PRÁCTICA 4
RESOLUCIÓN APROXIMADA DE ECUACIONES (2)
Asier Erramuzpe

Método de Horner:

• Explicación del algoritmo: Este método sólo es aplicable a las ecuaciones enteras, pero tiene la ventaja de que los cálculos necesarios son más sencillos que los usados en los métodos siguientes. La facilidad de cálculo es debida a que cada cifra de la raíz se determina individualmente.

Funciona de la siguiente manera: Supongamos que una ecuación entera dada f(x) = 0 tiene una raíz irracional que, correcta con 3 cifras decimales, es 2.124. Para determinar esta raíz primeramente veremos que la ecuación dada tiene una raíz entera. Después disminuiremos las raíces de f(x) = 0 en 2 unidades, obteniendo la nueva ecuación f1(x1)= 0 que tiene la raíz 0.124. Entonces hacemos ver que *f1{Xl}* = 0 tiene una raíz entre 0.1 y 0.2 y disminuimos sus raíces en 0.1, obteniendo una nueva ecuación f2(x2) = 0 que tiene la raíz 0.024. Repitiendo el paso anterior, mostramos que f*2(X2)* = O tiene una raíz entre 0.02 y 0.03 y disminuimos sus raíces en 0.02, obteniendo una nueva ecuación f3(x3) = O que tiene la raíz 0.004. Continuando este proceso, es posible obtener la raíz con el número de cifras decimales correctas que se desee.

Horner sólo evalúa polinomios, para encontrar raíces debemos combinarlo con el método de Newton (que también he implementado, el archivo se llama “newtonhorner.m”)

• Pseudocódigo y complejidad:

n=grado polinomio
a=vector de coeficientes
x=punto donde se quiere despejar el valor
P=valor del polinomio en el punto x

```
funcion  Horner(n, a, x) devuelve P;
        P = an;

        para i = 1 hasta n hacer
                P = P * x + ai - 1 ;
                i++;
        fpara;
        devolver P;

ffuncion;
```

La evaluación usando la forma monomial del polinomio de grado-*n* requiere al menos *n* sumas y (n^2+*n*)/2 multiplicaciones, si las potencias se calculan mediante la repetición de multiplicaciones. El algoritmo de Horner sólo requiere *n* sumas y *n* multiplicaciones. (Minimizar el número de multiplicaciones es lo más deseable porque necesitan mucha carga computacional y son inestables comparadas con la suma). El grado es O(n).

• Pruebas:

e$p_6(x) = (x - 3)(x + 3)(x + 5)(x + 8)(x - 2)(x - 7)$

$p_6(x) = x^6 + 4x^5 - 72x^4 - 214x^3 + 1127x^2 + 1602x - 5040.$

p=[1 4 -72 -214 1127 1602 -5040]

[a b]=newtonhorner(p,4)

a =

```
    3.0000
    2.0000
   -8.0000
    7.0000
   -3.0000
   -5.0000
```

(raices = correcto)

Método de Laguerre-Thibault :

• Explicación del algoritmo: El método de Laguerre se usa para solucionar la ecuación p(x)=0 para un polinomio 'p' dado. Una de las mejores propiedades del método es que es casi segura su convergencia a alguna raíz del polinomio, sin importar el valor inicial proporcionado.

La llamada Cota de Laguerre-Thibault para polinomio de coeficientes reales consiste en dividir p(x) por: "x-1", "x-2", "x-3", "x-m" etc. hasta encontrar un valor de "m" tal que el polinomio "cociente" y el polinomio "resto" de la división indicada tenga todos sus coeficientes positivos ó nulos. En tal caso, si p(x) tiene una raíz positiva, la misma estará en el intervalo (0, m).

Para calcular las negativas es lo mismo, pero evaluando el polinomio en p(-x).

• Pseudocódigo y complejidad:

