Métodos Numéricos | Laboratorio 3

Ricardo Largaespada

27 de marzo de 2025

1. Introducción

Durante la clase anterior, trabajamos con métodos cerrados para encontrar raíces de ecuaciones. En dichos métodos, la raíz se encuentra dentro de un intervalo predeterminado, delimitado por un valor inferior y uno superior. La aplicación repetida de estos métodos genera aproximaciones cada vez más cercanas a la raíz. Se dice que estos métodos son *convergentes* porque se acercan progresivamente a la raíz a medida que se avanza en el cálculo.

En esta sesión, estudiaremos los llamados métodos abiertos. A diferencia de los métodos cerrados, estos se basan en fórmulas que requieren únicamente de un valor inicial x (o un par de valores), pero que no necesariamente encierran la raíz dentro de un intervalo. Como resultado, estos métodos pueden —en algunos casos— alejarse de la raíz verdadera a medida que se itera. Sin embargo, cuando convergen, suelen hacerlo mucho más rápido que los métodos cerrados.
Comenzaremos el análisis de los métodos abiertos con una versión simple que es útil para ilustrar su funcionamiento general y también para introducir formalmente el concepto de convergencia.

2. Iteración por Punto Fijo

El método de iteración por punto fijo es una técnica sencilla y poderosa para aproximar raíces de ecuaciones no lineales. Es un método abierto que requiere una buena elección del valor inicial para asegurar su convergencia.

Idea del método

Partimos de una ecuación de la forma: $f(x) = 0$ y la reescribimos como: $x = g(x)$.
Esta nueva forma se denomina **forma iterativa**, ya que nos permite generar una secuencia de valores:

$$x_{i+1} = g(x_i)$$

A partir de un valor inicial x_0, se genera un nuevo valor usando $g(x_0)$, luego $g(x_1)$, y así sucesivamente. Este proceso continúa hasta que el valor calculado cambie muy poco con

respecto al anterior (es decir, hasta que el error relativo aproximado sea menor que un valor de tolerancia predefinido).

Criterio de convergencia

El método de punto fijo converge si en la vecindad de la raíz se cumple:

$$|g'(x)| < 1$$

Esto significa que la función iterativa $g(x)$ debe tener una pendiente suficientemente pequeña cerca de la solución para asegurar que las sucesivas aproximaciones se acerquen a la raíz.

Cálculo del error

El error relativo aproximado en cada iteración puede estimarse con:

$$\varepsilon_a = \left| \frac{x_{i+1} - x_i}{x_{i+1}} \right| \cdot 100\,\%$$

Ventajas y desventajas

- **Ventajas:** Es fácil de implementar y no requiere derivadas.
- **Desventajas:** No siempre converge. La elección de la función $g(x)$ es clave.

Pseudocódigo del método

```
1:  function PuntoFijo(x0, es, imax)
2:      xr ← x0
3:      iter ← 0
4:      repeat
5:          xrold ← xr
6:          xr ← g(xrold)
7:          iter ← iter + 1
8:          if xr ≠ 0 then
9:              ea ← |(xr − xrold)/xr| · 100
10:         end if
11:     until ea <es or iter ≥ imax
12:     return xr
13: end function
```

Observación importante

No toda reescritura de $f(x) = 0$ como $x = g(x)$ asegura convergencia. Por eso, antes de aplicar el método es recomendable verificar gráficamente o analíticamente si la función $g(x)$ cumple con el criterio de convergencia en el intervalo de interés.

3. Método de Newton-Raphson

El método de Newton-Raphson es uno de los métodos abiertos más populares para encontrar raíces de funciones no lineales. Su principal ventaja es que, si se aplica correctamente y con una buena estimación inicial, puede converger de manera muy rápida a la solución.
El único inconveniente es que requiere conocer no solo la función $f(x)$, sino también su derivada $f'(x)$. Por esta razón, es aplicable únicamente cuando la derivada puede ser calculada con facilidad.

Derivación de la fórmula

La fórmula del método de Newton-Raphson se puede obtener a partir de la serie de Taylor de $f(x)$ centrada en el punto x_i:

$$f(x_{i+1}) = f(x_i) + f'(x_i)(x_{i+1} - x_i) + O((x_{i+1} - x_i)^2) \tag{1}$$

Si asumimos que x_{i+1} es una raíz de $f(x)$, entonces $f(x_{i+1}) = 0$. Sustituyendo en la ecuación anterior y descartando el término de orden superior, se obtiene:

$$0 = f(x_i) + f'(x_i)(x_{i+1} - x_i) \tag{2}$$

Despejando x_{i+1} se llega a la forma iterativa:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{3}$$

Esta fórmula se aplica de forma repetida, generando nuevas aproximaciones de la raíz en cada paso.

