Métodos Numéricos

Departamento de Matemática Aplicada II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Lección 5: Ecuaciones y sistemas de ecuaciones no lineales. Optimización

Índice

El cálculo de las raíces o ceros de una función, o de cierto conjunto de funciones, es un problema que aparece reiteradamente en la computación científica. Por otro lado, solamente en un número muy reducido de casos existen fórmulas que permiten expresar simbólicamente tales ceros. Desde un punto de vista computacional, la estrategia para resolver este tipo de problemas consiste en diseñar algoritmos que generen sucesiones de valores que se aproximen tanto como uno desee a dichas raíces. El clásico método de Newton desempeña un papel fundamental en el diseño de tales algoritmos. En el caso de una única función, MATLAB® incorpora la orden fzero para el cálculo de sus raíces. Esta orden descansa en última instancia en los métodos de bisección y Newton. Asimismo, para calcular ceros de conjuntos de funciones, MATLAB® proporciona la orden fsolve. La extensión del método de Newton a varias variables desempeña un papel importante en este contexto.

1. Ecuaciones en una variable

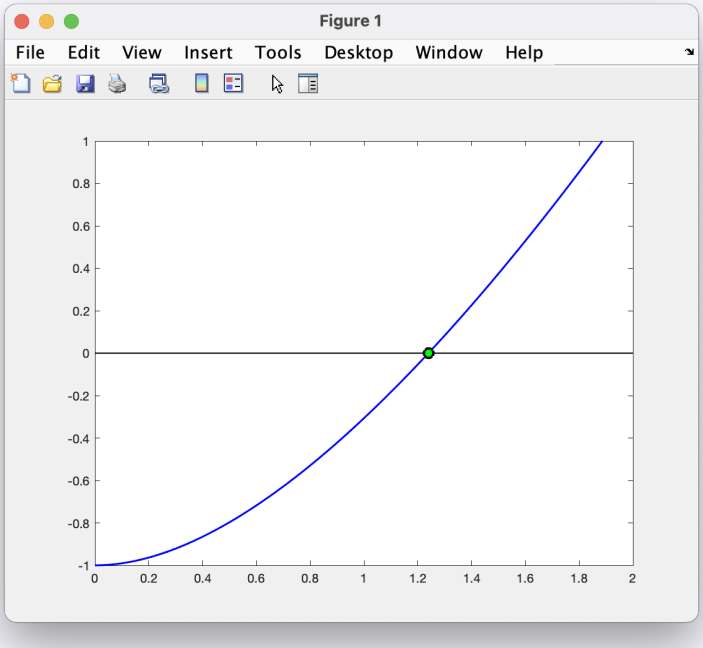
Una raíz o cero de una función $f : \mathbb{R} \to \mathbb{R}$ no lineal es cualquier número real α tal que $f(\alpha) = 0$. Se llama ecuación no lineal en una variable a la expresión $f(x) = 0$. Diremos que α es una solución[1] de esta ecuación.

Ejemplo 1.1 ▶ Ecuación no lineal en una variable

La ecuación no lineal

$$x \log(1 + x) - 1 = 0$$

tiene una única solución $\alpha \approx 1.24$ en el intervalo $[0, 2]$.



La función $f(x) = x \log(1 + x) - 1$ tiene a α como raíz o cero.

[1] O cero o raíz.

1.1. El método de bisección

Consideremos una función continua

$$f : [a, b] \to \mathbb{R}$$

tal que

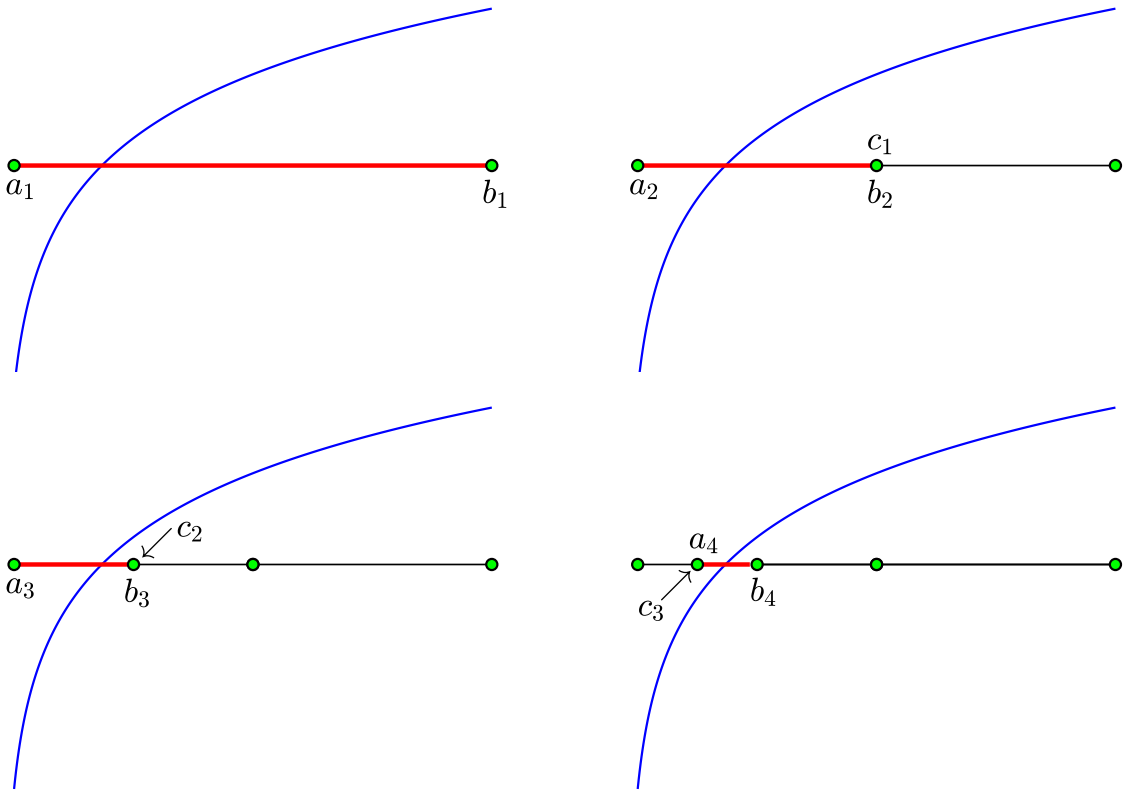
$$f(a)f(b) < 0.$$

El teorema de Bolzano garantiza que f tiene al menos un cero en (a, b). La estrategia del método de bisección consiste en dividir reiteradamente el intervalo por la mitad y seleccionar aquel subintervalo donde la función cambia de signo. En concreto, si $[a_1, b_1] := [a, b]$ y $c_1 := \frac{a+b}{2}$, pueden darse dos casos:

1. $f(c_1) = 0$. En este caso paramos, puesto que c_1 es una raíz de f.

