Revisión 1

Ronald Cardona Anderson Grajales
Sebastian Valencia Julian Sanchez

17 de septiembre de 2018

1. Solución numérica de ecuaciones de una variable

En esa seccion se describen algunos de los algoritmos numéricos principales para resolver ecuaciones de una variable. Estos algoritmos nos permiten ya sea encontrar intervalos en los que existe una raíz de la ecuación $f(x) = 0$ ó encontrar propiamente la raíz.

1.1. Búsquedas Incrementales

Leer x_0, $delta$, $niter$
$fx0 \leftarrow f(x_0)$
si $fx0 = 0$ **entonces**
  x_0 es raíz
en otro caso
  $x_1 \leftarrow x_0 + delta$
  $contador \leftarrow 1$
  $fx1 \leftarrow f(x_1)$
  mientras $fx0 * fx1 > 0$ ***y*** $contador < niter$ **hacer**
    $x_0 \leftarrow x_1$
    $fx0 \leftarrow fx1$
    $x_1 \leftarrow x_0 + delta$
    $fx1 \leftarrow f(x_1)$
    $contador \leftarrow contador + 1$
  fin
  si $fx1 = 0$ **entonces**
    x_1 es raíz
  si no, si $fx0 * fx1 < 0$ **entonces**
    Hay raíz entre x_0 y x_1
  en otro caso
    Fracasó en $niter$ iteraciones
  fin
fin

Algoritmo 1: Algoritmo de Búsqueda Incremental

1.2. Método de la bisección

```
Leer x_i, x_s, tolerancia, niter
fxi ← f(x_i)
fxs ← f(x_s)
si fxi = 0 entonces
    x_i es raíz
si no, si fxs = 0 entonces
    x_s es raíz
si no, si fxi * fxs < 0 entonces
    x_m ← (x_i + x_s)/2
    fxm = f(x_m)
    contador ← 1
    error ← tolerancia + 1
    mientras error > tolerancia y fxm ≠ 0 y contador < niter
     hacer
        si fxi * fxm < 0 entonces
            x_s ← x_m
            fxs ← fxm
        en otro caso
            x_i ← x_m
            fxi ← fxm
        fin
        x_aux ← x_m
        x_m ← (x_i + x_s)/2
        fxm ← f(x_m)
        error ← |x_m − x_aux|
        contador ← contador + 1
    fin
    si fxm = 0 entonces
        x_m es raíz
    si no, si error < tolerancia entonces
        x_m es aproximación a una raíz con una tolerancia =
         tolerancia
    en otro caso
        Fracasó en niter iteraciones
    fin
en otro caso
    El intervalo es inadecuado
fin
```

Algoritmo 2: Método de la Bisección

1.3. Método de la regla falsa

```
Leer $x_i$, $x_s$, $tolerancia$, $niter$
$fxi \leftarrow f(x_i)$
$fxs \leftarrow f(x_s)$
si $fxi = 0$ entonces
|  $x_i$ es raíz
si no, si $fxs = 0$ entonces
|  $x_s$ es raíz
si no, si $fxi * fxs < 0$ entonces
|  $x_m \leftarrow x_i - \frac{f(x_i)*(x_s-x_i)}{f(x_s)-f(x_i)}$
|  $fxm = f(x_m)$
|  $contador \leftarrow 1$
|  $error \leftarrow tolerancia + 1$
|  mientras $error > tolerancia$ y $fxm \neq 0$ y $contador < niter$
|   hacer
|   |  si $fxi * fxm < 0$ entonces
|   |  |  $x_s \leftarrow x_m$
|   |  |  $fxs \leftarrow fxm$
|   |  en otro caso
|   |  |  $x_i \leftarrow x_m$
|   |  |  $fxi \leftarrow fxm$
|   |  fin
|   |  $x_{aux} \leftarrow x_m$
|   |  $x_m \leftarrow x_i - \frac{f(x_i)*(x_s-x_i)}{f(x_s)-f(x_i)}$
|   |  $fxm \leftarrow f(x_m)$
|   |  $error \leftarrow |x_m - x_{aux}|$
|   |  $contador \leftarrow contador + 1$
|  fin
|  si $fxm = 0$ entonces
|  |  $x_m$ es raíz
|  si no, si $error < tolerancia$ entonces
|  |  $x_m$ es aproximación a una raíz con una tolerancia =
|  |   $tolerancia$
|  en otro caso
|  |  Fracasó en $niter$ iteraciones
|  fin
en otro caso
|  El intervalo es inadecuado
fin
```

Algoritmo 3: Método de la Regla Falsa

1.4. Método de Punto Fijo

Leer *tolerancia*, x_a, *niter*
$fx \leftarrow f(x_a)$
$contador \leftarrow 0$
$error \leftarrow tolerancia + 1$
mientras $fx \neq 0$ ***y*** $error > tolerancia$ ***y*** $contador < niter$ **hacer**
  $x_n \leftarrow g(x_a)$
  $fx \leftarrow f(x_n)$
  $error \leftarrow |x_n - x_a|$
  $x_a \leftarrow x_n$
  $contador \leftarrow contador + 1$
fin
si $fx = 0$ **entonces**
  x_a es raíz
si no, si $error < tolerancia$ **entonces**
  x_a es aproximación con una tolerancia = *tolerancia*
en otro caso
  El método fracasó en *niter* iteraciones
fin

