Implementación de métodos numéricos para resolución de sistemas de ecuaciones no lineales*

Luis Enrique Sablón Fernández

lesablon@mecanica.ismm.edu.cu

Arianna De Arma Hernández

adhernandez@info.ismm.edu.cu

Ingeniería mecánica e Ingeniería informática

Instituto Superior Minero Metalúrgico de Moa (Cuba).

Resumen: Se realizó una implementación en NetBeans para el método de Newton-Raphson, además de una en el ambiente de desarrollo MATLAB 2013. Este último fue elegido por la facilidad para el tratamiento de los datos tipo matrices y las operaciones asociadas a ellas. En total se implementaron dos métodos principales en MATLAB y uno auxiliar para resolver sistemas de ecuaciones lineales. Estos algoritmos sirven para la resolución de sistemas de ecuaciones no lineales y durante el desarrollo de la propuesta de solución se obtuvieron ficheros con historiales de soluciones y errores. Su utilidad radica en el reforzamiento didáctico de la disciplina Matemática Computacional. Se obtienen gráficos que brindan a los estudiantes una mejor percepción del desempeño de los algoritmos estudiados.

Palabras clave: Newton-Raphson; Punto Fijo; ecuaciones no lineales; Matemática Computacional; algoritmos.

* Recibido: 5 agosto 2018/ Aceptado: 13 enero 2019.

Implementation of numerical methods for solving systems of non-linear equations

Abstract: An implementation in NetBeans for the Newton-Raphson method, as well as one in the MATLAB 2013 development environment was made. For the ease of processing matrix data and the operations associated with them the latter was chosen. Two main methods were implemented in total in MATLAB, and an auxiliary one to solve systems of linear equations. These algorithms are used to solve systems of nonlinear equations, and during developing the proposal, files with record of solutions and errors were obtained. Its usefulness lies in the didactic reinforcement of the Computational Mathematics discipline. In addition, graphics that provide students with a better perception of the performance of the algorithms studied were obtained.

Key words: Newton-Raphson; Fixed Point; non-linear equations; Computational Mathematics; algorithms.

Introducción

La programación es el proceso de diseñar, escribir, depurar y mantener el código fuente de programas computacionales. Su propósito es crear programas que exhiban un comportamiento deseado. El proceso de escribir código requiere frecuentemente conocimientos en varias áreas, además del dominio del lenguaje a utilizar, algoritmos especializados y lógica formal. Programar no involucra necesariamente otras tareas tales como el análisis y diseño de la aplicación (pero sí el diseño del código), aunque sí suelen estar fusionadas en el progreso de pequeñas aplicaciones.

Cuando se resuelven problemas del ámbito ingenieril, muchas veces las soluciones radican en la resolución de ecuaciones, o sistemas de ecuaciones lineales o no lineales. Por otra parte, existe una gran variedad de situaciones ingenieriles que conducen a la modelación mediante ecuaciones diferenciales (ordinarias o en derivadas parciales), cuyas soluciones se obtienen en el campo de los sistemas de ecuaciones lineales. En el Instituto Superior Minero Metalúrgico de Moa se imparten asignaturas de matemática en diferentes carreras del perfil ingenieril, especialmente en la carrera de Ingeniería Mecánica, donde se dota a los estudiantes de herramientas para el tratamiento de fenómenos donde se involucren fluidos newtonianos.

Durante la asignatura de Mecánica de los Fluidos los estudiantes abordan el uso de las ecuaciones de Navier-Stokes aplicando los principios de conservación de la mecánica y la termodinámica, sin embargo, en ocasiones no son capaces de calcular eficientemente las ecuaciones en derivadas parciales no lineales que describen el movimiento de un fluido, porque no dominan con exactitud los métodos matemáticos que permiten darle solución a estos problemas. Como parte del perfeccionamiento de las asignaturas de Análisis Matemático I, II y III, y Álgebra, el Departamento de Matemática prepara modificaciones en los programas de estas asignaturas para que los estudiantes aprendan a utilizar los métodos numéricos desde los primeros años de la carrera.

La manera de estudiar y predecir el comportamiento de un fluido es tratándolo como un medio continuo. De esta forma, las variables de estado del material, tales como la presión, la densidad y la velocidad podrán ser consideradas como funciones continuas del espacio y del tiempo, conduciendo naturalmente a la descripción de los fluidos

como un conjunto de campos vectoriales y escalares, que coevolucionan a medida que una masa de fluido se deplaza como un todo o cambia de forma, por lo que se puede establecer un sistema de ecuaciones no lineales, sobre la base de un modelo exponencial, tal como explica Mahmodian y demás colaboradores (2012).

Este problema entra dentro de la gama de ejemplos que se puede resolver aplicando el método de Newton-Raphson, el cual goza de un excelente prestigio en la comunidad de programadores y matemáticos (Press *et al.* 2007). La idea del Departamento de Matemática es implementar el método antes mencionado, para mejorar la didáctica de la disciplina de Matemática Computacional. Tomando como base los antecedentes anteriores, el problema existente es de carácter pedagógico, y consiste en la no existencia en el Departamento de Matemática, de materiales didácticos que aborden completamente los métodos de Newton-Raphson y Punto Fijo para la solución de sistemas de ecuaciones no lineales.

El objetivo del presente trabajo es implementar los métodos de Newton-Raphson y Punto Fijo para completar la unidad didáctica de Solución de sistemas de ecuaciones, en la disciplina de Matemática Computacional.

Las asignaturas relacionadas durante el trabajo son las de Álgebra, Análisis Matemático I, II y III, Matemática Numérica, Programación, Arquitectura de Computadoras y Métodos de Investigación. Se utilizó el método de análisis y síntesis para la extracción de los segmentos teóricos que acompañan las secciones didácticas de los métodos numéricos estudiados, además de la modelación y el desarrollo algorítmico.

Requisitos del proyecto

El proyecto está destinado a ampliar la unidad didáctica de Solución de sistemas de ecuaciones, dentro de la disciplina de Matemática Computacional. En la carrera de Ingeniería Informática se abordan los métodos de Gauss, Gauss-Jordan, Gauss-Seidel y Jacobi para resolver sistemas de ecuaciones lineales, pero no se realiza un estudio detenido de la extensión del método de Newton-Raphson para sistemas no lineales. En la Figura 1 se muestra la estructura lógica de la propuesta de solución al problema planteado anteriormente.