2. $f(c_1) \neq 0$. En este caso elegimos el intervalo, $[a_1, c_1]$ o $[c_1, b_1]$, en cuyos extremos la función f toma signos opuestos.

Y así sucesivamente. Por tanto, o bien el proceso anterior llega a una raíz de f en un número finito de pasos o se genera una sucesión $\{I_n\}$ de subintervalos encajados de $[a, b]$ cuyas longitudes tienden a cero, de modo que cada subintervalo I_n contiene (al menos) un cero de f.



Método de bisección

1.2. El método de Newton en una variable

Sea α una raíz de cierta función $f : \mathbb{R} \to \mathbb{R}$ derivable de todo orden. Supongamos que, de alguna manera, hemos obtenido un punto x_0 que, presumiblemente, es una aproximación a la raíz α. Como nuestro planteamiento es determinar con gran precisión α, parece razonable buscar una «corrección» $x_0 + \xi_0$, de modo que el nuevo punto $x_0 + \xi_0$ aproxime mejor la raíz α que el punto inicial x_0. Lo ideal sería tener $f(x_0 + \xi_0) = 0$, pero esto es, usualmente, falso.

Sin embargo, recordando el teorema de Taylor, se tiene que, siempre que ξ_0 sea suficientemente pequeño,

$$f(x_0 + \xi_0) \approx f(x_0) + f'(x_0)\xi_0.$$

Por tanto, parece lógico imponer

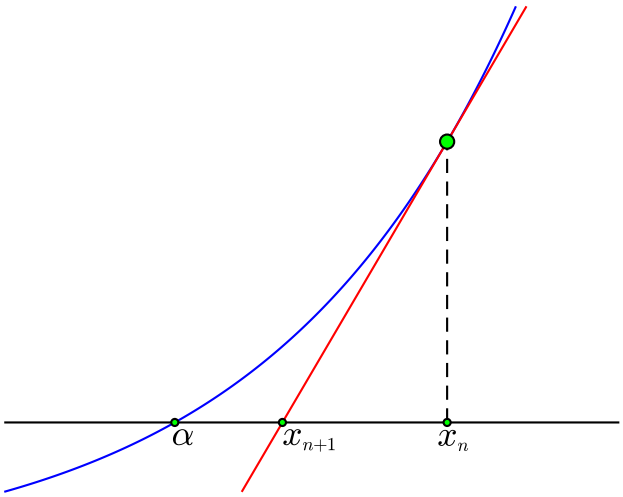
$$f(x_0) + f'(x_0)\xi_0 = 0,$$

despejar ξ_0 y, de este modo, considerar la «corrección»

$$x_1 := x_0 + \xi_0 = x_0 - \frac{f(x_0)}{f'(x_0)}. \tag{1}$$

Esta es la idea matriz del método de Newton.

Cierta experimentación gráfica puede ayudar a convencernos de que la estrategia anterior es bastante razonable. Es más, nos hace ver que el método de Newton tiene una interpretación geométrica simple: en el punto $(x_0, f(x_0))$, perteneciente a la gráfica de f, se dibuja la correspondiente recta tangente; se obtiene el corte de dicha recta con el eje OX y este punto es la corrección $x_0 + \xi_0$ [2].



Método de Newton

Repitiendo el proceso una vez y otra vez, obtenemos una sucesión $\{x_n\}$ definida por la recurrencia

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n \geqslant 0,$$

que, previsiblemente, debe tender a α.

[2] El corte de la recta tangente $y = f(x_0) + f'(x_0)(x - x_0)$ con $y = 0$ es el punto $x = x_0 - \frac{f(x_0)}{f'(x_0)}$, es decir, la corrección $x_0 + \xi_0$.

En general, el método de Newton sólo funciona localmente. En concreto, si $\alpha \in (a, b)$ es una raíz simple de f (esto es, $f'(\alpha) \neq 0$), existe un número $\delta > 0$ tal que, partiendo de cualquier $x_0 \in (-\delta + \alpha, \delta + \alpha)$, el método de Newton genera una sucesión $\{x_n\}$ convergente a α. Es más, esta convergencia es cuadrática[3], es decir, existe $C > 0$ tal que, para n suficientemente grande,

$$|x_{n+1} - \alpha| \approx C|x_n - \alpha|^2,$$

esto es, el error en una iteración es aproximadamente el cuadrado del error de la iteración anterior. En la práctica, esto se traduce en que, de una iteración a la siguiente, se duplica el número de dígitos significativos correctos en la aproximación[4].

Ejercicio 1.1

Diseña una función en MATLAB® que implemente el método de Newton para resolver la ecuación $f(x) = 0$ y muestre las n primeras iteradas. Los argumentos de entrada deben ser la función f, su derivada, el punto inicial x_0 y el número n.

Utiliza dicha función, con punto inicial $x_0 = 0.2$ y n a elegir, para aproximar seis cifras significativas correctas de la solución positiva de la ecuación

$$\frac{1}{1 + x^2} = 0.3.$$

1.3. El método de la secante y el método IQI

Una desventaja importante del método de Newton, desde un punto de vista computacional, es que requiere el cálculo de la derivada de f. En muchas ocasiones, esto no es posible o, simplemente, es demasiado costoso. Para evitar este problema, los códigos no suelen usar propiamente el método de Newton, sino ciertas variantes de este método, en concreto el método de la secante y/o el método de interpolación cuadrática inversa (en inglés, IQI, de *Inverse Quadratic Interpolation*).

El método de la secante es una variante del método de Newton que se basa en sustituir la recta tangente por la recta secante generada por f y los puntos x_n y x_{n-1} y definir x_{n+1} como el corte de dicha recta con el eje OX. En otras palabras, x_{n+1} se obtiene a partir del corte del eje OX con el interpolante lineal generado por f y los puntos x_n y x_{n-1}.

