Tarea 12

CIMAT

Nombre:	**Jairo Saul Diaz Soto**
Maestría:	Ciencias Computacionales
Modulo:	Métodos Numéricos
Instructor:	Dr. Luis Daniel Blanco Cocom
Fecha de entrega:	2023 - 11 - 12

1. Diferenciación

Calcular la derivada de una función es sumamente importante para determinar y resolver muchos tipos de problemas, en ocasiones es difícil la evaluación o quizá no se tiene la función explicita pero se conocen algunos puntos de la curva.

Por esto anterior, es que existen diferentes métodos numéricos que nos permiten determinar la evaluación de la derivada en algún punto en particular, existen diferentes métodos los cuales son más precisos pero más complejos y algunos más sencillos aunque carezcan de buenas aproximaciones, a continuación, se muestran algunas subrutinas para calcular derivadas

Algorithm 1: Derivada hacia adelante

Input: $f(x)$, $f(x+h)$, h
Output: Derivada hacia adelante $f'(x)$

1. $f'(x) \leftarrow \frac{f(x+h)-f(x)}{h}$;
2. **return** $f'(x)$;

Algorithm 2: Derivada hacia atrás

Input: $f(x)$, $f(x-h)$, h
Output: Derivada hacia atrás $f'(x)$

1. $f'(x) \leftarrow \frac{f(x)-f(x-h)}{h}$;
2. **return** $f'(x)$;

Algorithm 3: Derivada centrada

Input: $f(x-h)$, $f(x+h)$, h
Output: Derivada centrada $f'(x)$

1. $f'(x) \leftarrow \frac{f(x+h)-f(x-h)}{2h}$;
2. **return** $f'(x)$;

Algunas de las aplicaciones interesantes de las derivadas, para obtener más herramientas a la hora de resolver problemas, son la matriz jacobiana y la matriz hessiana, las cuales se muestran a continuación algoritmos para implementarlas

2. Sistemas de ecuaciones no lineales

Los sistemas de ecuaciones no lineales son más comunes de lo que se muestra en la academia, es por eso que es de suma importancia el poder resolver estos, sobre todo de manera numérica, pues no siempre es sencillo obtener una solución analítica.

Existen diferentes enfoques para abordar este tipo de problemas, a continuación se despliega una serie de algoritmos los cuales tratan este tipo de sistemas

Algorithm 4: Derivada de 3 puntos hacia adelante

Input: $f(x)$, $f(x+h)$, $f(x+2h)$, h
Output: Derivada de 3 puntos hacia adelante $f'(x)$

$f'(x) \leftarrow \frac{1}{2h}\left(-3f(x) + 4f(x+h) - f(x+2h)\right)$;
return $f'(x)$;

Algorithm 5: Derivada de 3 puntos centrada

Input: $f(x-h)$, $f(x+h)$, h
Output: Derivada de 3 puntos centrada $f'(x)$

$f'(x) \leftarrow \frac{f(x+h)-f(x-h)}{2h}$;
return $f'(x)$;

Algorithm 6: Derivada de 5 puntos centrada

Input: $f(x-2h)$, $f(x-h)$, $f(x+h)$, $f(x+2h)$, h
Output: Derivada de 5 puntos centrada $f'(x)$

$f'(x) \leftarrow \frac{1}{12h}\left(f(x-2h) - 8f(x-h) + 8f(x+h) - f(x+2h)\right)$;
return $f'(x)$;

Algorithm 7: Derivada de 5 puntos hacia adelante

Input: $f(x)$, $f(x+h)$, $f(x+2h)$, $f(x+3h)$, $f(x+4h)$, h
Output: Derivada de 5 puntos hacia adelante $f'(x)$

$f'(x) \leftarrow \frac{1}{12h}\left(-25f(x) + 48f(x+h) - 36f(x+2h) + 16f(x+3h) - 3f(x+4h)\right)$;
return $f'(x)$;

Algorithm 8: Segunda derivada por punto fijo

Input: $f(x-h)$, $f(x)$, $f(x+h)$, h
Output: Segunda derivada por punto fijo $f''(x)$

