USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA
FACULTADE DE MATEMÁTICAS

Traballo Fin de Grao

Resolución numérica del problema no lineal de mínimos cuadrados. Aplicaciones a la estimación de parámetros de modelos matemáticos.

Dídac Blanco Morros

Curso Académico

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

GRAO DE MATEMÁTICAS

Traballo Fin de Grao

Resolución numérica del problema no lineal de mínimos cuadrados. Aplicaciones a la estimación de parámetros de modelos matemáticos.

Dídac Blanco Morros

Febrero, 2022

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Trabajo propuesto

Área de Coñecemento: Matemática Aplicada
Título: Resolución numérica do problema non linear de mínimos cadrados. Aplicacións á estimación de parámetros de modelos matemáticos.
Breve descrición do contido
O problema non linear de mínimos cadrados xurde en moitas aplicacións da ciencia e da enxeñería: no axuste dun conxunto de datos a un modelo matemático, na estimación de parámetros, na aproximación de funcións, etc. O obxectivo do traballo fin de grao é o estudo de métodos numéricos para abordar o problema de minimización resultante. centrándose especialmente no algoritmo de Levenberg-Marquardt. O estudante estudará o método, no marco dos métodos de optimización con rexión de confianza e familiarizarase co seu uso mediante o comando Isqnonlin de Matlab. As metodoloxías estudadas aplicaranse a exemplos académicos e á estimación de parámetros de distintos modelos matemáticos a partir de datos experimentais.

Índice

en record de la meva mare,
Mercè

Resumen

En este trabajo se estudia la resolución numérica del problema no lineal de los mínimos cuadrados, el cual se utiliza principalmente para la estimación de parámetros en distintos campos científicos. Para ello se comienza con una introducción teórica de la optimización sin restricciones y del problema de mínimos cuadrados. Posteriormente el estudio se enfoca en el método de Levenberg-Marquardt, considerado precursor de los métodos de tipo región de confianza. Este método afronta el problema tomando una aproximación lineal para resolverla de forma más sencilla, y para que la aproximación se mantenga acorde, se limita el radio de acción en el que se busca un mínimo. Finalmente se implementa el algoritmo en Matlab® para comprobar su funcionamiento y compararlo con la alternativa propia del programa.

Abstract

This work studies the numerical resolution of the nonlinear least squares problem, which is mainly used for parameter estimation in various scientific fields. To do this, it begins with a theoretical introduction to unconstrained optimization and the least squares problem. Subsequently, the study focuses on the Levenberg-Marquardt method, considered a precursor of trust region type methods. This method faces the problem by taking a linear approximation to solve it more simply, and to maintain the approximation, the action radius in which a minimum is sought is limited. Finally, the algorithm is implemented in Matlab® to verify its operation and compare it with the program's own alternative.

Introducción

El problema de los mínimos cuadrados es un problema de optimización utilizado para la estimación de parámetros en una amplia variedad de campos científicos. En este trabajo enfocamos la resolución del problema desde un punto de vista numérico después de una revisión previa a los conceptos necesarios para llegar al método de Levenberg-Marquardt, el cual se estudiará en profundidad.

La computación numérica ha ido evolucionando muy rápido desde sus inicios. Con la llegada de los primeros ordenadores, era extremadamente importante que los algoritmos fueran los más eficientes posibles, pues los recursos eran muy limitados y costosos. A lo largo de los años, las nuevas tecnologías han permitido que los ordenadores sean cada vez más potentes y se podría argumentar que la eficiencia de los algoritmos de resolución numérica ha dejado de ser un problema y no siempre es necesaria. Hoy en día la estimación de parámetros forma parte de los nuevos paradigmas tecnológicos. En concreto, es necesaria para lo que hoy en día se conoce como *Inteligencia Artificial*, basada en redes neuronales que cuentan con una enorme cantidad de parámetros que deben ser estimados a partir de datos experimentales. Por esta razón, en este caso en particular se mantiene la necesidad de eficiencia. La necesidad de la computación numérica es evidente, pues nos permite obtener aproximaciones muy precisas de soluciones a problemas que serían imposibles de resolver analíticamente.

El método de Levenberg-Marquardt es un método numérico que se utiliza para resolver problemas de mínimos cuadrados. Este método fue desarrollado por los estadísticos estadounidenses Kenneth Levenberg [4] y Donald Marquardt [5] en la década de 1950, ambos descubrieron este algoritmo por separado. En 1978, el autor J.J. Moré implementa su propia versión numérica a partir de las ideas de estos, y será objeto de nuestro estudio.

El método de Levenberg-Marquardt se considera el precursor de los llamados métodos de región de confianza para resolver problemas de minimización sin restricciones, y nace de querer solucionar el problema de mínimos cuadrados. Problema muy utilizado en estadística para ajustar parámetros a partir de datos experimentales.

Desde entonces, se ha convertido en uno de los métodos más utilizados para resolver problemas

de optimización no lineal y ha sido aplicado en una amplia gama de campos, incluyendo la ingeniería, la economía, la física y la biología. En este trabajo, en el capítulo 3 se describirá en detalle el método de Levenberg-Marquardt después de echar un vistazo general a los métodos de optimización sin restricciones en el capítulo 1 y de presentar el problema de mencionado en el capítulo 2. Los dos primeros capítulos se basan en el trabajo de [7] y [8], por lo que se recomienda su lectura para profundizar en la teoría de la optimización. En el capítulo 4 implementaremos este algoritmo en Matlab® una versión propia y simplificada de la versión de Moré, lo aplicaremos a una serie de ejemplos y lo comparamos con el comando `lsqnonlin` de Matlab®.

Capítulo 1

Fundamentos de la optimización sin restricciones

Un problema de optimización sin restricciones tiene la forma

$$\min_x f(x), \tag{1.1}$$

donde $x \in \mathbb{R}^n$ y $f : \mathbb{R}^n \to \mathbb{R}$ es continuamente diferenciable, a f se le llama **función objetivo**. Notar que podemos usar esta formulación para referirnos tanto a los problemas de minimización como de maximización, basta sustituir f por $-f$.

La dificultad de un problema como este viene por no conocer el comportamiento global de f. Normalmente solo disponemos de la evaluación de f en algunos puntos y, a lo mejor, de algunas de sus derivadas. El trabajo de los algoritmos de optimización es identificar la solución sin usar demasiado tiempo ni almacenamiento computacional. Veamos una serie de definiciones y resultados necesarios.

Definición 1.1. A una aplicación $\|\cdot\|$ se le llama *norma* si y sólo si cumple:

1. $\|x\| \geq 0$, $\forall \mathbb{R}^n$ y $\|x\| = 0$ si y solo si $x = 0$.
2. $\|\alpha x\| = |\alpha| \|x\|$, $\forall \alpha \in \mathbb{R}$, $x \in \mathbb{R}^n$.
3. $\|x + y\| \leq \|x\| + \|y\|, \forall x, y \in \mathbb{R}^n$.

Un ejemplo muy común es la *norma* l_2, también llamada *norma euclídea*, a la cual nos referiremos cuando no se especifique lo contrario, se define

$$\|x\|_2 = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{\frac{1}{2}}. \tag{1.2}$$

Definición 1.2. Un punto x se dice *punto estacionario* de f si $\nabla f(x) = 0$.

Definición 1.3. Un punto x^* se dice *mínimo local* si existe $\delta > 0$ tal que $f(x^*) \leq f(x)$ para todo $x \in \mathbb{R}^n$ que satisface $\|x - x^*\| < \delta$. Un punto x^* se dice *mínimo local estricto* si existe $\delta > 0$ tal que $f(x^*) < f(x)$ para todo $x \in \mathbb{R}^n$ que satisface $\|x - x^*\| < \delta$ con $x \neq x^*$.

Definición 1.4. Un punto x^* se dice *mínimo global* si $f(x^*) \leq f(x)$ para todo $x \in \mathbb{R}^n$. Un punto x^* se dice *mínimo global estricto* si $f(x^*) < f(x)$ para todo $x \in \mathbb{R}^n$ con $x \neq x^*$.

Definición 1.5. Sea $f : \mathbb{R}^n \to \mathbb{R}$ diferenciable en $x \in \mathbb{R}^n$ tal que $\langle \nabla f(x), d \rangle < 0$, entonces a d se le llama *dirección descendente* de f en x.

Teorema 1.6 (Teorema de Taylor)**.** *Sea $f : \mathbb{R}^n \to \mathbb{R}$ continuamente diferenciable y sea $p \in \mathbb{R}^n$, tenemos que*

$$f(x + p) = f(x) + \nabla f(x + tp)^T p, \quad t \in (0, 1). \tag{1.3}$$

Si además, f es dos veces continuamente diferenciable

$$\nabla f(x + p) = \nabla f(x) + \int_0^1 \nabla^2 f(x + tp) p \, dt, \tag{1.4}$$

y

$$f(x + p) = f(x) + \nabla f(x)^T p + \frac{1}{2} p^T \nabla^2 f(x + tp) p, \ t \in (0, 1). \tag{1.5}$$

Proposición 1.7. *Partiendo de la reformulación (1.5) y tomando el último término como el error de aproximación $o(t)$*

$$f(x_k + td) = f(x_k) + t \nabla f(x_k)^T d + o(t), \tag{1.6}$$

se cumple que

$$\exists \delta > 0 \text{ tal que } f(x_k + td) < f(x_k) \quad \forall t \in (0, \delta) \tag{1.7}$$

si y solo si d es una dirección descendiente de f en x_k.

Tratemos ahora las condiciones de optimalidad, es decir, las condiciones necesarias y suficientes para que un punto x^* sea un mínimo local de f, necesarias para la construcción de un algoritmo de optimización.

Definición 1.8. Una matriz H es *definida positiva* si $x^T H x > 0$ para todo $x \neq 0$.

Definición 1.9. Una matriz H es *semidefinida positiva* si $x^T H x \geq 0$ para todo $x \neq 0$.

Teorema 1.10 (Condición Necesaria de Primer Orden)**.** *Sea $f : D \subset \mathbb{R}^n \to \mathbb{R}$ continuamente diferenciable en un conjunto abierto D. Si x^* es un mínimo local de (1.1), entonces $\nabla f(x^*) = 0$.*

Teorema 1.11. *(Condición Necesaria de Segundo Orden) Sea $f : D \subset \mathbb{R}^n \to \mathbb{R}$ dos veces continuamente diferenciable en un conjunto abierto D. Si x^* es un mínimo local de (1.1), entonces $\nabla f(x^*) = 0$ y $\nabla^2 f(x^*)$ es definida positiva.*

Teorema 1.12 (Condición Suficiente de Segundo Orden)**.** *Sea $f : D \subset \mathbb{R}^n \to \mathbb{R}$ dos veces continuamente diferenciable en un conjunto abierto D. Si $\nabla f(x^*) = 0$ y $\nabla^2 f(x^*)$ es definida positiva, entonces $x^* \in D$ es un mínimo local.*

Para finalizar con los resultados previos, tratemos el concepto de convexidad, una propiedad que nos permitirá encontrar mínimos globales.

Definición 1.13. Sea $S \subset \mathbb{R}^n$ y sean $x_1, x_2 \in S$ cualesquiera. Si $\alpha x_1 + (1-\alpha)x_2 \in S$ para todo $\alpha \in [0,1]$, entonces se dice que D es un *conjunto convexo.*

Definición 1.14. Sea $S \subset \mathbb{R}^n$ un conjunto convexo no vacío. Sea $f : S \subset \mathbb{R}^n \to R$. Si para cualquiera $x_1, x_2 \in S$ y $\alpha \in (0,1)$, se cumple que

$$f(\alpha x_1 + (1-\alpha)x_2) \leq \alpha f(x_1) + (1-\alpha) f(x_2), \tag{1.8}$$

se dice que f es una función convexa en S.

Teorema 1.15. *Sea $S \subset \mathbb{R}^n$ un conjunto convexo no vacío y $f : S \subset \mathbb{R}^n \to \mathbb{R}$ una función convexa. Si x^* es mínimo local de (1.1), entonces también es mínimo global.*

Teorema 1.16. *Sea $f : \mathbb{R}^n \to \mathbb{R}$ convexa y diferenciable, entonces x^* es un mínimo global si y solo si $\nabla f(x^*) = 0$.*

1.1. Generalidades y estructura de los algoritmos

Desde el punto de vista numérico o computacional, un problema de optimización se enfoca de otro modo. Seguimos necesitando resultados teóricos para poder obtener resultados, pero estos se obtienen a través de lo que llamamos *algoritmos*, que son un conjunto de órdenes con unas normas y condiciones que les damos a un ordenador para realizar una tarea. En este conjunto de órdenes se tiene en cuenta el funcionamiento de los ordenadores, sus puntos fuertes y sus puntos débiles, con estructuras de bucles iterativos que van aproximando la solución, para ser lo más eficientes posibles en tiempo de computación. Por esta razón, no se malgastan recursos en tener conocimiento global del comportamiento de f y la mayoría de algoritmos solo encuentran mínimos locales, lo cual es suficiente en muchos casos prácticos, ya que puede darnos una idea de la zona por donde empezar a buscar. Para encontrar mínimos globales, podemos usar una secuencia de otros algorítmos de optimización local, o en cambio aprovechar propiedades de algunas funciones para asegurar un mínimo global.

