenidos por los datos o etiquetas $\boldsymbol{y}^1, \boldsymbol{y}^2, \ldots, \boldsymbol{y}^p$ y además, los vectores de los valores predichos por la red, se denotan por $\hat{\boldsymbol{y}}^\mu \in \mathbb{R}^m$ con $\mu = 1, 2, \ldots, p$ así, se tiene que

$$f(\boldsymbol{w}) = \frac{1}{2}\sum_{\mu=1}^{p}\sum_{i=1}^{m}(\boldsymbol{y_i}^\mu - \hat{\boldsymbol{y}}_{\boldsymbol{i}}^{\,\mu})^2 = \frac{1}{2}\sum_{\mu=1}^{p}||\boldsymbol{y_i}^\mu - \hat{\boldsymbol{y}}_{\boldsymbol{i}}^{\,\mu}(\boldsymbol{w})||^2,$$

donde $\boldsymbol{y_i}^\mu(\boldsymbol{w}) \in \mathbb{R}^m$, con $\mu = 1, 2, \ldots, p$ y $\boldsymbol{y_i}^\mu \in \mathbb{R}^n$ representan las etiquetas que son conocidas por los valores de los datos (Aprendizaje supervisado).

Note que

$$\hat{\boldsymbol{y}}_{\boldsymbol{k}}^{\,\mu} = g\left(\sum_{j=0}^{q} w'_{k,j}\boldsymbol{y_j}^\mu\right) = g\left(w'_{j,k}\,\sigma\left(\sum_{i=0}^{n} w_{j,i}\boldsymbol{x_i}^\mu\right)\right),$$

donde g es la función de activación de las neuronas de salida en su respectiva capa,

dependiendo del problema. σ es la función de activación de las capas ocultas. Así, se tiene que

$$f(w_{j,i}, w_{k,j}) = \frac{1}{2}\sum_{\mu=1}^{p}\sum_{k=1}^{m}\left(\boldsymbol{y_i}^{\mu} - g\left(\sum_{j=0}^{q} w'_{k,j}\boldsymbol{y_j}^{\mu}\right)\right)^2,$$

el *Feed forward* o propagacion hacia adelante y el *Backpropagation* o retropropagación de la red, se explicó en los capítulos (5.2) y (5.3); respectivamente, con la condición de que las funciones de activación en las capas ocultas (Sigmoide,Tangente hiperbólica, Gaussiana, etc)[39] como la de salida (Identidad, Escalón, Relu, Softmax, Elu, etc)[39] sean diferenciables, para realizar el proceso de entrenamiento de la red neuronal. Para inicializar los pesos $\boldsymbol{w}$, se recomienda utilizar pesos aleatorios, sin embargo, si los pesos no son buenos pesos iniciales, los algoritmos están implementados, para usar estrategias de globalización vistos en el Capítulo 2 permitiendo iniciar el proceso iterativo de búsqueda lineal en cualquier punto del dominio de la función a optimizar.

Para actualizar los pesos en cada iteración, se determina una dirección de descenso $\boldsymbol{d_k}$ y luego se usa búsqueda lineal (2.2), para encontrar un tamaño de paso $\alpha > 0$; es decir

$$\boldsymbol{w_{k+1}} = \boldsymbol{w_k} + \alpha \boldsymbol{d_k}.$$

Para el criterio de parada, se usa el tamaño del gradiente de la función objetivo $||\nabla f(\boldsymbol{w})||$ y la cantidad de iteraciones ($Iter$).

Si $||\nabla f(\boldsymbol{w})|| < Tol$ se dice que el algoritmo converge, si $Iter > N$ se dice que el algoritmo diverge, donde $Iter$ es el número de iteraciones, Tol es la tolerancia

---

[39] Luis Llano et al. «Comparación del Desempeño de Funciones de Activación en Redes Feedforward para aproximar Funciones de Datos con y sin Ruido». En: *Avances en Sistemas e Informática* 4.2 (2007).

usada y N es el máximo número de iteraciones.
Con respecto a lo anterior, se presenta la estructura general del algoritmo y el proceso de entrenamiento para una red neuronal profunda.

