Propuestas de Solución

Diana Sofía Carrillo

01

Algoritmo babilónico

Enunciado

1. Algoritmo

Encontrar la raíz cuadrada de un valor x es un proceso que tiene sus origenes en Babilonia. En el siguiente seudocodigo se encuentra el algoritmo para encontrar la raíz cuadrada de cualquier valor $x \geq 0$.

```
Entra:          n       Dato
                E       Error permitido
                x       Valor inicial
Sale:           y       Respuesta calculada con error E
y ← 1/2(x + n/x)
Repetir mientras  |x - y|  >  E tolerancia
        x ← y
        y ← 1/2(x + n/x)
Fin
```

Figura 1: *Algoritmo de la raíz cuadrada*

PROBLEMA: Implemente el algoritmo para evaluar $\sqrt{7}$ utilice una tolerancia de 10^{-8} y de 10^{-16}.Tenga en cuenta que debe imprimir varias salidas con la tolerancia deseada y validar la respuesta

Desarrollo

Entra:	n	Dato
	E	Error permitido
	x	Valor inicial
Sale:	y	Respuesta calculada con error E

$y \leftarrow \frac{1}{2}(x + \frac{n}{x})$

Repetir mientras $|x - y| > E$ tolerancia

$x \leftarrow y$

$y \leftarrow \frac{1}{2}(x + \frac{n}{x})$

Fin

```python
def metodo_babilonico(n, e):

    x = n
    y = (1 / 2) * (x+(n / x))

    iteraciones = 0

    while abs(x - y) > e:

        x = y
        y = (1 / 2) * (x+(n / x))
        iteraciones += 1

    print('\nNumero de iteraciones: ', iteraciones)
    return y
```

Desarrollo

```python
n = 7

print(
    "\nLa raiz cudrada de",
    n,
    ", con el metodo de la biblioteca math, es",
    math.sqrt(n))
```

```python
print(
    "La raiz cuadrada de",
    n,
    ", con tolerancia 10 a la -8, es",
    metodo_babilonico(n, 1e-8),
)
print(
    "La raiz cuadrada de",
    n,
    ", con tolerancia 10 a la -16, es",
    metodo_babilonico(n, 1e-16),
)
```

```
La raiz cudrada de 7 , con el metodo de la biblioteca math, es 2.6457513110645907

Numero de iteraciones:  5
La raiz cuadrada de 7 , con tolerancia 10 a la -8, es 2.6457513110645907

Numero de iteraciones:  6
La raiz cuadrada de 7 , con tolerancia 10 a la -16, es 2.6457513110645907
```

02

Eficiencia de un Algoritmo

Enunciado

3. Eficiencia de un Algoritmo

PROBLEMA: Evaluar el valor de un polinomio $P(x)$ es una tarea que involucra para la maquina realizar un número de operaciones entre multiplicaciones y sumas, la cual deben ser mínimas para que sea eficiente. Implemente un contador de operaciones y evaluar el polinomio $f(x) = x^3 - 2x^2 + 4x/3 - 8/27$ en $x = 4$ de la manera más eficiente es decir, utilizando el menor número de operaciones.

Desarrollo: Método de Horner

Example: Evaluate the polynomial $f(x) = x^4 + 3x^3 + 5x^2 + 7x + 9$ at $x = 2$

Solution:

Since the polynomial is of the 4^{th} degree, then $n = 4$

K	4	3	2	1	0
Step	$b_4 = 1$	$b_3 = 3 + 2 * 1$	$b_2 = 5 + 2 * 5$	$b_1 = 7 + 2 * 15$	$b_0 = 9 + 2 * 37$
Result	1	5	15	37	83

(Autor, 2021)

Complejidad de las multiplicaciones: O(n)

Complejidad usando sustitución: O(n^2 + n)/2

Desarrollo: Método de Horner

```python
def metodo_horner(coeficientes, x):
  rta = coeficientes[0]
  n = len(coeficientes)
  numOperaciones = 0

  for i in range(1, n):
    rta = coeficientes[i] + x * rta
    numOperaciones +=1

  print("Numero de operaciones (sumas y multiplicaciones): ",
  numOperaciones)
  return rta

coeficientes = [1, -2, 4/3, -8/27] #grado 3
x = 4

print("El valor del polinomio es" , metodo_horner(coeficientes, x))
```

```
Numero de operaciones (sumas y multiplicaciones):  3
El valor del polinomio es 37.03703703703704
```

03

Sumar Elementos de una Matriz

Enunciado

Crear un algoritmo que permita sumar los elementos del triángulo inferior y superior de una matriz cuadrada. ¿Cuántas operaciones toma?