Todo algoritmo de optimización sin restricciones comienza con un punto de partida, denotado normalmente como x_0. Aunque generalmente el usuario introduce una estimación razonable, el punto puede ser elegido por el algoritmo, tanto de forma sistemática como aleatoria. El algoritmo

itera sobre x_0, creando una sucesión $\{x_k\}_{k=0}^n$e. Para decidir como se avanza de un x_k al siguiente, los algoritmos utilizan información sobre $f(x_k)$ o incluso en los puntos anteriores x_0, x_1, ..., x_{k-1} con el objetivo de que $f(x_{k+1}) < f(x_k)$. Además, para que esta sucesión sea finita, se imponen unas condiciones de parada, que suelen ser cuando se cumple un error aceptable de la estimación o cuando se llega a un máximo de iteraciones, esta última condición es necesaria para evitar que el algoritmo se quede en un bucle infinito. Con estas premisas, podemos esquematizar la estructura de la siguiente forma:

Algoritmo 1.17 (Esquema de algoritmo)**.**

Paso 1. Paso inicial. Se definen las condiciones iniciales, suelen ser el primer punto de la sucesión $\{x_k\}$, valores para definir las condiciones de parada como el número máximo de iteraciones o el error aceptable, y otros parámetros.

Paso 2. Test de parada. Se comprueba si se cumple alguna condición de parada.

Paso 3. Proceso principal. Se realizan los cálculos necesarios para avanzar de x_k a x_{k+1}.

Paso 4. Actualización y bucle. Se actualiza el valor de x_k a x_{k+1}, así como otros parámetros necesarios. Se repite el proceso desde el paso 2.

En este contexto de optimización numérica, el estudio de la convergencia tiene un especial interés. Es necesario que un algoritmo converga para asegurarnos que estas iteraciones lleven a un resultado en algún momento, además de que la velocidad de convergencia también resulta un factor importante a la hora de elegir un algoritmo. Es por esto que mantenemos la terminología de convergencia local y global. El tipo de convergencia es una medida de efectividad, es decir, nos dará una idea de la velocidad con la que llega a una solución y como se comporta en los distintos casos. Sea $\{x_k\}$ una sucesión generada por un algoritmo convergente en norma a x^*, esto es,

$$\lim_{k\to\infty} \|x_k - x^*\| = 0. \tag{1.9}$$

Definición 1.18. Si existen $\alpha \geq 1$ y β constante positiva, tales que

$$\lim_{k\to\infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^\alpha} = \beta, \tag{1.10}$$

diremos que el algoritmo converge con orden α. En particular, si $\alpha = 1$ y $\beta < 1$, se dice que la convergencia es *lineal* y, si $\alpha = 2$, *cuadrática.*

Hablaremos de las dos estrategias fundamentales que se utilizan para avanzar de x_k a x_{k+1}, *búsqueda de línea* y *región de confianza.*

1.2. Búsqueda de línea

En este caso el algoritmo tiene dos puntos clave en cada iteración, primero elegir una *dirección* d_k y, tomando el punto de partida, buscar en esa dirección el nuevo valor. Es decir, dado x_k, obtiene

$$x_{k+1} = x_k + \alpha_k d_k, \tag{1.11}$$

donde la clave está en encontrar el vector dirección d_k con la misma dimensión que x_k, y un paso α_k que nos indica cuánto avanzar en esa dirección. Si definimos

$$\phi(\alpha) = f(x_k + \alpha_k d_k), \tag{1.12}$$

el problema que parte de este x_k y d_k elegido previamente, y encuentra un α_k tal que

$$\phi(\alpha_k) < \phi(0), \tag{1.13}$$

se dice que realiza una búsqueda de línea. Si $alpha_k$ soluciona el problema

$$\min_{\alpha_k > 0} f\left(x_k + \alpha_k d_k\right), \tag{1.14}$$

se dice que realiza una búsqueda de línea *exacta* u *óptima*. Para evitar el gran coste computacional que puede llegar a tomar, lo más común es tomar un α_k que aporte un descenso aceptable, en cuyo caso se le llama búsqueda de línea *inexacta* o *aproximada*. Desde el nuevo punto se busca otra dirección y paso para repetir el proceso. Veamos brevemente cómo se elige d_k.

La mayor parte de algoritmos de este tipo necesitan que d_k sea una dirección de descenso, esto es, $d_k^T \nabla f_k < 0$, lo cual asegura que en esa dirección se podrá reducir el valor de f. Esta suele tener la forma

$$d_k = -B_k^{-1} \nabla f_k \tag{1.15}$$

con B_k una aproximación de la matriz Hessiana $\nabla^2 f(x_k)$ simétrica y no singular. Según lo que acabamos de decir, necesitamos que B_k sea definida positiva. En las tres corrientes principales se elige un B_k distinto, en el *método del descenso máximo* o *descenso del gradiente*, se usa la matriz identidad I. En el *método de Newton* se usa la matriz exacta, mientras que en los *métodos Quasi-Newton* la matriz Hessiana es aproximada para cada x_k.

1.2.1. Método de Newton con búsqueda de línea

No se hablará en detalle de los métodos de Newton, pero sí es útil saber que la idea principal detras de estos es usar la aproximación cuadrática $q^{(k)}$ de la función objetivo,

$$q^{(k)}(p) = f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T \nabla^2 f(x_k) p, \tag{1.16}$$

si $f : \mathbb{R}^n \to \mathbb{R}$ dos veces continuamente diferenciable, $x_k \in \mathbb{R}^n$ y la Hessiana $\nabla^2 f(x_k)$ es definida positiva. En tal caso aproximamos $f(x_k + p) \approx q^{(k)}(p)$.

Minimizando $q^{(k)}(p)$ obtenemos la fórmula de Newton, denotando $G_k = \nabla^2 f(x_k)$ y $g_k = \nabla f(x_k)$:

$$x_{k+1} = x_k - G_k^{-1} g_k. \tag{1.17}$$

Teorema 1.19 (Teorema de Convergencia del Método de Newton)**.** *Sea $f \in \mathcal{C}^2$ y x_k lo suficientemente cerca a la solución x^* del problema de minimización con $g(x^*) = 0$. Si la Hessiana $G(x^*)$ es definida positiva y $G(x)$ satisface la condición de Lipschitz*

$$|G_{ij}(x) - G_{ij}(y)| \leq \beta \|x - y\|, \text{ para algún } \beta, \text{ y para todo } i, j, \tag{1.18}$$

siendo $G_{ij}(x)$ el elemento en la posición (i, j) de la matriz $G(x)$, entonces para todo k, la iteración (1.17) está bien definida y la sucesión $\{x_k\}$ generada converge a x^* de forma cuadrática.

1.3. Región de confianza

Veremos esta sección con un poco más de profundidad, ya que el método que estudiaremos en el tercer capítulo forma parte de este grupo. Aunque no conozcamos el comportamiento de una función de manera global, podemos tomar una pequeña región donde la función se comporte de una manera predecible, y en esta región aplicar un método de optimización conocido. En esta idea se basan los métodos de región de confianza, primero se fija una distancia máxima Δ_k para definir una región, generalmente de la forma

$$\Omega_k = \{x : \|x - x_k\| \leq \Delta_k\}, \tag{1.19}$$

y luego se busca el punto que minimice la función dentro de este. Para aproximar el comportamiento de la función en la región elegida, se suele utilizar el modelo cuadrático $q^{(k)}$ de Newton, aprovechando la notación usada:

$$m_k(p) := q^{(k)}(p) = f(x_k) + g_k^T p + \frac{1}{2} p^T G_k p. \tag{1.20}$$

La elección de Δ_k dependerá de si el m_k es una buena aproximación o no, pues cuanto más grande sea la región de confianza, más complicado es que m_k y f se parezcan. Para este método no es necesario que la Hessiana sea definida positiva, lo cual nos supone una ventaja. En cada iteración, una vez elegido Δ_k se resuelve el siguiente problema:

$$\begin{aligned} \min_p \quad & m_k(p) = f(x_k) + g_k^T p + \frac{1}{2} p^T B_k p \\ \text{s.a.} \quad & \|p\| \leq \Delta_k. \end{aligned} \tag{1.21}$$

Donde se recupera la aproximación B_k, más usual que la Hessiana exacta G_k debido a su coste computacional.

Definición 1.20. Un punto x_k se dice *factible* si cumple las restricciones impuestas en el problema de optimización. El conjunto de puntos factibles se llama *conjunto factible*.

Como este subproblema es de optimización con restricciones, las condiciones de optimalidad se ven un poco alteradas, para más exactitud referimos a [7, Capítulo 12] o a [8, Capítulo 8].

La restricción impuesta puede variar según la necesidad de cada caso. Un cambio en la norma puede ser conveniente, ya que se cambia la forma de la región de confianza según el caso particular. Estudiaremos el caso más general, tomando como región de confianza la bola definida por $\|p\|_2 \leq \Delta_k$.

Caractericemos la solución del subproblema (1.21) en el siguiente teorema. Por conveniencia, omitimos los subíndices para una notación más clara,

$$\begin{aligned} \min_{p} \quad & m(p) = f + g^T p + \frac{1}{2} p^T B p, \\ \text{s.a.} \quad & \|p\| \leq \Delta. \end{aligned} \tag{1.22}$$

Teorema 1.21. *El vector p^* es una solución global de (1.22) si y solo si p^* es factible y existe un escalar $\lambda^* \geq 0$ tal que*

1. $(B + \lambda^* I)p^* = -g$,
2. $\lambda^*(\Delta - \|p^*\|) = 0$,
3. $(B + \lambda^* I)$ *es semidefinida positiva.*

Demostración. Dejemos el tercer punto para el final. Supongamos que p^* es solución del subproblema (1.22). De las condiciones de optimalidad con restricciones (véase [7] o [8]), se tiene que existe un multiplicador de Lagrange $\lambda^* \geq 0$ tal que se cumplen las condiciones 1 y 2. Problemos ahora el punto 3, que nos dice que $(B + \lambda^* I)$ es semidefinida positiva.

Si $\|p^*\| < \Delta$, entonces $\lambda^* = 0$ y p^* es una solución sin restricciones y, además, $(B + \lambda^* I)$ es semidefinida positiva por serlo B. Si $\|p^*\| = \Delta$, por la condición necesaria de segundo orden (véase [8, 8.3]), se tiene que $p^T(B + \lambda^* I)p \geq 0$ para todo p tal que $p^T p^* = 0$. En el caso contrario de que $p^T p^* \neq 0$, tomamos $t = -\frac{2p^T p^*}{\|p\|^2}$, entonces $\|p^* + tp\| = \Delta$. Por definición de p^*, se tiene que

$$m(p^* + tp) + \frac{1}{2}\lambda^* \|p^* + tp\|^2 \geq m(p^*) + \frac{1}{2}\lambda^* \|p^*\|^2 . \tag{1.23}$$

Desarrollando y reordenando términos, se tiene que

$$\frac{1}{2}t^2 p^T(B + \lambda^* I)p \geq -t(p^T(g + (B + \lambda^* I)p^*)). \tag{1.24}$$

Teniendo en cuenta el primer punto, $(B+\lambda^* I)p^* = -g$, se tiene que el lado derecho de la anterior ecuación es 0, lo que resulta en

$$p^T(B + \lambda^* I)p \geq 0, \tag{1.25}$$

para todo p tal que $p^T p^* \neq 0$. Por lo tanto, $(B + \lambda^* I)$ es semidefinida positiva.

Veamos ahora la implicación contraria. Asumimos que existe $\lambda^* \geq 0$ tal que p^* cumple los 3 puntos del teorema, además de ser factible. Por lo tanto, para todo p factible, se tiene que

$$\begin{aligned} m(p) &= f + g^T p + \frac{1}{2}p^T(B + \lambda^* I)p - \frac{1}{2}\lambda^* \|p\|^2 \\ &\geq f + g^T p + \frac{1}{2}(p^*)^T(B + \lambda^* I)p - \frac{1}{2}\lambda^* \|p^*\|^2 \\ &= m(p^*) + \frac{1}{2}\lambda^* \left(\|p^*\|^2 - \|p\|^2\right). \end{aligned} \tag{1.26}$$

A partir del punto 2, se tiene que $\lambda^*(\Delta^2 - (p^*)^T p^*) = 0$, así que la desigualdad anterior se puede escribir como

$$\begin{aligned} m(p) &\geq m(p^*) + \frac{1}{2}\lambda^* \left[\left(\|p^*\|^2 - \Delta^2\right) + \left(\Delta^2 - \|p\|^2\right)\right] \\ &= m(p^*) + \frac{1}{2}\lambda^* \left(\Delta^2 - \|p\|^2\right). \end{aligned} \tag{1.27}$$

Por lo tanto, como $\lambda^* \geq 0$ y $\|p\| \leq \Delta$, se tiene que $m(p) \geq m(p^*)$ para todo p factible y, finalmente, que p^* es una solución de (1.22).

