Método de Diferencias Finitas

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

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

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 está 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 cuadricula como se puede ver en la figura

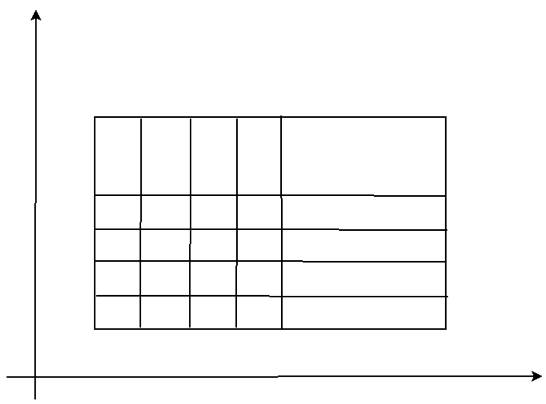

Figura 1: Cuadrícula de $n \times m$

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

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

Las rectas correspondientes a $x = x_i$ y $y = y_i$ son las lineas que forman la cuadricula 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

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

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

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

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

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

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

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

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

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

2. Método de Diferencias Finitas para ecuación de Poisson

2.1. Algoritmo diferencias finitas(sistema 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

Algorithm 2 Finite Difference Linear System

$h \leftarrow (b-a)/n$
$k \leftarrow (d-c)/n$
$x \leftarrow$ numpy.linspace$(a, b, n+1)$
$y \leftarrow$ numpy.linspace$(c, d, m+1)$
$W_{ij} \leftarrow$ zeros $(((n-1)\times(m-1)), ((n-1)\times(m-1)))$
$w \leftarrow$ 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
    $w[L(i,j,n,m)] \leftarrow -h^2 f(x[i], y[j])$
  end for
end for
return W_{ij}, w

2.2. Algoritmo de Generalización Factorización de Crout para matrices tridiagonales 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: $sol \in \mathbb{R}^{N}$

$N \leftarrow \lfloor \frac{\text{shape}(A)[1]}{n} \rfloor$
$W \leftarrow []$
$B_1 \leftarrow A_{1:n,1:n}$
$C_1 \leftarrow A_{1:n,n:n+n}$
$W.append(\text{inv}(B_1) \cdot C_1)$
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

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 $Ax = b$.

```
k ← 1
xo ← x^(0)
tolerance ← True
while k < max_iter and tolerance do
    for i = 1 to n do
        s ← Σ_{j=1}^{i-1} a_ij x_j + Σ_{j=i+1}^{n} a_ij xo_j
        x_i ← (1 − ω) xo_i + (ω / a_ii)(b_i − s)
    end for
    if ||x − xo|| < ε then
        tolerance ← False
    end if
    k ← k + 1
    xo = x
end while
if k = max_iter then
    print "Se ha alcanzado el número máximo de iteraciones"
end if
return x
```

Referencias

[1] Richard L. Burden, J. Douglas Faires *Numerical Analysis*, (Ninth Edition). Brooks/Cole, Cengage Learning. 978-0-538-73351-9