p=vector de coeficientes
maxiter=nº máximo de iteraciones
inf=cota inferior de las raíces del polinomio
sup=cota superior de las raíces del polinomio

```
funcion  laguerre(p, maxiter) devuelve (sup, inf);

        //Parte cota positiva
        cota=falso;
        iter=0;
        L=1;
        si p(1)<0 → p1=-p
          [] p1=p
        fsi
        mientras ¬cota y iter<maxiter hacer
                  resto y cociente de p1 entre (1-L);
                  cociente = [0 cociente];
                  i=1;
                  mientras i<length(cociente) y (cociente(i)>=0 y resto(i)>=0) hacer
                            i=i+1;
                  fmientras;
        cota=  cociente(i)>=0 y resto(i)>=0;
        iter=iter+1;
        L=L+1;
        fmientras;
        si cota → sup=L
          [] sup=0
        fsi

        //Parte cota negativa
        cota=falso;
        iter=0;
        L=1;
        //evaluar p(-x) ?¿
        si p(1)<0 → p2=-p
          [] p2=p
        fsi
        mientras ¬cota y iter<maxiter hacer
                  resto y cociente de p2 entre (1-L);
                  cociente = [0 cociente];
                  i=1;
                  mientras i<length(cociente) y (cociente(i)>=0 y resto(i)>=0) hacer
                            i=i+1;
                  fmientras;
        cota=  cociente(i)>=0 y resto(i)>=0;
```

```
        iter=iter+1;
        L=L+1;
        fmientras;
        si cota → inf=L
           [] inf=0
        fsi
        devolver sup,inf;
ffuncion;
```

La complejidad de este algoritmo, pese a tener 2 bucles, es de O(n), porque ninguno de los dos bucles es de orden superior y los bucles internos no suponen una carga cuadrática.

• Pruebas:

e$p_6(x) = (x - 3)(x + 3)(x + 5)(x + 8)(x - 2)(x - 7)$

$p_6(x) = x^6 + 4x^5 - 72x^4 - 214x^3 + 1127x^2 + 1602x - 5040.$

p=[1 4 -72 -214 1127 1602 -5040]

[a b]=laguerre(p,9) ---> hasta la novena iteración no es capaz de descubrir la cota superior

a = 0

b = 10

[a b]=laguerre(p,9)---> dos iteraciones más tarde, es capaz de acotar también la cota inferior bastante aproximado

a = -12

b = 10

Método de Sturm :

• Explicación del algoritmo: Este teorema tiene como principal objetivo, descubrir los ceros de una función polinómica. Sirve para saber cuantas raíces hay en un espacio acotado.

Funciona de esta manera:

Sea $f(x)$ un polinomio de coeficientes reales tal que $f(x) = 0$ no tiene raíces múltiples. Sean a y b números reales, $a < b$ y ninguno de los dos es raíz de $f(x) = 0$. Entonces la cantidad de raíces reales de $f(x) = 0$ entre a y b es la diferencia entre el número de variaciones de signo del sistema de Sturm

$$f(x), f_1(x), f_2(x),..., f_{k-1}(x), f_k(x)$$

para $x = b$ y el número de variaciones del sistema para $x = a$. Los términos que den cero deben ser descartados antes de contar los cambios de signo.

• Pseudocódigo y complejidad:

p1=vector de coeficientes
p2=vector de coeficientes de la derivada de p1
a=cota inferior
b=cota superior
num=numero de raices que tiene el polinomio en el intervalo

```
funcion  sturm(p1, p2, a, b) devuelve num;

        grado=longitud(p);

        //utilizo horner para evaluar el polinomio en un punto determinado
        si horner(p1,a)*horner(p2,a)<0 → x=1;
          [] x=0;
        fsi
        si horner(p1,b)*horner(p2,b)<0 → y=1;
          [] y=0;
        fsi

        mientras i=2<grado
                 resto y cociente de p1 entre p2;
                 p1=p2;
                 p2=-resto;
                si horner(p1,a)*horner(p2,a)<0 → x=x+1;
                fsi
                si horner(p1,b)*horner(p2,b)<0 → y=y+1;
                fsi
                i=i++;
        fmientras;

        num = abs(x-y);
        devuelve num
ffuncion
```

La complejidad de este algoritmo es $O(n^2)$, ya que dentro del bucle principal llamamos a horner que es de complejidad O(n), quedándonos un algoritmo de orden superior.

• Pruebas:

Otra vez probamos con nuestro polinomio habitual:

e$p_6(x) = (x - 3)(x + 3)(x + 5)(x + 8)(x - 2)(x - 7)$

$p_6(x) = x^6 + 4x^5 - 72x^4 - 214x^3 + 1127x^2 + 1602x - 5040.$

p=[1 4 -72 -214 1127 1602 -5040]

>> sturm(p,-2,6) ans = 2

>> sturm(p,-9,8) ans = 6

>> sturm(p,-4,3) ans = 3

Los resultados son correctos.

Para la realización de esta práctica, bibliografía:

Calculo Cientifico Con MATLAB y Octave
Escrito por A. Quarteroni,F. Saleri

Fin Práctica 5