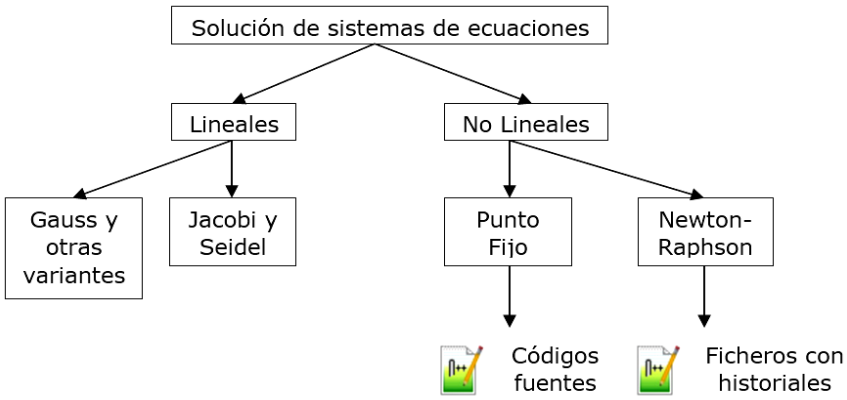

Figura 1. Propuesta de solución didáctica.

El trabajo de programación se inició en el ambiente de desarrollo de NetBeans versión 7.3, usando lenguaje de programación Java versión 8.0. Durante el desarrollo del trabajo surgieron inconvenientes que obligaron al uso de la herramienta Matlab 2013 para comparar y verificar algunos resultados. Para el desarrollo de las prácticas se utilizó un procesador Intel (R) Core (TM) i5-4570S CPU@ 2.90 GHz (Quad-Core), sistema operativo Windows 7. Aunque las características de la computadora empleada proporcionan buena velocidad de cómputo, los algoritmos implementados se pueden ejecutar en computadoras que posean un solo núcleo y 512 MB de memoria RAM, sin que esto afecte la eficiencia computacional de los métodos.

Materiales y métodos

Método de Newton-Raphson

El método de Newton-Raphson puede ser extendido en varios sentidos. En este caso se abordan dos de estas extensiones: la aplicación del método a sistemas de dos o más ecuaciones y su adaptación para calcular raíces complejas de polinomios.

El método de Newton Raphson para simplificar ecuaciones

Para simplificar la notación solo se consideran sistemas de dos ecuaciones con dos incógnitas, pero la extensión a problemas de mayor dimensión es inmediata (Álvarez, Guerra & Lau, 2004). Sea entonces el sistema:

$$\begin{cases} f(x,y)=0 \\ g(x,y)=0 \end{cases} \quad (1)$$

Si se utiliza la notación matricial, el par (x,y) se puede representar como la matriz columna: $x=\begin{bmatrix} x \\ y \end{bmatrix}$

Así que el sistema se expresaría como: $\begin{cases} f(x)=0 \\ g(x)=0 \end{cases}$

Empleando la función vectorial: $f(x)\begin{bmatrix} f(x) \\ g(x) \end{bmatrix}$

El sistema de ecuaciones adquiere el aspecto de una ecuación matricial:

$$f(x)=0 \tag{2}$$

Donde 0 representa la matriz columna nula de orden 2.

El análogo matricial de la derivada $f'(x)$ se define como la matriz jacobiana de $f(x)$ respecto a x y será denotada por $W(x)$, es decir:

$$W(x)=\begin{bmatrix} f_x(x,y) & f_y(x,y) \\ g_x(x,y) & g_y(x,y) \end{bmatrix}$$

Entonces la extensión del algoritmo de Newton-Raphson para un sistema de ecuaciones sería:

$$x_n = x_{n-1} - W^{-1}(x_{n-1})f(x_{n-1}) n = 1,2,3,... \tag{3}$$

Donde: $x_0=\begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$ es la aproximación inicial a la solución buscada.

Teorema 1. Sea x^* una raíz del sistema $f(x)=0$ y R una región que contiene a x^* tal que: f, g y sus primeras y segundas derivadas parciales son continuas y acotadas en R y W es no singular en R. Entonces, si x_0 se escoge en R y lo suficientemente cerca de x^*, la sucesión de vectores x_0, x_1, x_2, generada por el algoritmo $x_n = x_{n-1} - W^{-1}(x_{n-1})f(x_{n-1})$ converge hacia x^*.

Cuando el proceso iterativo converge lo hace en forma cuadrática, lo cual garantiza una rápida aproximación hacia la raíz. En el caso de los sistemas de ecuaciones el error en la aproximación x_n, se denota $E(x_n)$ y se define como la norma $-\infty$ del vector $x^* - x_n$, esto es: $E(x_n) = \|x^* - x_n\|_\infty = \max\{|x^* - x_n|, |y^* - y_n|\}$

Puede probarse que si la norma -∞ de la diferencia de dos iteraciones sucesivas es pequeña puede tomarse como la cota del error, es decir: $E_m(x_n) = \|x_n - x_{n-1}\| = \max\{|x_n - x_{n-1}|, |y_n - y_{n-1}|\}$

Y, por lo tanto, el proceso iterativo se debe detener tan pronto como ambos: $|x_n - x_{n-1}|$ y $|y_n - y_{n-1}|$

Son menores que la tolerancia que se permitirá.