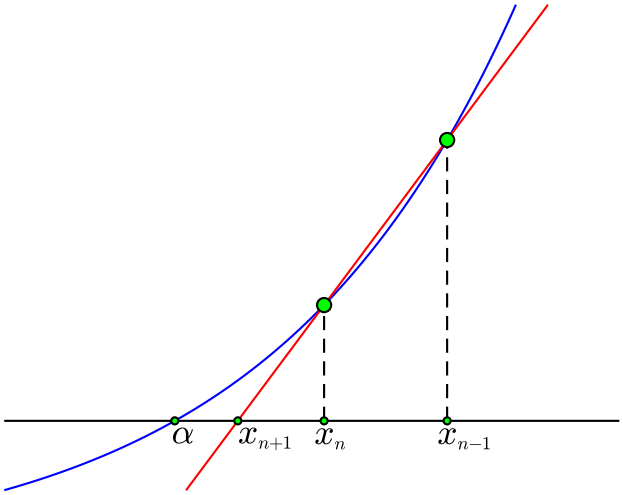
En concreto, dados $x_0, x_0^* \in [a, b]$, con $x_0 \neq x_0^*$, este método se basa en la siguiente recurrencia:

$$\begin{cases} x_1 = x_0, \\ x_2 = x_0^*, \\ x_{n+1} = x_n - f(x_n)\dfrac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}, \quad n \geqslant 2. \end{cases}$$

[3] O de orden dos.

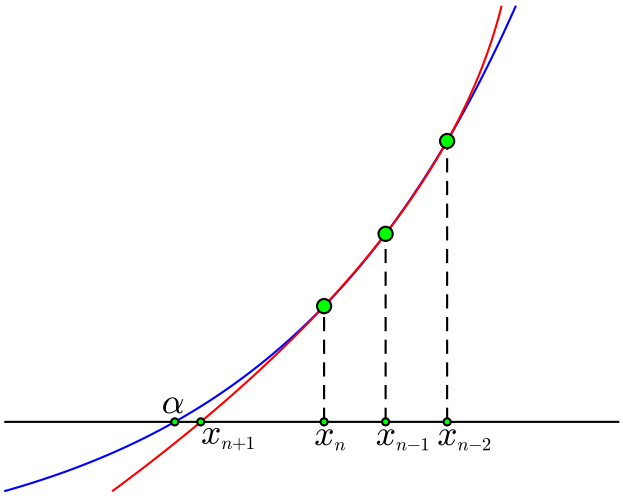
[4] En MATLAB®, la convergencia cuadrática significa, a grandes rasgos, que si en un paso de la iteración se ha obtenido alguna cifra inicial significativa correcta de α, las cifras restantes se obtienen en las siguientes tres o cuatro iteraciones.

La convergencia del método de la secante es ligeramente menos rápida que la del método de Newton. En concreto, su orden de convergencia coincide con el «número de oro», cuyo valor es $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$.



Método de la secante

El método IQI da un paso más y obtiene el punto x_{n+1} a partir del corte del eje OX con el interpolante cuadrático (la parábola) generado por f y los tres puntos anteriores de la iteración, x_n, x_{n-1} y x_{n-2}. Para garantizar el corte con el eje OX, la parábola se construye con eje horizontal[5], es decir, se interpolan los puntos $(f(x_{n-2}), x_{n-2}), (f(x_{n-1}), x_{n-1}), (f(x_n), x_n)$.



Método IQI

Respecto al método de la secante, el método IQI tiene la ventaja de un mayor orden de convergencia[6], pero tiene el inconveniente de que se requiere que las abscisas de los puntos, en este caso $f(x_{n-2})$, $f(x_{n-1})$ y $f(x_n)$, sean distintas entre sí. En consecuencia, definir las iteradas con el método IQI suele ser más delicado que con el método de la secante.

[5] De ahí el calificativo de «inversa» en el nombre del método.

[6] El orden de convergencia del método IQI es, aproximadamente, 1.8393.

Proyecto 1.1 (solución en página 17)

Repite el ejercicio 1.1, pero sustituye el método de Newton por el método de la secante. Elige convenientemente los puntos iniciales.

1.4. La orden fzero

Las técnicas computacionales para determinar ceros o raíces de funciones combinan siempre dos estrategias:

1. Un método numérico en general lento, pero con buenas propiedades de convergencia global, que actúa como «técnica de salvaguardia» (*safeguard technique*).

2. Uno o varios métodos numéricos muy rápidos (*fast methods*), pero que, usualmente, sólo poseen propiedades de convergencia local.

La orden fzero de MATLAB® sigue esta estrategia y se basa en el denominado método de Dekker-Brent, que combina el método de bisección, como técnica de salvaguardia, con los métodos de la secante y de interpolación cuadrática inversa, como métodos rápidos.

Para utilizar la orden fzero basta indicar la función f de la que queremos encontrar el cero junto con, o bien un punto inicial x_0, o bien dos valores x_0 y x_1 donde la función f toma signos distintos. En el primer caso, y antes de nada, MATLAB® trata de encontrar un intervalo que contenga a x_0 y donde f cambie de signo. Si no es capaz de generar tal intervalo, MATLAB® devuelve el valor NaN. En el segundo caso, MATLAB® proporciona un mensaje de error si detecta que f no cambia de signo en los puntos x_0 y x_1.

Tras determinar el intervalo, el proceso iterativo es como sigue: para generar el siguiente iterante/subintervalo, se intenta aplicar inicialmente el método IQI; si esto no es posible, se usa el método de la secante y si ambos métodos dan malos resultados se emplea, finalmente, el método de bisección. En definitiva, el método de bisección actúa, tal como se ha comentado antes, como una técnica de salvaguardia.

Las opciones que utiliza fzero por defecto se pueden modificar mediante la orden optimset.

Ejercicio 1.2

Considera la ecuación no lineal

$$e^x(x-1) = x.$$

1. *Comprueba gráficamente que la ecuación anterior tiene dos soluciones, una positiva y otra negativa.*

2. *Calcula la solución positiva de la ecuación utilizando la orden de MATLAB® fzero con los valores por defecto del vector de opciones y partiendo del punto $x_0 = 1$.*

3. *¿Qué ocurre si ejecutamos la orden fzero partiendo de $x_0 = 20$?*

4. *Calcula de nuevo la solución positiva de la ecuación, partiendo tanto de $x_0 = 1$ como de $x_0 = 20$, pero en esta ocasión imponiendo en el vector de opciones los parámetros TolX=1e-8 y Display='iter'. Con la información obtenida, describe el proceso iterativo que realiza MATLAB® en ambos casos.*

2. Sistemas de ecuaciones no lineales

Consideremos un conjunto de funciones escalares

$$f_k : \mathbb{R}^m \to \mathbb{R},$$

para $k = 1, \ldots, p$, componentes de la función vectorial no lineal[7]

$$f : \mathbb{R}^m \to \mathbb{R}^p \,^{8}.$$

Una raíz o cero de f es cualquier vector $\alpha \in \mathbb{R}^m$ tal que $f(\alpha) = 0$, es decir, $f_k(\alpha) = 0$, para todo $k = 1, \ldots, p$.