Algoritmo 5.5.1. *(Red neuronal profunda y el problema de Mínimos Cuadrados No Lineales)*

Inicialización. *Tomemos* $\alpha_0 = 1$, $\lambda = 0.0001$ *y* $H_0 = I_n$

- *Inicializar los* q *pesos aleatorios* $\boldsymbol{w_0}$.
- *Calcular la salida de la red que involucra las* p *características o etiquetas en el entrenamiento.*
- *Calcular el error de la salida entre el valor de los datos con respecto a los valores predichos por la red.*

Mientras $||\nabla f(\boldsymbol{w_k})|| > Tol$ *y* $k < N$ *hacer*

- ***Dirección de búsqueda.*** *Calcular* H_k *y* $\boldsymbol{s_k}$ *tal que*

$$H_k \boldsymbol{s_k} = -\nabla f(\boldsymbol{w_k}).$$

- ***Tamaño del paso.***

$$f(\boldsymbol{w_k} + \alpha_k \boldsymbol{s_k}) \leq f(\boldsymbol{w_k}) + \alpha_k \lambda \nabla f(\boldsymbol{w_k})^T \boldsymbol{s_k}.$$

- ***Actualización de los pesos***

$$\boldsymbol{w_{k+1}} = \boldsymbol{w_k} + \alpha_k \boldsymbol{s_k}.$$

- *Calcular la salida de la red que involucra las* p *características o etiquetas en el entrenamiento.*

- *Calcular el error de la salida entre el valor de los datos con respecto a los valores predichos por la red.*
- *Calcular H_{k+1} utilizando el algoritmo (4.2.2).*

Actualizar la información.

Aunque el propósito principal de este trabajo es teórico, se incluyen algunas pruebas numéricas con el fin de verificar la efectividad del algoritmo que, en teoría, funciona bien. Sin embargo, se busca identificar posibles dificultades que puedan surgir durante su ejecución.

5.5.1. Regresión lineal usando una red neuronal

El **primer problema** aborda un problema de interpolación lineal, que utiliza 10 datos de entrenamiento obtenidos de un archivo (**.xlsx**) de Excel. Estos datos consisten en el número de acciones vendidas (en millones) de varias empresas de productos tecnológicos y el precio esperado (el promedio del precio mínimo y el precio máximo) en euros de 10 acciones.

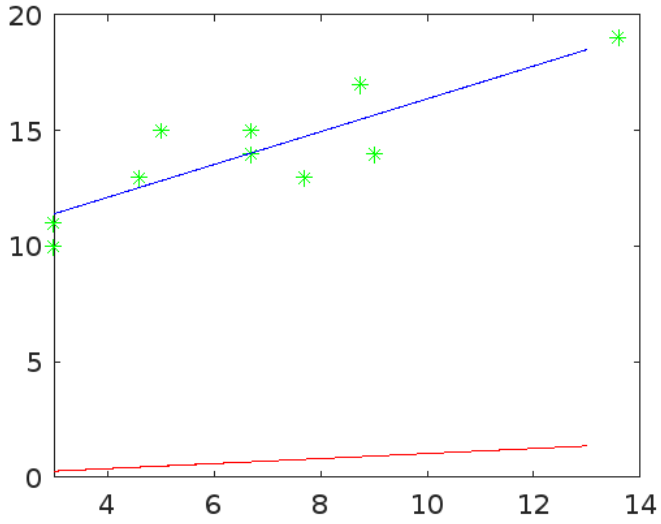

Figura 11. Gráfica de la interpolación lineal basada en el primer problema. Hecho por el autor en MATLAB.

En la figura 11, los puntos de color verde representan los datos de entrenamiento. La función en color rojo modela la interpolación lineal sin entrenar los pesos y sesgos. La función en color azul modela la interpolación lineal después de entrenar los pesos y sesgos. En la iteración (Iter=289) de un máximo de (Max=500) iteraciones, el algoritmo secante estructurado BFGS converge superlinealmente, donde la norma del gradiente es 0.0009834 y la norma del residuo es 4.0144933. En este proceso, Se utilizaron 3 capas profundas, cada una con dos neuronas y una neurona en la capa de salida, y se empleo la función de activación σ identidad tanto en las capas profundas como en la capa de salida.