Como la operación de invertir la matriz W es muy costosa en tiempo es preferible transformar la expresión (3) de la siguiente manera: $x_n - x_{n-1} = -W^{-1}(x_{n-1}) f(x_{n-1})$

Transponiendo el término x_{n-1} y llamando Δx_n al vector diferencial: $\Delta x_n = -W^{-1}(x_{n-1}) f(x_{n-1})$

Si se premultiplica en ambos miembros por la matriz jacobiana:

$$W(x_{n-1})\Delta x_n = -f(x_{n-1}) \quad (4)$$

Ahora, el vector Δx_n, se halla resolviendo el sistema lineal (4) por algún método numérico eficiente, por ejemplo, el método de Gauss. En el caso de sistemas de dos ecuaciones, se pueden aplicar incluso métodos más elementales, como Cramer. El sistema (4), si se prefiere, puede ser expresado en forma escalar, escribiendo las matrices en términos de sus componentes:

$$\begin{bmatrix} f_x(x_{n-1}, y_{n-1}) & f_y(x_{n-1}, y_{n-1}) \\ g_x(x_{n-1}, y_{n-1}) & g_y(x_{n-1}, y_{n-1}) \end{bmatrix} \begin{bmatrix} \Delta x_n \\ \Delta y_n \end{bmatrix} = -\begin{bmatrix} f(x_{n-1}, y_{n-1}) \\ f(x_{n-1}, y_{n-1}) \end{bmatrix}$$

Donde:

$$\begin{aligned} f_x(x_{n-1},y_{n-1})\Delta x_n + f_y(x_{n-1},y_{n-1})\Delta y_n &= -f(x_{n-1},y_{n-1}) \\ g_x(x_{n-1},y_{n-1})\Delta x_n + g_y(x_{n-1},y_{n-1})\Delta y_n &= -g(x_{n-1},y_{n-1}) \end{aligned} \quad (5)$$

Resolviendo el sistema (5) se obtienen Δx_n y Δy_n y con ellos se determinan x_n y y_n:

$$x_n = x_{n-1} + \Delta x_n$$
$$y_n = y_{n-1} + \Delta y_n$$

El algoritmo se detiene cuando $|\Delta x_n|$ y $|\Delta y_n|$ son ambos menores que la tolerancia.

Algoritmo del método de Newton Raphson

Se desea resolver el sistema de ecuaciones:

$$\begin{cases} f(x,y) = 0 \\ g(x,y) = 0 \end{cases}$$

El cual posee una raíz $x^* = (x^*, y^*)$ en una región *R* del plano *xy*. Se supone que las funciones y sus derivadas de primer y segundo orden son continuas y acotadas en R, que W no es singular en R y que el par (x_0, y_0), formado por las aproximaciones iniciales, está en R y suficientemente próximo a x^*, de modo que el proceso iterativo converge.

El algoritmo requiere como datos: las funciones f, g, f_x, f_y, g_x, g_y, las aproximaciones iniciales x_0 y y_0, la tolerancia ε que se permitirá.

$$x := x_0$$
$$y := y_0$$

repeat

Resolver el sistema $\begin{cases} f_x(x,y)\Delta x + f_y(x,y)\Delta y = -f(x,y) \\ g_x(x,y)\Delta x + g_y(x,y)\Delta y = -g(x,y) \end{cases}$

Error: $\{|\Delta x|,|\Delta y|\}$

$$x := x + \Delta x$$
$$y := y + \Delta y$$

Until *Error* < ε

La solución del sistema es (x,y) con error menor *Error*

Terminar

Método de Broyden

Un notorio inconveniente del método de Newton-Raphson multivariable es que en cada iteración se requiere resolver un sistema lineal de *nxn*, lo cual implica un esfuerzo computacional de $O(n)$. El aporte de este método es que la inversa de la matriz jacobiana ahora es actualizada en cada iteración mediante la *fórmula de inversión* de *Sherman y Morrison* (Conde & Schiavi, 2010):

$$\left[A^{(k)}\right]^{-1} = \left[A^{(k-1)}\right]^{-1} + \frac{\left(s^k - \left[A^{k-1}\right]^{-1} y^k\right)\left[s^k\right]^T \left[A^{(k-1)}\right]^{-1}}{\left[s^k\right]^T \left[A^{(k-1)}\right]^{-1} y^{(k)}}$$

Donde **A** es la matriz jacobiana, $y = f(x) - f(x)$ y $s = x - x$. Esta fórmula solo incluye multiplicaciones matriciales y, por lo tanto, demanda no más de $O(n)$ cálculos aritméticos. $\lim_{k\to\infty} \frac{\left\|x^{k+1} - p\right\|}{\left\|x^k - p\right\|}$

Donde **p** es la solución de $f(x) = 0$. Sin embargo, una reducción en el orden de convergencia de dos a superlineal es aceptable, ya que compensa por la reducción en el número de cálculos requeridos por iteracción.

Algoritmo del método de Broyden

% Inicio

v = F(x_0)

M = inv(DF(x_0))

% Inversa Jacobiano

```
s = -M*v;
x = x0 + s;
  % Paso de Newton
incr = norm(s);
while incr > tol
  w = v;          % F(x(k-1))
v = F(x);
  y = v-w;        % F(x(k)) – F(x(k-1))
z = - M*y;        % -inv(A(k-1))*y(k)
p = - s´*z;
  q = s´*M;       % s(k)´*inv(A(k-1))
R = (s+z)*q/p;
M = M+R;          % inversa de A(k)
s = -M*v;
x = x+s;          % Paso de Brpyden
incr = norm(s);
end
```

Método de Punto Fijo

El método de aproximaciones sucesivas (o del punto fijo) para determinar una solución de la ecuación no lineal $f(x)=0$ se basa en el teorema del punto fijo. Para ello el primer paso que se realiza en este método consiste en reescribir la ecuación $f(x)=0$ en la forma $x=g(x)$. Adviértase que existen múltiples posibilidades para transformar la ecuación $f(x)=0$ en otra del tipo $x=g(x)$. Por ejemplo, podría despejarse (de la forma que sea) x de la expresión de la ecuación $f(x)=0$. O podría sumarse la variable x en ambos lados de la ecuación y designar por $g(x)$ a $(f(x)+x)$:

$0=f(x)\Leftrightarrow x=f(x)+x=g(x)$

O siendo $\alpha=0$ podrá realizarse el proceso: $0=f(x)\Leftrightarrow x=\alpha\cdot f(x)+x=g(x)$

O bien: $0=f(x)\Leftrightarrow x^k=\alpha\cdot f(x)+x^k=g(x)\Leftrightarrow x=\sqrt[k]{\alpha\cdot f(x)+x^k}=g(x)$

Teorema 1. Si $g(x)$ es una contradicción definida sobre el intervalo $[a,b]$ entonces el método de aproximaciones sucesivas que se acaba de describir genera, a partir de

cualquier valor $x_0 \in [a,b]$, una sucesión $\{x_i\}^{\infty}{}_i = 0$ que converge hacia la única solución de la ecuación $x = g(x)$ en el intervalo $[a,b]$.