$f''(x) \leftarrow \frac{1}{h^2}\left(f(x-h) - 2f(x) + f(x+h)\right)$;
return $f''(x)$;

Algorithm 9: Cálculo del Jacobiano de una función

Input: Vector de funciones f, Punto x, Paso h, Dimensión sz
Output: Matriz Jacobiana J

```
J ← InicializarMatriz(sz, sz);
for i ← 0 to sz − 1 do
    for j ← 0 to sz − 1 do
        J[i][j] ← (f_i(x + h_j) − f(x − h_j)) / (2h_j);
return J;
```

$J[i][j] \leftarrow \frac{f_i(x+h_j)-f(x-h_j)}{2h_j}$

Algorithm 10: Cálculo del Hessiano de una función

Input: Vector de funciones f, Punto x, Paso h, Dimensión sz

Output: Matriz Hessiana H

$H \leftarrow$ InicializarMatriz(sz, sz);
for $i \leftarrow 0$ **to** $sz - 1$ **do**
  for $j \leftarrow 0$ **to** $sz - 1$ **do**
    $H[i][j] \leftarrow \frac{1}{h_i h_j} \cdot (f_i(x + h_i + h_j) + f_i(x) - f_i(x + h_i) - f_i(x + h_j))$;
return H;

Algorithm 11: Sistemas de Ecuaciones No Lineales por Punto Fijo

Input: Funciones f, Punto inicial x_0, Dimensión sz, Tolerancia TOL, Número máximo de iteraciones $nMax$

Output: Punto de convergencia x

$i \leftarrow 0$;
$x \leftarrow x_0$;
while $i < nMax$ **do**
  `// Evaluar el punto en la función`
  $x \leftarrow f(x_0)$;
  `// Verificar la tolerancia`
  if *inf_norm*$(x_0, x, sz) \leq TOL$ **then**
    `// Se alcanzó la convergencia`
    break;
  `// Actualizar el punto para la siguiente iteración`
  for $j \leftarrow 0$ **to** $sz - 1$ **do**
    $x_0[j] \leftarrow x[j]$;
  $i \leftarrow i + 1$;
if $i \geq nMax$ **then**
  `// No se alcanzó la convergencia`
  `printf`("No se alcanzó la convergencia tras %d iteraciones", i);
else
  `printf`("El programa finalizó tras %d iteraciones", i);
return x;

Algorithm 12: Método de Newton para sistemas de ecuaciones no lineales

Input: f: Conjunto de funciones vectoriales, x_0: Punto inicial, h: Tamaño de paso, sz: Dimensión, TOL: Tolerancia, $nMax$: Número máximo de iteraciones

Output: x_0: Punto aproximado

$i \leftarrow 0$;
while $i < nMax$ **do**
 $b \leftarrow \text{zeros}(sz)$;
 for $j \leftarrow 0$ **to** $sz - 1$ **do**
  $b[j] \leftarrow -f[j](x_0)$;
 $A \leftarrow$ `JacobianMatrix`(f, x_0, h, sz);
 $qr \leftarrow$ `QR`(A, sz);
 $qt \leftarrow$ `TransposeMatrix`$(qr.Left, sz, sz)$;
 $b1 \leftarrow$ `MultiplyMatrixVector`(qt, b, sz, sz);
 $x \leftarrow$ `USolve`$(qr.Right, b1, sz)$;
 for $j \leftarrow 0$ **to** $sz - 1$ **do**
  $x_0[j] \leftarrow x_0[j] + x[j]$;
 if $||x|| < TOL$ **then**
  break;
 $i \leftarrow i + 1$;
if $i \geq nMax$ **then**
 Imprimir(”No se alcanzó la convergencia tras i iteraciones”);
else
 Imprimir(.El programa finalizó tras $i + 1$ iteraciones”);