Se denomina sistema de ecuaciones no lineales a la expresión $f(x) = 0$, esto es,

$$\left\{ \begin{array}{l} f_1(x_1, \ldots, x_m) = 0, \\ f_2(x_1, \ldots, x_m) = 0, \\ \qquad\qquad \vdots \\ f_p(x_1, \ldots, x_m) = 0. \end{array} \right. \tag{2}$$

Diremos que α es una solución[9] del sistema anterior.

Tal como hicimos en la lección segunda al estudiar los sistemas lineales, supondremos de nuevo que $p \geqslant m$, es decir, solamente trataremos el caso en el que hay un número mayor o igual de ecuaciones que de incógnitas. Es lo que ocurre casi exclusivamente en la práctica. Además, supondremos que $f : \mathbb{R}^m \to \mathbb{R}^p$ es una función diferenciable de todo orden, esto es, todas sus funciones componentes $f_j : \mathbb{R}^m \to \mathbb{R}$ admiten derivadas parciales de todo orden.

Teniendo en cuenta la eficiencia de la orden fzero al tratar el caso $p = m = 1$ y el papel tan relevante del método de Newton en su diseño, es natural plantearse si es posible obtener una formulación de dicho método en este contexto más general. En la siguiente sección veremos que la respuesta es afirmativa.

Ejemplo 2.1 ▶ Sistema de ecuaciones no lineales

Consideremos el sistema de ecuaciones no lineales

$$\left\{ \begin{array}{rcl} x^2 - y^2 & = & 4, \\ x^2 + y^2 & = & 9. \end{array} \right.$$

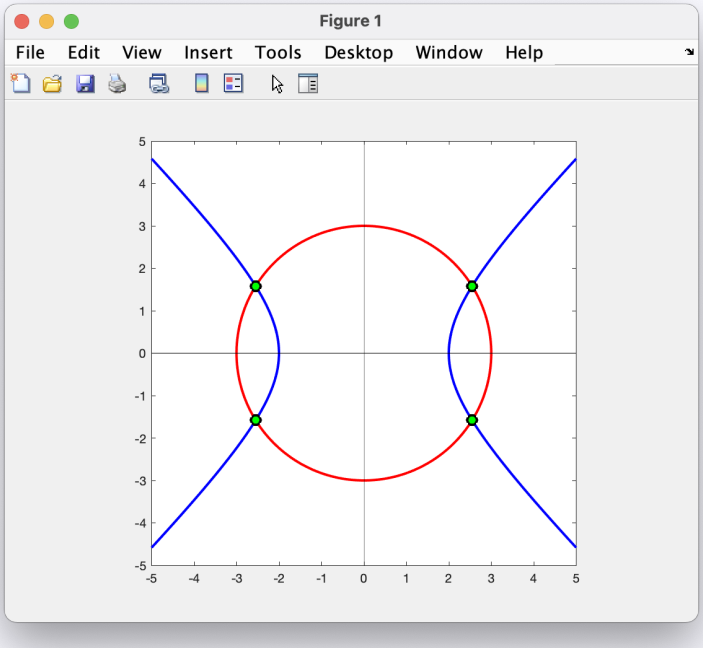
Este sistema tiene cuatro soluciones en $\mathbb{R}^2$, cuyo cálculo analítico es muy simple[a]:

$$\left[\pm\sqrt{\tfrac{13}{2}}, \pm\sqrt{\tfrac{5}{2}}\right]^{\mathsf{T}}, \quad \left[\pm\sqrt{\tfrac{13}{2}}, \mp\sqrt{\tfrac{5}{2}}\right]^{\mathsf{T}}.$$

[7] Se trata, en realidad, de una función general. No obstante, dado que en esta lección estamos interesados en resolver sistemas de ecuaciones no lineales, asumimos que f es una función no lineal.

[8] Así, $f = [f_1, f_2, \ldots, f_p]^{\mathsf{T}}$.

[9] O cero o raíz.



Las cuatro soluciones del sistema son raíces o ceros de la función $f : \mathbb{R}^2 \to \mathbb{R}^2$ dada por

$$f(x, y) = \begin{bmatrix} x^2 - y^2 - 4 \\ x^2 + y^2 - 9 \end{bmatrix}.$$

[a] Estimaremos numéricamente una de las soluciones en el ejemplo 2.2.

2.1. El método de Newton en varias variables

Sea $\alpha \in \mathbb{R}^m$ una solución del sistema anterior (2), o sea, $f(\alpha) = 0$. Dado un vector $x_0 \in \mathbb{R}^m$ suficientemente cerca de α, e imitando la idea central del método de Newton en una variable, buscamos un nuevo punto x_1 que aproxime mejor la raíz α que el punto inicial x_0. Si aproximamos $f(x_1)$ mediante los términos de primer grado del polinomio de Taylor de f en torno al punto x_0 se tiene que

$$f(x_1) \approx f(x_0) + J_f(x_0)(x_1 - x_0).$$

Recordemos que $J_f(x_0)$ denota la matriz jacobiana de f en el punto x_0. Esta matriz tiene orden $p \times m$. Nuevamente, tal como se hizo en el caso unidimensional, para determinar el valor de x_1 planteamos la ecuación

$$f(x_0) + J_f(x_0)(x_1 - x_0) = 0.$$

Para despejar x_1 tenemos que considerar dos casos por separado:

1. $p = m$: en este caso, la matriz $J_f(x_0)$ es cuadrada. Si $J_f(\alpha)$ es invertible, podemos afirmar que $J_f(x_0)$ es también invertible, puesto que x_0 está suficientemente cerca de

α, y deducir que

$$x_1 = x_0 - J_f(x_0)^{-1} f(x_0).$$

Repitiendo el proceso una vez y otra vez, obtenemos una sucesión $\{x_n\}$ definida por la recurrencia $x_{n+1} = x_n + d_n$, donde el vector d_n es la solución (clásica) del sistema de ecuaciones lineales compatible determinado

$$J_f(x_n) d_n = -f(x_n)$$

y da lugar al método de Newton o método de Newton cuadrado (*square*).

