Métodos Numéricos Aplicados a la Ingeniería Civil

1.5-Métodos iterativos

Juan Nicolás Ramírez Giraldo
jnramirezg@unal.edu.co

Departamento de Ingeniería Civil
Facultad de Ingeniería y Arquitectura
Universidad Nacional de Colombia
Sede Manizales

UNIVERSIDAD NACIONAL DE COLOMBIA

"Cum cogitaveris quot te antecedant, respice quot sequantur"
Séneca

Métodos Iterativos

1. Método de Jacobi
2. Método de Gauss-Seidel

Método de Jacobi

1. Construcción del método

La propuesta del método de Jacobi busca despejar de cada ecuación una de las incógnitas, de tal manera que, se puedan evaluar iterativamente las demás. Más adelante se discutirán los cuidados numéricos que se deben considerar para garantizar la convergencia.

Para la construcción del método se usa inicialmente un sistema de 3x3 simbólico:

Dado la matriz de coeficientes constantes: $\underline{\underline{A}} = \begin{bmatrix} i_0 & i_1 & i_2 \\ j_0 & j_1 & j_2 \\ k_0 & k_1 & k_2 \end{bmatrix}$

Y el vector de constantes: $\underline{B} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$

Método de Jacobi

1. Construcción del método

Dado la matriz de coeficientes constantes: $\underline{\underline{A}} = \begin{bmatrix} i_0 & i_1 & i_2 \\ j_0 & j_1 & j_2 \\ k_0 & k_1 & k_2 \end{bmatrix}$

Y el vector de constantes: $\underline{B} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$

Se busca hallar el vector de soluciones $\underline{X}$ de la expresión: $\underline{\underline{A}}.\underline{X} = \underline{B}$

En este caso, se define para efectos de programación en Python: $\underline{X} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$

Método de Jacobi

[1]:
```python
# Importación de liberías.
import sympy as sp
```

[2]:
```python
# Incógnitas simbólicas.
x0, x1, x2 = sp.symbols('x_0 x_1 x_2')
```

[3]:
```python
# Constantes simbólicas.
b0, b1, b2 = sp.symbols('b_0 b_1 b_2')
```

[4]:
```python
# Coeficientes constantes simbólicos.
i0, i1, i2 = sp.symbols('i_0 i_1 i_2')  # Se asocian a la primera ec.
j0, j1, j2 = sp.symbols('j_0 j_1 j_2')  # Se asocian a la segunda ec.
k0, k1, k2 = sp.symbols('k_0 k_1 k_2')  # Se asocian a la tercera ec.
```

Método de Jacobi

[5]:
```python
# Definición de la matriz de coeficientes constantes.
A = sp.Matrix([[i0, i1, i2],
               [j0, j1, j2],
               [k0, k1, k2]])
```

[6]:
```python
A
```

[6]:

$$\begin{bmatrix} i_0 & i_1 & i_2 \\ j_0 & j_1 & j_2 \\ k_0 & k_1 & k_2 \end{bmatrix}$$

[7]:
```python
X = sp.Matrix([x0, x1, x2])
```

[8]:
```python
X
```

[8]:

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

Método de Jacobi

[9]:
```python
B = sp.Matrix([b0, b1, b2])
```

[10]:
```python
B
```

[10]:

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$$

[11]:
```python
P = A*X - B  # Esto es igual a 0 porque AX = B
```

[12]:
```python
P
```

[12]:

$$\begin{bmatrix} -b_0 + i_0x_0 + i_1x_1 + i_2x_2 \\ -b_1 + j_0x_0 + j_1x_1 + j_2x_2 \\ -b_2 + k_0x_0 + k_1x_1 + k_2x_2 \end{bmatrix}$$

Método de Jacobi

Por lo que, el sistema queda de la siguiente manera:

$$\begin{bmatrix} -b_0 + i_0 x_0 + i_1 x_1 + i_2 x_2 \\ -b_1 + j_0 x_0 + j_1 x_1 + j_2 x_2 \\ -b_2 + k_0 x_0 + k_1 x_1 + k_2 x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Se despeja x_0, x_1 y x_2 de la primera, segunda y tercera ecuación, respectivamente.