□

Este teorema caracteriza la solución según el primer punto. El segundo punto es una condición complementaria que nos dice que al menos uno de los dos factores es 0. Esto es, si $\|p\| < \Delta$, λ tendrá que ser 0 y $Bp^* = -g$ con B definida positiva (puntos 1 y 3). En el caso de que $\|p\|$ se maximice, lo cual da a entender que la solución óptima no se encuentra dentro de la región de confianza, λ podrá tomar valores positivos. Para la demostración referimos a [7, Sección 4.3] o [8, Teorema 6.1].

Respecto a la convergencia de este algoritmo, veremos un resultado relevante para el estudio realizado en el tercer capítulo. Si fuera de interés, se puede consultar [8, Sección 6.1.2].

La efectividad de cada iteración depende de la elección del radio Δ_k, es por ello que puede que la primera elección de este no sea la definitiva. Es decir, se toma un radio a raíz de la información que se tenga, esta puede incluír la de pasos anteriores, y luego se decide si este radio nos da un resultado aceptable. Un radio demasiado pequeño nos puede hacer perder la oportunidad de ser mucho más rápidos, pero para un paso demasiado grande, el mínimo de la función modelo m_k puede estar lejos del mínimo de la función objetivo. Por esta razón, en cada paso se valora actualizar Δ_k para que el radio sea más grande o más pequeño.

Veamos ahora de forma detallada como se elige el radio Δ_k en cada iteración. Esta elección se toma según el parecido entre la función f y el modelo m_k tomado en las iteraciones previas. Dado p_k, definimos el ratio

$$\rho_k = \frac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k)}, \tag{1.28}$$

donde el numerador es la *reducción real*, mientras que el denominador es la *reducción prevista*. La reducción prevista será positiva por definición, pues p_k es elegido para tener el menor valor posible y el 0 es una posibilidad. Por tanto, si ρ_k es negativo, el nuevo valor $f(x_k + p_k)$ no es menor que $f(x_k)$ y este paso ha de ser rechazado. Por otro lado, si ρ_k es cercano a 1, esto quiere decir que f y m_k se comportan de manera similar en la región tomada en la iteración actual, por tanto podemos agrandar el radio con seguridad. En resumen, nos quedamos con el radio elegido si ρ_k no tiene un valor muy cercano a 0 o a 1.

Una vez tomado el radio, encontrar el mínimo es directo en el caso de que B_k sea definida positiva, basta tomar $p_k^B = -B_k^{-1} g_k$, conocido como *paso completo*. En caso contrario tampoco supone una tarea muy costosa ya que sólo se necesita una solución aproximada para garantizar la convergencia. El proceso se describe en el siguiente algoritmo. Aclarar que tomaremos η_1 y η_2 como parámetros de control, los cuales servirán para decidir si hace falta modificar el radio Δ_k. Los parámetros γ_1 y γ_2 se usarán para modificar el radio Δ_k en caso de que sea necesario, ϵ es la tolerancia para el criterio de parada y $\bar{\Delta}$ es el radio máximo.

Algoritmo 1.22 (Región de confianza)**.**

Paso 1. Dados $x_0, \bar{\Delta}, \Delta_0 \in (0, \bar{\Delta}), \epsilon \geq 0, 0 < \eta_1 \leq \eta_2 < 1$ *y* $0 < \gamma_1 < 1 < \gamma_2, k := 0$.

Paso 2. Si $\|g_k\| \leq \epsilon$ *terminar.*

Paso 3. Aproximar p_k *resolviendo (1.21)-(1.22) según la caracterización del teorema 1.21.*

Paso 4. Calcular $f(x_k + p_k)$ *y* ρ_k*. Definir*

$$x_{k+1} = \begin{cases} x_k + p_k, & \text{si } \rho_k \geq \eta_1, \\ x_k, & \text{en otro caso.} \end{cases}$$

Paso 5. Si $\rho_k < \eta_1$ *entonces* $\Delta_{k+1} \in (0, \gamma_1 \Delta_k]$.
Si $\rho_k \in [\eta_1, \eta_2)$ *entonces* $\Delta_{k+1} \in [\gamma_1 \Delta_k, \Delta_k]$.
Si $\rho_k \geq \eta_2$ *y* $\|p_k\| = \Delta_k$ *entonces* $\Delta_{k+1} \in [\Delta_k, \text{mín}\{\gamma_2 \Delta_k, \bar{\Delta}\}]$.

Paso 6. Calcular B_{k+1}*, actualizar* $m^{(k)}$ *y* $k := k + 1$*. Ir al Paso 2.*

Si nos fijamos, en el paso 5 se define la actualización de Δ_k, la cual no siempre supone un cambio. El primer caso se dice que resulta en una *iteración no exitosa*, mientras que el segundo

caso marcará una *iteración exitosa* y el tercero una *iteración muy exitosa*. Añadir que $\bar{\Delta}$ es el máximo radio de la región para toda iteración. Normalmente se elige $\eta_1 = 0{,}25$ y $\eta_2 = 0{,}75$, mientras que $\gamma_1 = 0{,}25$ y $\gamma_2 = 2$. Además, la elección de Δ^{k+1} se puede obtener por interpolación polinómica dentro del intervalo dado, aunque es común tomar el máximo del intervalo.

Teorema 1.23. *Se supone $\|B_k\|$ uniformemente acotada para cada k y que el conjunto de nivel $\{x \,|\, f(x) \leq f(x_0)\}$ es acotado, con $f : \mathbb{R}^n \to \mathbb{R}$ continuamente diferenciable. Además, para generalizar el subproblema (1.22) y calcular cada $\|p_k\|$, se permite un exceso sobre la cota con un pequeño margen, esto es*

$$\|p_k\| \leq \tilde{\eta}\Delta_k, \tag{1.29}$$

siendo $\tilde{\eta}$ constante positiva. Si el algoritmo (1.22) tiene las suficientes iteraciones exitosas, entonces la sucesión que genera cumplirá

$$\liminf_{k\to\infty} \|g_k\| = 0. \tag{1.30}$$

Capítulo 2

Mínimos Cuadrados

El problema de mínimos cuadrados surge de la necesidad de ajustar modelos que nos permitan predecir ciertos comportamientos en una amplia variedad de campos. Dados unos datos $(t_1, y_1), (t_2, y_2), \cdots, (t_m, y_m)$, queremos ajustar una función $\phi(t, x) : \mathbb{R}^n \to \mathbb{R}^m$ de forma que se minimicen los residuos $r_i(x) = \phi(t_i, x) - y_i$ para $i = 1, \cdots, m$

$$\min_{x \in \mathbb{R}^n} f(x) = \frac{1}{2} r(x)^T r(x) = \frac{1}{2} \sum_{i=1}^{m} r_i^2(x), \quad m \geq n, \tag{2.1}$$

donde $r(x) = (r_1(x), r_2(x), \ldots, r_m(x))^T$, con $r_i : \mathbb{R}^n \to \mathbb{R}$ funciones continuamente diferenciables y x es el parámetro de dimensión n a estimar.

Ejemplo 2.1 (Problema lineal)**.** Partimos de una muestra aleatoria de 20 puntos en el intervalo $t \in [0, 5]$ e $y \in [0, 7]$. En un contexto estadístico, a t se le llama *variable explicativa* y a y la *variable respuesta*. Las cruces azules en la figura 2.1 representan los puntos de la muestra. El objetivo del problema es ajustar una recta a los datos, es decir, encontrar los parámetros a y b que minimicen la diferencia entre los puntos de la muestra y su valor esperado, representado por la recta roja definida por $y = at + b$. Recordar que en un contexto estadístico se tienen en cuenta una serie de hipótesis sobre la distribución de los datos, por lo que el ajuste de una recta es un modelo muy simple. En la práctica, se suelen utilizar modelos más complejos, como por ejemplo, polinomios de grado n.

En la figura, marcamos con una línea verde la diferencia que se intenta minimizar en el problema de mínimos cuadrados para un solo dato en particular, el problema obtiene esta diferencia para el resto de puntos de la muestra y trata de minimizarlo.

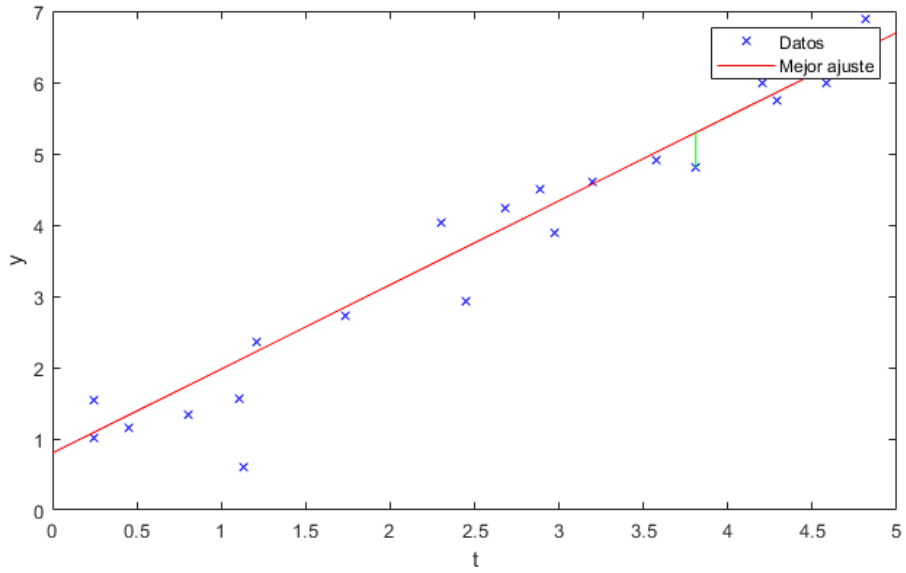

Figura 2.1: Ejemplo de ajuste de una recta a una muestra de puntos

Veamos las propiedades de este modelo concreto de optimización sin restricciones y cómo se pueden aprovechar para formular algoritmos eficientes y robustos. Sea $J(x)$ la matriz Jacobiana de $r(x)$,

$$J(x) = \begin{bmatrix} \nabla r_1(x)^T \\ \nabla r_2(x)^T \\ \vdots \\ \nabla r_m(x)^T \end{bmatrix} = \begin{bmatrix} \frac{\partial r_1}{\partial x_1}(x) & \frac{\partial r_1}{\partial x_2}(x) & \cdots & \frac{\partial r_1}{\partial x_n}(x) \\ \frac{\partial r_2}{\partial x_1}(x) & \frac{\partial r_2}{\partial x_2}(x) & \cdots & \frac{\partial r_2}{\partial x_n}(x) \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial r_m}{\partial x_1}(x) & \frac{\partial r_m}{\partial x_2}(x) & \cdots & \frac{\partial r_m}{\partial x_n}(x) \end{bmatrix}. \tag{2.2}$$

El gradiente y la Hessiana de f se pueden expresar como sigue:

$$g(x) = \nabla f(x) = \sum_{i=1}^{m} r_i(x) \nabla r_i(x) = J(x)^T r(x), \tag{2.3}$$

$$\begin{aligned} G(x) = \nabla^2 f(x) &= \sum_{i=1}^{m} \nabla r_i(x) \nabla r_i(x)^T + \sum_{i=1}^{m} r_i(x) \nabla^2 r_i(x) \\ &= J(x)^T J(x) + S(x). \end{aligned} \tag{2.4}$$

Si nos fijamos en la formulación de la matriz Hessiana, el cálculo del primer termino es directo gracias a que ya obtenemos $J(x)$ para calcular el gradiente (2.3), así que el coste de obtener $G(x)$ se reduce al segundo término, que hemos denotado $S(x)$. Adaptando el modelo cuadrático (1.16)

$$q^{(k)}(x) = f(x_k) + g_k^T (x - x_k) + \frac{1}{2}(x - x_k)^T G_k (x - x_k), \tag{2.5}$$

y sustituyendo según (2.1), (2.3) y (2.4), obtenemos el método de Newton para (2.1),

$$x_{k+1} = x_k - (J(x_k)^T J(x_k) + S(x_k))^{-1} J(x_k)^T r(x_k). \tag{2.6}$$

2.1. El Problema Lineal

El primer caso más sencillo es si $\phi(t, x)$ es una función lineal, en cuyo caso los residuos $r_i(x)$ también serán lineales. Por ser ϕ lineal, se puede representar como Jx, con J una matriz $m \times n$. Realizaremos un estudio del caso lineal para tener un conocimiento de como se enfocan estos problemas, que nos servirá para el caso no lineal. Si escribimos el vector residuo como $r(x) = Jx - y$, la función objetivo nos queda de la forma

$$f(x) = \frac{1}{2}\|Jx - y\|^2. \tag{2.7}$$

En consecuencia, tomando como referencia (2.3) y (2.4) y teniendo en cuenta que en este caso particular $\nabla^2 r_i = 0$, nos queda

$$\nabla f(x) = J^T(Jx - y), \qquad \nabla^2 f(x) = J^T J. \tag{2.8}$$

Como f es convexa, dado un punto x^* tal que $\nabla f(x^*) = 0$, este será mínimo global (1.16). Por tanto, x^* satisface el siguiente sistema lineal:

$$J^T J x^* = J^T y. \tag{2.9}$$

Antes de ver como se resuelve numéricamente este sistema de ecuaciones, conocidas como *ecuaciones normales* de (2.7), veamos los conceptos previos necesarios.

Definición 2.2. Un problema numérico se dice *bien condicionado* si su solución no se ve afectada por pequeñas perturbaciones a cualquiera de los datos que definen el problema.