Convergencia del método

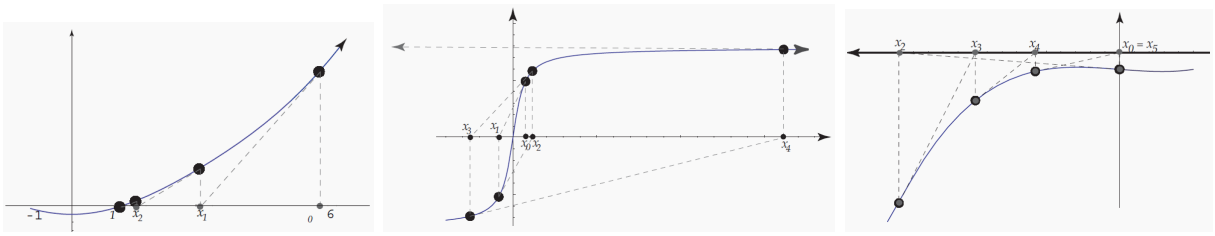
Si x es la raíz exacta, el error en la iteración i se define como $E_i = x - x_i$. Puede demostrarse que el error en la siguiente iteración está dado por:

$$E_{i+1} = -\frac{f''(x_i)}{2f'(x_i)} E_i^2$$

Este resultado implica que el método tiene **convergencia cuadrática**: si x_i está suficientemente cerca de la raíz, el número de cifras significativas se duplica aproximadamente en cada iteración.

3.1. Algoritmo del método

La Figura (1) muestra gráficamente cómo funciona la iteración: se traza la recta tangente a $f(x)$ en el punto x_i, y la intersección de esta recta con el eje x se usa como la siguiente aproximación x_{i+1}.
El algoritmo puede describirse así:



(a) Converge (b) Diverge (c) Ciclo

Figura 1: Comportamiento del método de Newton-Raphson

1. Elegir un valor inicial x_0 como estimación de la raíz.
2. Calcular $\Delta x = -\dfrac{f(x)}{f'(x)}$.
3. Actualizar $x \leftarrow x + \Delta x$.
4. Repetir los pasos 2 y 3 hasta que $|\Delta x| < \varepsilon$ (criterio de convergencia).

Nota: Aunque el método es rápido si se inicia cerca de la raíz, puede fallar si la derivada se anula o si se está lejos de la solución. Por eso, a veces se combina con métodos cerrados como la bisección.

3.2. Versión segura del método

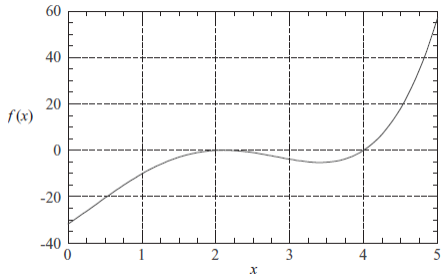
Una alternativa práctica es usar una versión "híbrida"que combina Newton-Raphson con bisección. Se parte de un intervalo (a, b) que encierra la raíz, se inicia con el punto medio, y si una iteración de Newton-Raphson sale del intervalo, se sustituye por una iteración de bisección.
El siguiente archivo implementa esta versión:

```
function root = newtonRaphson_segura(func, dfunc, a, b, tol)
% Método de Newton–Raphson combinado con bisección para encontrar una raíz de f(x) = 0.
% USO:
%   root = newtonRaphson_segura(func, dfunc, a, b, tol)
% ENTRADAS:
%   func  – función manejable que evalúa f(x)
%   dfunc – función manejable que evalúa f'(x)
%   a, b  – extremos del intervalo que encierra la raíz
%   tol   – tolerancia de error (opcional, por defecto es 1.0e6 * eps)
% SALIDA:
%   root – raíz encontrada (o NaN si no hay convergencia)
if nargin < 5; tol = 1.0e6 * eps; end          % Tolerancia por defecto
```

```
fa = feval(func, a); fb = feval(func, b);        % Evaluar extremos del intervalo
% Verifica si alguno de los extremos ya es raíz
if fa == 0; root = a; return
elseif fb == 0; root = b; return
end
% Verifica que la raíz esté encerrada
if fa * fb > 0
    error('La raíz no está encerrada en (a, b)')
end
x = (a + b) / 2;                                 % Punto inicial: punto medio
for i = 1:30
    fx = feval(func, x);                         % Evaluar f(x)
    if abs(fx) < tol                             % Verifica si ya encontró la raíz
        root = x; return
    end
    % Ajuste del intervalo usando bisección
    if fa * fx < 0
        b = x; fb = fx;
    else
        a = x; fa = fx;
    end
    dfx = feval(dfunc, x);                       % Evaluar f'(x)
    if abs(dfx) == 0                             % Evitar división por cero
        dx = b - a;
    else
        dx = -fx / dfx;                          % Paso de Newton-Raphson
    end
    x = x + dx;                                  % Nueva estimación
    % Si la estimación sale del intervalo, usar bisección
    if (x < a) || (x > b)
        dx = (b - a) / 2;
        x = a + dx;
    end
    % Verificación de convergencia
    if abs(dx) < tol * max(abs(b), 1.0)
        root = x; return
    end
end
root = NaN;                                      % Si no converge en 30 iteraciones
end
```