Desarrollo

Matriz de prueba (3 x 3):

i/j	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

i < j

i = j

i > j

Desarrollo

```
def suma(matriz, tamanio, triangulo):
    suma = 0
    operaciones = 0
    for i in range(0, tamanio):
        for j in range(0, tamanio):
            if triangulo == 1:
                if (i <= j):
                    suma += matriz[i][j]
                    operaciones += 1
            elif triangulo == 2:
                if (i >= j):
                    suma += matriz[i][j]
                    operaciones += 1
            else:
                suma += matriz[i][j]
                operaciones += 1

    print("\nLa suma toma: ",operaciones, " operaciones.")
    return suma
```

Desarrollo

```python
tamanio = 3
matriz = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

#tria                                                              y
cualq
print                                                              uma
(matr

print                                                              uma
(matr

print("La suma de toda la matriz es: ", suma(matriz, tamanio, 3))
```

```
La suma toma:  6  operaciones.
La suma del triangulo superior de la matriz es:  26

La suma toma:  6  operaciones.
La suma del triangulo inferior de la matriz es:  34

La suma toma:  9  operaciones.
La suma de toda la matriz es:  45
```

04

Prueba por Inducción

Enunciado

Probar por inducción matemática las complejidades de los siguientes métodos.

Teorema de Horner Sea $P(x) = 2x^4 - 3x^2 + 3x - 4$ un polinomio incompleto de grado 4 para cualquier x_0 el polinomio puede ser evaluado de diferentes maneras:

Método 1:

$P(x_0) = 2 * x_0 * x_0 * x_0 * x_0 - 3 * x_0 * x_0 + 3 * x_0 - 4$

Método 2:

$P(x_0) = -3 * x * (x) + 0 * x * (x^2) + 2 * x * (x^3) + 3 * x - 4$

Método 3:

$P(x_0) = -4 + x * (3 - x * (-3 + x * (x * (2))))$

En resumen:

Método	Multiplicaciones
Método 1	$\frac{n(n+1)}{2}$
Método 2	$2n - 1$
Método 3	n

Desarrollo: Método 1

Peor caso: polinomio completo.

Forma general de un polinomio: $P(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \ldots + a_1 x^1 + a_0$

Multiplicaciones:

$$P(n) = n + (n-1) + (n-2) + \ldots + 3 + 2 + 1$$

$$P(n) = (n+1) + (n+1) + (n+1) + \ldots(n+1) + (n+1) + (n+1)$$

$$P(n) = n(n+1)/2$$

Desarrollo: Método 2

Peor caso: polinomio completo.

Forma general de un polinomio: $P(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \ldots + a_1 x^1 + a_0$

Multiplicaciones: $P(x) = a_2 \cdot x \cdot (x) + a_3 \cdot x \cdot (x^2) + \ldots + a_n \cdot x \cdot (x^{n-1}) + a_1 \cdot x + a_0$

$P(n) = 2 \; multiplicaciones + 2\,m. + \ldots + 2m + 1m$

$P(n) = 2n - 1$

Desarrollo: Método 3

Peor caso: polinomio completo.

Forma general de un polinomio: $P(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \ldots + a_1 x^1 + a_0$

Multiplicaciones: $P(x) = (\ldots(((a_n \cdot x + a_{n-1}) \cdot x + a_{n-2}) \cdot x + \ldots + a_2) \cdot x + a_1) \cdot x + a_0$

$P(n) = 1\ multiplicación + 1\ m. + \ldots + 1\ m. + 1\ m.$

$P(n) = n$

05

Sistema de Ecuaciones

Enunciado

Solucionar un sistema de ecuaciones. Contar las operaciones necesarias para hallar la solución y averiguar qué pasa cuando hay un error en la entrada.

Mejorar la solución del sistema usando métodos como el pivote parcial o total.