[13]:
```
# Este código por el momento NO es necesario estudiarlo.
e0 = sp.solve(P[0],x0)[0]
e1 = sp.solve(P[1],x1)[0]
e2 = sp.solve(P[2],x2)[0]
```

[14]:
```
# De la ecuación 1, se despeja x0
e0
```

[14]:

$$\frac{b_0 - i_1 x_1 - i_2 x_2}{i_0}$$

Método de Jacobi

Por lo que, el sistema queda de la siguiente manera:

$$\begin{bmatrix} -b_0 + i_0 x_0 + i_1 x_1 + i_2 x_2 \\ -b_1 + j_0 x_0 + j_1 x_1 + j_2 x_2 \\ -b_2 + k_0 x_0 + k_1 x_1 + k_2 x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Se despeja x_0, x_1 y x_2 de la primera, segunda y tercera ecuación, respectivamente.

[13]:
```
# Este código por el momento NO es necesario estudiarlo.
e0 = sp.solve(P[0],x0)[0]
e1 = sp.solve(P[1],x1)[0]
e2 = sp.solve(P[2],x2)[0]
```

[15]:
```
# De la ecuación 2, se despeja x1
e1
```

[15]: $$\frac{b_1 - j_0 x_0 - j_2 x_2}{j_1}$$

Método de Jacobi

Por lo que, el sistema queda de la siguiente manera:

$$\begin{bmatrix} -b_0 + i_0x_0 + i_1x_1 + i_2x_2 \\ -b_1 + j_0x_0 + j_1x_1 + j_2x_2 \\ -b_2 + k_0x_0 + k_1x_1 + k_2x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Se despeja x_0, x_1 y x_2 de la primera, segunda y tercera ecuación, respectivamente.

[13]:
```python
# Este código por el momento NO es necesario estudiarlo.
e0 = sp.solve(P[0],x0)[0]
e1 = sp.solve(P[1],x1)[0]
e2 = sp.solve(P[2],x2)[0]
```

[16]:
```python
# De la ecuación 3, se despeja x2
e2
```

[16]:

$$\frac{b_2 - k_0x_0 - k_1x_1}{k_2}$$

Método de Jacobi

Es decir,

$$x_0 = \frac{b_0 - i_1 x_1 - i_2 x_2}{i_0}$$

$$x_1 = \frac{b_1 - j_0 x_0 - j_2 x_2}{j_1}$$

$$x_2 = \frac{b_2 - k_0 x_0 - k_1 x_1}{k_2}$$

Luego, se asume para la iteración 0 que:

$$\begin{bmatrix} x_0^{(m)} \\ x_1^{(m)} \\ x_2^{(m)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Con las ecuaciones organizadas así:

$$\begin{bmatrix} x_0^{(m+1)} \\ x_1^{(m+1)} \\ x_2^{(m+1)} \end{bmatrix} = \begin{bmatrix} \frac{b_0 - i_1 x_1^{(m)} - i_2 x_2^{(m)}}{i_0} \\ \frac{b_1 - j_0 x_0^{(m)} - j_2 x_2^{(m)}}{j_1} \\ \frac{b_2 - k_0 x_0^{(m)} - k_1 x_1^{(m)}}{k_2} \end{bmatrix}$$

Se obtienen después de la primera iteración que x_0, x_1 y x_2 toman nuevos valores, es decir, los valores que se usarán en la iteración 1. Así, sucesivamente ¿hasta qué iteración?

Es allí, donde se definen unos criterios de error, los cuales se mostrarán más adelante.

Método de Jacobi

1.1. Generalización del método a mxm

Resulta altamente complejo realizar despejes particulares para cada sistema de ecuaciones, por lo que, se busca una estructura matricial simple que llegue a la forma general mxm:

De caso $3x3$ sabemos que la ecuación del método es:

$$\begin{bmatrix} x_0^{(m+1)} \\ x_1^{(m+1)} \\ x_2^{(m+1)} \end{bmatrix} = \begin{bmatrix} \frac{b_0 - i_1 x_1^{(m)} - i_2 x_2^{(m)}}{i_0} \\ \frac{b_1 - j_0 x_0^{(m)} - j_2 x_2^{(m)}}{j_1} \\ \frac{b_2 - k_0 x_0^{(m)} - k_1 x_1^{(m)}}{k_2} \end{bmatrix}$$

Por lo tanto, a partir de la ec(0) se busca construir una forma matricial general que no sea particular para $m = 3$

Método de Jacobi

1.1. Generalización del método a mxm

[17]:
```
# Se construye un vector con los resultados a los que queremos llegar.
ec0 = sp.Matrix([e0, e1, e2])
ec1 = sp.Matrix([e0, e1, e2]) # Se crea una copia para realizar operaciones.
```

[18]:
```
ec0
```

[18]:

$$\begin{bmatrix} \frac{b_0 - i_1 x_1 - i_2 x_2}{i_0} \\ \frac{b_1 - j_0 x_0 - j_2 x_2}{j_1} \\ \frac{b_2 - k_0 x_0 - k_1 x_1}{k_2} \end{bmatrix}$$