Algorithm 13: Método de Broyden para sistemas de ecuaciones no lineales

Input: f: Conjunto de funciones vectoriales, x_0: Punto inicial, h: Tamaño de paso, sz: Dimensión, TOL: Tolerancia, $nMax$: Número máximo de iteraciones

Output: x_n: Punto aproximado

```
A_0 ← JacobianMatrix(f, x_0, h, sz);
v ← zeros(sz);
s, z, u, p ← zeros(sz);
xn ← zeros(sz);
w, y, temp ← zeros(sz);
A_inv ← InverseMatrix(A_0, sz);
for i ← 0 to sz − 1 do
    v[i] ← f[i](x_0);
s ← MultMatrixVector(A_inv, v, sz, sz);
for i ← 0 to sz − 1 do
    s[i] ← −1,0 × s[i];
    xn[i] ← x_0[i] + s[i];
k ← 1;
while k < nMax do
    for i ← 0 to sz − 1 do
        w[i] ← v[i];
        v[i] ← f[i](xn);
        y[i] ← v[i] − w[i];
    z ← MultMatrixVector(A_inv, y, sz, sz);
    for i ← 0 to sz − 1 do
        z[i] ← −1,0 × z[i];
    p ← −1,0× MultMatrixVector(s, z, sz);
    u ← MultMatrixVector(A_inv, s, sz, sz);
    for i ← 0 to sz − 1 do
        temp[i] ← s[i] + z[i];
    for i ← 0 to sz − 1 do
        for j ← 0 to sz − 1 do
            A_inv[i][j] ← A_inv[i][j] + (1,0/p) × (temp[i] × u[j]);
    s ← MultMatrixVector(A_inv, v, sz, sz);
    for i ← 0 to sz − 1 do
        s[i] ← −1,0 × s[i];
        xn[i] ← xn[i] + s[i];
    if ||s|| < TOL then
        break;
    k ← k + 1;
if k ≥ nMax then
    Imprimir("No se alcanzó la convergencia tras k iteraciones");
else
    Imprimir("El programa finalizó tras k + 1 iteraciones");
```

Algorithm 14: Método de gradiente conjugado de Fletcher - Reeves

Input: f: Conjunto de funciones vectoriales, x_0: Punto inicial, dh: Tamaño de paso, sz: Dimensión, TOL: Tolerancia, $nMax$: Número máximo de iteraciones

Output: x_n: Punto aproximado

$z, z_0, j, jt, g_0, gv_0 \leftarrow$ 0,0, 0,0, zeros(sz, sz), zeros(sz, sz), 0,0, zeros(4), zeros(3), zeros(4), 0,0;
$temp, zv, a \leftarrow$ zeros(sz), zeros(sz), zeros(4);
$k \leftarrow 0$;
while $k < nMax$ **do**
    $z \leftarrow \nabla g(x)$;
    if $k > 0$ **then**
        $g_0 \leftarrow \mathrm{dot}(g, g, sz)$;
        $gv_0 \leftarrow \mathrm{dot}(gv, gv, sz)$;
        for $i \leftarrow 0$ **to** $sz - 1$ **do**
            $z[i] \leftarrow z[i] + (g_0/gv_0) \times zv[i]$;
    $g[1] \leftarrow 0{,}0$;
    for $i \leftarrow 0$ **to** $sz - 1$ **do**
        $g[1] \leftarrow g[1] + f[i](x_0) \times f[i](x_0)$;
    $z_0 \leftarrow \sqrt{\mathrm{dot}(z, z, sz)}$;
    if $|z_0| < 1e - 12$ **then**
        Imprimir(”Gradiente 0”);
        break;
    z/z_0;
    $a[3] \leftarrow 1{,}0$;
    $g[3] \leftarrow g(x_0[i] - (a[3] \times z[i]))$;
    while $g[3] \geq g[1]$ **do**
        $a[3] \leftarrow a[3]/2{,}0$;
        $g[3] \leftarrow g(x_0[i] - (a[3] \times z[i]))$;
    if $a[3] < 1e - 12/2$ **then**
        Imprimir(”No hay mejora”);
        break;
    $a[2] \leftarrow a[3]/2{,}0$;
    $g[2] \leftarrow g(x_0[i] - (a[2] \times z[i]))$;
    $h[0] \leftarrow (g[2] - g[1])/a[2]$;
    $h[1] \leftarrow (g[3] - g[2])/(a[3] - a[2])$;
    $h[2] \leftarrow (h[1] - h[0])/a[3]$;
    $a[0] \leftarrow 0{,}5 \times (a[2] - (h[0]/h[2]))$;
    $g[0] \leftarrow x_0[i] - (a[0] \times z[i])$;
    $y \leftarrow 0$;
    $g_{\min} \leftarrow min\{g_i\}$;
    $x_0 \leftarrow x_0 - (a_{min} \times z)$;
    if $|g_{min} - g[1]| < TOL$ **then**
        break;
    for $i \leftarrow 0$ **to** 3 **do**
        $gv[i] \leftarrow g[i]$;
        $zv[i] \leftarrow z[i]$;
    $k \leftarrow k + 1$;
if $k \geq nMax$ **then**
    Imprimir(”No se alcanzó la convergencia tras k iteraciones”);
else
    Imprimir(.El programa finalizó tras $k + 1$ ”iteraciones”);

