

Método de Diferencias Finitas para ecuaciones elípticas

Julio A. Medina
Universidad de San Carlos
Escuela de Ciencias Físicas y Matemáticas
Maestría en Física
julioantonio.medina@gmail.com

1. Ecuaciones diferenciales parciales elípticas

La ecuación diferencial parcial elíptica a considerar es la ecuación de Poisson

$$\nabla^2 u(x, y) \equiv \frac{\partial^2 u}{\partial x^2}(x, y) + \frac{\partial^2 u}{\partial y^2}(x, y) = f(x, y) \quad (1)$$

en $R = \{(x, y) \mid a < x < b, c < y < d\}$, con $u(x, y) = g(x, y) \in S$, donde S denota al contorno de R . Si f y g son continuas en su dominio entonces hay una única solución a la ecuación.

1.1. Seleccionando un retículo

El método a utilizar es una adaptación bidimensional del método de diferencias finitas para problemas con fronteras lineales como se discute en [1]. El primer paso es escoger enteros n y m para definir el tamaño de los pasos (*steps*) $h = (b - a)/n$ y $k = (d - c)/m$ particionando de esta manera el intervalo $[a, b]$ en n partes iguales de ancho h y el intervalo $[c, d]$ en m partes iguales con ancho k , formando un retículo o cuadrícula como se puede ver en la figura

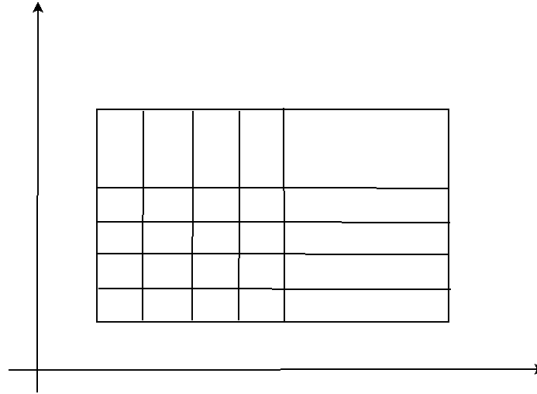


Figura 1: Cuadrícula de $n \times m$

Este retículo se construye formalmente al dibujar líneas verticales y horizontales sobre el dentro del rectángulo R en los puntos con coordenadas (x_i, y_j) ,

donde

$$x_i = a + ih, \text{ para cada } i = 0, 1, 2, \dots, n \text{ y } y = a + jk, \text{ para cada } j = 0, 1, 2, \dots, m \quad (2)$$

Las rectas correspondientes a $x = x_i$ y $y = y_j$ son las lineas que forman la cuadrícula y sus intersecciones son los puntos del retículo. Para cada punto interior del retículo se (x_i, y_j) , para $i = 1, 2, \dots, n - 1$ y $j = 1, 2, \dots, m - 1$, se puede utilizar una serie de Taylor en la variable x alrededor del punto x_i para generar una fórmula de diferencia centrada

$$\frac{\partial^2 u}{\partial x^2}(x_i, y_j) = \frac{u(x_{i+1}, y_j) - 2u(x_i, y_j) + u(x_{i-1}, y_j))}{h^2} - \frac{h^2}{12} \frac{\partial^4 u}{\partial x^4}(\xi_i, y_j) \quad (3)$$

donde $\xi_i \in (x_{i-1}, x_{i+1})$. De igual manera se puede encontrar la serie de Taylor en la variable y alrededor del punto y_j para hallar la diferencia centrada

$$\frac{\partial^2 u}{\partial y^2}(x_i, y_j) = \frac{u(x_i, y_{j+1}) - 2u(x_i, y_j) + u(x_i, y_{j-1}))}{k^2} - \frac{k^2}{12} \frac{\partial^4 u}{\partial y^4}(x_i, \eta_j) \quad (4)$$

donde $\eta_j \in (y_{j-1}, y_{j+1})$, sustituyendo estas fórmulas en la ecuación 1 permite expresar la ecuación de Poisson en los puntos (x_i, y_j) como

$$\begin{aligned} & \frac{u(x_{i+1}, y_j) - 2u(x_i, y_j) + u(x_{i-1}, y_j))}{h^2} + \frac{u(x_i, y_{j+1}) - 2u(x_i, y_j) + u(x_i, y_{j-1}))}{k^2} \\ &= f(x_i, y_j) + \frac{h^2}{12} \frac{\partial^4 u}{\partial x^4}(\xi_i, y_j) + \frac{k^2}{12} \frac{\partial^4 u}{\partial y^4}(x_i, \eta_j) \end{aligned} \quad (5)$$

para cada $i = 1, 2, \dots, n - 1$ y $j = 1, 2, \dots, m - 1$. Las condiciones de contorno son

$$u(x_0, y_j) = g(x_0, y_j) \text{ y } u(x_n, y_j) = g(x_n, y_j), \text{ para cada } j = 0, 1, \dots, m;$$

$$u(x_i, y_0) = g(x_i, y_0) \text{ y } u(x_i, y_m) = g(x_i, y_m), \text{ para cada } i = 1, 2, \dots, n - 1;$$