Ejemplo 3.1. Encuentre la menor raíz positiva de

$$f(x) = x^4 - 6.4x^3 + 6.45x^2 + 20.538x - 31.752$$



Al observar el gráfico, parece que hay una raíz doble cerca de $x = 2$. La bisección no funcionaría bien en este caso, ya que la función no cambia de signo en una raíz múltiple. Sin embargo, el método de Newton-Raphson estándar sí puede aplicarse.
El siguiente programa muestra cómo usar este método para encontrar la raíz:

```
function [root, numIter] = newton_simple(func, dfunc, x, tol)
% Versión simple del método de Newton–Raphson.
% USO:
%   [root, numIter] = newton_simple(func, dfunc, x, tol)
% ENTRADAS:
%   func  – función manejable que evalúa f(x)
%   dfunc – función manejable que evalúa f'(x)
%   x     – valor inicial (x0)
%   tol   – tolerancia al error (opcional, por defecto 1.0e6*eps)
% SALIDAS:
%   root    – aproximación de la raíz
%   numIter – número de iteraciones realizadas
if nargin < 4; tol = 1.0e6 * eps; end    % Establecer tolerancia
   por defecto
numIter = 0;
for i = 1:50                             % Máximo de 50 iteraciones
    fx = feval(func, x);                 % Evaluar función
    dfx = feval(dfunc, x);               % Evaluar derivada
    if dfx == 0                          % Evita división por cero
        root = NaN; return
    end
    dx = -fx / dfx;                      % Paso de Newton
    x = x + dx;                          % Actualiza estimación
    if abs(dx) < tol                     % Criterio de convergencia
        root = x; numIter = i; return
    end
end
root = NaN;                              % No convergió en 50 iteraciones
end
```

Las funciones utilizadas en este ejemplo son:

```
function y = fej3_1(x)
y = x^4 - 6.4*x^3 + 6.45*x^2 + 20.538*x - 31.752;
```

```
function y = dfej3_1(x)
y = 4.0*x^3 - 19.2*x^2 + 12.9*x + 20.538;
```

Y el resultado se obtiene mediante:

```
[root,numIter] = newton_simple(@fej3_1,@dfej3_1,2.0)
root =
    2.1000
numIter =
    35
```

3.3. Raíces múltiples y convergencia modificada

Cuando una función tiene una raíz múltiple, el método de Newton-Raphson ya no tiene convergencia cuadrática, sino **lineal**. Esto explica por qué se requieren más iteraciones en el ejemplo anterior.
Para mejorar la convergencia en estos casos, se puede modificar la fórmula original del método así:

$$x_{i+1} = x_i - m\frac{f(x_i)}{f'(x_i)} \tag{4}$$

donde m es la multiplicidad conocida de la raíz (en el ejemplo anterior, $m = 2$). Esta modificación permite recuperar la convergencia rápida esperada.