Primer derivada en 1.3 con h = 0.1		
f'(x)	error abs	Metodo
28.191100	1.909400	Derivada hacia delante
24.527000	1.754700	Derivada hacia atras
26.359050	0.077350	Derivada centrada
Primer derivada en 1.3 con h = 0.01		
f'(x)	error abs	Metodo
26.465000	0.183300	Derivada hacia delante
26.100000	0.181700	Derivada hacia atras
26.282500	0.000800	Derivada centrada
Segunda derivada en 1.3		
f''(x)	error abs	h
36.641000	0.047470	0.1
36.500000	0.093530	0.01

Figura 1: Calculo de derivadas utilizando varios métodos evaluado en un punto.

Solucion sistema 1			
x1	x2	x3	Metodo
0.500000	0.000000	-0.523599	Punto fijo
0.500000	0.000000	-0.523599	Newton
0.500000	0.000000	-0.523599	Broyden
0.501106	0.000130	-0.523642	Gradiente
Solucion sistema 2			
x1	x2	Metodo	
0.000000	3.000000	Punto fijo	
-0.000000	3.000000	Newton	
-0.000000	3.000000	Broyden	
-0.006893	2.999851	Gradiente	

Figura 2: Resolución de un dos sistemas de ecuaciones empleando diferentes métodos, las tolerancias utilizadas son de 1e-12, 1e-12 y 1e-6 para verificar el cambio de gradiente, la mejora de minimización la tolerancia del método respectivamente, las primeras 2 son fijas.

3. Resultados

A continuación se muestran resultados obtenidos de aplicar estos métodos

3.000000	0.001000	-0.001000
0.200000	-32.400000	0.995004
-0.099005	-0.099005	20.000000

Figura 3: Matriz Jacobiana para uno de los sistemas anteriores evaluado en un punto.

0.000000	-0.000000	0.000000
0.000000	-162.000000	0.000000
0.000000	0.000000	0.000000

Figura 4: Matriz Hessiana para uno de los sistemas anteriores evaluado en un punto.

4. Conclusión

Se puede observar que aunque hay en el caso de los métodos de diferenciación, algunos demasiados sencillos, tienen muy buenas aproximaciones, sin embargo, hay que tener bastante cuidado a la hora de elegir el tamaño de paso para la derivada ya que se puede inducir un error de representación.

Mientras que para los métodos para sistemas no lineales, resultan sumamente eficaces, pues en muy pocas iteraciones, estos tienen aproximaciones demasiado buenas.

5. Referencias

[1] Burden, R.L., Faires, J.D. and Burden, A.M. 2015. *Numerical analysis.* Cengage learning.

[2] Quarteroni, A., Sacco, R. and Saleri, F. 1999. *Numerical Mathematics.* Springer.