El **segundo problema**, aborda una cadena de restaurantes que se especializa en pizzas y otros platos italianos. La gerencia está interesada en saber la cantidad de ventas diarias de pizzas en su restaurante y para ello, han recopilado datos acerca de una población de estudiantes con respecto a la ventas realizadas. La gerencia desea construir un modelo de regresión lineal que le permita predecir las ventas diarias de pizza en función de los datos mencionados anteriormente.

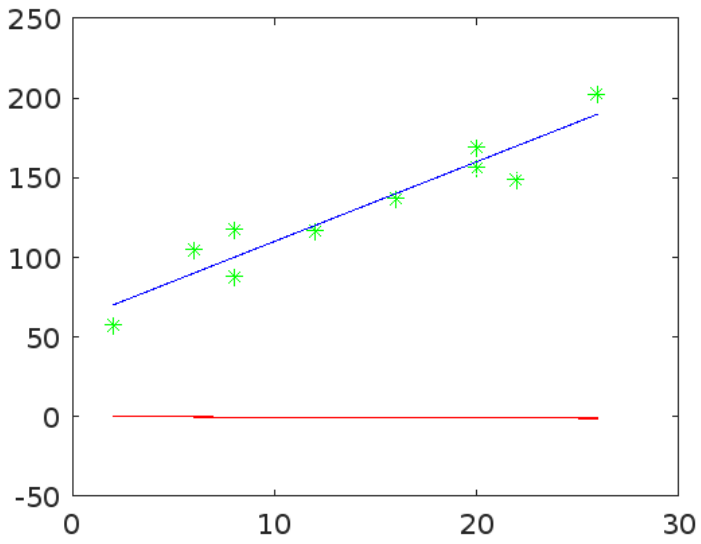

Figura 12. Gráfica de la interpolación lineal basada en los datos del segundo problema. Hecho por el autor en MATLAB.

Se han recopilado 10 datos para el entrenamiento mediante un archivo (**.xlsx**) de Excel. Estos datos consisten en la cantidad de población de estudiantes y el precio en ventas de la pizza por día, en la gráfica 12 se representan los datos mediante los puntos verdes. La función de color rojo representa la interpolación lineal sin entrenar los pesos y sesgos. La función en color azul modela la interpolación lineal después de entrenar los pesos y sesgos. En la iteración (Iter=727) de un máximo de (Max=1000) iteraciones, el algoritmo secante estructurado BFGS converge, donde la norma del gradiente es 0.0009531 y la norma del residuo es 39.1152144. En este proceso, se utilizaron 4 capas profundas, cada una con dos neuronas y una neurona en la capa de salida, y se empleo la función de activación σ identidad tanto en las capas profundas como en la capa de salida.

A. Apéndice

El siguiente código, implementado en el lenguaje de programación MATLAB, corresponde a la implementación del algoritmo 4.2.2 en una red neuronal totalmente conectada.

