

Auxiliar Métodos Iterativos

Aux: Pablo Pizarro

Métodos iterativos

- Métodos numéricos para sistemas lineales
 - Métodos directos
 - Ejemplo: Gauss
- Métodos iterativos
 - Jacobi
 - Gauss-Seidel
 - Sobrerelajación sucesiva

Métodos iterativos

- Métodos numéricos para sistemas lineales
 - Métodos directos
 - Ejemplo: Gauss
- Métodos iterativos
 - Jacobi
 - Gauss-Seidel
 - Sobrerelajación sucesiva

Métodos iterativos

- Métodos numéricos para sistemas lineales
 - Métodos directos
 - Ejemplo: Gauss
 - Métodos iterativos
- Métodos iterativos
 - Jacobi
 - Gauss-Seidel
 - Sobrerelajación sucesiva

Método directo/iterativo

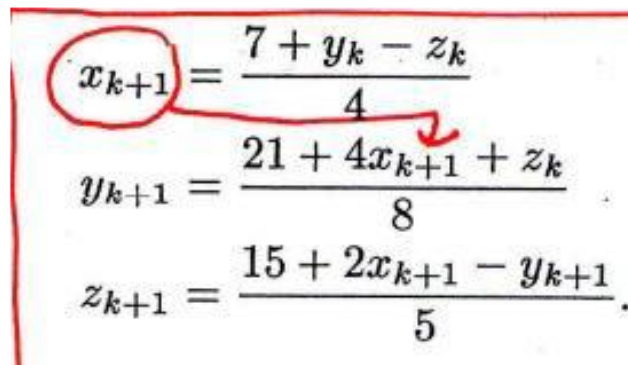
- Directo
 - Ya conoce las ecuaciones que gobiernan el sistema
 - Ecuaciones lineales -> Sistemas representables por matrices

$$\begin{aligned}x_1 + 2x_2 + x_3 + 4x_4 &= 13 \\2x_1 + 0x_2 + 4x_3 + 3x_4 &= 28 \\4x_1 + 2x_2 + 2x_3 + x_4 &= 20 \\-3x_1 + x_2 + 3x_3 + 2x_4 &= 6.\end{aligned}$$

$$\begin{aligned}\text{pivote} &\rightarrow \\m_{21} &= 2 \\m_{31} &= 4 \\m_{41} &= -3\end{aligned} \left[\begin{array}{cccc|c} \underline{1} & 2 & 1 & 4 & 13 \\ 2 & 0 & 4 & 3 & 28 \\ 4 & 2 & 2 & 1 & 20 \\ -3 & 1 & 3 & 2 & 6 \end{array} \right].$$

Método directo/iterativo. Resolución

- Eliminación de Gauss
 - Igual que Álgebra Lineal
- Método de Jacobi
- Gauss-Seidel
 - Basado en el método de Jacobi, la idea es que cada vez que tenemos un nuevo valor, usamos ese para obtener el valor de la siguiente ecuación


$$\begin{aligned}x_{k+1} &= \frac{7 + y_k - z_k}{4} \\y_{k+1} &= \frac{21 + 4x_{k+1} + z_k}{8} \\z_{k+1} &= \frac{15 + 2x_{k+1} - y_{k+1}}{5}\end{aligned}$$

Método iterativo

- Sobrerelajación sucesiva
 - Permite solucionar problemas que involucren EDP de forma mucho más rápida que sólo calculando el promedio para el caso de Laplace.

$$h(i, j) = h_{i,j} = \frac{1}{4} (h_{i+1,j} + h_{i-1,j} + h_{i,j+1} + h_{i,j-1})$$