2. $p > m$: en este caso, la matriz $J_f(x_0)$ no es cuadrada. Ahora bien, $J_f(x_0)^{\mathsf{T}} J_f(x_0)$ es una matriz cuadrada de orden m. Es más,

$$J_f(x_0)^{\mathsf{T}} J_f(x_0)(x_1 - x_0) = -J_f(x_0)^{\mathsf{T}} f(x_0). \tag{3}$$

Si $J_f(\alpha)^{\mathsf{T}} J_f(\alpha)$ es invertible, podemos afirmar que $J_f(x_0)^{\mathsf{T}} J_f(x_0)$ es también invertible, puesto que x_0 está suficientemente cerca de α, y deducir que

$$x_1 = x_0 - (J_f(x_0)^{\mathsf{T}} J_f(x_0))^{-1} J_f(x_0)^{\mathsf{T}} f(x_0).$$

Repitiendo el proceso una vez y otra vez, obtenemos una sucesión $\{x_n\}$ definida por la recurrencia $x_{n+1} = x_n + d_n$, donde el vector d_n es, en este caso, la solución en el sentido de los mínimos cuadrados[10] del sistema de ecuaciones lineales sobredeterminado

$$J_f(x_n) d_n = -f(x_n)$$

y da lugar al método de Gauss-Newton.

Por consiguiente, ambos métodos están definidos mediante la misma relación de recurrencia: dado un vector inicial x_0, se define el iterante x_{n+1}, para $n \geqslant 0$, en la forma[11]

$$\begin{cases} J_f(x_n) d_n = -f(x_n), \\ \qquad x_{n+1} = x_n + d_n. \end{cases}$$

El análisis numérico de los métodos de Newton y de Gauss-Newton es muy similar al caso unidimensional del método de Newton. En concreto, la convergencia de ambos métodos es cuadrática, esto es, para n suficientemente grande, el error en una iteración es aproximadamente el cuadrado del error de la iteración anterior. Esto significa que, siempre que se parta de un vector inicial x_0 suficientemente cercano (por ejemplo, usando la norma vectorial euclídea) a un vector solución α simple (esto es, la matriz jacobiana $J_f(\alpha)$ tiene rango máximo), la sucesión generada por ambos métodos está bien definida y converge muy rápidamente al vector α. En otras palabras, si se tiene una «buena» estimación de una solución de un sistema, ambos métodos permiten (en general) obtener aproximaciones de dicha solución con la precisión que uno quiera.

[10] Observemos que el sistema lineal (3) constituye unas ecuaciones normales de Gauss.

[11] Cuando $p = m$, la relación de recurrencia se puede expresar como

$$x_{n+1} = x_n - J_f^{-1}(x_n) f(x_n),$$

siempre que la matriz jacobiana J_f sea invertible en x_n.

Problema 2.1

1. *Diseña una función de MATLAB® que implemente el método de Newton para aproximar una solución de un sistema de ecuaciones no lineales. El proceso iterativo debe parar cuando la distancia entre dos iteraciones consecutivas sea menor que cierta tolerancia predefinida. Los argumentos de entrada deben ser la función que define el sistema, su matriz jacobiana, el vector inicial, la tolerancia y un número máximo de iteraciones. Los argumentos de salida deben ser la aproximación a la solución del sistema, el error cometido y el número de iteraciones realizadas.*

2. *Dado el sistema de ecuaciones no lineales*

$$\begin{cases} 2x - y = e^{-x}, \\ -x + 2y = e^{-y}, \end{cases}$$

haz un dibujo en el cuadrado $[-1, 1] \times [-1, 1]$ *de las curvas que lo definen. ¿Cuántas soluciones tiene el sistema en dicho cuadrado? Calcúlalas usando la función diseñada en el apartado anterior. Utiliza una tolerancia de* 10^{-9} *y guíate del dibujo realizado para elegir los puntos iniciales del método de Newton.*

2.2. La orden fsolve

La orden fsolve (*Function SOLVE*) permite resolver sistemas de ecuaciones no lineales enfocándolos como problemas de optimización sin restricciones [12]. En concreto, fsolve calcula una solución del sistema no lineal $f(x) = 0$ calculando un mínimo de la función escalar

$$x \in \mathbb{R}^m \;\mapsto\; g(x) := f_1^2(x) + f_2^2(x) + \cdots + f_p^2(x),$$

siendo $f_1, f_2, \ldots, f_p$ las componentes de la función vectorial f, esto es, $f = [f_1, f_2, \ldots, f_p]^\mathsf{T}$.

Es fácil comprobar que una solución del sistema no lineal es un mínimo global de la función g. Conviene subrayar que el recíproco de esta afirmación no tiene por qué ser cierto. Es decir, g puede tener mínimos relativos que no sean soluciones del sistema no lineal. Es más, puede ocurrir que la orden fsolve «converja» a un vector que no sea una solución del sistema, aunque sí una buena aproximación de un mínimo de g. Cuando esto ocurre, MATLAB® suele mostrar el correspondiente mensaje y sugiere que se elija un vector inicial diferente. Este fenómeno ocurre cuando, iterando, se llega a un vector donde la matriz jacobiana del sistema es singular o cercana a ser singular.

Como ocurre con todas las órdenes de MATLAB® del bloque de optimización, se pueden alterar diversos parámetros de ejecución de esta orden mediante la orden optimset.

La orden fsolve permite utilizar varios métodos de la optimización sin restricciones. En concreto, y por defecto, utiliza un método de regiones de confianza. Asimismo, asignando (en el vector de opciones) al parámetro Algorithm el valor levenberg-marquardt, utiliza el denominado método simplificado de Levenberg-Marquardt. Desde un punto de vista práctico, la principal diferencia entre ambas opciones es que la primera opción (opción por defecto) sólo puede usarse con sistemas cuadrados ($p = m$); en cambio, la segunda no impone restricciones

[12] Los problemas de optimización sin restricciones se estudian en la sección 3.

ni a p ni a m. Eso sí, para sistemas cuadrados, la primera opción suele ser más robusta y eficiente que la segunda. [13]

Siempre que sea posible, es más que recomendable proporcionar la matriz jacobiana de la función f. En caso contrario, MATLAB® la estima mediante fórmulas de diferencias. [14].

Ejemplo 2.2 ▶ La orden fsolve

Consideremos de nuevo el sistema de ecuaciones no lineales dado en el ejemplo 2.1.