Función main

```
%+===================================================================+
%| neuronal_net                                                      |
%| Red Neuronal multicapa                                            |
%| xred es el vector de entrenamiento                                |
%| tred es el vector de de salida                                    |
%| el entrenamiento se efectuará con BFGS ESTRUCTURADO               |
%+===================================================================+
clear all;
%clc
% cargamos ---> llamar datos
datos=xlsread('celsius.xlsx');
[datFilas,datCol]=size(datos);
datMax=15; %indica el numero de registros que se usaran
%en el entrenamiento
xred=datos(1:datMax,1:datCol-1);
l_min=min(xred(:,1));
l_max=max(xred(:,1));
xred=xred';
```

```
% la útima columna son los valores esperados

tred=datos(1:datMax,datCol);
tred=tred';
% plot(xred,tred)
% datos para la verificacion
%%%%%%%%%% inicia experimento cambiando la capa oculta %%%%%%%%%
hiddenLayer=[2 3]; % ejemplo [3 2 1]

layerNumb=length(hiddenLayer); % numero de capas ocultas
global nvar;
nvar=(datCol)*hiddenLayer(1);% esta variable determina
%la longitud de w0

% se calcula el numero total de pesos y bias
for k=2:layerNumb
        nvar=nvar+(hiddenLayer(k-1)+1)*hiddenLayer(k);
end
        nvar=nvar+ hiddenLayer(layerNumb)+1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Tol=10^(-3); %Tolerancia
% se genera vector inicial
w0=0.5*randn(nvar,1);
%w0=iniciar();

% se procede con el entrenamiento
[w,n,tiempo,exito]=structural_BFGS(w0,xred,tred,hiddenLayer,Tol);
```

```
% retorna los pesos y bias despues del entrenamiento

if exito==1
% realizaremos una prueba grafica del resultado del entrenamiento
% Graficaremos en rojo la interpolación antes y luego
% Graficaremos en azul la interpolación despues del entrenamiento

    dom = l_min:1:l_max;
    ndom=length(dom);
    % antes
    for m=1:ndom
        ran(m)=exit_net(dom(m),w0,hiddenLayer);
    end
    % despues
    for m=1:ndom
        ran2(m)=exit_net(dom(m),w,hiddenLayer);
    end
    % grafica
    figure
    plot(xred,tred,'g*')
    hold on
    plot(dom,ran,'r')
    hold on
    plot(dom,ran2,'b')

else
    run main.m
```

```
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Función structural_BFGS

```
function[w,n,tiempo,exito]=structural_BFGS(w0,xred,tred
,hiddenLayer,Tol);
des=length(w0);
A=eye(des);
%+=================================================================+
%| structural_BFGS                                                 |
%| metodo BFGS estructurado                                        |
%| w0 es punto inicial                                             |
%| A es la matriz inicial                                          |
%+=================================================================+
jac=JN(w0,xred,tred,hiddenLayer); %aproximacion numerica del
%jacobiano
r=residue_net(w0,xred,tred,hiddenLayer); % vector residuo
g=jac'*r; % gradiente
n=0;
exito=1;
niter=1000;
j0=jac;
tic;
while (norm(g)>Tol) && (n<niter)
 B=jac'*jac+A;
 B=correction_net(B);
```

```
 if rcond(B)>0.01 % si B está bien condicionada use
 %la dirección BFGS
    d=-B\g;
    d=-jac'*r;
 else
    d=-jac'*r; % si no use la dirección de menos el gradiente
 end
 % Tamano del paso w esta actualizado.
 [l,cont]=linearsearch_net(d,w0,xred,tred,hiddenLayer,jac,r);
 fprintf('l= %2.7f  k=%2.0f des? %2.0f g= %2.7f

 \n',l,n,sign(g'*d),norm(g));
 % actualizar.
 n=n+1;
 w1=w0+l*d;
 s=w1-w0;
 jac=JN(w1,xred,tred,hiddenLayer);%aproximation
 jacobian
 r=residue_net(w1,xred,tred,hiddenLayer);
 g=jac'*r;% gradiente
 A=update_net(A,B,w1,w0,s,jac,j0,r); %Necesita A y B
 j0=jac; % esta matriz se almacena para usar de nuevo en update_net
 w0=w1;  % este vector se almacena para usar de nuevo en update_net
end
  tiempo=toc;
  w=w0;
```

```
% prueba para definir si hubo exito
if n>=niter || norm(w1)>10^10
    exito=0;
    fprintf('Entrenamiento incompleto, ejecute de nuevo \n')
    fprintf('Norma del residuo %2.7f \n',norm(r));
    fprintf('Norma del gradiente %2.7f \n',norm(g));
 else
    fprintf('Norma del residuo %2.7f \n',norm(r));
    fprintf('Norma del gradiente %2.7f \n',norm(g));
  end
end