LAPLACE

$$h_{i,j} = \begin{cases} 1/4 \cdot (2h_{i-1,j} + h_{i,j+1} + h_{i,j-1}) & (i+1) \text{ es borde} \\ 1/4 \cdot (2h_{i+1,j} + h_{i,j+1} + h_{i,j-1}) & (i-1) \text{ es borde} \\ 1/4 \cdot (h_{i+1,j} + h_{i-1,j} + 2h_{i,j+1}) & (j-1) \text{ es borde} \\ 1/4 \cdot (h_{i+1,j} + h_{i-1,j} + 2h_{i,j-1}) & (j+1) \text{ es borde} \end{cases}$$

$$h_{i,j} = \begin{cases} 1/2 \cdot (h_{i+1,j} + 2h_{i,j+1}) & (i-1), (j-1) \text{ son bordes} \\ 1/2 \cdot (h_{i+1,j} + 2h_{i,j-1}) & (i-1), (j+1) \text{ son bordes} \\ 1/2 \cdot (h_{i-1,j} + 2h_{i,j+1}) & (i+1), (j-1) \text{ son bordes} \\ 1/2 \cdot (h_{i-1,j} + 2h_{i,j-1}) & (i+1), (j+1) \text{ son bordes} \end{cases}$$

Método iterativo

- Sobrerrelajación sucesiva

$$(22) \quad \begin{aligned} u_{i,j} &= u_{i,j} + \omega \left(\frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}}{4} \right) \\ &= u_{i,j} + \omega r_{i,j}, \end{aligned}$$

en la que el parámetro ω verifica $1 \leq \omega < 2$. En el método de sobrerrelajación sucesiva, cada paso de la iteración consiste en hacer un barrido de la malla con la fórmula recursiva (22) hasta que se tenga $|r_{i,j}| < \varepsilon$. Para elegir el valor óptimo del parámetro ω hay que estudiar los autovalores de la matriz que caracteriza el método iterativo que estamos usando para resolver un sistema lineal; en nuestro caso, dicho valor óptimo viene dado por la fórmula

$$(23) \quad \omega = \frac{4}{2 + \sqrt{4 - \left(\cos \left(\frac{\pi}{n-1} \right) + \cos \left(\frac{\pi}{m-1} \right) \right)^2}}.$$

Método iterativo

- Sobrerelajación sucesiva

Las ecuaciones de Poisson y Helmholtz

Consideremos la ecuación de Poisson

$$(28) \quad \nabla^2 u = g(x, y).$$

Usando la notación $g_{i,j} = g(x_i, y_j)$, la extensión de la fórmula (20) para resolver la ecuación (28) sobre una malla rectangular es

$$(29) \quad u_{i,j} = u_{i,j} + \frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j} - h^2 g_{i,j}}{4}.$$

IMPORTANTE

Las ecuaciones de Poisson y Helmholtz

Consideremos la ecuación de Poisson

$$(28) \quad \nabla^2 u = g(x, y).$$

Usando la notación $g_{i,j} = g(x_i, y_j)$, la extensión de la fórmula (20) para resolver la ecuación (28) sobre una malla rectangular es

$$(29) \quad u_{i,j} = u_{i,j} + \frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j} - h^2 g_{i,j}}{4}.$$

Nuevo valor

Iteración anterior

Ejemplo

- Mismo problema del río, implementarlo ahora con sobrerelajación sucesiva.
- 1) Definir tolerancia error máximo en el constructor
 - 2) Crear función **_single_iteration** que calcule sólo 1 iteración.
 - 3) Crear función **_convergio** que indique si el sistema convergió, o sea, que el máximo error es menor que la tolerancia.
 - 4) Modificar función **start** para crear dos matrices, la de la iteración nueva y pasada, usar **_single_iteration** y **_convergio** para verificar convergencia, retornar número de iteraciones realizadas.

Ejemplo

1) Definir tolerancia error máximo en el constructor

```
15 class Rio:
16     def __init__(self, ancho, largo, dh, tol):
17         """
18         Constructor
19         :param ancho: Ancho
20         :param largo: Largo
21         :param dh: Tamaño grilla diferencial
22         :param tol: Tolerancia
23         :type ancho: int,float
24         """
25         self._ancho = ancho # privada
26         self._largo = largo
27         self._dh = dh
28
29         self._h = int(float(ancho) / dh)
30         self._w = int(float(largo) / dh)
31
32         self._matrix = np.ones((self._h, self._w))
33
34         self.tol = tol
```

Ejemplo

1) Definir tolerancia error máximo en el constructor

```
157  
158  
159 # Instancia rio  
160 r = Rio(3, 7, 0.1, 0.001)  
161 r.cb(10)  
162
```

Ejemplo

2) Crear función **_single_iteration** que calcule sólo 1 iteración.