1.2. Método de Diferencias Finitas

En forma de ecuación de diferencias el Método de diferencias finitas es

$$2 \left[\left(\frac{h}{k} \right)^2 + 1 \right] w_{ij} - (w_{i+1,j} + w_{i-1,j}) - \left(\frac{h}{k} \right)^2 (w_{i,j+1} + w_{i,j-1}) = -h^2 f(x_i, y_j) \quad (6)$$

para cada $i = 1, 2, \dots, n - 1$ y $j = 1, 2, \dots, m - 1$, donde

$$\begin{aligned} w_{0j} &= g(x_0, y_j) \text{ y } w_{nj} = g(x_n, y_j), \text{ para cada } j = 0, 1, \dots, m; \\ w_{i0} &= g(x_i, y_0) \text{ y } w_{im} = g(x_i, y_m), \text{ para cada } i = 1, 2, \dots, n - 1 \end{aligned} \quad (7)$$

donde w_{ij} aproxima $u(x_i, y_j)$. Este método tiene un error de truncación del orden $O(h^2 + k^2)$. La ecuación 6 involucra aproximaciones para $u(x, y)$ en los puntos

$$(x_{i-1}, y_j), (x_i, y_j), (x_i, y_{j-1}), \text{ y } (x_i, y_{j+1}).$$

Utilizando la información proveída por las condiciones de contorno 7 cuando el sistema definido por 6 lo permite, i.e. en los puntos adyacentes al contorno

definido por la cuadrícula construida previamente(1.1).

Con esto se obtiene un sistema de ecuaciones con una matriz de incógnitas de dimensiones $(n-1)(m-1) \times (n-1)(m-1)$ donde las incógnitas son las aproximaciones w_{ij} para $u(x_i, y_j)$ en los puntos interiores del retículo(1.1).

El sistema lineal para las incógnitas w_{ij} se puede representar más eficientemente para realizar operaciones matriciales si se realiza una re-etiquetación de los puntos interiores del retículo, una recomendación para realizar este esquema de re-etiquetación puede hallarse en [2] de la siguiente manera

$$P_l = (x_i, y_j), \text{ y } w_l = w_{ij} \quad (8)$$

donde $l = i + (m-1-j)(n-1)$ para cada $i = 1, 2, \dots, n-1$ y $j = 1, 2, \dots, m-1$, con este esquema de etiquetan a los puntos interiores del retículo de izquierda a derecha y de arriba hacia abajo y con esto se asegura que el sistema lineal resultante para hallar w_{ij} es una matriz con bandas de ancho máximo $2n-1$. Esta técnica fue utiliza para implementar este algoritmo en Python, ver algoritmos 1 y 2.

2. Algoritmo Método de Diferencias Finitas para ecuación de Poisson

Para hallar un algoritmo que genere y resuelva el método de diferencias finitas para las ecuaciones parciales elípticas descrito en la sección anterior se usa el esquema de re-etiquetado expuesto anteriormente, con esto se asegura que se obtiene un sistema tri-diagonal por bloques. Para estos sistemas tri-diagonales por bloques es conveniente aplicar una generalización del algoritmo de Factorización de Crout [2] para matrices tri-diagonales por bloques. Otro algoritmo iterativo que se puede utilizar es el método de relajamiento SOR. Con esto se puede delinear una estrategia para construir un algoritmo para el método de diferencias finitas:

- Construir el sistema asociado a las aproximaciones w_{ij} para $u(x_i, y_j)$, utilizando el esquema de etiquetado expuesto anteriormente.
- Aplicar un método de resolución de sistemas lineales: Generalización de factorización de Crout, método iterativo SOR.
- Resolver para w_{ij}

2.1. Algoritmo diferencias finitas(sistema lineal)

2.1.1. Función de re-etiquetado 1

Se implementa la función l para re-etiquetar el sistema de ecuaciones lineal

Algorithm 1 Función L para re-etiquetar las variables del sistema lineal

```

function L( $i, j, n, m$ )
  return  $(i+1) + (m-1-(j+1)) \times (n-1) - 1$ 
end function