Definición 2.3. Una matriz cuadrada A se dice *definida positiva* si existe un $\alpha \in \mathbb{R}^+$ tal que

$$x^T A x \geq \alpha x^T x, \quad \text{para todo } x \in \mathbb{R}^n. \tag{2.10}$$

Esta es *semidefinida postiva* si

$$x^T A x \geq 0, \quad \text{para todo } x \in \mathbb{R}^n. \tag{2.11}$$

Definición 2.4. Una matriz $n \times n$ cuadrada A se dice *no singular* si para cada $b \in \mathbb{R}^n$, existe $x \in \mathbb{R}^n$ tal que $Ax = b$.

Definición 2.5. Una matriz cuadrada Q se dice *ortogonal* si cumple $QQ^T = Q^T Q = I$

Definición 2.6. Si tomamos los sistemas de vectores de una una matriz $A_{m\times n}$, $\{u_i\}_{i=1}^n$ y $\{v_i\}_{i=1}^m$, al número máximo de vectores linealmente independientes se le llama *rango*, tanto de los sistemas de vectores como de la matriz A. Si $n < m$, se dice que A es de *rango completo* si su rango es n, que es el máximo posible.

Definición 2.7. Sea A una matriz cuadrada. Un vector x se dice *autovector* (vector propio o eigenvector en inglés) de A si existe un escalar λ tal que $Ax = \lambda x$. A este escalar se le llama *autovalor* (valor propio o eigenvalue) de A.

Definición 2.8. Dado un autovalor, el conjunto de autovectores asociados a este forma un subespacio llamado *subespacio propio*. Al conjunto de todos los autovalores de A se le llama *espectro de A*.

Veamos ahora como se resuelve numéricamente el sistema de ecuaciones normales de (2.9), para más información sobre los algoritmos de factorización, ver por ejemplo [9] o [2] para profundizar en el tema.

Lo más común en este caso es usar distintos tipos de factorización sobre la matriz J^TJ o sobre J, para luego resolver con sustituciones triangulares. El primer algoritmo que se plantea es a partir de la **factorización de Cholesky**, comenzando por computar la matriz de coeficientes J^TJ y el lado derecho J^Ty. Después se computa la factorización de Cholesky

$$J^TJ = \bar{R}^T\bar{R}. \tag{2.12}$$

Para que esta exista, necesitamos que $m \geq n$ y que J sea de rango n, lo que permite que J^TJ sea simétrica y definida positiva. Se termina realizando las dos sustituciones triangulares con los factores de Cholesky para encontrar x^*. La principal desventaja de este método es que el condicionamiento de J^TJ es el cuadrado del condicionamiento de J, y esto puede llevar a errores de aproximación. Además, si J está mal condicionada, ni si quiera se puede llevar a cabo la factorización.

Una segunda posibilidad es basarse en la **factorización QR**, que evita el problema de depender del cuadrado del condicionamiento de J, ya que aplicaremos la factorización directamente a J. Se aprovecha que la norma euclídea no se ve afectada por trasformaciones ortogonales para partir de la igualdad

$$\|Jx - y\| = \left\|Q^T(Jx - y)\right\|, \tag{2.13}$$

siendo Q una matriz ortogonal $m \times m$. Realizando la factorización QR con una permutación Π, que para una matrix $A \in \mathbb{R}^{m\times n}$ tiene la forma

$$A\Pi = QR, \tag{2.14}$$

donde Π es una matriz permutación $n \times n$, y por tanto ortogonal; Q es una matriz ortogonal $m \times m$ y R es una matriz $n \times n$ triangular superior con elementos positivos en la diagonal. Se tiene

$$J\Pi = Q \begin{bmatrix} R \\ 0 \end{bmatrix} = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix} = Q_1R, \tag{2.15}$$

donde Q_1 son las primeras n columnas de Q y Q_2 las restantes. Por (2.13) y (2.15) se llega a

$$\begin{aligned} \|Jx - y\| &= \left\| \begin{bmatrix} Q_1^T \\ Q_2^T \end{bmatrix} (J\Pi\,\Pi^T x - y) \right\|^2 \\ &= \left\| \begin{bmatrix} R \\ 0 \end{bmatrix} (\Pi^T x) - \begin{bmatrix} Q_1^T y \\ Q_2^T y \end{bmatrix} \right\|^2 \\ &= \left\| R(\Pi^T x) - Q_1^T y \right\|^2 + \left\| Q_2^T y \right\|^2 . \end{aligned} \tag{2.16}$$

Notar que el segundo término no depende de x, por tanto para minimizar la expresión solo igualamos a 0 el primer término, lo que nos lleva a la solución

$$x^* = \Pi R^{-1} Q_1^T y. \tag{2.17}$$

En este caso, el error relativo es proporcional al condicionamiento de J y no de su cuadrado. Aún así, hay situaciones en las que necesitamos asegurar que la obtención sea de algún modo más robusta o en las que queremos más información acerca de la sensibilidad de la solución a perturbaciones en J o en y, típicas durante la computación.

Nos basaremos ahora en la **descomposición de valores singulares (DVS)**, cuya resolución una vez realizada la descomposición se enfoca de forma similar a la anterior. Primero se realiza el algoritmo DVS para obtener

$$J = U \begin{bmatrix} S \\ 0 \end{bmatrix} V^T = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} S \\ 0 \end{bmatrix} V^T = U_1 S V^T, \tag{2.18}$$

con U matriz $m \times m$, U_1 las primeras n columnas de U, U_2 las restantes, V matriz $n \times n$, ambas ortogonales y S matriz $n \times n$ de elementos diagonales $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n > 0$. Aprovechamos estas propiedades para obtener

$$x^* = V S^{-1} U_1^T y. \tag{2.19}$$

Denotando por $u_i \in \mathbb{R}^m$ y $v_i \in \mathbb{R}^n$ las columnas de U y V respectivamente, escribimos

$$x^* = \sum_{i=1}^{n} \frac{i_i^T y}{\sigma_i} v_i. \tag{2.20}$$

Fórmula de donde obtenemos información útil como la sensibilidad al aproximar x^*.

Las 3 opciones son buenas según las condiciones en las que nos encontremos. La resolución basada en Cholesky es útil cuando $m \gg n$ y es práctico trabajar almacenando $J^T J$ en lugar de J, siempre y cuando J sea de rango completo y bien condicionada. Si esto último no se cumple, la factorización QR es un enfoque más equilibrado, mientras que DVS es el más costoso a cambio de ser el más fiable.

Por último, mencionar que existen métodos para problemas muy grandes, en los que se usan técnicas iterativas como el método de gradientes conjugados para resolver el sistema.

2.2. El método de Gauss-Newton

Comenzamos los métodos de minimización del problema no lineal (2.1) con el método de Gauss-Newton. Se caracteriza por despreciar el término de segundo orden $S(x)$ del término G_k en el modelo cuadrático (2.5). Así, resulta

$$\begin{aligned} q^{(k)}(x) =& \frac{1}{2} r(x_k)^T r(x_k) + (J(x_k)^T r(x_k))^T (x - x_k) + \\ &+ \frac{1}{2}(x - x_k)^T (J(x_k)^T J(x_k))(x - x_k), \end{aligned} \tag{2.21}$$

y por tanto, aplicando búsqueda de línea y despreciando el término de segundo orden, obtenemos

$$x_{k+1} = x_k + p_k = x_k - (J(x_k)^T J(x_k))^{-1} J(x_k)^T r(x_k). \tag{2.22}$$

Notar que para que esté bien definido, la matriz Jacobiana $J(x)$ tiene que ser de rango completo. Gracias a la aproximación $\nabla^2 f(x_k) \approx J(x_k)^T J(x_k)$, hacemos que la única dificultad del algoritmo sea resolver un sistema lineal, ya que evitamos computar $\nabla^2 r_j$, $j = 1, 2, \ldots, m$. En algunas situaciones, cuando nos vamos acercando a la solución x^*, esta aproximación suele ser más precisa, ya sea porque los residuos r_i o $\left\|\nabla^2 r_i\right\|$ es cercano a cero. La eficacia del método dependerá por tanto de lo buena que sea esta aproximación.

Algoritmo 2.9 (Método de Gauss-Newton)**.**

Paso 1. x_0 *y* $\epsilon > 0$ *dados,* $k := 0$

Paso 2. Si $\|g_k\| \leq \epsilon$*, terminar.*

Paso 3. Obtener el paso p_k *resolviendo*

$$J(x_k)^T J(x_k) p_k = -J(x_k)^T r(x_k) \tag{2.23}$$

Paso 4. Definimos $x_{k+1} = x_k + p_k$ *y actualizamos* $k = k + 1$*. Ir a Paso 2.* □

Siempre y cuando J tenga rango completo y el gradiente $\nabla f_k = J(x_k)^T r(x_k)$ sea no nulo, la dirección p_k es una dirección descendente. Como vemos en el *Paso 3* resolvemos un caso análogo al problema lineal. Debido a esto, p_k es también la solución de

$$\min_{p_k} \frac{1}{2} \|J(x_k) p_k + r_k\|^2, \tag{2.24}$$

y por eso decimos que el método de Gauss-Newton es en realidad una linealización del problema no lineal de mínimos cuadrados y, como el método de Newton, resultará localmente convergente de manera cuadrática bajo las condiciones de este. Por tanto, el error de aproximación de la solución dependerá también de como resolvamos el problema lineal, y aplican los casos vistos en el apartado anterior.

Teorema 2.10. *Sea $f : \mathbb{R}^n \to \mathbb{R}$ y $f \in \mathcal{C}^2$. Supongamos que x^* es mínimo local del problema (2.1), $J(x^*)^T J(x^*)$ es definida positiva y la sucesión $\{x_k\}$ generada por el algoritmo (2.9) converge a x^*. Si $G(x)$ y $(J(x)^T J(x))^{-1}$ son lipschitzianas en una vecinidad de x^*, entonces*

$$\|x_{k+1} - x^*\| \leq \|(J(x^*)^T J(x^*))^{-1}\| \, \|S(x^*)\| \, \|x_k - x^*\| + O(\|x_k - x^*\|). \tag{2.25}$$

Este teorema nos dice esencialmente que la convergencia del método depende de $S(x^*)$. Cuando $S(x^*) = 0$ este converge de forma cuadrática y, según aumente, la convergencia es menor.

Teorema 2.11. *Sea $f : D \subset \mathbb{R}^n \to \mathbb{R}$ y $f \in \mathcal{C}^2(D)$, con D conjunto abierto convexo. Sea $J(x)$ lipschitziana en D y $\|J(x)\| \leq \alpha$, $\forall x \in D$. Supongamos que existe $x^* \in D$ y $\lambda, \sigma \geq 0$ tal que $J(x^*)^T r(x^*) = 0$, λ es el autovalor más pequeño de $J(x^*)^T J(x^*)$, y*

$$\left\|(J(x) - J(x^*))^T r(x^*)\right\| \leq \sigma \|x - x_k\|, \, \forall x \in D. \tag{2.26}$$

Si $\sigma < \lambda$, para cualquier $c \in (1, \lambda/\sigma)$, existe $\epsilon > 0$ tal que para todo $x_0 \in N(x^*, \epsilon)$, la sucesión resultante $\{x_k\}$ del algoritmo 2.9 está bien definida, converge a x^* y satisface

$$\|x_{k+1} - x^*\| \leq \frac{c\sigma}{\lambda} \|x_k - x^*\| + \frac{c\alpha\sigma}{2\lambda} \|x_k - x^*\|^2 \tag{2.27}$$

y

$$\|x_{k+1} - x^*\| \leq \frac{c\alpha + \lambda}{2\lambda} \|x_k - x^*\| < \|x_k - x^*\|. \tag{2.28}$$

Teorema 2.12. *Manteniendo las suposiciones de los dos teoremas 2.10 y 2.11, si $r(x^*) = 0$, entonces existe $\epsilon > 0$ tal que para cualquier $x_0 \in N(x^*, \epsilon)$, la sucesión $\{x_k\}$ obtenida del método de Gauss-Newton converge a x^* con orden cuadrático.*

Para concluir observamos que el método encaja dentro de los métodos de búsqueda de línea, tomando $d_k = p_k$ en (1.11) y calculando una longitud de paso α_k,

$$x_{k+1} = x_k - \alpha_k (J(x_k)^T J(x_k))^{-1} J(x_k)^T r(x_k). \tag{2.29}$$

Capítulo 3

El método de Levenberg-Marquardt

El método de Levenberg-Marquardt soluciona la necesidad de que J sea de rango completo cambiando el enfoque de búsqueda de línea por el de región de confianza. Lo hace manteniendo la raíz del método de Gauss-Newton, la linealización del problema obviando el término cuadrático, esto es, usando la aproximación $\nabla^2 f(x_k) \approx J(x_k)^T J(x_k)$. Este cambio de enfoque surge de que la linealización pierde efectividad según nos alejamos de x_k, por lo que conviene restringir el tamaño de $p = (x - x_k)$, resultando en el subproblema

$$\min_p \quad \frac{1}{2} \|J_k p + r_k\|^2, \quad \text{s.a. } \|p\| \leq \Delta_k, \tag{3.1}$$

donde $\Delta_k > 0$ es el radio de la región de confianza. Esta forma de afrontar el problema caracteriza los métodos de región de confianza (1.3) y, de hecho, el método de Levenberg-Marquardt es, para muchos, el precursor de estos métodos. Otra forma de escribir el modelo es

$$m_k(p) = \frac{1}{2} \|r_k\|^2 + p^T J_k^T r_k + \frac{1}{2} p^T J_k^T J_k p, \quad \text{s.a. } \|p\| \leq \Delta_k. \tag{3.2}$$

Lema 3.1. *El vector p es solución del subproblema (3.1) si y solo si p es factible y existe un $\lambda \geq 0$ tal que*

$$(J^T J + \lambda I)p = -J^T r, \tag{3.3}$$

$$\lambda(\Delta - \|p\|) = 0. \tag{3.4}$$

Demostración. Es consecuencia del teorema 1.21. Solo hay que ver que se cumplen las 3 condiciones, las dos primeras se siguen del propio lema. La tercera pide que $(J^T J + \lambda I)$ sea semidefinida positiva, y lo es por serlo $J^T J$ y por ser $\lambda > 0$ □

En concreto, como $(J^T J + \lambda I)$ es definida positiva, la solución de (3.3) es una dirección descendente. Lo que nos quiere decir este lema es que si la solución obtenida por el método de

Gauss-Newton cae estrictamente dentro de la región de confianza, esta solucionará también el subproblema (3.1). En otro caso, existe un $\lambda > 0$ que permite encontrar una solución a (3.3) con $\|p\| = \Delta$. Si $\lambda = 0$, la solución del problema es la de Gauss-Newton, y según aumenta λ, la solución se acerca a la del método de máximo descenso.