ANTES:

```

48 def start(self):
49     """
50     Inicia calculo
51     :return:
52     """
53     for _ in tqdm.tqdm(range(1000)):
54         for x in range(1, self._w):
55             for y in range(self._h):
56
57                 # General
58                 if 1 < y < self._h - 2 and x < self._w - 2:
59                     self._matrix[y][x] = 0.25 * (
60                         self._matrix[y - 1][x] + self._matrix[y + 1][x] + self._matrix[y][x - 1] +
61                         self._matrix[y][x + 1])
62

```

Ejemplo

2) Crear función **_single_iteration** que calcule sólo 1 iteración.

AHORA:

```
def _single_iteration(self, matrix_new, matrix_old, omega):  
    """  
    Inicia calculo sólo 1 iteración  
    :return:  
    """  
    for x in range(1, self._w):  
        for y in range(self._h):  
  
            # Valor anterior de la matriz promediado  
            prom = 0  
  
            try:  
                # General  
                if 1 < y < self._h - 2 and x < self._w - 2:  
                    prom = 0.25 * (matrix_old[y - 1][x] + matrix_old[y + 1][x] + matrix_old[y][x - 1] +  
                                   matrix_old[y][x + 1] - 4 * matrix_old[y][x])
```

Ejemplo

2) Crear función **_single_iteration** que calcule sólo 1 iteración.

AHORA:

```
try:
    # General
    if 1 < y < self._h - 2 and x < self._w - 2:
        prom = 0.25 * (matrix_old[y - 1][x] + matrix_old[y + 1][x] + matrix_old[y][x - 1] +
                      matrix_old[y][x + 1] - 4 * matrix_old[y][x])

    # Borde superior
    if y == 0 and x < self._w - 2:
        prom = 0.25 * (2 * matrix_old[y + 1][x] + matrix_old[y][x - 1] +
                      matrix_old[y][x + 1] - 4 * matrix_old[y][x])
```

$$(22) \quad \begin{aligned} u_{i,j} &= u_{i,j} + \omega \left(\frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}}{4} \right) \\ &= u_{i,j} + \omega r_{i,j}, \end{aligned}$$

en la que el parámetro ω verifica $1 \leq \omega < 2$. En el método de sobrerelajación sucesiva, cada paso de la iteración consiste en hacer un barrido de la malla con la fórmula recursiva (22) hasta que se tenga $|r_{i,j}| < \varepsilon$. Para elegir el valor óptimo del parámetro ω hay que estudiar los autovalores de la matriz que caracteriza el método iterativo que estamos usando para resolver un sistema lineal; en nuestro caso, dicho valor óptimo viene dado por la fórmula

$$(23) \quad \omega = \frac{4}{2 + \sqrt{4 - \left(\cos\left(\frac{\pi}{n-1}\right) + \cos\left(\frac{\pi}{m-1}\right) \right)^2}}.$$

Ejemplo

2) Crear función **_single_iteration** que calcule sólo 1 iteración.

AHORA:

```
83         # Borde inferior derecho
84         if y == self._h - 1 and x == self._w - 1:
85             prom = 0.25 * (2 * matrix_old[y - 1][x] + 2 * matrix_old[y][x - 1])
86         except:
87             continue
88
89         # Calcula nuevo valor
90         matrix_new[y][x] = matrix_old[y][x] + prom * omega
91
```

Ejemplo

3) Crear función **_convergio** que indique si el sistema convergió, o sea, que el máximo error es menor que la tolerancia.

```
@staticmethod
def _convergio(mat_old, mat_new, tol):
    """
    Retorna un booleano indicando si el problema convergió o no.

    :param mat_old: Vector de soluciones previo
    :param mat_new: Vector de soluciones posterior
    :param tol: Error máximo admisible
    :return:
    """
    not_zero = (mat_new != 0)
    diff_relativa = (mat_old - mat_new)[not_zero] / mat_new[not_zero]
    max_diff = np.max(np.fabs(diff_relativa))
    return [max_diff < tol, max_diff]
```