Calculamos la solución del sistema situada en el primer cuadrante:

Command Window

```
>> format long
>> F=@(x)[x(1)^2-x(2)^2-4;x(1)^2+x(2)^2-9];
>> cero=fsolve(F,[1 1]);
>> cero

   2.549509756796426   1.581138830084192
```

Sabiendo que la solución exacta es

$$\left[\sqrt{\tfrac{13}{2}}, \sqrt{\tfrac{5}{2}}\right]^{\mathsf{T}},$$

evaluamos el error relativo cometido:

Command Window

```
>> format short e
>> norm([sqrt(13/2) sqrt(5/2)]-cero)/norm(cero)

ans =

   1.1122e-14
```

Comprobamos que el valor de F en la solución calculada está cerca de cero:

Command Window

```
>> F(cero)

ans =

   1.6165e-13
   1.7586e-13
```

[13] La orden fsolve está diseñada para resolver sistemas de ecuaciones no lineales, aunque se puede utilizar también para resolver sistemas de ecuaciones lineales. No obtante, para sistemas lineales, la orden «barra invertida» de MATLAB® es muy superior (numéricamente hablando) a la orden fsolve.

[14] Las fórmulas de diferencias para aproximar la derivada de una función se estudiaron en la lección 4.

Problema 2.2

Considera de nuevo el sistema de ecuaciones no lineales dado en el problema 2.1.

Utilizando la orden de MATLAB® fsolve, calcula las soluciones del sistema en el cuadrado $[-1,1]\times[-1,1]$. *Proporciona la matriz jacobiana del sistema y activa el parámetro DerivativeCheck del vector de opciones para estimar la bondad de dicha matriz jacobiana.*

Proyecto 2.1 (solución en página 19)

Considera el sistema de ecuaciones no lineales

$$\begin{cases} x^2+y^2=9, \\ (x-3)^2+y^2=9, \\ x=\frac{3}{2}. \end{cases}$$

Observa que se trata de un sistema con más ecuaciones que incógnitas.

1. *Calcula analíticamente todas las soluciones del sistema y dibuja las tres curvas que lo definen.*
2. *Aproxima las soluciones del sistema con la orden de MATLAB® fsolve utilizando el método simplificado de Levenberg-Marquardt. Proporciona la matriz jacobiana del sistema y activa el parámetro DerivativeCheck del vector de opciones para estimar la bondad de dicha matriz jacobiana. Elige los puntos iniciales guiándote del dibujo que has realizado en el primer apartado.*
3. *Calcula el error absoluto cometido.*
4. *Define un nuevo sistema de ecuaciones no lineales que sea equivalente al anterior y que tenga el mismo número de ecuaciones que de incógnitas. Aplica la orden fsolve a este nuevo sistema utilizando los dos algoritmos que la orden incorpora. Compara la eficiencia de ambos algoritmos evaluando en cada caso el error cometido.*

3. Optimización sin restricciones

Abordamos en esta última sección el problema de maximizar o minimizar una función no lineal de varias variables, $f: S\subset\mathbb{R}^m\to\mathbb{R}$, a la que nos referiremos como función objetivo.

En primer lugar, recordamos conceptos y resultados elementales relativos a optimización. Para ello, consideremos el problema de optimización[15]

$$\min_{x\in S\subset\mathbb{R}^m} f(x).$$

[15]La búsqueda de máximos de f se efectúa resolviendo el problema de optimización para $-f$.

Un punto $x^* \in S$ es un mínimo global de f si $f(x) \geqslant f(x^*)$, para todo $x \in S$. Si existe un valor $\varepsilon > 0$ tal que $f(x) \geqslant f(x^*)$, para todo $x \in S$ que verifique $||x - x^*|| < \varepsilon$, x^* es un mínimo local de f. De forma análoga se definen máximos globales y locales. La búsqueda de extremos globales constituye la rama llamada optimización global.

Si f es diferenciable, se puede definir su vector gradiente,

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_m} \end{bmatrix},$$

y si es de clase C^2, se puede definir su matriz hessiana,

$$H_f = \begin{bmatrix} \frac{\partial^2 f}{\partial {x_1}^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_m} \\ \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial {x_2}^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_m} & \frac{\partial^2 f}{\partial x_2 \partial x_m} & \cdots & \frac{\partial^2 f}{\partial {x_m}^2} \end{bmatrix}.$$

Los siguientes códigos permiten calcular, en un punto x, el vector gradiente y la matriz hessiana[16] de una función escalar, así como la matriz jacobiana de una función vectorial.

Editor ▶ Vector gradiente gradiente.m

```
function g=gradiente(f,x)
h=1e-5;
m=length(x);
I=eye(m);
g=zeros(m,1);
for j=1:m
    g(j)=(f(x+h*I(:,j))-f(x-h*I(:,j)))/(2*h);
end
end
```

Editor ▶ Matriz jacobiana jacobiana.m

```
function J=jacobiana(f,x)
h=1e-5;
m=length(x);
I=eye(m);
J=zeros(m);
for j=1:m
    J(:,j)=(f(x+h*I(:,j))-f(x-h*I(:,j)))/(2*h);
end
end
```

[16] La matriz hessiana de una función escalar coincide con la matriz jacobiana de su vector gradiente.

Editor ▶ Matriz hessiana hessiana.m

```
function H=hessiana(f,x)
G=@(x)gradiente(f,x);
H=jacobiana(G,x);
end
```

Recordamos, finalmente, las condiciones necesarias y suficiente de extremos de funciones de varias variables.

Sean $f : S \subset \mathbb{R}^m \to \mathbb{R}$, donde S es un conjunto abierto, y $x^* \in S$.

Condiciones necesarias de extremo:

i) Si f es diferenciable y x^* es un mínimo local, entonces $\nabla f(x^*) = 0$.

ii) Si $f \in C^2$ y x^* es un mínimo local, entonces la matriz $H_f(x^*)$ es semidefinida positiva.

Condición suficiente de extremo:

Si $\nabla f(x^*) = 0$ y la matriz $H_f(x^*)$ es definida positiva, entonces x^* es un mínimo local.

3.1. El método de Newton para optimización

El método de Newton para optimización surge como caso particular del método de Newton estudiado en la sección 2.1 cuando este se aplica a la resolución del sistema de ecuaciones no lineales

$$\nabla f(x) = 0.$$

Los ceros de ∇f serán, en general, puntos críticos de f. Un análisis posterior permitirá dilucidar si estos puntos críticos son mínimos, máximos o puntos de silla. Cuando hayamos verificado que un punto crítico es un mínimo, habremos resuelto el problema de optimización.