4. Sistema de Ecuaciones No Lineales

Hasta ahora, hemos trabajado con funciones de una sola variable, es decir, con ecuaciones de la forma $f(x) = 0$. Sin embargo, muchos problemas reales requieren resolver sistemas de ecuaciones no lineales con varias variables. Este tipo de problemas puede expresarse como:

$$f(x) = 0$$

donde $f(x)$ es un vector de funciones y x es un vector de variables. En notación escalar, esto equivale a:

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ &\vdots \\ f_n(x_1, x_2, \ldots, x_n) &= 0 \end{aligned}$$

Resolver este tipo de sistemas es más complejo que el caso de una sola variable. Uno de los principales desafíos es que no existe un método sencillo para asegurar que la solución esté contenida dentro de un intervalo determinado. Por lo tanto, es fundamental partir de una buena estimación inicial, la cual muchas veces proviene del análisis físico del problema.
Uno de los métodos más utilizados para resolver sistemas de ecuaciones no lineales es el **método de Newton-Raphson**, ya que es relativamente fácil de implementar y converge rápidamente si se parte de una buena estimación inicial.

4.1. El método de Newton-Raphson para sistemas

Para extender el método de Newton-Raphson al caso de varias variables, utilizamos una expansión de Taylor de primer orden para cada función f_i alrededor de un punto x:

$$f_i(x + \Delta x) = f_i(x) + \sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j} \Delta x_j + O(\Delta x^2) \tag{5}$$

Descartando los términos de segundo orden, esta expresión se puede escribir de manera matricial como:

$$f(x + \Delta x) = f(x) + J(x)\Delta x$$

donde $J(x)$ es la matriz Jacobiana, cuyos elementos son:

$$J_{ij} = \frac{\partial f_i}{\partial x_j}$$

Esta matriz contiene las derivadas parciales de todas las funciones con respecto a todas las variables, y representa la generalización de la derivada en el caso multivariable.
Ahora, si asumimos que x es la estimación actual del vector solución, buscamos una corrección Δx tal que:

$$f(x + \Delta x) = 0$$

Sustituyendo en la expansión anterior, obtenemos:

$$J(x)\Delta x = -f(x)$$

Este sistema de ecuaciones lineales se puede resolver para encontrar Δx, y luego se actualiza el valor de x:

$$x \leftarrow x + \Delta x$$

Pasos del algoritmo de Newton-Raphson para sistemas

1. Estimar un valor inicial para el vector x.
2. Evaluar $f(x)$.
3. Calcular la matriz Jacobiana $J(x)$.
4. Resolver el sistema lineal $J(x)\Delta x = -f(x)$.
5. Actualizar: $x \leftarrow x + \Delta x$.
6. Repetir desde el paso 2 hasta que $||\Delta x||$ sea menor que la tolerancia.

Implementación

En el siguiente código, se implementa el método de Newton-Raphson para sistemas de ecuaciones. La función requiere que el usuario proporcione una función que evalúe el vector $f(x)$. La matriz Jacobiana se aproxima por diferencias finitas.

```
function root = newtonRaphson_sistema(func, x, tol)
% Método de Newton–Raphson para encontrar la raíz de un sistema de
% ecuaciones no lineales: f_i(x_1,x_2,...,x_n) = 0, para i = 1,2,...,n.
% USO:
%   root = newtonRaphson_sistema(func, x, tol)
% ENTRADAS:
%   func – función manejable que retorna el vector [f1, f2, ..., fn]
%   x    – vector columna con la estimación inicial [x1; x2; ...; xn]
%   tol  – tolerancia al error (opcional, por defecto 1.0e4 * eps)
% SALIDA:
%   root – vector solución (aproximación de la raíz)
if nargin == 2; tol = 1.0e4 * eps; end              % Tolerancia por defecto
if size(x,1) == 1; x = x'; end                      % Asegura que x sea columna
for i = 1:30                                        % Máximo 30 iteraciones
    [jac, f0] = jacobian(func, x);                  % Calcula Jacobiana y f(x)
    % Criterio de convergencia por magnitud de f(x)
    if sqrt(dot(f0, f0) / length(x)) < tol
        root = x;
        return
    end
    dx = jac \ (-f0);                               % Resuelve J dx = -f
    x = x + dx;                                     % Actualiza solución
    % Criterio de convergencia por cambio en x
    if sqrt(dot(dx, dx) / length(x)) < tol * max(abs(x), 1.0)
        root = x;
        return
    end
end
error('Muchas iteraciones: el método no convergió.');
end
```

```
function [jac, f0] = jacobian(func, x)
% Calcula la matriz Jacobiana y el vector f(x) en el punto x.
% USO:
%   [jac, f0] = jacobian(func, x)
% ENTRADAS:
%   func - función manejable que devuelve el vector f(x)
%   x    - vector columna de variables [x1; x2; ...; xn]
% SALIDAS:
%   jac - matriz Jacobiana (n x n) aproximada por diferencias finitas
%   f0  - evaluación de f(x) en el punto actual
h = 1.0e-4;                          % Incremento para diferencia finita
n = length(x);                       % Número de variables
jac = zeros(n);                      % Inicializa matriz Jacobiana
f0 = feval(func, x);                 % Evalúa f(x) en el punto actual
for i = 1:n
    temp = x(i);                     % Guarda el valor original de x(i)
    x(i) = temp + h;                 % Perturba x(i)
    f1 = feval(func, x);             % Evalúa f(x + h)
    x(i) = temp;                     % Restaura el valor original
    jac(:, i) = (f1 - f0) / h;       % Aproxima derivada parcial
end
end
```