Ejemplo

4) Modificar función **start** para crear dos matrices, la de la iteración nueva y pasada, usar **_single_iteration** y **_convergio** para verificar convergencia, retornar número de iteraciones realizadas.

```
107 def start(self, omega):
108     """
109     Soluciona el sistema
110     :return:
111     """
112
113     # Clonamos las matrices
114     mat_new = np.copy(self._matrix)
115
116     # Inicia variables
117     niters = 0
118     run = True
119     converg = []
120     omega = omega - 1
121     if not 0 <= omega <= 1:
122         raise Exception('Omega tiene un valor incorrecto')
```

Ejemplo

4) Modificar función **start** para crear dos matrices, la de la iteración nueva y pasada, usar **_single_iteration** y **_convergio** para verificar convergencia, retornar número de iteraciones realizadas.

```
124     while run:
125         mat_old = np.copy(mat_new)
126         self._single_iteration(mat_new, mat_old, omega)
127         niters += 1
128         converg = self._convergio(mat_old, mat_new, self.tol)
129         run = not converg[0]
130
131     print 'El programa terminó en {0} iteraciones, con error {1}'.format(niters, converg[1])
132     self._matrix = np.copy(mat_new)
133     return niters
```

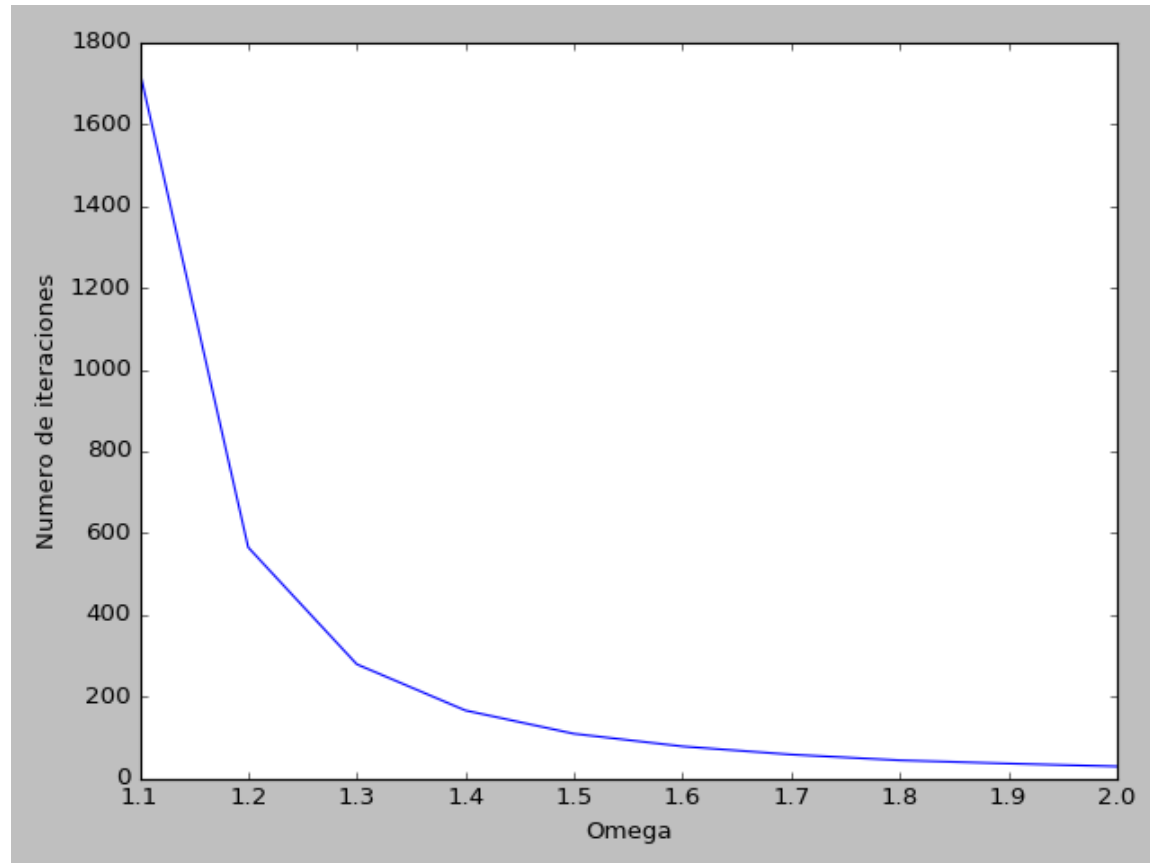
Ejemplo

Veamos qué pasa con el número de iteraciones en función de omega:

```
157
158     # Instancia rio
159     r = Rio(3, 7, 0.1, 0.001)
160     r.cb(10)
161
162     # Se prueban varios w
163     omegas = [1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0]
164     iters = []
165     errors = []
166     for w in omegas:
167         iters.append(r.start(w))
168     r.plot()
```