Puesto que la matriz jacobiana de ∇f es la matriz hessiana H_f de f, el método de Newton consistirá en elegir un punto inicial $x_0 \in \mathbb{R}^m$ y construir a partir de él una sucesión de puntos $x_1, x_2, \ldots$ mediante la relación de recurrencia [17]

$$\begin{cases} H_f(x_n)d_n = -\nabla f(x_n), \\ x_{n+1} = x_n + d_n. \end{cases}$$

Para la búsqueda sin restricciones de mínimos locales de funciones escalares de varias variables, MATLAB® dispone, entre otras, de las órdenes fminsearch y fminunc. La primera aplica un método de búsqueda directa que no utiliza el gradiente (numérico o analítico) y es aplicable a funciones con discontinuidades. La segunda determina el gradiente numéricamente si no se le proporciona analíticamente. Permite la utilización de dos métodos distintos que se usan, respectivamente, para problemas de tamaño medio o grande.

[17] O, escrito de otra manera,

$$x_{n+1} = x_n - H_f^{-1}(x_n)\nabla f(x_n),$$

siempre que la matriz hessiana H_f sea invertible en x_n.

Problema 3.1

Considera la función

$$f(x,y) = xy\log(x^2+y^2), \quad x,y \neq 0,$$

1. *Determina analíticamente sus mínimos relativos.*

2. *Dibuja su gráfica en el cuadrado* $[-1,1]\times[-1,1]$. *Haz también un mapa de curvas de nivel en el mismo cuadrado.*

3. *Utilizando la orden de MATLAB® fminsearch, calcula los mínimos de* f. *Vuelve a calcularlos, pero utilizando esta vez la orden fminunc. Compara los resultados obtenidos con ambas órdenes usando argumentos de salida adecuados. En ambas órdenes, exige una tolerancia en los cálculos de* 10^{-9}. *En la orden fminunc, utiliza las opciones GradObj para proporcionar analíticamente el gradiente de la función y DerivativeCheck para estimar la bondad de dicho gradiente. Elige los puntos iniciales guiándote de los dibujos realizados en el apartado segundo.*

4. *Diseña una función de MATLAB® que implemente el método de Newton para optimización. Sus argumentos de entrada deben ser la función objetivo, el punto inicial, la tolerancia y un número máximo de iteraciones. Sus argumentos de salida deben ser el punto crítico calculado, el valor de la función en el punto crítico, el error relativo cometido y el número de iteraciones realizadas. Calcula el vector gradiente y la matriz hessiana de la función objetivo mediante los códigos de MATLAB® dados, respectivamente, en las páginas 13 y 14.*

5. *Aplica la función del apartado anterior al cálculo de los mínimos relativos de la función* f. *Utiliza una tolerancia de* 10^{-9}. *Verifica numéricamente que, efectivamente, se trata de mínimos. Elige los puntos iniciales guiándote de nuevo de los dibujos realizados en el apartado segundo.*

4. Soluciones de los proyectos

Proyecto 1.1

La siguiente función de MATLAB® implementa el método de la secante, mostrando una tabla con un número prefijado de iteraciones.

Editor ▶ Método de la secante secante.m

```
function secante(f,x0,x1,maxiter)
for n=1:maxiter
    d=f(x1)*(x1-x0)/(f(x1)-f(x0));
    x=x1-d;
    fprintf('x(%2d)=%17.15f \n',n,x)
    x0=x1;
    x1=x;
    end
end
```

Vamos a utilizar esta función para aproximar la solución positiva de la ecuación

$$\frac{1}{1+x^2} = 0.3.$$

con seis cifras significativas correctas. Tomamos $x_0 = 0.2$ y $x_1 = 0.3$ como puntos iniciales y efectuamos 10 iteraciones:

Command Window

```
>> f=@(x)1./(1+x.^2)-0.3;
>> secante(f,0.2,0.3,10)
x( 1)=1.699840000000004
x( 2)=1.608906665147614
x( 3)=1.518893133547937
x( 4)=1.527944432964537
x( 5)=1.527527360486938
x( 6)=1.527525231126108
x( 7)=1.527525231651947
x( 8)=1.527525231651947
x( 9)=1.527525231651947
x(10)=1.527525231651947
```

Es fácil comprobar que la solución positiva de la ecuación vale exactamente $x = \sqrt{\frac{7}{3}}$:

Command Window

```
>> format long
>> sqrt(7/3)

ans =

   1.527525231651947
```

Por lo tanto, el método de la secante ha aproximado la solución positiva de la ecuación no solo con seis cifras significativas correctas, tal como pedía el enunciado del ejercicio, sino con dieciséis. Esto es fácil de explicar, ya que el método de la secante brinda una convergencia muy rápida, del orden de

$$\phi = \frac{1+\sqrt{5}}{2} \approx 1.618.$$

Para finalizar, podemos comprobar numéricamente su orden de convergencia. Para ello, simplemente observa que en la sexta iteración hay 10 dígitos significativos correctos y en la séptima, 16. Por lo tanto,

$$\frac{16}{10} = 1.6 \approx \phi = \frac{1+\sqrt{5}}{2}.$$

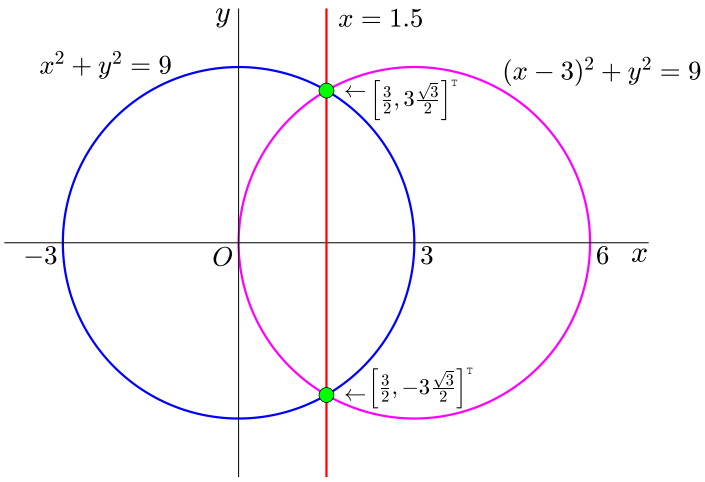
Proyecto 2.1

El sistema de ecuaciones no lineales del proyecto consta de tres ecuaciones con dos incógnitas. Dos de las tres ecuaciones corresponden a sendas circunferencias y la tercera, a una línea recta. Sustituyendo la tercera ecuación, $x = \frac{3}{2}$, en la primera, $x^2 + y^2 = 9$, se tiene que $y = \pm\sqrt{9 - x^2} = \pm 3\frac{\sqrt{3}}{2}$. Es inmediato comprobar que los dos puntos obtenidos,

$$\left[\tfrac{3}{2}, -3\tfrac{\sqrt{3}}{2}\right]^{\mathsf{T}}, \quad \left[\tfrac{3}{2}, 3\tfrac{\sqrt{3}}{2}\right]^{\mathsf{T}},$$

verifican también la segunda ecuación, por lo que ambos puntos son las únicas soluciones del sistema no lineal.