Además, podemos intercambiar la matriz identidad I por una matriz diagonal D_k definida positiva, lo que nos dejaría el problema (3.3) en

$$(J_k^T J_k + \lambda D_k)p = -J_k^T r_k. \tag{3.5}$$

Lo que resulta en una dirección p combinación del método Gauss-Newton y la dirección de máximo descenso respecto a la métrica definida por la matriz D_k. Veamos una serie de propiedades del método de Levenberg-Marquardt y de p en función de λ.

3.1. Propiedades del método de Levenberg-Marquardt

Teorema 3.2. *Si λ aumenta desde cero monótonamente, $\|p(\lambda)\|$ decrece de forma estrictamente monótona.*

Demostración. Por un lado,

$$\frac{\mathrm{d}}{\mathrm{d}\lambda}\|p\| = \frac{\mathrm{d}}{\mathrm{d}\lambda}(p^T p)^{\frac{1}{2}} + \frac{p^T \frac{\mathrm{d}p}{\mathrm{d}\lambda}}{\|p\|}. \tag{3.6}$$

Derivando (3.3) respecto a λ, obtenemos

$$p + (J^T J + \lambda I)\frac{\mathrm{d}p}{\mathrm{d}\lambda} = 0, \tag{3.7}$$

de esta y (3.3) resulta

$$\frac{\mathrm{d}p}{\mathrm{d}\lambda} = (J^T J + \lambda I)^{-2} g, \tag{3.8}$$

con $g = J^T r$. Sustituyendo en (3.6) y usando (3.3), nos queda

$$\frac{\mathrm{d}}{\mathrm{d}\lambda}\|p\| = -\frac{g^T (J^T J + \lambda I)^{-3} g}{\|p\|}. \tag{3.9}$$

Cuando $\lambda \geq 0$, $J^T J + \lambda I$ es definida positiva. Por tanto $\|p(\lambda)\|$ decrece de forma estrictamente monótona. □

Teorema 3.3. *Sea $\lambda_k > 0$, si p_k es solución de (3.3), entonces p_k es solución global de el subproblema*

$$\min_p \quad m_k(p) = \frac{1}{2}\|J_k p + r_k\|^2, \quad s.a. \|p\| \leq \|p_k\|, \tag{3.10}$$

Demostración. Como p_k es solución de (3.3), entonces

$$\begin{aligned} m_k(p_k) &= \frac{1}{2} r_k^T r_k + r_k^T J_k p_k + \frac{1}{2} p_k^T J_k^T J_k p_k \\ &= \frac{1}{2} r_k^T r_k - p_k^T (J_k^T J_k + \lambda_k I) p_k + \frac{1}{2} p_k^T J_k^T J_k p_k \\ &= \frac{1}{2} r_k^T r_k - \lambda_k p_k^T p_k - \frac{1}{2} p_k^T J_k^T J_k p_k. \end{aligned} \tag{3.11}$$

Por otro lado, para un p cualquiera, tenemos

$$\begin{aligned} m_k(p) &= \frac{1}{2} r_k^T r_k + p^T J_k^T r_k + \frac{1}{2} p^T J_k^T J_k p \\ &= \frac{1}{2} r_k^T r_k - p^T (J_k^T J_k + \lambda_k I) p_k + \frac{1}{2} p^T J_k^T J_k p \\ &= \frac{1}{2} r_k^T r_k - \lambda_k p^T p_k - p^T J_k^T J_k p_k + \frac{1}{2} p_k^T J_k^T J_k p_k. \end{aligned} \tag{3.12}$$

Por tanto, para cualquier p tal que $\|p\| \leq \|p_k\|$, se cumple

$$\begin{aligned} m_k(p) - m_k(p_k) &= \frac{1}{2} (p_k - p)^T J_k^T J_k (p_k - p) + \lambda_k (p_k^T p_k - p^T p_k) \\ &\geq \frac{1}{2} (p_k - p)^T J_k^T J_k (p_k - p) + \lambda_k \|p_k\| (\|p_k\| - \|p\|) \\ &\geq 0, \end{aligned} \tag{3.13}$$

por lo que p_k es una solución global óptima del problema (3.10). □

Como hemos dicho, podemos intercambiar la matriz identidad I por una matriz diagonal definida positiva D, por lo tanto una caracterización más general del método de Levenberg-Marquardt es la dada por la ecuación

$$(J_k^T J_k + \lambda D_k) p = -J_k^T r_k, \tag{3.14}$$

donde, como siempre, $J_k = J(x_k)$, $r_k = r(x_k)$ y se introduce $D_k = D(x_k)$, matriz diagonal definida positiva. El factor α_k satisface *la regla de Armijo*:

$$f(x_k + \alpha_k p_k) \leq f(x_k) + \sigma \alpha_k g_k^T p_k, \quad \sigma \in \left(0, \frac{1}{2}\right), \tag{3.15}$$

con $g_k = J^T r$.

Teorema 3.4. *En (3.14), el condicionamiento de $J^T J + \lambda D$ decrece en función de λ.*

Esta última propiedad nos indica que el condicionamiento de las ecuaciones iniciales a resolver se mejora con el método de Levenberg-Marquardt.

3.2. Convergencia

Estudiemos ahora la convergencia del método de Levenberg-Marquardt.

Teorema 3.5. *Sea $\{x_k\}$ la sucesión generada por el método de Levenberg-Marquardt. Supongamos que la longitud de paso α_k es la determinada por la regla de Armijo (3.15). Si existe una subsucesion $\{x_{k_i}\}$ convergente a x^*, y la subsucesión correspondiente $\{J_{k_i}^T J_{k_i} + \lambda_{k_i} D_{k_i}\}$ converge a una matriz definida positiva P, donde mantenemos la notación $J_{k_i} = J(x_{k_i})$ y $D_{k_i} = D(x_{k_i})$ matriz diagonal definida positiva, entonces $g(x^*) = 0$.*

Demostración. Por reducción al absurdo. Supongamos que $g(x^*) \neq 0$. Sea

$$p_{k_i} = -(J_{k_i}^T J_{k_i} + \lambda D_{k_i})^{-1} J_{k_i}^T r_{k_i}, \tag{3.16}$$

$$p^* := \lim p_{k_i} = -P^{-1} J(x^*)^T r(x^*), \tag{3.17}$$

con $r_{k_i} = r(x_{k_i})$. Notemos que $g(x^*)^T p^* < 0$. Sean $\beta \in (0,1)$, $\sigma \in (0, \frac{1}{2})$ y m^* el mínimo entero no negativo m tal que

$$f(x^* + \beta^m p^*) < f(x^*) + \sigma \beta^m g(x^*)^T p(x^*). \tag{3.18}$$

Por continuidad, para k suficientemente grande, tenemos que

$$f(x_{k_i} + \beta^{m^*} p_{k_i}) \leq f(x_{k_i}) + \sigma \beta^{m^*} g(x_{k_i})^T p_{k_i}. \tag{3.19}$$

Por tanto

$$f(x_{k_i+1}) = f(x_{k_i} + \beta^{m_{k_i}} p_{k_i}) \leq f(x_{k_i}) + \sigma \beta^{m^*} g(x_{k_i})^T p_{k_i}. \tag{3.20}$$

Tomando límites a ambos lados, teniendo en cuenta la monotonía del método tenemos que $\lim f(x_{k_i+1}) = \lim f(x_{k_i}) = f(x^*)$, tenemos que

$$f(x^*) \leq f(x^*) + \sigma \beta^{m^*} g(x^*)^T p^* < 0. \tag{3.21}$$

Lo cual no es posible puesto que $\sigma \beta^{m^*} g(x^*)^T p^* < 0$, por lo que tenemos una contradicción, lo que concluye la demostración. □

Esta teorema afirma la convergencia de una subsucesión, la cual usaremos para demostrar la convergencia global del método a continuación y después veremos su orden de convergencia.

Teorema 3.6. *Supongamos que se cumplen las siguientes afirmaciones:*

1. *El conjunto de nivel*

$$L(\bar{x}) = \{x \mid f(x) \leq f(\bar{x})\}$$

es cerrado y acotado para cualquier $\bar{x} \in \mathbb{R}^n$.

2. *El número de puntos estacionarios para los que sus valores de f iguales es finito.*

3. *$J^T J$ es definida positiva para todo x.*

4. *$\lambda_k \leq M \leq \infty$, $\forall k$, esto es, M es cota superior de λ_k.*

Entonces el método de Levenberg-Marquardt es globalmente convergente, es decir, para cualquier punto inicial la sucesión obtenida converge a un punto estacionario de f.

Demostración. La sucesión $\{x_k\}$ se encuentra en un conjunto compacto $L(\bar{x})$ por (1) y por su monotonía y, por tanto, tendrá puntos de acumulación. Basta comprobar que estos son únicos para concluír la demostración.

Por (3), (4) y por el teorema 3.5, cada punto de acumulación de $\{x_k\}$ es único. Como $\{f(x)\}$ es una sucesión monótona decreciente, $f(x)$ tiene el mismo valor en los puntos de acumulación de $\{x_k\}$. Además, por (2) hay solo un número finito de puntos de acumulación.

Para una subsucesión $\{x_{k_i}\}$, se tiene que $x_{k_i} \to \hat{x}_k$ y $\lim_{k\to\infty} g(x_{k_i}) = g(\hat{x}_k) = 0$. Además,

$$p(\lambda_{k_i}) = -(J(x_{k_i})^T J(x_{k_i}) + \lambda_{k_i} D(x_{k_i}))^{-1} g(x_{k_i}). \tag{3.22}$$

Se sigue entonces de (3) y (4) que $p(\lambda_{k_i}) \to 0$, y por lo tanto para la sucesión $\{p(\lambda_k)\}$, se cumple que $p(\lambda_k) \to 0$.

Supongamos que $\{x_k\}$ tiene más de un punto de acumulación. Sea ϵ^* la menor distancia entre dos puntos de acumulación, como $\{x_k\}$ es un conjunto compacto, existe un entero positivo N tal que para todo $k \geq N$, x_k está contenido en una bola cerrada con centro algún punto de acumulación y radio $\epsilon^*/4$.

Por otro lado, existe un entero $N' \geq N$ tal que

$$\|p(\lambda_k)\| < \frac{\epsilon^*}{4}, \forall k \geq N'. \tag{3.23}$$

Por tanto, cuando $k \geq N'$, todo x_k está en la bola cerrada descrita. Por tanto tenemos una contradicción. □

Teorema 3.7. *Supongamos la sucesión $\{x_k\}$ generada por el método de Levenberg-Marquardt convergente a x^* punto estacionario. Sean l el menor autovalor de $J(x^*)^T J(x^*)$, M el máximo en valor absoluto de los autovalores de $S(x^*) = \sum_{i=1}^m r_i(x^*)\nabla^2 r_i(x^*)$. Si se cumple*

$$\tau = \frac{M}{l} < 1,$$

$$0 < \beta < \frac{1-\tau}{2} \ y$$

$$\lambda_k \to 0,$$

entonces, para k suficientemente grande, la longitud de paso es $\alpha_k = 1$,

$$\limsup \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} \leq \tau \tag{3.24}$$

y x^* es mínimo local de f.

Para la demostración remitimos al lector a [8, Teorema 7.3.8]

Algoritmo 3.8 (Región de confianza de Levenberg-Marquardt)**.**

Paso 1. Fijamos x_0, $\bar{\Delta}$, $\Delta_0 \in (0, \bar{\Delta})$, $\epsilon \geq 0$, $0 < \eta_1 \leq \eta_2 < 1$ y $0 < \gamma_1 < 1 < \gamma_2$, $k := 0$.