Demostración: en virtud del teorema del punto fijo (teorema 1), por ser $g(x)$ una contracción definida sobre el espacio métrico completo $([a,b],df)$ admitirá un único punto fijo $x*$ que será el límite de la sucesión $\{x_i\}^{\infty}{}_i = 0$.

Teorema 2: si $g(x)$ es una aplicación de clase $C^1([a,b])$ y que toma valores en $[a,b]$ verificando la condición: $\exists k < 1 / |g'(x)| \leq k \forall x \in [a,b]$

Entonces la sucesión $\{x_i\}^{\infty}{}_i = 0$ generada, a partir de cualquier $x_0 \in [a,b]$, converge hacia la única solución de la ecuación $x = g(x)$ en $[a,b]$.

Demostración: por aplicación del teorema del valor medio se verificará que: $\forall x,y \in [a,b] \exists z \in]a,b[/ \quad g(x) - g(y) = g'(z) \cdot (x - y)$

Y por haber supuesto que la primera derivada estaba acotada en valor absoluto se tendrá que: $\forall x,y \in [a,b]: \quad |g(x) - g(y)| \leq k \cdot |x - y| < |x - y|$

Por lo que, teniendo en cuenta que $g : [a,b] \to [a,b]$, resulta que $g(x)$ es una contracción. Aplicando el teorema precedente quedará totalmente demostrado este (Conde & Schiavi, 2010).

Teorema 3: si existe una solución $x*$ de la ecuación $x = g(x)$ en un intervalo $[a,b]$ en el que $g(x)$ es de clase $C^1([a,b])$ y $|g'(x*)| < 1$ entonces existe un valor $\delta > 0$ tal que si $|x* - x_i| < \delta$ la sucesión $\{x_{i+1} = g(x_i)\}^{\infty}{}_i = 0$ verifica que: $x* - x_i < \delta \quad \forall x_i$

a) $\lim_{i \to \infty} x_i = x*$

Demostración: por ser $g'(x)$ continua en todo $x \in [a,b]$ existirá un intervalo abierto de centro $x*$ y radio δ' tal que en él se verifique: $|g'(x)| \leq k < 1 \quad \forall x \in]x* - \delta', x* + \delta'[$

Considerando un valor $\delta < \delta'$ se tendrá por tanto que:

$$|g'(x)| \leq k < 1 \quad \forall x \in]x^* - \delta, x^* + \delta[$$

Y consecuentemente $g(x)$ es una contracción en $[x^* - \delta, x^* + \delta]$. Ello conduce a que:

$$\forall x_i \in \{x_i\}_{i=1}^{\infty} : |x_i - x^*| = |g(x_{i-1}) - g(x^*)| < k \cdot |x_{i-1} - x^*| < k^2 \cdot |x_{i-2} - x^*| < k^i \cdot |x_0 - x^*| < k^i \cdot \delta < \delta$$

Por otra parte, al ser $k < 1$ bastará con escoger el índice i suficientemente elevado para que todos los elementos de la sucesión con índice mayor que i sean tan cercanos a x^* como se desee. En otros términos $x^* = \lim_{i \to \infty} x_i$.

Teorema 4: siendo $g(x)$ una contracción definida en el intervalo $[a,b]$ la distancia entre la única solución x^* de la ecuación $x = g(x)$ y cualquier valor $x_0 \in [a,b]$, está acotada mediante la expresión: $|x^* - x_n| \leq \frac{k^n}{1-k} \cdot |x_1 - x_0|$

Donde k es la constante de Lipschitz de la contracción.

Nota: una interpretación gráfica del método consiste simplemente en buscar entre la bisectriz del primer cuadrante y la contracción $g(x)$ mediante sucesivos escalones comprendidos entre la gráfica $g(x)$ y la bisectriz del primer cuadrante, como se muestra en la Figura 2.

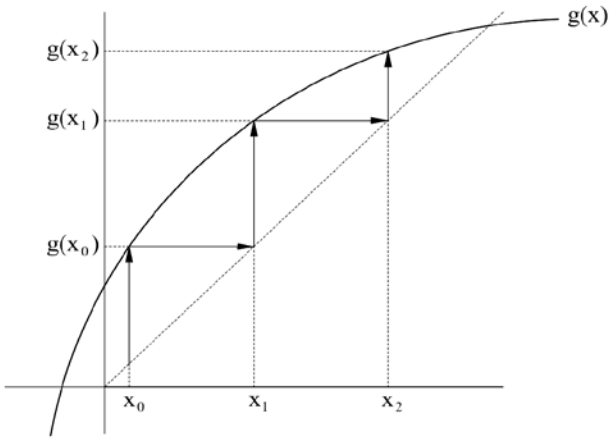

Figura 2. Interpretación gráfica del método de aproximaciones sucesivas.

En la práctica, en lugar de calcular a priori el número de iteraciones a realizar, se va estimando en cada iteración la distancia del valor en ella hallado a la solución exacta. Esta estimación se realiza simplemente evaluando la diferencia entre las dos últimas aproximaciones halladas que, cuando $g(x)$ es una contracción, son un indicador de la cercanía a la solución exacta en virtud del siguiente teorema:

Teorema 5: siendo $g(x)$ una contracción definida en el intervalo $[a,b]$ la distancia entre la única solución $x*$ de la ecuación $x = g(x)$ y cualquier elemento de la sucesión $\{xn = g(x_{n-1})\}_{n=0}^{\infty}$, generado a partir de cualquier valor $x0 \in [a,b]$, está acotada mediante la expresión: $|x* - x_n| \geq \frac{k}{1-k} \cdot |x_n - x_{n-1}|$.

Donde k es la constante de Lipschitz de la contracción.

Cuando $g(x)$ sea una contracción, al ser $k < 1$, bastará con hacer un número de iteraciones tal que $|x_n - x_{n-1}|$ sea suficientemente pequeño para asegurar que $|x* - x_n|$ también es pequeño. En los algoritmos que recojan el método, este control de la convergencia debe acompañarse con la limitación del número de iteraciones a realizar en previsión de los casos en los que, no siendo $g(x)$ una contracción, el método no converja. Más concretamente, un algoritmo del método de aproximaciones sucesivas, en el que se parte de la ecuación equivalente $x = g(x)$, es el siguiente:

Algoritmo del método de aproximaciones sucesivas

Dada la ecuación $x = g(x)$, el indicador de precisión ε, un valor máximo del número de iteraciones que se permiten realizar (maxiter) y un punto $x0$ con el que inicializar el proceso.