En la siguiente imagen se muestran las tres curvas y los dos puntos de corte comunes, es decir, las dos soluciones del sistema.



Al resolver el sistema no lineal con la orden de MATLAB® fsolve, no podemos utilizar el algoritmo por defecto, puesto que este algoritmo solamente se puede aplicar a la resolución de sistemas con el mismo número de ecuaciones que de incógnitas. Vamos a aplicar, por tanto, el algoritmo de Levenberg-Marquardt, que es apto para todo tipo de sistemas, sean o no cuadrados. Además, tal como pide el enunciado del proyecto, vamos a proporcionar la matriz jacobiana del sistema, calculada analíticamente,

$$J_f(x, y) = \begin{bmatrix} 2x & 2y \\ 2(x-3) & 2y \\ 1 & 0 \end{bmatrix}.$$

Poniendo en común estas ideas en MATLAB®, generamos el código:

Editor ▶ Sistema rectangular: Levenberg-Marquardt proyecto21a.m

```
format long
opciones=optimset('Algorithm','Levenberg-Marquardt',...
    'Jacobian','on','DerivativeCheck','on');
cero=fsolve(@f,[1 1],opciones);
disp(cero)

function [f,Jf]=f(x)
f=[x(1)^2+x(2)^2-9;(x(1)-3)^2+x(2)^2-9;x(1)-3/2];
Jf=[2*x(1) 2*x(2);2*(x(1)-3) 2*x(2);1 0];
end
```

La ejecución en MATLAB® del código anterior genera los siguientes resultados:

Command Window

```
>> proyecto21a
_____________________________________________________________
   CheckGradients Information

Objective function derivatives:
Maximum relative difference between supplied
and finite-difference derivatives = 2.43355e-08.

CheckGradients successfully passed.
_____________________________________________________________


Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the value of the function tolerance, and
the problem appears regular as measured by the gradient.

<stopping criteria details>
   1.499999999998880   2.598076211353393
```

Hemos obtenido una aproximación al cero del sistema situado en el primer cuadrante. Observa que hemos introducido la opción 'DerivativeCheck','on' en optimset, con el objeto de que MATLAB® evalúe la bondad del cálculo analítico de la matriz jacobiana.

El error absoluto cometido es

Command Window

```
>> err=norm(cero-[1.5 3*sqrt(3)/2])

err =

     1.122181887927609e-12
```

Para finalizar, vamos a comparar la eficiencia de los dos métodos incorporados en la orden fsolve. Puesto que el algoritmo por defecto solo se puede aplicar a sistemas con el mismo número de ecuaciones que de incógnitas, vamos a suprimir la tercera ecuación del sistema. Es trivial comprobar que el sistema original y el sistema con la tercera ecuación suprimida son equivalentes. Sea, por tanto, el sistema de ecuaciones no lineales

$$\begin{cases} x^2+y^2=9, \\ (x-3)^2+y^2=9, \end{cases}$$

cuya matriz jacobiana viene dada por

$$J_f(x,y)=\begin{bmatrix} 2x & 2y \\ 2(x-3) & 2y \end{bmatrix}.$$

Lo resolvemos, en primer lugar, con el algoritmo de Levenberg-Marquardt:

Editor ▶ Sistema cuadrado: Levenberg-Marquardt proyecto21b.m

```
format long
opciones=optimset('Algorithm','Levenberg-Marquardt',...
    'Jacobian','on','DerivativeCheck','on');
cero=fsolve(@f,[1 1],opciones);
disp(cero)

function [f,Jf]=f(x)
f=[x(1)^2+x(2)^2-9;(x(1)-3)^2+x(2)^2-9];
Jf=[2*x(1) 2*x(2);2*(x(1)-3) 2*x(2)];
end
```

Ejecutamos este código en MATLAB®:

Command Window

```
>> proyecto21b
------------------------------------------------------------
   CheckGradients Information

Objective function derivatives:
Maximum relative difference between supplied
and finite-difference derivatives = 1.42688e-08.

CheckGradients successfully passed.
------------------------------------------------------------


Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the value of the function tolerance, and
```

```
the problem appears regular as measured by the gradient.

<stopping criteria details>
   1.499999999998514   2.598076211353413
```

El error cometido es:

Command Window

```
>> err=norm(cero-[1.5 3*sqrt(3)/2])

err =

     1.489323415349799e-12
```

En segundo lugar, resolvemos el sistema mediante el algoritmo por defecto:

Editor ▶ Sistema cuadrado: Algoritmo por defecto proyecto21c.m

```
format long
opciones=optimset('Jacobian','on','DerivativeCheck','on');
cero=fsolve(@f,[1 1],opciones);
disp(cero)

function [f,Jf]=f(x)
f=[x(1)^2+x(2)^2-9;(x(1)-3)^2+x(2)^2-9];
Jf=[2*x(1) 2*x(2);2*(x(1)-3) 2*x(2)];
end
```

Su ejecución en MATLAB® genera como resultado:

Command Window

```
>> proyecto21c
------------------------------------------------------------
   CheckGradients Information

Objective function derivatives:
Maximum relative difference between supplied
and finite-difference derivatives = 4.93264e-09.

CheckGradients successfully passed.
------------------------------------------------------------


Equation solved.
```

```
fsolve completed because the vector of function values is near zero
as measured by the value of the function tolerance, and
the problem appears regular as measured by the gradient.

<stopping criteria details>
   1.500000000000000   2.598076211360909
```

El error cometido es:

Command Window

```
>> err=norm(cero-[1.5 3*sqrt(3)/2])

err =

     7.593481399226221e-12
```

Observamos un nivel de precisión ligeramente superior en el primer caso.