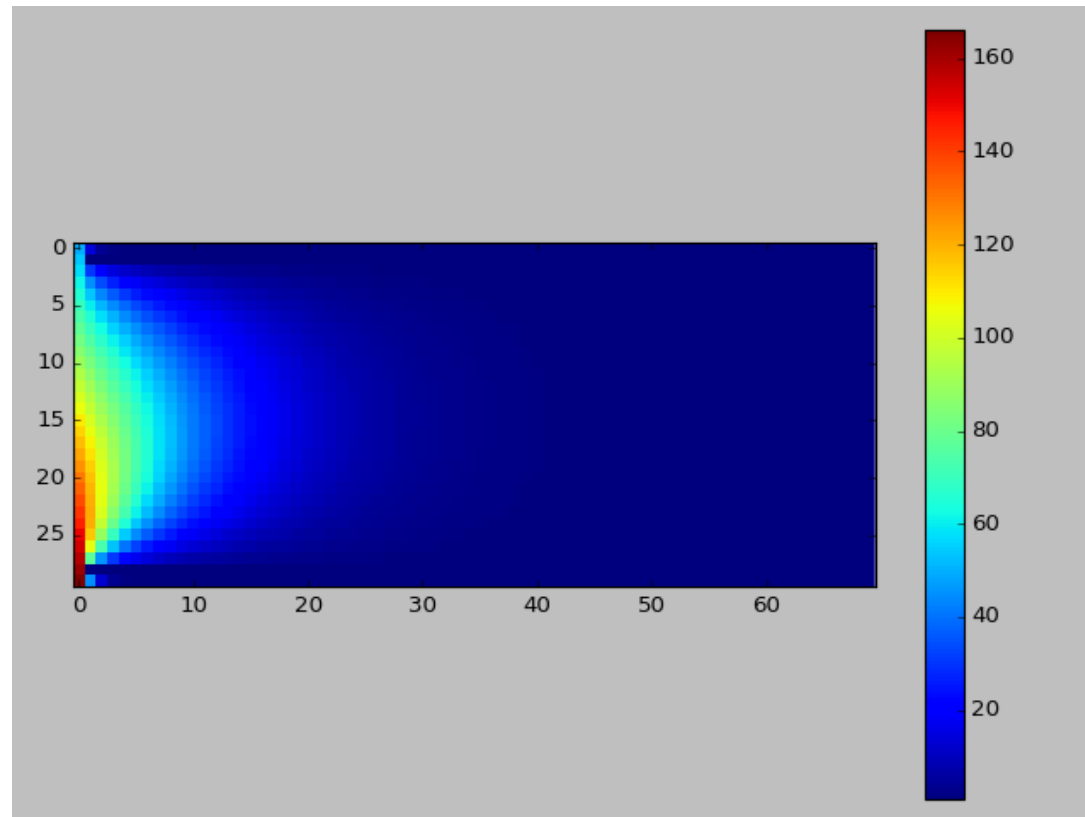
Ejemplo

Veamos qué pasa con el número de iteraciones en función de omega:



Ejemplo

Veamos qué pasa con el número de iteraciones en función de omega:



Ejemplo

¿Qué pasa si $\omega > 2$?

```
CC3501-2018-1/aux 3/rio.py:67: RuntimeWarning: invalid value encountered in double_scalars
CC3501-2018-1/aux 3/rio.py:77: RuntimeWarning: overflow encountered in double_scalars
CC3501-2018-1/aux 3/rio.py:76: RuntimeWarning: overflow encountered in double_scalars
_old[y][x + 1] - 4 *
CC3501-2018-1/aux 3/rio.py:72: RuntimeWarning: overflow encountered in double_scalars
```


¿Preguntas?