Paso 2. Si $\|g_k\| = \left\|J_k^T r_k\right\| \leq \epsilon$, terminamos.

Paso 3. Resolvemos de forma aproximada el problema linealizado con restricciones (3.1) para obtener p_k.

Paso 4. Calcular

$$\rho_k = \frac{f(x_k) - f(x_k + p_k)}{f(x_k) - m_k(p_k)}.$$

Paso 5. Si $\rho_k < \eta_1$, $\Delta_k = \gamma_1 \Delta_k$ e ir a Paso 3.

Paso 6. Definimos $x_{k+1} = x_k + p_k$,

$$\Delta_{k+1} = \begin{cases} \min\{\gamma_2 \Delta_k, \bar{\Delta}\}, & \text{si } \rho_k \geq \eta_2 \text{ y } \|p_k\| = \Delta_k, \\ \Delta_k, & \text{en otro caso.} \end{cases}$$

Paso 7. $k := k + 1$ e ir a Paso 2. □

Como consecuencia del Teorema 1.23 enunciamos el siguiente resultado.

Teorema 3.9. *Supongamos que la función $f(x) = \frac{1}{2}\sum_{i=1}^{m} [r_i(x)]^2$ es dos veces continuamente diferenciable y que $\mathcal{L}(x_0) = \{x \,|\, f(x) \leq f(x_0)\}$ es un conjunto de nivel acotado. Si existen $M_1, M_2 > 0$ tales que*

$$\left\|\nabla^2 f(x)\right\| \leq M_1, \quad \forall x \in \mathcal{L}(x_0),$$
$$\left\|J(x)^T J(x)\right\| \leq M_2, \quad \forall x \in \mathcal{L}(x_0).$$

Entonces,

$$\lim_{k\to\infty} \nabla f(x_k) = \lim_{k\to\infty} J_k^T r_k = 0. \tag{3.25}$$

3.3. Implementación

En esta sección elegimos la implementación propuesta por Jorge J. Moré en [6], ya que otras implementaciones son poco robustas o su justificación teórica es débil. Por tanto, veremos sus fuertes propiedades de convergencia, robustez y eficiencia. Además veremos las principales propiedades de esta implementación.

A partir de la transformación del problema no lineal de mínimos cuadrados a uno linealizado con restricciones

$$\text{mín} \|r_k + J_k p\| \quad \text{sujeto a} \quad \|D_k p\| \leq \Delta_k, \tag{3.26}$$

Moré propone un método para obtener p resolviendo las ecuaciones correspondientes,

$$p(\lambda) = -\left(J_k^T J_k + \lambda D_k^T D_k\right)^{-1} J_k^T r_k. \tag{3.27}$$

Si J no es de rango completo y $\lambda = 0$, entonces las ecuaciones previas se definen mediante

$$D_k p(0) \equiv \lim_{\lambda \to 0^+} D_k p(\lambda) = -\left(J_k D_k^{-1}\right)^* r_k.^1 \tag{3.28}$$

Existen dos posibilidades, que $\lambda = 0$ y $\|D_k p(0)\| \leq \Delta_k$, en cuyo caso $p(0)$ es la solución de (3.26), donde se alcanza el mínimo de $\|D_k p\|$; o que $\lambda > 0$ y $\|D_k p(\lambda)\| = \Delta_k$, entonces $p(\lambda)$ es la única solución de (3.26). Llegamos así al siguiente algoritmo.

Algoritmo 3.10 (Algoritmo de Levenberg-Marquardt)**.**

Paso 1. Dado Δ_0, encontrar $\lambda_k \geq 0$ tal que

$$(J_k^T J_k + \lambda_k D_k^T D_k) p = -J_k^T r_k.$$

O bien $\lambda_k = 0$ y $\|D_k p_k\| \leq \Delta_k$, o bien $\lambda_k > 0$ y $\|D_k p_k\| = \Delta_k$.

Paso 2. Si $\|r(x_k + p_k)\| < \|r(x_k)\|$, entonces $x_{k+1} := x_k + p_k$ y calcular J_{k+1}, en otro caso $x_{k+1} := x_k$ y $J_{k+1} := J_k$.

Paso 3. Elegir Δ_{k+1} y D_{k+1}. □

Veamos ahora las propiedades que caracterizan la implementación de Moré, ya que para la resolución de los pasos del algoritmo previo existen varias posibilidades entre las que elegir.

3.3.1. Resolución del problema lineal

En la sección 2.1 mostramos 3 formas prácticas de solucionar el problema lineal (3.27). Veamos este caso particular, en el que Moré elige la factorización QR. Si bien es cierto que la resolución

[1] $A^* := Adj(A)$

de las ecuaciones normales

$$\left(J_k^T J_k + \lambda_k D_k^T D_k\right) p = -J_k^T r_k \tag{3.29}$$

mediante Cholesky puede ser más simple y rápida, la factorización QR es más robusta, particularmente cuando $\lambda = 0$ y J_k no es de rango completo, además de que la evaluación de $J_k^T J_k$ y $D_k^T D_k$ puede no ser adecuada. Por esto, resolvemos el siguiente problema

$$\begin{bmatrix} J_k \\ \lambda_k^{1/2} D_k \end{bmatrix} p \cong - \begin{bmatrix} r_k \\ 0 \end{bmatrix}, \tag{3.30}$$

por descomposición QR, problema que tiene por ecuaciones normales (3.29). Se procede en dos etapas, la primera consiste en descomponer J produciendo una matriz ortogonal Q y una permutación π de las columnas de J,

$$Q J_k \pi = \begin{bmatrix} T & S \\ 0 & 0 \end{bmatrix}, \tag{3.31}$$

donde T es una matriz triangular superior no singular con el mismo rango que J. Si $\lambda_k = 0$, el problema (3.30) se reduce a resolver el sistema según lo visto en la sección 2.1,

$$p = \pi \begin{bmatrix} T^{-1} & 0 \\ 0 & 0 \end{bmatrix} Q r_k \equiv J^- r_k, \tag{3.32}$$

donde J^- es la pseudo-inversa de J, es decir JJ^- es simétrica y $JJ^-J = J$. En el caso de que $\lambda_k > 0$, notamos que

$$\begin{bmatrix} Q & 0 \\ 0 & \pi^T \end{bmatrix} \begin{bmatrix} J_k \\ \lambda_k^{1/2} D_k \end{bmatrix} \pi = \begin{bmatrix} R \\ 0 \\ D_\lambda \end{bmatrix}, \tag{3.33}$$

donde R es una matriz triangular superior y $D_\lambda = \lambda^{1/2} \pi^T D_k \pi$ es una matriz diagonal.

El segundo paso es descomponer el lado derecho de (3.33) con el objetivo de eliminar la matriz D_λ. Se obtiene

$$W \begin{bmatrix} R \\ 0 \\ D_\lambda \end{bmatrix} = \begin{bmatrix} R_\lambda \\ 0 \end{bmatrix}, \tag{3.34}$$

donde W es la rotación de Givens y R_λ es una matriz triangular superior de orden n no singular. Entonces la solución a (3.30) es

$$p = -\pi R_\lambda^{-1} u, \tag{3.35}$$

con $u \in \mathbb{R}^n$ determinado por

$$W \begin{bmatrix} Q r_k \\ 0 \end{bmatrix} = \begin{bmatrix} u \\ v \end{bmatrix}. \tag{3.36}$$

Destacar que un cambio en λ solo afecta al segundo paso.

3.3.2. Actualización del radio Δ_k

Como ya adelantamos en la sección 1.3, es muy útil utilizar la proporción entre la reducción obtenida y la que se predice para cada paso. En este caso particular el ratio viene dado por

$$\rho = \frac{\|r(x_k)\|^2 - \|r(x_k + p_k)\|^2}{\|r(x_k)\|^2 - \|r(x_k) + J(x_k)p_k\|^2}. \tag{3.37}$$

Como la predicción se hace por el modelo lineal, en el caso de que $r(x)$ sea lineal, $\rho = 1$, y cuando $J(x_k)^T r(x_k) \neq 0$, cuanto menor sea el paso p_k, mayor será el ratio ρ. Para evitar errores numéricos por desbordamiento, multiplicamos en ambos lados de las ecuaciones normales (3.29) por $2p^T$, que resulta en

$$\|r_k\|^2 - \|r_k + J_k p\|^2 = \|J_k p\|^2 + 2\lambda_k \|D_k p\|^2 . \tag{3.38}$$

Sustituyendo en (3.37) obtenemos

$$\rho = \frac{1 - \left[\frac{\|r(x_k+p_k)\|}{\|r(x_k)\|}\right]^2}{\left[\frac{\|J_k p\|}{\|r(x_k)\|}\right]^2 + 2\left[\frac{\lambda^{1/2}\|D_k p\|}{\|r(x_k)\|}\right]^2}. \tag{3.39}$$

Es fácil ver de (3.38) que $\|J_k p\| \leq \|r_k\|$ y $\lambda^{1/2}\|D_k p\| \leq \|r_k\|$, por lo que esta vez la obtención de ρ no llevará a errores numéricos y además el denominador es no negativo. Mencionar que si $\|r(x_k + p_k)\| \gg \|r(x_k)\|$ sí que hay desbordamiento, pero como estamos interesados en valores no negativos de ρ, basta con tomar $\rho = 0$ y omitir el cálculo. Una vez calculado ρ, se actualiza el radio Δ_{k+1} según decidamos. Moré no especifica el factor utilizado para aumentar Δ_{k+1}, pero sí recomienda mantener el factor en el intervalo $[\frac{1}{10}, \frac{1}{2}]$.

3.3.3. El parámetro Levenberg-Marquardt

En esta implementación, se acepta $\alpha > 0$ como parámetro de Levenberg-Marquardt Si

$$|\phi(\alpha)| \leq \sigma\Delta, \tag{3.40}$$

siendo

$$\phi(\alpha) = \left\|D(J^T J + \alpha D^T D)^{-1} J^T r\right\| - \Delta, \tag{3.41}$$

y $\sigma \in (0, 1)$ es un parámetro que controla la aceptación de α. Si $\phi(\alpha) \leq 0$, el parámetro requerido es $\alpha = 0$, así que de aquí en adelante asumimos que $\phi(\alpha) > 0$. Por tanto, ϕ es una función decreciente para $\alpha \in [0, \infty)$ y $\phi(\alpha)$ tiende a $-\Delta$ en el infinito y, en consecuencia, existe un único $\alpha^* > 0$ tal que $\phi(\alpha^*) = 0$. Para determinar el parámetro Levenberg-Marquardt comenzamos con un $\alpha_0 > 0$ y calculamos la sucesión $\{a_k\}$ que converge a α^*.

Por ser ϕ convexa, parece conveniente utilizar el método de Newton para encontrar α^*, pero resulta muy poco eficiente. En su lugar, podemos implementar el método de Hebden [3] dado a la estructura particular del problema. Para ello, notemos que

$$\phi(\alpha) = \left\|(J^TJ + \alpha I)^{-1}J^Tr\right\| - \Delta, \quad \tilde{J} = JD^{-1}, \tag{3.42}$$

siendo $\tilde{J} = U\Sigma V^T$ la descomposición SVD de J. Por tanto,

$$\phi(\alpha) = \sum_{i=1}^{m}\left[\frac{\sigma_i^2 z_i^2}{(\sigma_i^2+\alpha)^2}\right]^{-1/2} - \Delta. \tag{3.43}$$

donde $z = U^Tr$ y σ_i son los valores singulares de J. En consecuencia, asumimos que

$$\phi(\alpha) \doteq \frac{a}{a+b} - \Delta \equiv \tilde{\phi}(\alpha), \tag{3.44}$$

y elegimos a y b de forma que $\tilde{\phi}(\alpha_k) = \phi(\alpha_k)$ y $\tilde{\phi}'(\alpha_k) = \phi'(\alpha_k)$. De esta forma, $\tilde{\phi}(\alpha_{k+1}) = 0$ si

$$\alpha_{k+1} = \alpha_k - \left[\frac{\phi(\alpha_k)+\Delta}{\Delta}\right]\left[\frac{\phi(\alpha_k)}{\phi'(\alpha_k)}\right]. \tag{3.45}$$

Para que este método converja, es necesario controlar las iteraciones, Hebden propuso tomar cotas inferiores y superiores l_k y u_k y obligar a que la diferencia entre iteraciones no sea mayor a $(u_k - l_k)/10$. Sin embargo, esta restricción puede ser muy dañina, ya que en muchos casos $u_k \gg l_k$, por lo que Moré solo modifica (3.45) si se sale del intervalo $[l_k, u_k]$. Partiendo de (3.42) podemos definir la cota superior

$$u_0 = \frac{\left\|(JD^{-1})^Tr\right\|}{\Delta}. \tag{3.46}$$

Si J es de rango completo, $\phi'(0)$ está definido y por la convexidad de ϕ definimos la cota inferior

$$l_0 = -\frac{\phi(0)}{\phi'(0)}. \tag{3.47}$$

En otro caso, $l_0 = 0$. Escribamos un esquema algoritmo con los parámetros que hemos explicado. Normalmente se toma un valor no muy pequeño de σ, por ejemplo 0,1.