Tol ← 2 · ε
Iteración ← 0

Mientras ((iteración < maxiter) y (tol > ε)), hacer:

x1 ← g($x0$)

Tol ← $|x1-x0|$

Iteración ← iteración + 1

$x0 \leftarrow x1$

Fin bucle condicional.

Si (tol < ε) entonces:

Tomar $x1$ como solución

si no:

Escribir un mensaje de error en el proceso de cálculo

Fin condición.

Fin del algoritmo.

Descripción de la solución propuesta

Para el acaparamiento interno de la investigación se utilizaron arreglos, matrices y variables simples. La herramienta NetBeans fue buena para el trabajo realizado, pero no resultó precisa la solución, al menos no hubo la eficacia de la herramienta de trabajo Matlab porque los algoritmos implementados requieren del uso de estructuras cíclicas y el almacenamiento eficiente de sucesiones de soluciones en formato de matrices.

En el desarrollo de dicha implementación se verificó que la programación en MATLAB posee varias ventajas respecto a NetBeans cuando se trabajan datos estructurados tipo arreglos de n-dimensiones, también el ahorro de memoria interna. La selección del lenguaje Java está justificada por las habilidades adquiridas durante los dos años de curso, pero no por facilidad, pues resulta menos eficaz en el tratamiento de funciones por la entrada estándar, ya que se necesitan ficheros precompilados con las funciones y sus derivadas parciales. En NetBeans la declaración, inicialización y ampliación de matrices requiere operaciones de orden n en muchos casos, mientras que en MATLAB para el usuario programador estas instrucciones tienen una complejidad de orden 1. Por otra parte, MATLAB permite la entrada de varias funciones mediante el comando inline, propiciando un manejo simbólico de las variables, sin que esto afecte la dimensión aritmética de sus evaluaciones posteriores (Yang *et al*. 2005).

Implementaciones del método de Newton-Raphson

En lenguaje Java se implementó la clase Newton que posee tres métodos públicos principales que se exponen a continuación: *public double[] Newton(double sol[], double Tol, int MaxIt, int cod)*

Los códigos fuentes del paquete *newtonrapson* y el programa principal de la aplicación se muestran a continuación:

Métodos públicos de la clase Newton

```
package newtonrapson;
public class Newton {
public Newton() {
    }
    public double F1(double x, double y) {
        return x*y-1;
    }
    public double G1(double x, double y) {
        return x-y*y;
    }
    public double F1x(double x, double y) {
        return y;
    }
    public double F1y(double x, double y) {
        return x;
    }
    public double G1x(double x, double y)
        return 1;
    }
    public double G1y(double x, double y)
        return -2*y;
    }
    public double F2(double x, double y) {
        return x*x+y*y-4;
    }
    public double G2(double x, double y) {
        return 4*Math.pow(x-1, 2)+ 9 *Math.pow(y-2, 2)-36;
    }
```

```
public double F2x(double x, double y) {
    return 2*x;
}
public double F2y(double x, double y) {
    return 2*y;
}
public double G2x(double x, double y) {
    return 8*(x-1);
}
public double G2y(double x, double y) {
    return 18*(y-2);
}
public double F3(double x, double y) {
    return 5*x*x+2*y*y-7;
}
public double G3(double x, double y) {
    return Math.sin(x)+Math.cos(2*y)-1;
}
public double F3x(double x, double y) {
    return 10*x;
}
public double F3y(double x, double y) {
    return 4 * y;
}
public double G3x(double x, double y) {
    return Math.cos(x);
}
public double G3y(double x, double y) {
    return -2 * Math.sin(2 * y);
}

public double[] Newton(double sol[], double Tol, int MaxIt, int cod) {
    double error= Double.MAX_ VALUE;
    for (int i = 0;i<MaxIt && error > Tol;i++){
        System.out.println("It: "+(i+1));
        double [][] matriz = LLenarMatriz(cod, sol[0], sol[1]);
        double tmp[] = Gauss(matriz, String.valueOf(Tol).length()+2);
        System.out.println("Soluciones Gauss: "+tmp[0]+" "+tmp[1]);
        sol[0] += tmp[0];
```

```
            sol[1] += tmp[1];
            System.out.println("Soluciones Sistema: "+sol[0]+" "+sol[1]);
            error= Math.max(Math.abs(tmp[0]), Math.abs(tmp[1]));
            System.out.println("Error: "+error);
        }
        return sol;
    }
    public double[][] LLenarMatriz(int cod, double x, double y) {
        double [][] matriz = new double[2][3];
        switch (cod){
            case 1:{
                matriz[0][0] = F1x(x,y);
                matriz[0][1] = F1y(x,y);
                matriz[0][2] = -F1(x,y);
                matriz[1][0] = G1x(x,y);
                matriz[1][1] = G1y(x,y);
                matriz[1][2] = -G1(x,y);
            }break;
            case 2:{
                matriz[0][0] = F2x(x,y);
                matriz[0][1] = F2y(x,y);
                matriz[0][2] = -F2(x,y);

                matriz[1][0] = G2x(x,y);
                matriz[1][1] = G2y(x,y);
                matriz[1][2] = -G2(x,y);
            }break;
            case 3:{
                matriz[0][0] = F3x(x,y);
                matriz[0][1] = F3y(x,y);
                matriz[0][2] = -F3(x,y);

                matriz[1][0] = G3x(x,y);
                matriz[1][1] = G3y(x,y);
                matriz[1][2] = -G3(x,y);
            }break;
        }
        return matriz;
    }
```

```
    public double[] Gauss(double [][] matriz, int l) {
        double tmp = 0, m;
        double X[] = new double[2];
        if (matriz[0][0]==0 || matriz[0][0]<matriz[1][0]){
            for (int i = 0;i<2;i++){
                tmp = matriz[0][1];
                matriz[0][i] = matriz[1][i];
                matriz[1][i] = tmp;
            }
        }
        m = matriz[1][0] / matriz[0][0];
        matriz[1][1] -= m*matriz[0][1];
        matriz[1][2] -= m*matriz[0][2];
        X[1] = redondear(matriz[1][2]  / matriz[1][1], l);
        X[0] = redondear((matriz[0][2] - matriz[0][1]*X[1]) / matriz[0][0], l);
        return X;
    }

    public double redondear(double a, int cifras){
        String cadena = String.valueOf(a);
        if (cadena.contains("-")){
            cifra++;
        }
        cadena = cadena.substring(0, cifras+1);
```