%% funcion jacobiano de r aproximado
function Jac=JN(w0,x,t,hiddenLayer)
% esta funcion calcula el jacobiano aproximado
% de la funcion residuo, residue_net, en la variable W0
 h=0.000001;
 n=length(w0);
 e=eye(n);
 Y0=residue_net(w0,x,t,hiddenLayer);
    for j=1:1:n
            stepFoward=w0+h*e(:,j);
            Y1=residue_net(stepFoward,x,t,hiddenLayer);
            Jac(:,j)=(Y1-Y0)/(h);
    end
```

```
 end

 %% funcion Residuo
function r=residue_net(w,x,t,hiddenLayer)
 % x,t son los p valores de entrenamiento de entrada y salida resp.
  p=length(t);
  r=zeros(1,p);
  for mu=1:p
     r(mu)=t(mu)-exit_net(x(:,mu),w,hiddenLayer);
  end
  r=r';
end



% esta funcion corrige  a la matriz a para que sea definida positiva
function B=correction_net(B)
  n=length(B);
  I=eye(n);
  [L,p]=chol(B,'lower');
  % si p=0 , B>0
  if(p~=0)
       i=0;
       while(p~=0)&&(i<100)
         B=B+0.1*I;
         [L,p]=chol(p,'lower');
         i=i+1;
```

```
        end
    end
 end


% esta funcion calcula el tamaño de paso adecuado
% para que se genere un decenso suficiente
function [l,k]=linearsearch_net(d,w,xred,tred,hiddenLayer,J,R)%Tamaño
%del paso.
% esta funcion aplica la condición de Armijo
% J es el jacobiano inicial en w
% R es el residuo anterior
  alfa=0.00001;
  niter=25;
  l=1;
  j=J;
  r=R;
  g=j'*r;
  fes=0.5*norm(r)^2;
  wnew=w+l*d;
  r1=residue_net(wnew,xred,tred,hiddenLayer); %Residuo nuevo
  f1=0.5*norm(r1)^2;
  k=0;
  while (f1 > fes+real(alfa*l*g'*d))&&(k<niter)
    %fprintf('%2.0f \t %5.5f \n ',k,l);
    l=0.5*l;
    wnew=w+l*d;
```

```
    r1=residue_net(wnew,xred,tred,hiddenLayer); %Residuo actualizado
    f1=0.5*norm(r1)^2;
    k=k+1;
  end

 end
```

```
function A1=update_net(A,B,w,w0,s,j,j0,r)
  des=length(w);
  ynum=(j-j0)'*r;
  ye=ynum+j'*j*s;
  v=ye+sqrt(((ye'*s)/(s'*B*s)))*(B*s);
  a=ynum-A*s;
  b=v'*s;
  delta=((a*v'+v*a')/b)-(((a'*s)/b^2))*(v*v');
  A1=A+delta;

end
```

Función exit_net

```
%% funcion redneuronal, evalua la red en un punto x
function t=exit_net(z,W,hiddenLayer)
  xNumb=length(z);
```

```
  H=[xNumb hiddenLayer 1];% indicador de todas las capas, ocultas y visibles
  HNumb=length(H);% total de capas
  lim2=0;
  %primero los pesos en vectores mas pequeños
  wt=cell(HNumb-1,1);
  for p=2:1:HNumb
     lim1=lim2+1;
     lim2=lim1+H(p)*H(p-1)-1;
     wt{p-1}=W(lim1:lim2);
  end
  % ahora los bias
  wbias=cell(HNumb-1,1);
    for q=2:1:HNumb
      lim1=lim2+1;
      lim2=lim1+H(q)-1;
      wbias{q-1}=W(lim1:lim2);
    end
% ahora llevamos los pesos a matrices tri-indizadas ws(i,j,k)
% k indica la capa
% i indica la neurona de la capa
% j indica la neurona de la capa anterior
 for p=1:1:HNumb-1
    ws{p}=reshape(wt{p},H(p+1),H(p));% creamos un vector de matrices
 end
% FIN DE LA SEPARACION
% aqui va la función de activación
 for capa=1:1:HNumb-1
```