Desarrollo: Solución del Sistema

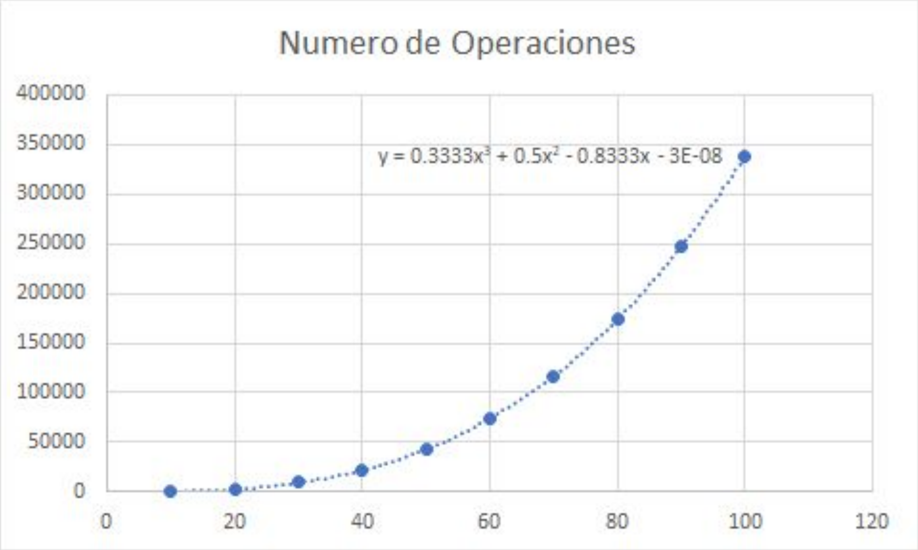
```python
def gauss(a, b):
  n = len(b) #Cantidad de ecuaciones
  c = np.zeros([n, n+1]) #Genera matriz de n * n+1 llena de ceros
  for i in range (n):
    for j in range (n):
      c[i][j] = a[i][j] #Se crea la matriz aumentada
    c[i][n] = b[i]
  for k in range(n): #Se empieza con el proceso de normalización y reducción
    t = c[k][k]
    for j in range (k, n+1):
      c[k][j] = c[k][j]/t #Se normaliza la fila k
    for i in range(k+1, n): #Se reducen las filas debajo
      t = c[i][k]
      for j in range (k, n+1):
        c[i][j] = c[i][j]-t*c[k][j]
  x = np.zeros([n, 1]) #Se genera la matriz llena de ceros para el vector solución
  x[n-1] = c[n-1][n] #Se llena la última posición del arreglo con la solución
                     #de la última ecuación (la que no necesita despeje)
  for i in range(n-2, -1, -1): #Sistema triangular
      s = 0
      for j in range(i+1, n):
          s += c[i][j]*x[j]
      x[i] = c[i][n] - s

  return x
```

```python
a = [[2, 3, 7], [-2, 5, 6], [8, 9, 4]]
b = [3, 5, 8]

x = gauss(a, b)
print ("\n--Resultado--")
print(x)
```

Desarrollo: Num operaciones



El número de operaciones está representado por: T(n) = n^3/3 + n^2/2 + 5n/6

El método es de tercer orden: O(n^3).

Desarrollo: Resultados y Error Relativo

```
--Resultado--
[[-0.055556]
 [ 0.915033]
 [ 0.052288]]
```

```python
def error_relativo(v_obtenido, v_real):
    #v_real = np.linalg.solve(a, b)
    rta = np.linalg.norm(v_obtenido - v_real) / np.linalg.norm(v_real)
    return rta
```

```python
a = [[1.9, 3, 7], [-2, 5, 6], [8, 9, 4]]
x1 = gauss(a, b)
```

```
Error Relativo:
0.001749518928297893
```

```python
a = [[2.1, 3, 7], [-2, 5, 6], [8, 9, 4]]
x1 = gauss(a, b)
```

```
Error Relativo:
0.0017110679628406802
```

Desarrollo: Estrategia pivote

```
def gaussPivote(a, b):
    n = len(b)
    c = np.zeros([n, n+1])
    for i in range(n):
        for j in range(n):
            c[i][j] = a[i][j] #Matriz aumentada
        c[i][n] = b[i]
    for k in range(n):
        p = k
        for i in range(k+1, n): #Selección del pivote
            if (abs(c[i][k])>abs(c[p][k])):
                p = i
        for j in range(k, n+1): #Intercambio de filas
            t = c[k][j]
            c[k][j] = c[p][j]
            c[p][j] = t
        t = c[k][k]
        if abs(t) < 1e-10: #Verificar que el sistema es singular
            return []
        for j in range(k, n+1): #Normalizar fila e
            c[k][j] = c[k][j]/t
        for i in range(k+1, n): #Reducir filas debajo
            t = c[i][k]
            for j in range(k, n+1):
                c[i][j] = c[i][j] - t*c[k][j]
    x = np.zeros([n,1]) #Celdas para el vector solución
    x[n-1] = c[n-1][n]
    for i in range (n-2, -1, -1): #Resolver el sistema triangular
        s = 0
        for j in range(i+1, n):
            s = s + c[i][j]*x[j]
        x[i] = c[i][n] - s
    return x
```

Para disminuir el error de redondeo, la estrategia del pivote busca reducir el valor de los operandos que intervienen en la multiplicación.

Se trata de normalizar las filas con el elemento de mayor magnitud de esa columna. De esta forma el cociente termina siendo de menor valor, este mismo elemento se usa posteriormente en la multiplicación.

De esta forma se reduce el error en la etapa de reducción de las otras filas.

El método verifica la unicidad de la solución.

Gracias

¿Preguntas?

Referencias

Author. (2021). *Horner's Rule*. Math10.com.

https://www.math10.com/en/algebra/horner.html