Algoritmo 4: Método de Punto Fijo

1.5. Método de Newton

Leer tol, x_0, $niter$
$fx \leftarrow f(x_0)$
$dfx = f'(x_0)$
$contador \leftarrow 0$
$error \leftarrow tol + 1$
mientras $fx \neq 0$ ***y*** $dfx \neq 0$ ***y*** $error > tol$ ***y*** $contador < niter$ **hacer**
| $x_1 \leftarrow x_0 - \frac{fx}{dfx}$
| $fx \leftarrow f(x_1)$
| $dfx \leftarrow f'(x_1)$
| $error \leftarrow |x_1 - x_0|$
| $x_0 \leftarrow x_1$
| $contador \leftarrow contador + 1$
fin
si $fx = 0$ **entonces**
| x_a es raíz
si no, si $error < tol$ **entonces**
| x_a es aproximación con una tolerancia = $tolerancia$
si no, si $dfx = 0$ **entonces**
| x_1 es una posible raíz múltiple
en otro caso
| El método fracasó en $niter$ iteraciones
fin

Algoritmo 5: Método de Newton

1.6. Método de la Secante

Leer x_0, x_1, $niter$, tol
$fx_0 \leftarrow f(x_0)$
si fx_0 = *0* **entonces**
| *imprimir* x_0 es raíz
en otro caso
| $fx_1 \leftarrow f(x_1)$
| $contador = 0$
| $error = tol + 1$
| $denominador \leftarrow fx_1 - fx_0$
| **mientras** $error > tol$ ***y*** $fx_1 \neq 0$ ***y*** $denominador \neq 0$ ***y*** $contador < niter$ **hacer**
| | $x_2 \leftarrow x_1 - \frac{fx_1 * (x_1 - x_0)}{denominador}$
| | $error \leftarrow \left|\frac{x_2 - x_1}{x_2}\right|$
| | $x_0 \leftarrow x_1$
| | $fx_0 \leftarrow fx_1$
| | $x_1 \leftarrow x_2$
| | $fx_1 \leftarrow f(x_1)$
| | $denominador \leftarrow fx_1 - fx_0$
| | $contador \leftarrow contador + 1$
| **fin**
| **si** $fx_1 = 0$ **entonces**
| | *imprimir* x_1 es raíz
| **si no, si** $error < tol$ **entonces**
| | *imprimir* x_1 es una aproximación con tolerancia = tol
| **si no, si** $denominador = 0$ **entonces**
| | *imprimir* probablemente existe una raiz multiple
| **en otro caso**
| | *imprimir* fracasó en $niter$ iteraciones
| **fin**
fin

Algoritmo 6: Método de la Secante

1.7. Método de Raices Multiples

Leer x_0, $niter$, tol
$fx \leftarrow f(x_0)$
$dfx \leftarrow f'(x_0)$
$ddfx \leftarrow f''(x_0)$
$denominador \leftarrow dfx^2 - fx * ddfx$
$contador \leftarrow 0$
$error \leftarrow tol + 1$
mientras $error > tol$ ***y*** $fx \neq 0$ ***y*** $denominador \neq 0$ ***y*** $contador < niter$ **hacer**
| $x_1 \leftarrow x_0 - \frac{fx*dfx}{denominador}$
| $fx \leftarrow f(x_1)$
| $dfx \leftarrow f'(x_1)$
| $ddfx \leftarrow f''(x_1)$
| $denominador \leftarrow dfx^2 - fx * ddfx$
| $error \leftarrow |x_1 - x_0|$
| $x_0 \leftarrow x_1$
| $contador \leftarrow contador + 1$
fin
si $fx = 0$ **entonces**
| *imprimir* x_0 es raíz
si no, si $error < tol$ **entonces**
| *imprimir* x_0 es una aproximación con tolerancia = tol
en otro caso
| *imprimir* fracasó en *niter* iteraciones
fin

Algoritmo 7: Método de Raices Multiples

2. Metodos de optimizacion para la solución numérica de ecuaciones de una variable

En algunos casos es necesario optimizar algunos métodos numéricos con la finalidad de alcanzar una tolerancia en el menor numero de iteraciones posible. Es por esto, que acá se presenta un método de optimización a el método de Punto Fijo, que hace que la convergencia sea mucho mas rápida y exacta.