```

2.1.2. Sistema lineal asociado

Con está función se crea el sistema asociado al método de diferencias finitas

Algorithm 2 Finite Difference Linear System

```

function FINITE_DIFFERENCE_LINEAR_SYSTEM( $a, b, c, d, n, m, f, g$ )
     $h \leftarrow (b - a)/n$ 
     $k \leftarrow (d - c)/n$ 
     $x \leftarrow \text{numpy.linspace}(a, b, n + 1)$ 
     $y \leftarrow \text{numpy.linspace}(c, d, m + 1)$ 
     $W_{ij} \leftarrow \text{zeros}(((n - 1) \times (m - 1)), ((n - 1) \times (m - 1)))$ 
     $w \leftarrow \text{zeros}((n - 1) \times (m - 1))$ 
    for  $i \leftarrow 0$  to  $n - 2$  do
        for  $j \leftarrow 0$  to  $m - 2$  do
             $W_{ij}[L(i, j, n, m), L(i, j, n, m)] \leftarrow 2((h/k)^2 + 1)$ 
            if  $i \neq 0$  and  $i \neq n - 2$  then
                 $W_{ij}[L(i, j, n, m), L(i + 1, j, n, m)] \leftarrow -1$ 
                 $W_{ij}[L(i, j, n, m), L(i - 1, j, n, m)] \leftarrow -1$ 
            else
                if  $i = 0$  then
                     $W_{ij}[L(i, j, n, m), L(i + 1, j, n, m)] \leftarrow -1$ 
                     $w[l(i, j, n, m)] \leftarrow g(x[i], y[j + 1], a, b, c, d)$ 
                end if
                if  $i = n - 2$  then
                     $W_{ij}[L(i, j, n, m), L(i - 1, j, n, m)] \leftarrow -1$ 
                     $w[L(i, j, n, m)] \leftarrow g(x[i + 2], y[j + 1], a, b, c, d)$ 
                end if
            end if
            if  $j \neq 0$  and  $j \neq m - 2$  then
                 $W_{ij}[L(i, j, n, m), L(i, j + 1, n, m)] \leftarrow -1$ 
                 $W_{ij}[L(i, j, n, m), L(i, j - 1, n, m)] \leftarrow -1$ 
            else
                if  $j = 0$  then
                     $W_{ij}[L(i, j, n, m), L(i, j + 1, n, m)] \leftarrow -(h/k)^2$ 
                     $w[L(i, j, n, m)] \leftarrow g(x[i + 1], y[j], a, b, c, d)$ 
                end if
                if  $j = m - 2$  then
                     $W_{ij}[L(i, j, n, m), L(i, j - 1, n, m)] \leftarrow -(h/k)^2$ 
                     $w[L(i, j, n, m)] \leftarrow g(x[i + 1], y[j + 2], a, b, c, d)$ 
                end if
            end if
             $w[L(i, j, n, m)] \leftarrow -h^2 f(x[i], y[j]) + w[L(i, j, n, m)]$ 
        end for
    end for
    return  $W_{ij}, w$ 
end function

```

2.2. Algoritmo de Generalización Factorización de Crout para matrices tridiagonales por bloques

Esta es la implementación de la generalización del algoritmo de factorización de Crout para matrices tri-diagonales por bloques, el algoritmo de factorización de Crout sólo es valido para matrices tri-diagonales y no para matrices tri-diagonales por bloques.

Algorithm 3 Crout Generalization Algorithm for Tridiagonal Block Matrices

Require: $A \in \mathbb{R}^{N \times N}$, $K \in \mathbb{R}^N$, $n \in \mathbb{N}$