Algoritmo 3.11 (Parámetro Levenberg-Marquardt)**.**

Paso 1. Definir valores iniciales α_0, l_0 y u_0. Definir coeficiente de parada σ.

Paso 2. Si $\alpha_k \notin (l_k, u_k)$, definir $\alpha_k = \max\{0{,}001u_k, (l_ku_k)^{1/2}\}$.

Paso 3. Calcular $\phi(\alpha_k)$ y $\phi'(\alpha_k)$. Si $|\phi(\alpha)| < \sigma\Delta$, terminar.

Paso 4. Actualizar l_k y u_k:

$$l_{k+1} = \max\{l_k, \alpha_k - \frac{\phi(\alpha_k)}{\phi'(\alpha_k)}\},$$

$$u_{k+1} = \begin{cases} \alpha_k, & \textit{si } \phi(\alpha_k) < 0, \\ u_k, & \textit{en otro caso.} \end{cases}$$

Paso 5. Actualizar α_{k+1} según (3.45). Volver al Paso 2 □

En el Paso 1, se elige α_k con tendencia a la cota inferior, se añade el factor $0{,}001u_k$ para contrarrestar los casos en los que l_k es muy pequeño, o en particular $l_k = 0$. No es dificil probar la convergencia de este algoritmo, y en la práctica se observa que converge en muy pocos pasos, de media en menos de 2 se cumple (3.40) si $\sigma = 0{,}1$.

Por completitud, mostrar que

$$\phi'(\alpha) = -\left\|q(\alpha)\right\| \left\| (R_\alpha^{-1})^T \left[\frac{\pi^T D^T q(\alpha)}{\|q(\alpha)\|} \right] \right\|^2, \tag{3.48}$$

con $q(\alpha) = Dp(\alpha)$ y p definido por (3.27) y π permutación de la factorización QR.

3.3.4. Escalado

En el algoritmo de Levenberg-Marquardt se utiliza una matriz diagonal D_k para reducir los problemas de escalado. Algunos autores eligen

$$D_k = diag(d_1^{(k)}, \ldots, d_n^{(k)}), \tag{3.49}$$

con

$$d_i^{(k)} = \left\|\partial_i r(x_k)\right\|, k \geq 0. \tag{3.50}$$

Siempre que $\|\partial_i r(x_k)\|$ no crezca respecto a k, esta elección es adecuada. En caso contrario, se ve afectada la efectividad del algoritmo y por esta razón se propone la elección

$$\begin{aligned} d_i^{(0)} &= \left\|\partial_i r(x_0)\right\|, \\ d_i^{(k)} &= \text{máx}\{d_i^{(k-1)}, \|\partial_i r(x_k)\|\}, k \geq 1. \end{aligned} \tag{3.51}$$

Entre elecciones posibles, siempre que D sea diagonal con elementos positivos, el algoritmo genera las mismas iteraciones tanto si aplicamos el algoritmo a $r(x)$ empezando en x_0 como si aplicamos el algoritmo a $r(D^{-1}x)$ empezando en Dx_0. Por esta razón se dice que el algoritmo es invariante al escalado. Para decir esto se asume que la elección de Δ se basa únicamente en (3.37), lo que mantiene la invarianza.

3.3.5. La versión de Moré

Concluyamos este capítulo presentando la versión propuesta por Moré del algoritmo de Levenberg-Marquardt.

Algoritmo 3.12 (Versión de Moré)**.**

Paso 1. Definir valores iniciales x_0, Δ_0 *y* ϵ *y determinar* J_k, D_k, *y* $f(x_0)$ *a partir de la función* f *determinada.*

Paso 2. Si $\left\|J_k(D_k)^{-1}f(x_k)\right\| < \epsilon$, *terminar.*

Paso 3. Sea $\sigma \in (0,1)$. *Si* $\left\|D_k J_k^- r_k\right\| \leq (1+\sigma)\Delta_k$, *asignar* $\lambda_k = 0$ *y* $p_k = -J_k^- r_k$. *En otro caso, encontar* λ_k *tal que si*

$$\begin{bmatrix} J_k \\ \lambda_k^{1/2} D_k \end{bmatrix} p_k \cong - \begin{bmatrix} r_k \\ 0 \end{bmatrix}, \tag{3.52}$$

entonces

$$(1-\sigma)\Delta_k \leq \|D_k p_k\| \leq (1+\sigma)\Delta_k. \tag{3.53}$$

Paso 4. Determinar la relación ρ_k *entre reducción real y la esperada.*

Paso 5. Si $\rho_k \leq 0{,}0001$, *asignar* $x_{k+1} = x_k$ *y* $J_{k+1} = J_k$.
Si $\rho_k > 0{,}0001$, *asignar* $x_{k+1} = x_k + p_k$ *y calcular* $J_{k+1} = J(x_{k+1})$.

Paso 6. Si $\rho_k \leq \frac{1}{4}$, *asignar* $\Delta_{k+1} \in \left[\frac{1}{10}\Delta_k, \frac{1}{4}\Delta_k\right]$.
Si $\rho_k \in \left[\frac{1}{4}, \frac{3}{4}\right]$ *y* $\lambda_k = 0$, *o si* $\rho_k \geq \frac{3}{4}$, *asignar* $\Delta_{k+1} = 2\|D_k p_k\|$.

Paso 7. Actualizar D_k *por (3.49) y (3.51) y volver al Paso 2.*

Teorema 3.13 (Convergencia de la versión de Moré)**.** *Sea* $r : \mathbb{R}^n \to \mathbb{R}^m$ *una función continuamente diferenciable y sea* $\{x_k\}$ *generada por el algoritmo 3.12. Entonces,*

$$\liminf_{k\to+\infty} \left\|(J_k D_k^{-1})^T r_k\right\| = 0. \tag{3.54}$$

Lo que garantiza que el gradiente escalado será lo suficientemente pequeño. Si además $\{J_k\}$ *es acotada, esto implica que*

$$\liminf_{k\to+\infty} \left\|J_k^T r_k\right\| = 0. \tag{3.55}$$

Por último, si $\nabla r(x)$ *es uniformemente continua, entonces*

$$\lim_{k\to+\infty} \left\|J_k^T r_k\right\| = 0. \tag{3.56}$$

Capítulo 4

Implementación en Matlab®

4.1. Tests de funcionamiento

Después del análisis teórico, pasamos a la implementación del algoritmo en Matlab®. Este programa cuenta con el comando `lsqnonlin` [1] que permite ser configurado para que utilice el algoritmo de Levenberg-Marquardt:

```
opts = optimoptions(@lsqnonlin,'Algorithm','levenberg-marquardt');
x = lsqnonlin(fun,x0,[],[],opts);
```

Donde `fun` es la función objetivo escrita como *function handle*, `x0` es el punto inicial y `opts` son las opciones que hemos configurado para que se use el algoritmo de Levenberg-Marquardt. En este caso, no se ha configurado nada más, por lo que se usan los valores por defecto. Los argumentos vacíos `[]` indican que no hay restricciones en las variables, es decir, que no hay límites inferior ni superior a la hora de buscar parámetros. De aquí en adelante, se asume que se ha configurado el comando `lsqnonlin` para que utilice el algoritmo de Levenberg-Marquardt.

Podemos realizar otras configuraciones para que se nos muestren más información. Por ejemplo, la figura 4.1 es la salida resultante de utilizar la función resiudo

$$f(x) = \begin{pmatrix} 10(x_3 - 10\,\theta(x_1, x_2)) \\ 10((x_1^2 + x_2^2)^{1/2} - 1) \\ x_3 \end{pmatrix}, \tag{4.1}$$

donde

$$\theta(x_1, x_2) = \begin{cases} \frac{1}{2\pi} arctan\left(\frac{x_2}{x_1}\right), & \text{si } x_1 > 0 \\ \frac{1}{2\pi} arctan\left(\frac{x_2}{x_1}\right) + \frac{1}{2}, & \text{si } x_1 < 0, \end{cases} \tag{4.2}$$

```
>> options = optimoptions(@lsqnonlin,'Algorithm','levenberg-marquardt','Display','iter');
d = linspace(-2,2);
fun = @(x) [10*(x(3)-10*t(x(1),x(2))),10*((x(1)^2+x(2)^2)^(1/2)-1),x(3)];
x0=[-1,0,0];
[x,resnorm,residual,exitflag,output] = lsqnonlin(fun,x0,[],[],options);
```

Iteration	Func-count	Residual	First-Order optimality	Lambda	Norm of step
0	4	2500	796	0.01	
1	8	1405.95	297	0.001	3.1293
2	14	1259.26	306	0.1	5.05482
3	18	134.073	113	0.01	3.78513
4	22	9.48967	38.3	0.001	0.936633
5	26	0.177279	5.99	0.0001	0.248342
6	30	5.42548e-05	0.107	1e-05	0.0302099
7	34	4.72921e-12	3.15e-05	1e-06	0.000515594
8	38	4.00314e-26	2.67e-12	1e-07	1.52101e-07

```
Local minimum found.

Optimization completed because the size of the gradient is less than
1e-4 times the value of the function tolerance.

<stopping criteria details>
```

Figura 4.1: Ejemplo de ejecución del comando `lsqnonlin` mostrando información sobre las iteraciones.

y $x_0 = (-1, 0, 0)^T$.

Como vemos en la figura, se muestra información para cada iteración, donde `Func-count` es el número de evaluaciones de la función, `Resiudal` es el valor de la función objetivo, `First-Order optimality` es una medida de optimalidad de primer orden, `lambda` hace referencia a los multiplicadores de Lagrange y `Norm of step` es la norma del paso. Además, en la salida se guardan `[x,resnorm,residual,exitflag,output]`, que en orden son el punto final, la norma del valor de la función objetivo en el punto final, el valor de la función objetivo en el punto final, el código de salida y la información de salida, para más información ver [1].

Con el objetivo de entender mejor el funcionamiento del algoritmo y ponerlo en práctica, se ha implementado una versión simplificada de la descrita en el capítulo anterior con el propósito de compararla con el comando `lsqnonlin`. De aquí en adelante, cuando mostremos el comando `myLevMar` nos referimos a esta función, proporcionada en el Anexo I que toma como argumentos `fun` y `x0`.

Para comenzar, veamos el ejemplo ya propuesto. El interés de este es la discontinuidad en el plano $x_1 = 0$, el cual se tiene que cruzar ya que definimos $x_0 = (-1, 0, 0)$. En $x = (1, 0, 0)$ la suma de residuos vale 0.

```
fun = @(x) [10*(x(3)-10*theta(x(1),x(2))),10*((x(1)^2+x(2)^2)^(1/2)
    -1),x(3)];
```

```
myX = myLevMar(fun,x0);
x = lsqnonlin(fun,x0,[],[],options);
```

La función `theta()` es la definida según (4.2):

```
function y = theta(x1,x2)
if x1 > 0
        y = 1/(2*pi)*atan(x2/x1);
else
        y = 1/(2*pi)*atan(x2/x1) + 0.5;
end
```

El resultado de la ejecución es el mismo para ambos, $x = (1, 0, 0)$. Para comprobar que la implementación es correcta, la probamos ahora con otros ejemplos. El siguiente código es un ejemplo de la documentación de `lsqnonlin` [1] en el que se genera una muestra con un error aleatorio normalmente distribuído de media 0. Este ejemplo es interesante porque la matriz jacobiana no es cuadrada como en el primer caso y nos sirve para ver que la implementación es funciona correctamente. El resultado se muestra en la figura 4.2, también se muestran las curvas de e^{-xt} con $x = 2, 3, 4$ para ver mejor el efecto del ajuste.

```
rng default % para que sea reproducible
d = linspace(0,3);
y = exp(-1.3*d) + 0.05*randn(size(d));
fun = @(r)exp(-d*r)-y;
x0 = 4;
x = myLevMar(fun,x0);
plot(d,y,'ko',d,exp(-x*d),'b-',d,exp(-4*d),'r:',d,exp(-3*d),'r:',d,
   exp(-2*d),'r:')
legend('Datos','Mejor ajuste','exp(-x*d) con x=2,3,4')
xlabel('t')
ylabel('exp(-tx)')
```

Este es un ejemplo de estimación de parámetros en un modelo de regresión. Se parte de unos datos muestrales y una función paramétrica que se cree que puede encajar con los datos, normalmente esta función se puede obtener por el contexto o, en su defecto, con alguna técnica de inferencia. En este caso se trata de la función $f(x) = e^{-xt}$, siendo x el parámetro que queremos estimar. Para ello, definimos la función residuo $r(x) = e^{-xt} - y$, siendo y los datos muestrales.