2.1. Método de Steffensen

```
// Tomado de:
   https://en.wikipedia.org/wiki/Steffensen%27s_method
```

Leer x_0, $niter$, tol
$error$ = 1+tol
$contador = 0$
mientras $contador < niter$ ***y*** $error > tol$ **hacer**
| $x_1 = f(x_0)$
| $x_2 = f(x_1)$
| $f(x) = f(x_0)$
| $p = x_0 - \frac{(x_1 - x_2)^2}{x_2 - 2 * x_1 + x_0}$
| $error = |x - x_0|$
| $x_0 = p$
| $contador = contador + 1$
fin
si $error \leq tol$ **entonces**
| Hay una raíz en p
en otro caso
| El método fracasó después de $niter$ iteraciones
fin

Algoritmo 8: Método de Steffensen

3. Solución numérica de sistemas de ecuaciones lineales

Muchos problemas del mundo real se formulan como sistemas de ecuaciones de n variables y m incógnitas que bajo condiciones ideales(n y m no son valores muy grandes), se pueden resolver de manera analítica. Sin embargo, cuando n y m tienden a ser valores muy grandes la solución analítica a estos problemas es muy difícil de calcular ya que requiere de mucho tiempo y claramente no es la forma más eficiente hacerlo. Debido a esto, desde el campo del *Análisis Numérico* se plantean diversas formas computacionales, ya sean algoritmos u otras técnicas que nos permitan resolver estos sistemas rápidamente, teniendo en cuenta que hay una ***propagación de error*** en cada cálculo dependiendo de la capacidad de la computadora donde se ejecuten estos. En este documento se presentan algunos algoritmos numéricos que son de gran ayuda a la hora de resolver sistemas de ecuaciones lineales.

3.1. Determinantes

Sea $A = [a_{ij}]$ una matriz de tamaño $n \times n$. El cofactor C_{ij} de a_{ij} se define como $(-1)^{i+j} \det M_{ij}$, donde M_{ij} es la matriz de tamaño $(n-1) \times (n-1)$, que se obtiene al eliminar la fila i y la columna j de la matriz.[1]

Sea $A = [a_{ij}]$ una matriz de tamaño $n \times n$. [1]

- Para cada $1 \leq i \leq n$ se cumple que: $\det A = a_{i1}C_{i1} + a_{i2}C_{i2} + ... + a_{in}C_{in}$
- Para cada $1 \leq j \leq n$ se cumple que: $\det A = a_{1j}C_{1j} + a_{2j}C_{2j} + ... + a_{nj}C_{nj}$

De acuerdo al teorema 1.1 se puede definir una ecuación de recurrencia para encontrar el determinante de una matriz $A = [a_{ij}]$ de la siguiente manera:

$$det(A, n)_{1 \leq i \leq n} = \{ \, a_{11} \, , if$$

n = 1.
$(-1)^{i+1} \times a_{1i} \times det(A', n-1), if$n ¿1. (1)

Donde A' es la matriz que se obtiene al eliminar la columna i y la fila n de A. De esta manera, para una matriz B de $n \times n$, la solución se entrega de la forma: $det(B, n)$.

3.2. Multiplicación de matrices

Dadas las matrices $A \in M_{m \times n}$ y $B \in M_{n \times p}$, entonces el producto de A con B, denotado AB, es una matriz $C \in M_{m \times p}$, dada por: [1]

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + a_{i3}b_{3j} + \ldots + a_{in}b_{nj} = \sum_{k=1}^{n} a_{ik}b_{kj}$$

con $i = 1, ..., m$ y $j = 1, ..., p$

3.3. Escalonamiento de matrices

Sea $A = [a_{ij}]$ una matriz de $n \times n$. Decimos que A está escalonada si $\forall i, j, 1 \leq i \leq j \leq n$, $a_{ij} = 0$.

```
Leer A, b
si A ∉ ℜ^{n×n} ó b ∉ ℜ^n entonces
|  A debe ser cuadrada y b debe ser un arreglo de n posiciones
si no, si det A = 0 entonces
|  A debe ser invertible
en otro caso
|  para k = 1 a n − 1 hacer
|  |  si A_kk = 0 entonces
|  |  |  j ← k + 1
|  |  |  mientras j < n y A_jk = 0 hacer
|  |  |  |  j ← j + 1
|  |  |  fin
|  |  |  si j < n entonces
|  |  |  |  para l = k a n hacer
|  |  |  |  |  A_kl ← A_kl + A_jl
|  |  |  |  fin
|  |  |  |  b_k ← b_k + b_j
|  |  |  fin
|  |  fin
|  |  para i = k + 1 a n hacer
|  |  |  si A_ki ≠ 0 entonces
|  |  |  |  m ← A_ik / A_kk
|  |  |  |  para l = k a n hacer
|  |  |  |  |  A_il ← A_il − m × A_kl
|  |  |  |  fin
|  |  |  |  b_i ← b_i − m × b_k
|  |  |  fin
|  |  fin
|  fin
fin
La solución (A, b)
```

Algoritmo 9: Algoritmo para escalonar matrices

```
Leer A, b, n
Ab ← FormaMatrizAumentada(A, b)
para k = 1 a n − 1 hacer
    para i = k + 1 a n hacer
        multiplicador ← Ab_ik / Ab_kk
        para j = k a n + 1 hacer
            Ab_ij ← Ab_ij − multiplicador ∗ Ab_kj
        fin
    fin
fin
return Ab
```

Algoritmo 10: Algoritmo de Eliminacion Gaussiana simple

Referencias

[1] Orlando García Jaimes, Jairo A. Villegas Gutiérrez, Jorge Iván. *Álgebra Lineal*. Editorial EAFIT, Medellín 2012.