Paquete newtonrapson: programa principal de la aplicación

```
package newtonrapson
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class NewtonRapson {
    public static void main(String[] args) throws IOException{
        Newton newton = new Newton();
        BufferedReader buffer = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Entre el código del Sistema a tartar: ");
        System.out.println("1: xy = 1");
        System.out.println("    x = y^2");
```

```
System.out.println();
System.out.println("2:         x^2+y^2 = 4");
System.out.println("  4(x-1)^2+9(y-2^2) = 36");
System.out.println();
System.out.println("3:         5x^2+2y^2 = 7");
System.out.println("  senx+cos2y = 1");
int cod = Integer.parseInt(buffer.readLine());
System.out.println("Entrar soluciones iniciales!!!!\nEntre x0: ");
double x0 = Double.parseDouble(buffer.readline());
System.out.println("Entre y0: ");
double y0 = Double.parseDouble(buffer.readline());
double sol[] = {x0, y0};
System.out.println("Entre la cantidad de iteraciones: ");
int MaxIt = Integer.parseInt(buffer.readline());
System.out.println("Entre la tolerancia: ");
double Tol = Double.parseDouble(buffer.readline());
sol = newton.Newton(sol, Tal, MaxIt, cod);
System.out.println("Soluciones: ");
System.out.println("x: "+sol[0]);
System.out.println("y: "+sol[1]); }}
```

Este método permite calcular la solución de un sistema de ecuaciones no lineales y recibe como parámetros una aproximación inicial, la tolerancia, un máximo de iteraciones y un código para identificar una función específica de las que están previamente compiladas (métodos públicos Fi y Gi). En este último parámetro y su funcionalidad radica una de las deficiencias de la implementación en MATLAB, aunque para este trabajo no tiene repercusión negativa por tratarse de una propuesta didáctica (Yang *et al*. 2005).

public double[][] LlenarMatriz(int cod, double x, double y)

Este método permite inicializar una matriz de 2x3 y llenarla con la evaluación de las funciones Fi y Gi, según el tipo de función ilustrativa que se esté tratando. Recibe como parámetros el código de la función y la solución actual que se evalúa en ella.

public double[] Gauss(double [][] matriz, int I)

Se resuelve el sistema de ecuaciones lineales resultante de la evaluación en las funciones y las derivadas. Recibe como parámetros una matriz de 2x3 dimensiones y la cantidad de cifras significativas para efectuar algunos redondeos intermedios y evitar overflows.

En lenguaje MATLAB se realizó la implementación del método de Newton-Raphson obteniendo dos ficheros con la programación de los métodos que se explican a continuación:

function [x,y,hSol,e,hErr,it] = NewtonRaph(f,g,fx,fy,gx,gy,x0,y0,Tol,maxIter)

Esta es una variante didáctica preparada para cualquier sistema no lineal de dos ecuaciones y dos incógnitas. Los parámetros desde *f* hasta *gy* representan las funciones y sus derivadas parciales, respectivamente (pasados por la línea de comandos). Una aproximación inicial, la tolerancia y un máximo de iteraciones para definir fracaso en caso de que las soluciones sean muy costosas o impracticables. El método maneja internamente un vector de soluciones que almacena el historial de las mismas a través de cada iteración, un vector con la evolución de los errores y un contador de iteraciones para detectar cuándo exactamente se experimenta la parada.

function [x,y] = MGauss(matriz)

Este método permite la resolución de un sistema de ecuaciones lineales de 2x3 dimensiones, es una adaptación del método general de Gauss y se utilizó la estrategia parcial de pivote. Como parámetro se recibe una matriz de 2x3 dimensiones.

Los códigos fuentes de las implementaciones explicadas anteriormente se exponen a continuación:

Código fuente del método Newton-Raphson

```
function [x,y,hSol,e,hErr,it] = NewtonRaph(f,g,fx,fy,gx,gy,x0,y0,Tol,maxIter)
% Esta función permite la resolución de un Sistema de ecuaciones no lineales, con dos ecuaciones y dos incógnitas.
% f y g son funciones pasadas desde el comando inline o almacenadas en Math-files.
% fx, fy, gx y gy son funciones que se corresponden con las derivadas parciales de f e y, respectivamente.
```

```
% x0, y0 es la solución inicial.
% Tol es la tolerancia máxima para todos los cálculos y para la precisión de la solución final.
% maxIter es un máximo de iteraciones para definir que las soluciones a partir de ahí no son aconsejables por el desgaste de la eficiencia computacional.
% hSol almacena el historial de soluciones para estudiar las características de la convergencia del método.
% hErr almacena el historial de los errores que se cometen en cada aproximación.
% it devuelve la iteración exacta donde el algoritmo concluye, sirve para estudiar la eficiencia computacional.
%-------------------------------------------------------------------------------------------------
% Inicialización de variables
hSol=zeros(1,2);
tmp=zeros(1,2);
x=x0;
y=y0;
hSol(1,1)=x0;
hSol(1,2)=y0;
deltaX=0;
deltaY=0;
MS=zeros(2,3); % se reserva espacio para la matriz de sistema de casa iteración
error=Tol+1;
it=1;
%-------------------------------------------------------------------------------------------------
while it<=maxIter & error>Tol
    % aquí se forma la matriz del sistema para cada iteración
        MS(1,1)=feval(fx,x,y); % feval evalúa en la función definida para caso a partir de (x,y)
        MS(1,2)=feval(fy,x,y);
        MS(1,3)=(-1)-feval(f,x,y);
        MS(2,1)=feval(gx,x,y);
        MS(2,2)=feval(gy,x,y);
        MS(2,3)=(-1)-feval(g,x,y);
        [deltaX,deltaY]=MGauss(MS); %aquí se resuelve el Sistema
        error=max(abs(deltaX),abs(deltaY));
        % aquí se actualiza el vector historial de errores, útil en el estudio de la eficacia computacional
        if it==1, hErr=error;
        else hErr=[hErr;error];
        end
        x=x+deltaX;
```

```
        y=y+deltaY;
        % está sección es para actualizar la matriz de historial de soluciones
        tmp(1,1)=x;
        tmp(1,2)=y;
        hSol=[hSol;tmp];
        it=it+1;
end
% preparación de soluciones finales

it=it-1;
e=error;
end
```