```
      y=ws{capa}*z+wbias{capa};
      if capa~=HNumb-1
          %z=tansig(y);
          %z=atan(y);
          %z=sigmoide(y);
          %z=max(0,y);
          %z=sign(y);
          z=gauss(y);
          %z=y;
      else
% aqui va la función de activación para la última neurona
          %z=tansig(y);
          %z=atan(y);
          %z=sigmoide(y);
          %z=max(0,y);
          %z=sign(y);
          %z=gauss(y);
          z=y;
      end
 end

 t=z;
 end

 function z=sigmoide(x)
  z=1./(1+exp(-1.*x));
end
```

```
function z=tansig(x)
z=2./(1+exp(-2.*x))-1;
end

function z=gauss(x)
z=exp((-1/sqrt(2*pi)).*x.^2);
end
```

BIBLIOGRAFÍA

Aguilar, Luis Joyanes. *Big Data, Análisis de grandes volúmenes de datos en organizaciones*. Alfaomega Grupo Editor, 2016 (vid. pág. 10).

Ahn, Kwangjun, Jingzhao Zhang y Suvrit Sra. «Understanding the unstable convergence of gradient descent». En: *International Conference on Machine Learning*. PMLR. 2022, págs. 247-257 (vid. pág. 12).

Andrés, Acevedo Vázquez Julio. «Implementación De Los Métodos Cuasi-Newton». Tesis doct. BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA, 2019 (vid. pág. 65).

Andrychowicz, Marcin et al. «Learning to learn by gradient descent by gradient descent». En: *Advances in neural information processing systems* 29 (2016) (vid. págs. 97, 107, 108).

Arenas, Favián y Hevert Vivas. «Un software interactivo para el entrenamiento de redes neuronales multicapa usando el método secante estructurado». En: *Revista colombiana de tecnologías de avanzada (RCTA)* 2.34 (2019), págs. 85-90 (vid. pág. 13).

Boggs, Paul T, Jon W Tolle y Pyng Wang. «On the local convergence of quasi-Newton methods for constrained optimization». En: *SIAM journal on control and optimization* 20.2 (1982), págs. 161-171 (vid. pág. 12).

Burden, Richard L et al. «Análisis numérico». En: (2017) (vid. pág. 19).

Buscema, Massimo. «Back propagation neural networks». En: *Substance use & misuse* 33.2 (1998), págs. 233-270 (vid. págs. 98, 106, 107).

Castañeda, Pablo. «Matemática Computacional». En: () (vid. pág. 87).

Corso, Cynthia Lorena. «Aplicación de algoritmos de clasificación supervisada usando Weka». En: *Córdoba: Universidad Tecnológica Nacional, Facultad Regional Córdoba* (2009) (vid. pág. 117).

De Villiers, Jacques y Etienne Barnard. «Backpropagation neural nets with one and two hidden layers». En: *IEEE transactions on neural networks* 4.1 (1993), págs. 136-141 (vid. págs. 98, 106).

Dennis, JE, Héctor J Martinez y Richard A Tapia. «Convergence theory for the structured BFGS secant method with an application to nonlinear least squares». En: *Journal of Optimization Theory and Applications* 61 (1989), págs. 161-178 (vid. pág. 92).

Dennis Jr, John E y Jorge J Moré. «Quasi-Newton methods, motivation and theory». En: *SIAM review* 19.1 (1977), págs. 46-89 (vid. pág. 77).

Dennis Jr, John E y Robert B Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations*. SIAM, 1996 (vid. págs. 14, 27, 81, 82, 90, 92, 96).

Fernando, Chrisantha et al. «Pathnet: Evolution channels gradient descent in super neural networks». En: *arXiv preprint arXiv:1701.08734* (2017) (vid. pág. 103).

García, Roberto. «El perceptrón: una red neuronal artificial para clasificar datos». En: *Revista de investigación en modelos matematicos aplicados a la gestion la economia* (2021) (vid. pág. 11).