Nota: La matriz Jacobiana se vuelve a calcular en cada iteración. Esto puede representar un alto costo computacional si el número de variables n es grande o si $f(x)$ es costoso de evaluar. Si se está cerca de la solución, es posible fijar $J(x)$ y no recalcularla en cada paso para ahorrar tiempo.

Ejemplo 4.1. Encuentre una solución del siguiente sistema:

$$\begin{cases} \sin x + y^2 + \ln z - 7 = 0 \\ 3x + 2^y - z^3 + 1 = 0 \\ x + y + z - 5 = 0 \end{cases}$$

usando el método `newtonRaphson_sistema`, iniciando en el punto $[1, 1, 1]$.

Si denotamos $x = x_1$, $y = x_2$ y $z = x_3$, la función que define el sistema se implementa en:

```
function y = fej4_1(x)
y = [sin(x(1)) + x(2)^2 + log(x(3)) - 7; 3*x(1) + 2^x(2) - x(3)^3 + 1; x(1) + x(2) + x(3) - 5];
```

Para resolver el sistema, se ejecuta: `newtonRaphson_sistema(@fej4_1, [1; 1; 1])`.
La solución obtenida es: $x = 0.5991$, $y = 2.3959$, $z = 2.0050$.

5. Problemas Propuestos

Problema 5.1. a) Aplique el método de Newton-Raphson a la función $f(x) = \tanh(x^2 - 9)$ con $x_0 = 3.2$. Realice al menos cuatro iteraciones.

b) ¿Converge el método a la raíz real? Apoye su análisis con una gráfica.

Problema 5.2. La función $f(x) = x^3 - 2x^2 - 4x + 8$ tiene una raíz doble en $x = 2$. Resuélvala con:

a) Newton-Raphson estándar

b) Newton-Raphson modificado

Compare la velocidad de convergencia usando $x_0 = 1.2$.

Problema 5.3. Resuelva el siguiente sistema por:

a) Iteración de punto fijo

b) Newton-Raphson

$$\begin{cases} y &= -x^2 + x + 0.75 \\ y + 5xy &= x^2 \end{cases}$$

Use $x = y = 1.2$ como valor inicial. Analice los resultados.

Problema 5.4. Encuentre las raíces del siguiente sistema:

$$\begin{cases} (x-4)^2 + (y-4)^2 &= 5 \\ x^2 + y^2 &= 16 \end{cases}$$

Estime gráficamente los valores iniciales y refine la solución con Newton-Raphson.

Problema 5.5. Repita el problema (5.4) con el sistema:

$$\begin{cases} y &= x^2 + 1 \\ y &= 2\cos x \end{cases}$$

Determine la raíz positiva.

Problema 5.6. El balance de masa de un contaminante en un lago bien mezclado se expresa como:

$$V\frac{dc}{dt} = W - Qc - kV\sqrt{c}$$

Dado: $V = 1 \times 10^6\,\mathrm{m}^3$, $Q = 1 \times 10^5\,\mathrm{m}^3/\text{año}$, $W = 1 \times 10^6\,\mathrm{g}/\text{año}$, $k = 0.25\,\mathrm{m}^{0.5}/\text{año}$. Para el problema, se puede escribir la ecuación como:

$$c = \left(\frac{W - Qc}{kV}\right)^2 \quad \text{o bien} \quad c = \frac{W - kV\sqrt{c}}{Q}$$

De las dos formas, solo una converge para $2 < c < 6$. Determine cuál y explique por qué.

Problema 5.7. Desarrolle un programa amigable para el usuario que implemente el método de Newton-Raphson.

Problema 5.8. Desarrolle un programa amigable para el método de Newton-Raphson aplicado a sistemas de ecuaciones.