Método auxiliar para resolver un sistema de ecuaciones lineales

```
function [x,y] = MGauss(matriz)
% Esta función permite la resolución de un sistema de ecuaciones lineales de 2x3 dimensiones.
% Matriz es e la matriz ampliada del sistema Ax=b……A|b
n=length(matriz(1,:));
m=length(matriz(:,1));
if n ~=3|| m~=2, error(`la matriz del sistema no tiene dimensiones correctas´); return;
end
tvar=0;
% se comprueba y ejecuta la estrategia parcial de pivote
if matriz(1,1)==0 || matriz(1,1)<matriz(2,1);
   for i=1:3 % intercambio de filas
      tvar=matriz(1,i);
      matriz(1,i)=matriz(2,i);
      matriz(2,i)=tvar;
   end
end
p=matriz(2,1)/matriz(1,1);
% matriz(2,1)=0 es opcional porque no se utiliza este valor en ningún cálculo
%modificación de los coeficientes de la matriz A|b
matriz(2,2)=matriz(2,2)-p*matriz(1,2);
matriz(2,3)=matriz(2,3)-p*matriz(1,3);
% proceso inverso del método de Gauss para obtener las soluciones
y=matriz(2,3)/matriz(2,2);
x=(matriz(1,3)-y*matriz(1,2))/matriz(1,1);
```

```
end
```

Implementación del método de Punto Fijo

En lenguaje MATLAB se realizó la implementación del método de Punto Fijo, obteniendo un fichero con la programación del método y el código fuente que se explica a continuación:

```
function [x,hSol,j,error,hErr] = PuntoFijo(g1,g2,g3,x0,Tol,maxIter)
% la función permite encontrar la solución de un sistema de tres ecuaciones no lineales con
% tres incógnitas
% aproximaciones sucesivas mediante el enfoque F(x)=0 <=> x=G(x)
% g1,g2,g3 son las funciones transformadas, deben ser pasadas mediante el comando
% inline()
% x0 es un vector que contiene la aproximación inicial a la solución deseada
% Tol es la tolerancia permitida para los cálculos y los resultados finales
% maxIter es un número máximo de iteraciones para determinar fracaso del algoritmo en
% caso de no hallar la solución
hSol = x0;
j = 1;
error = Tol + 1;
hErr = error;
while error > Tol & j<=maxIter
        %aquí se evalúan las funciones en la aproximación inicial
  x(1) = feval(g1,x0(2),x0(3));
  x(2) = feval(g2,x0(1),x0(3));
  x(3) = feval(g3,x0(1),x0(2));
        % se recalcula la condición de parada a través de la norma del vector diferencia entre
        la solución actual y la anterior
  error = norm(x – x0);
        j = j + 1;
        x0 = x;
  % se actualizan las variables con los historiales de soluciones y errores
  hSol = [hSol;x0];
  hErr = [hErr;error];
end
if error > Tol, disp(`El proceso no converge, debe comprobar el teorema 1 página 95 Álvarez et
al. 2004´);
end
```

Este método permite la resolución de un sistema de ecuaciones no lineales, a partir de la transformación $F(x) \Leftrightarrow x = G(x)$. Las funciones deben ser pasadas mediante el comando inline. Esta variante está preparada para tres ecuaciones con tres incógnitas, pero se puede variar fácilmente para sistemas n-dimensionales. Además, se tiene como parámetro un vector con la solución inicial, la tolerancia y un máximo de iteraciones para controlar si el proceso converge o no (Álvarez, Guerra & Lau, 2004. Teorema 1, página 95).

Pruebas realizables a los métodos implementados

Para realizar algunas comparaciones entre los métodos de Newton-Raphson y Punto Fijo se abordó un ejemplo de la literatura (Álvarez, Guerra & Lau, 2004). A continuación, se relaciona su fuente:

Ejemplo1. Ejemplo 2 Álvarez *et al*. 2004, página 28

En la Tabla 1 se muestran los resultados comparativos de una corrida mediante los métodos de Newton-Raphson y Punto Fijo para 100 iteraciones generales, con tolerancia igual a 0,000005. Del análisis de los resultados se evidencia que la variante de Newton-Raphson es más eficaz y eficiente, los resultados lo corroboran el error obtenido y el número de iteraciones reales, respectivamente.

En este ejemplo se evidencia el método Newton-Raphson como el más eficiente, preciso y eficaz, pues en las iteraciones la convergencia es más eficiente, la solución es más precisa y el error es más pequeño; hace una adecuada selección de las aproximaciones iniciales.

Tabla 1. Newton-Raphson vs. Punto Fijo: eficacia y eficiencia

Newton-Raphson			
x	y	Error	iteración
1,000000000001964	1,000000000002655	2,525357393213844e-0,6	4
Punto Fijo			
x	Y	Error	iteración
1,000003093458063	1,000001360607859	4,469583982722054e-0,6	36