Hecht-Nielsen, Robert. «Theory of the backpropagation neural network». En: *Neural networks for perception*. Elsevier, 1992, págs. 65-93 (vid. págs. 97, 98, 116).

Hernández, Luis Enrique Ruíz. «Teorema del Valor Medio para Derivadas en RN en Puntos Condicionados». En: (1990) (vid. pág. 67).

Hochreiter, Sepp, A Steven Younger y Peter R Conwell. «Learning to learn using gradient descent». En: *Artificial Neural Networks—ICANN 2001: International Conference Vienna, Austria, August 21–25, 2001 Proceedings 11*. Springer. 2001, págs. 87-94 (vid. pág. 97).

LeCun, Yann, Yoshua Bengio y Geoffrey Hinton. «Deep learning». En: *nature* 521.7553 (2015), págs. 436-444 (vid. pág. 99).

Llano, Luis et al. «Comparación del Desempeño de Funciones de Activación en Redes Feedforward para aproximar Funciones de Datos con y sin Ruido». En: *Avances en Sistemas e Informática* 4.2 (2007) (vid. pág. 120).

López, Javier. «Optimización multi-objetivo». Tesis doct. Universidad Nacional de La Plata, 2013 (vid. pág. 43).

Martín, B et al. «Redes neuronales y sistemas borrosos: un libro de texto en español». En: (1970) (vid. pág. 10).

Matlab, Starting. «Matlab». En: *The MathWorks, Natick, MA* (2012) (vid. pág. 15).

Moré, Jorge J. «The Levenberg-Marquardt algorithm: implementation and theory». En: *Numerical Analysis: Proceedings of the Biennial Conference Held at Dundee, June 28–July 1, 1977*. Springer. 2006, págs. 105-116 (vid. pág. 13).

Moreno, Antonio et al. *Aprendizaje automático*. 1994 (vid. pág. 10).

Poveda, Luz Marina Rondón. «Comparación de la eficiencia del método de optimización BFGS en C, OX y R para un modelo de regresión no lineal.» En: *Comunicaciones en Estadística* 2.2 (2009), págs. 175-188 (vid. pág. 74).

Ros Gómez, Ignacio. «Introducción al aprendizaje supervisado e implementación de una red neuronal en Python». En: (2018) (vid. pág. 10).

Sharma, Sagar, Simone Sharma y Anidhya Athaiya. «Activation functions in neural networks». En: *Towards Data Sci* 6.12 (2017), págs. 310-316 (vid. págs. 99, 100).

Solorio, Félix Calderón. «OPTIMIZACION NO–LINEAL (VER 1.10)». En: (2005) (vid. pág. 38).

Specht, Donald F. «Probabilistic neural networks and the polynomial adaline as complementary techniques for classification». En: *IEEE Transactions on Neural Networks* 1.1 (1990), págs. 111-121 (vid. pág. 11).

Tang, Jiexiong, Chenwei Deng y Guang-Bin Huang. «Extreme learning machine for multilayer perceptron». En: *IEEE transactions on neural networks and learning systems* 27.4 (2015), págs. 809-821 (vid. pág. 11).

Tseng, Paul y Sangwoon Yun. «A coordinate gradient descent method for nonsmooth separable minimization». En: *Mathematical Programming* 117 (2009), págs. 387-423 (vid. pág. 12).

Vivas, Hevert, Héctor Jairo Martínez y Rosana Pérez. «Método secante estructurado para el entrenamiento del perceptrón multicapa». En: (2018) (vid. pág. 13).

Voigtlaender, Felix. «The universal approximation theorem for complex-valued neural networks». En: *Applied and Computational Harmonic Analysis* 64 (2023), págs. 33-61 (vid. pág. 100).

Wang, Yong. «Gauss–newton method». En: *Wiley Interdisciplinary Reviews: Computational Statistics* 4.4 (2012), págs. 415-420 (vid. pág. 12).

Zaia, Alejandra C. «Algebra: Trabajo Práctico. Unidad Temática Nº 4: Diagonalización de matrices». En: (2018) (vid. pág. 68).

Zeiler, Matthew D. «Adadelta: an adaptive learning rate method». En: *arXiv preprint arXiv:1212.5701* (2012) (vid. pág. 108).