Ensure: $sol \in \mathbb{R}^N$

```

function CROUT_GENERALIZATION( $A, K, n$ )
     $N \leftarrow \lfloor \frac{\text{shape}(A)[1]}{n} \rfloor$ 
     $W \leftarrow []$ 
     $B_1 \leftarrow A_{1:n, 1:n}$ 
     $C_1 \leftarrow A_{1:n, n:n+n}$ 
     $W.append(\text{inv}(B_1) \cdot C_1)$ 
    for  $i \leftarrow 2$  to  $N - 1$  do
         $B_i \leftarrow A_{(i-1)n:in, (i-1)n:in}$ 
         $A_i \leftarrow A_{(i-1)n:in, (i-2)n:(i-1)n}$ 
         $C_i \leftarrow A_{(i-1)n:in, in+1:n+(i-1)n}$ 
         $W.append(\text{inv}(B_i - A_i \cdot W_{i-2}) \cdot C_i)$ 
    end for
     $B_N \leftarrow A_{1:n, 1:n}$ 
     $K_1 \leftarrow K_{1:n}$ 
     $G_1 \leftarrow \text{inv}(B_1) \cdot K_1$ 
     $G \leftarrow []$ 
     $G.append(G_1)$ 
    for  $i \leftarrow 2$  to  $N$  do
         $K_i \leftarrow K_{(i-1)n:in}$ 
         $B_i \leftarrow A_{(i-1)n:in, (i-1)n:in}$ 
         $A_i \leftarrow A_{(i-1)n:in, (i-2)n:(i-1)n}$ 
         $G.append(\text{inv}(B_i - A_i \cdot W_{i-2}) \cdot (K_i - A_i \cdot G_{i-2}))$ 
    end for
     $Z \leftarrow G[:]$ 
    for  $i \leftarrow N - 2$  to  $0$  step  $-1$  do
         $Z_i \leftarrow G_i - W_i \cdot Z_{i+1}$ 
    end for
     $sol \leftarrow \text{concatenate}(Z)$ 
    return  $sol$ 
end function

```

2.3. Método iterativo SOR para resolver sistemas lineales

Algorithm 4 Iterative SOR method for solving linear systems

Require: A es una $n \times n$ matriz, b es un vector de incógnitas de dimensión n , $x^{(0)}$ es una solución propuesta inicial, ω es el parámetro de relajamiento, ϵ es la tolerancia, and max_{iter} es el número máximo de iteraciones.

Ensure: x es una solución aproximada de $Ax = b$.

```
function SOR( $A, b, \omega, \epsilon, max_{iter}, x^{(0)}$ )
     $k \leftarrow 1$ 
     $xo \leftarrow x^{(0)}$ 
     $tolerance \leftarrow \mathbf{True}$ 
    while  $k < max_{iter}$  and  $tolerance$  do
        for  $i = 1$  to  $n$  do
             $s \leftarrow \sum_{j=1}^{i-1} a_{ij}x_j + \sum_{j=i+1}^n a_{ij}xo_j$ 
             $x_i \leftarrow (1 - \omega)xo_i + \frac{\omega}{a_{ii}}(b_i - s)$ 
        end for
        if  $\|x - xo\| < \epsilon$  then
             $tolerance \leftarrow \mathbf{False}$ 
        end if
         $k \leftarrow k + 1$ 
         $xo = x$ 
    end while
    if  $k = max_{iter}$  then
        print "Se ha alcanzado el número máximo de iteraciones"
    end if
    return  $x$ 
end function
```

2.4. Diferencia finitas usando generalización de factorización Crout

Algorithm 5 Algoritmo diferencias finitas usando generalización factorización de Crout

Dado un problema con condiciones de frontera de la forma 1, definir

$$a, b, c, d, n, m, f(x, y), g(x, y)$$

usando las funciones definidas en los algoritmos 2, 3

```
 $A, b \leftarrow \text{finite\_difference\_linear\_system}(a, b, c, d, n, m, f, g)$ 
 $solution \leftarrow \text{Crout\_generalization}(A, b, n-1)$ 
```

este algoritmo se ha implementado en Python, usando la librería Numpy. El repositorio se encuentra en https://github.com/Julio-Medina/Finite_Difference_Method/tree/main/code.

2.5. Diferencia finitas usando método de relajamiento SOR

Algorithm 6 Algoritmo diferencias finitas usando generalización factorización de Crout

Dado un problema con condiciones de frontera de la forma 1, definir

$$a, b, c, d, n, m, f(x, y), g(x, y), \omega, \epsilon, x^{(0)}$$

usando las funciones definidas en los algoritmos 2, 4

$A, b \leftarrow \text{finite_difference_linear_system}(a, b, c, d, n, m, f, g)$
 $solution \leftarrow \text{SOR}(A, b, \omega, \epsilon, \text{máx}_{iter}, x^{(0)})$

Referencias

- [1] Richard L. Burden, J. Douglas Faires *Numerical Analysis*, (Ninth Edition). Brooks/Cole, Cengage Learning. 978-0-538-73351-9
- [2] Richard S. Varga. *Matrix Iterative Analysis*. Second Edition. Springer. DOI 10.1007/978-3-642-05156-2