A continuación, se muestra el espacio de trabajo de MATLAB para una corrida del método de Punto Fijo:

$$x_1 = \frac{\cos(x_2 * x_3)}{3} + \frac{1}{6}$$

$$x_2 = \frac{1}{9}\sqrt{x_1^2 + sen(x_3) + 1.06} - 0.1$$

$$x_3 = \frac{1}{20}(1 - e^{-x_1 * x_2}) - \frac{\pi}{6}$$

```
>> g1=inline(`cos(x2*x3)./3+1./6´,`x2´,`x3´);
>> g2=inline(`1./9*sqrt(x1.^2+sin(x3)+1.06)-0.1´,`x1´,`x3´);
>> g3=inline(`1./20*(1-exp(-x1*x2))-pi./6´,`x1´,`x2´);
>> x0(1)=1.0;
>> x0(2)=1.0;
>> x0(3)=-1.0;
>> [x,hSol,j,error,hError] = PuntoFijo(g1,g2,g3,x0,0.00005, 100);
>> x
x =
  0.499999999996427  0.000000026241805          -0.523598996678042

>> hSol
hSol =
  0.346767435289380  0.022652224149387          -0.491992747656871
  0.499979299379800 -0.006517022420287  -0.523207561430399
  0.499998062263220  0.000019636310936          -0.523761960127409
  0.499999999982371 -0.000008843170194  -0.523598284694838
  0.499999999996427  0.000000026241805          -0.523598996678042
>> hErr
      1.000050000000000
      1.280622046894310
      0.159056860406648
      0.006560153758070
      0.000166145968439
      0.000008897942981

>> error
error =
      8.897942981453349e-06
```

Mediante la utilización de las funciones **plot** y **plot3** de los ficheros cabecera de MATLAB se implementó la opción de graficar el historial de soluciones para el sistema de ecuaciones, de modo que se pueda apreciar la convergencia del método hacia la solución deseada. En las Figuras 3 y 4 se muestran gráficos con el historial de soluciones.

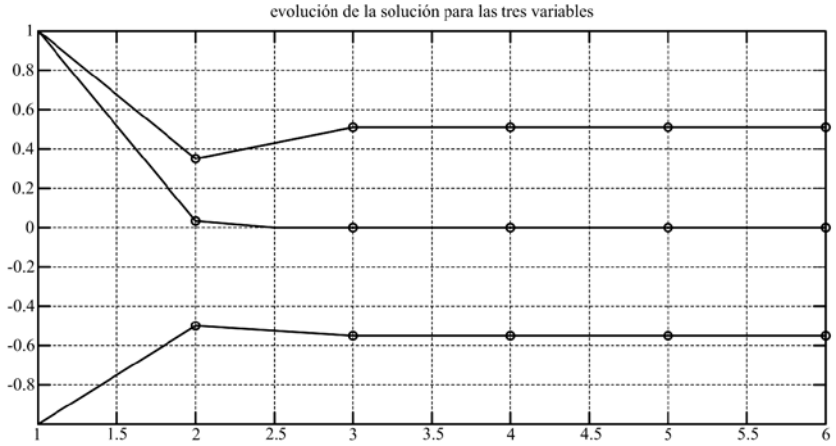

Figura 3. Gráfico en 2D del historial de soluciones para Ejemplo Punto Fijo.

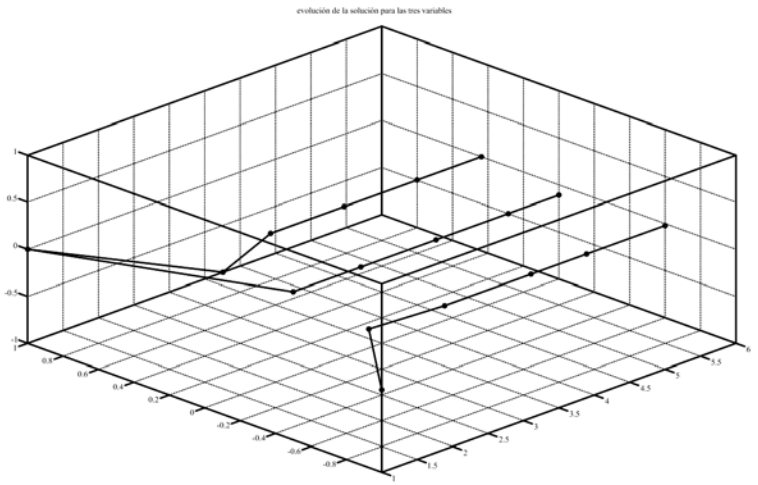

Figura 4. Gráfico en 3D del historial de soluciones para Ejemplo Punto Fijo.

Conclusiones

A través de la implementación de los métodos de Newton-Raphson y Punto Fijo para sistemas de ecuaciones no lineales se logra ampliar las opciones de la unidad didáctica Solución de sistemas de ecuaciones, de la disciplina Matemática Computacional. Se realiza la programación del método de Newton-Raphson en dos entornos de desarrollo,

propiciando comparaciones interesantes sobre la selección adecuada de las estructuras de datos para el almacenamiento de las variables y los resultados.

A través de las pruebas algorítmicas se estudia la eficacia y la eficiencia computacional de los métodos implementados, propiciando un marco ilustrativo para los estudiantes cuando los sistemas de ecuaciones resultantes de la modelación práctica no sean lineales.

A partir del análisis bibliográfico se conforma un marco teórico útil para la revisión de los aspectos matemáticos de ambos métodos. Se generan los códigos fuentes de ambas programaciones, precisando los detalles esenciales a través de comentarios especializados.

Referencias bibliográficas

ÁLVAREZ, M.; GUERRA, A. & LAU, R. 2004. *Matemática Numérica.* Félix Varela, La Habana.

CANALES, C. 2006. *Resolución numérica de sistemas de ecuaciones no lineales*. Trabajo de diploma. Universidad Pontificia Comillas. Madrid.

CONDE, C. & SCHIAVI, E. 2010. *Métodos numéricos de resolución de ecuaciones no lineales*. Disponible en: http://ocw.upm.es/matematica-aplicada/programacion-y-metodos-numericos/contenidos/TEMA_8/Apuntes/EcsNoLin.pdf

MAHMODIAN, M.; RAHMANI, R.; TASLIMI, E. & MEKHILEF, S. 2012. Step By Step Analyzing, Modeling and Simulation of Single and Double Array PV system in Different Environmental Variability. International Conference on Future Environment and Energy IPCBEE, Singapur, vol. 28: 37-42.

PRESS, W.; FLANNERY, B.; TEUKOLSKY, S. & VETTERLING, W. 2007. *Numerical Recipes: The Art of Scientific Computing.* 3er ed. Cambridge University Press, New York.

YANG, W.; CAO, W.; CHUNG, T. S. & MORRIS, J. 2005. *Applied Numerical Methods Using MATLAB*. John Wiley & Sons, New Jersey. 528 p.