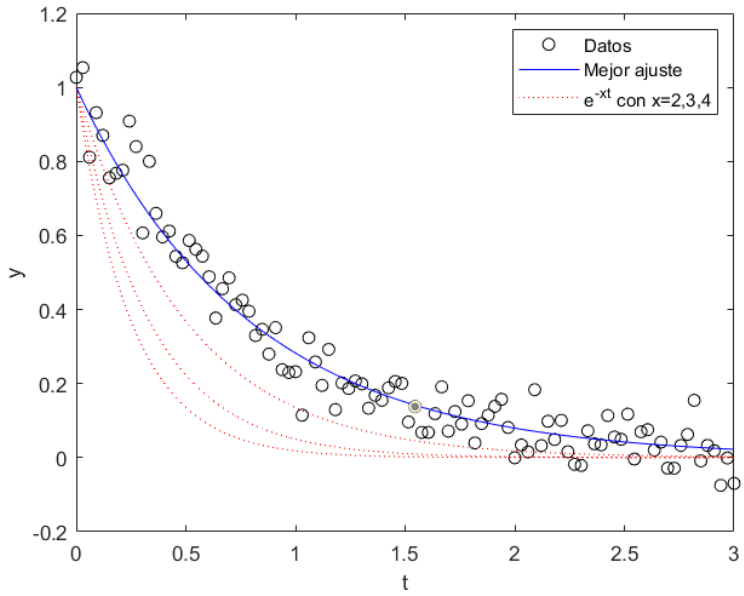

Figura 4.2: Resultado del ejemplo de ajuste exponencial.

4.2. Análisis de eficiencia

Una vez comprobado que funciona correctamente, veamos ahora el tiempo de ejecución necesario. Para poder analizar con criterio, lo pondremos frente al propio algoritmo de matlab `lsqnonlin`. Aprovecharemos los mismos ejemplos que hemos visto en la sección anterior. Para cada ejecución del siguiente código la salida cambia, seguramente debido a cache almacenado u otras herramientas del programa

```
clear all
rng default % para que sea reproducible
d = linspace(0,3);
y = exp(-1.3*d) + 0.05*randn(size(d));
fun = @(r)exp(-d*r)-y;
x0 = 4;
tic
x = myLevMar(fun,x0);
toc
opts = optimoptions(@lsqnonlin,'Algorithm','levenberg-marquardt');
tic
x = lsqnonlin(fun,x0,[],[],opts);
toc
```

En este caso se ha probado con la prueba aleatoriezada, la salida después del comando `myLevMar` ha sido "*Elapsed time is 0.104180 seconds*", mientras que para el comando `lsqnonlin` ha sido "*Elapsed time is 0.107986 seconds*". Aunque en esta ejecución la diferencia de tiempo es poco notable, siempre era más rápida la implementación `myLevMar`. El motivo es que se trata de una implementación más sencilla, y seguramente no sea tan robusta como la de `lsqnonlin` en casos más críticos que se escapan del estudio. Veamos los tiempos de ejecución del primer ejemplo mostrado.

```
1  d = linspace(-2,2);
2  fun = @(x) [10*(x(3)-10*theta(x(1),x(2))),10*((x(1)^2+x(2)^2)^(1/2)
       -1),x(3)];
3  x0 = [-1,0,0];
4  tic
5  x = myLevMar(fun,x0);
6  toc
7  opts = optimoptions(@lsqnonlin,'Algorithm','levenberg-marquardt');
8  tic
9  x = lsqnonlin(fun,x0,[],[],opts);
10 toc
```

En la primera ejecución de este caso, los resultados son contrarios a los anteriores, siendo la implementación de `lsqnonlin` más rápida que la de `myLevMar`. Mostrando tiempos de `0.095620` y `0.143244` respectivamente. En la segunda ejecución los resultados ya se ven más cercanos pero sigue siendo más rápido el primero. Para nuestros objetivos y seguramente la mayoría de los casos prácticos en los que se utilice, esta diferencia no supone ningún coste. Por el contrario, se puede requerir la máxima eficiencia posible en casos de ajuste de grandes cantidades de parámetros como puede ser el caso de las redes neuronales.

Las debilidades principales de esta implemetación son el cálculo explícito de la matriz jacobiana, el cálculo directo de cada matriz inversa necesaria en el algoritmo, como no eran el objetivo principal de estudio, estas se calcularon directamente eligiendo la sencillez a la eficiencia en este caso. Sin embargo, los resultados no son muy distantes al comando principal de Matlab®, por lo retocando pequeños detalles se podría conseguir un algoritmo muy eficiente, ya que en terminos teóricos ya lo es. Con todo, se puede considerar una implementación válida para casos de uso prácticos.

Comparemos ahora iteraciones, `lsqnonlin` itera 8 veces en el caso anterior, con 38 evaluaciones de f, mientras que `myLevMar` itera 17, la diferencia es que estas iteraciones son más reales pues cada iteración evalúa una vez, pero no se cuentan las veces en las que no se actualiza x_k.

Anexo I

Anexo: Código de la versión simplificada

Se muestra el código de Matlab® para la implementación del algoritmo de Moré simplificado. Se presenta en forma de función `myLevMar` que toma como argumentos `fun` en forma de *function handle* y `x0` el punto de partida. Se muestran las distintas funciones creadas para un código más limpio una tras otra. Para mayor comodidad, los archivos `.m` se pueden descargar desde el repositorio de Github `https://github.com/didacwhite/levenberg-marquardt` en la carpeta implementación.

```
1  function [xk, iter] = myLevMar(fun, x0)
2  % setup
3  [n,m,xk,Delta,funk,lambdak, maxit, maxit2, eps]= myLevMarSetup(fun,
       x0);
4
5  % calculo inicial de Jk y Dk
6  dkm1=zeros(length(xk));
7  [Jk, Dk, dkm1] = JDk(fun,xk,dkm1);
8
9  try
10     test = norm((Jk*inv(Dk))'*funk);
11 catch
12     test = norm((Jk*inv(Dk))'*funk');
13 end
14 iter=0;
15 while test > eps && iter < maxit
16     % solucion problema lineal
```

```
17      % descomposicion QR
18      [Q,R,Pi] = qr(Jk); % calcula Jk*Pi = Q*R
19      % ajuste de R
20      T = R(1:n,1:n);
21      S = R(1:n,n+1:end);
22
23      % matriz $J^{-}$
24      Jm = Jmfun(Pi,T,Q,n,m);
25
26      % parametro levenberg-marquardt
27      [pk, lambdak] = pkfun(Delta, Jk, Dk, funk, Jm, Q, Pi, n, m,
            lambdak, maxit2);
28
29      % 2. calculo de rho
30      rho = rhofun(funk,pk,lambdak,Dk,Jk,fun, xk);
31
32      % actualizacion de xk
33      if rho <= 0.0001
34          % xk+1 = xk y jk+1=jk
35          % continue
36      else
37          xk = xk+pk;
38          % calculo de Jk, Dk con el nuevo xk
39          funk = fun(xk)';
40          [Jk, Dk, dkm1] = JDk(fun,xk,dkm1);
41          iter=iter+1;
42          test = norm((Jk*inv(Dk))'*fun(xk)');
43      end
44
45      % actualizacion de Delta
46      if rho <= 0.25
47          Delta = Delta/2;
48      elseif rho >= 0.75
49          Delta = 2*norm(Dk*pk');
50      elseif lambdak == 0
51          Delta = 2*norm(Dk*pk');
52      end
53  end
```

```
54 xk = real(xk);
```

```
1  function [n,m,xk,Delta,funk,lambdak, maxit, maxit2, eps] =
       myLevMarSetup(fun,x0)
2  % editable
3  Delta = 3;
4  lambdak = 1;
5  maxit = 50;
6  maxit2 = 10; % miteraciones maximas en hebden.m
7  eps = 0.001;
8
9  % no editable
10 n = length(x0);
11 m = length(fun(x0));
12 xk = x0;
13 funk = fun(xk)';
```

```
1  function [Jk,Dk,v] = JDk(fun,xk,dkm1)
2
3  Jk = jacob(fun,xk);
4
5  % Compute diagonal of Dk
6  di = @(i) max(dkm1(i),norm(Jk(:,i)));
7
8  v = arrayfun(di, 1:length(xk));
9  Dk = diag(v);
```

```
1  function J = jacob(fun,xk)
2  h = 1e-5;
3  n = length(xk);
4  m = length(fun(xk));
5  J = zeros(m,n);
6  for j = 1:n
7      hj = zeros(1,n);
8      hj(j) = h;
9      %fun(xk+hj)
10     %fun(xk)
11     J(:,j) = (fun(xk+hj)-fun(xk))./h;
```

```
end
```

```
function Jm = Jmfun(Pi,T,Q,n,m)
Tm1 = zeros(m,m);
Tm1(1:n,1:n)=inv(T);
Jm = Pi*Tm1*Q';
```

```
function [pk, lambdak] = pkfun(Delta, Jk, Dk, funk, Jm, Q, Pi, n, m
    , lambdak, maxit)

phi = @(a) norm(Dk*inv(Jk'*Jk+a.*(Dk'*Dk))*Jk'*funk) - Delta;
if phi(0) == 0
    lambdak = 0;
    pk = -Jm*funk;
    pk = pk';
else
    lambdak = hebden(phi, Delta, Jk, Dk, funk, Q, Pi, n, m, lambdak
        , maxit);
    pk = pk2(funk, Q, Pi, Jk, Dk, lambdak, n, m);
end
```

```
function lambdak = hebden(phi, Delta, Jk, Dk, funk, Q, Pi, n, m,
   lambdak, maxit)
ak=lambdak;

Qpi = zeros(n+m, n+m);
Qpi(1:m,1:m)=Q';
Qpi(m+1:end,m+1:end)=Pi';
Jizq = zeros(n+m,n);
Jizq(1:m,1:n) = Jk;
Jizq(m+1:end,1:n) = sqrt(ak)*Dk;

dcha = Qpi*Jizq*Pi;

[W,Ra] = qr(dcha);
W = W';
Ra = Ra(1:n,1:n);

```

```
p = @(a) -inv(Jk'*Jk+a*Dk'*Dk)*Jk'*funk;
q = @(a) Dk*p(a);
fac = @(a) (Pi'*Dk'*q(a))/(norm(q(a)));
dphi = @(a) -norm(q(a))*norm(inv(Ra)'*(fac(a)))^2;

uk = norm((Jk*inv(Dk))'*funk)/Delta;
try
    lk = -phi(0)/dphi(0);
catch
    lk = 0;
end
phiak = phi(ak);
k=0;
while abs(phiak)>0.1*Delta && k<maxit
    if ak <= lk || ak >= uk
        ak = max(0.001*uk,sqrt(lk*uk));
    end
    phiak = phi(ak);
    dphiak = dphi(ak);
    if phiak < 0
        uk = ak;
    end
    lk = max(lk, ak-(phiak/dphiak));
    ak = ak-((phiak+Delta)/Delta)*(phiak/dphiak);
    k = k+1;
end
%k
lambdak = ak;
```

```
function pk = pk2(funk, Q, Pi, Jk, Dk, lambdak, n, m)
Qpi = zeros(n+m, n+m);
Qpi(1:m,1:m)=Q';
Qpi(m+1:end,m+1:end)=Pi';
Jizq = zeros(n+m,n);
Jizq(1:m,1:n) = Jk;
Jizq(m+1:end,1:n) = sqrt(lambdak)*Dk;

dcha = Qpi*Jizq*Pi;
```

```
10
11 [W,Rlambda] = qr(dcha);
12 W = W';
13 Rlambda = Rlambda(1:n,1:n);
14 Qf0 = zeros(1,size(W,1));
15 Qfunk = Q'*funk;
16
17 Qf0(1:length(Qfunk)) = Qfunk;
18
19 uv= W*Qf0';
20 u = uv(1:n);
21 pk = -Pi*inv(Rlambda)*u;
22 pk = pk';
```

```
1 function rho = rhofun(funk,pk,lambdak,Dk,Jk,fun,xk)
2 funplus = fun(xk+pk);
3
4 rhonum=1-(norm(funplus)/norm(funk))^2;
5 rhoden=(norm(Jk*pk')/norm(funk))^2+2*(sqrt(lambdak)*norm(Dk*pk')/
     norm(funk))^2;
6 rho = rhonum/rhoden;
```

Bibliografía

[1] *Resolver problemas de mínimos cuadrados no lineales (ajuste de datos no lineales) - MATLAB lsqnonlin - MathWorks españa*, `https://es.mathworks.com/help/optim/ug/lsqnonlin.html`, Accessed: 2023-1-23.

[2] Gene H Golub and Charles F Van Loan, *Matrix computations*, 4 ed., Johns Hopkins University Press, Baltimore, MD, 2013 (en).

[3] M. D. Hebden, *An algorithm for minimization using exact second derivatives*, 1973.

[4] Kenneth Levenberg, *A method for the solution of certain non-linear problems in least squares*, Quart. Appl. Math. **2** (1944), no. 2, 164–168 (en).

[5] Donald W Marquardt, *An algorithm for least-squares estimation of nonlinear parameters*, J. Soc. Ind. Appl. Math. **11** (1963), no. 2, 431–441.

[6] Jorge J Moré, *The Levenberg-Marquardt algorithm: Implementation and theory*, Lecture Notes in Mathematics, Springer Berlin Heidelberg, Berlin, Heidelberg, 1978, pp. 105–116.

[7] Jorge Nocedal and Stephen Wright, *Numerical optimization*, 2 ed., Springer, New York, NY, 2006 (en).

[8] Wenyu Sun and Ya-Xiang Yuan, *Optimization theory and methods: Nonlinear programming*, 2006 ed., Springer, New York, NY, 2006 (en).

[9] Lloyd N (lloyd Nicholas) Trefethen, *Numerical linear algebra*, Society for Industrial and Applied Mathematics, Philadelphia, MS, 1997 (en).