¿Cómo se organiza?

$$\begin{bmatrix} \frac{b_0 - i_1 x_1 - i_2 x_2}{i_0} \\ \frac{b_1 - j_0 x_0 - j_2 x_2}{j_1} \\ \frac{b_2 - k_0 x_0 - k_1 x_1}{k_2} \end{bmatrix} \text{ec(0)}$$

Método de Jacobi

1.1. Generalización del método a mxm

$$\boxed{\left[\begin{matrix}\frac{b_0 - i_1 x_1 - i_2 x_2}{i_0}\\\frac{b_1 - j_0 x_0 - j_2 x_2}{j_1}\\\frac{b_2 - k_0 x_0 - k_1 x_1}{k_2}\end{matrix}\right] \text{ec(0)}}$$

La propuesta es eliminar los denominadores de cada fila, de tal manera que, se multiplica la fila 0 por i_0, la fila 1 por j_1 y la fila 2 por k_2.

Luego, esto deberá ser compensado multiplicando por $1/i_0$, $1/j_1$ y $1/k_2$, correpondientemente, en un paso siguiente a fin de consevar la igualdad.

[19]:
```
ec1[0] = ec1[0]*(i0)
ec1[1] = ec1[1]*(j1)
ec1[2] = ec1[2]*(k2)
```

[20]:
```
ec1 # Ecuación 1
```

[20]:
$$\left[\begin{matrix}b_0 - i_1 x_1 - i_2 x_2\\b_1 - j_0 x_0 - j_2 x_2\\b_2 - k_0 x_0 - k_1 x_1\end{matrix}\right]$$

Método de Jacobi

1.1. Generalización del método a mxm

$$\begin{bmatrix} \frac{b_0 - i_1 x_1 - i_2 x_2}{i_0} \\ \frac{b_1 - j_0 x_0 - j_2 x_2}{j_1} \\ \frac{b_2 - k_0 x_0 - k_1 x_1}{k_2} \end{bmatrix} \text{ec(0)}$$

$$\begin{bmatrix} b_0 - i_1 x_1 - i_2 x_2 \\ b_1 - j_0 x_0 - j_2 x_2 \\ b_2 - k_0 x_0 - k_1 x_1 \end{bmatrix} \text{ec(1)}$$

Recordando que:

$$\underline{\underline{A}} = \begin{bmatrix} i_0 & i_1 & i_2 \\ j_0 & j_1 & j_2 \\ k_0 & k_1 & k_2 \end{bmatrix} \qquad \underline{B} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} \qquad \underline{X} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

¿Qué pasa si se realiza esta operación?

$$\underline{B} - \underline{\underline{A}}.\underline{X}$$

[21]:
```
ec2 = B - A*X   # Ecuación 2
```

Método de Jacobi

1.1. Generalización del método a mxm

$$\begin{bmatrix} \frac{b_0 - i_1 x_1 - i_2 x_2}{i_0} \\ \frac{b_1 - j_0 x_0 - j_2 x_2}{j_1} \\ \frac{b_2 - k_0 x_0 - k_1 x_1}{k_2} \end{bmatrix} \text{ec(0)}$$

$$\begin{bmatrix} b_0 - i_1 x_1 - i_2 x_2 \\ b_1 - j_0 x_0 - j_2 x_2 \\ b_2 - k_0 x_0 - k_1 x_1 \end{bmatrix} \text{ec(1)}$$

$$\begin{bmatrix} b_0 - i_0 x_0 - i_1 x_1 - i_2 x_2 \\ b_1 - j_0 x_0 - j_1 x_1 - j_2 x_2 \\ b_2 - k_0 x_0 - k_1 x_1 - k_2 x_2 \end{bmatrix} \text{ec(2)}$$

¿Qué similitudes tienen la ecuación 1 y la ecuación 2 ($\underline{B} - \underline{\underline{A}}.\underline{X}$)?

Aplicamos la resta de las ecuaciones y revisamos qué elementos quedan:

[22]:
```
ec2-ec1
```

[22]:

$$\begin{bmatrix} -i_0 x_0 \\ -j_1 x_1 \\ -k_2 x_2 \end{bmatrix}$$

Método de Jacobi

1.1. Generalización del método a mxm

$$\begin{bmatrix} \frac{b_0 - i_1 x_1 - i_2 x_2}{i_0} \\ \frac{b_1 - j_0 x_0 - j_2 x_2}{j_1} \\ \frac{b_2 - k_0 x_0 - k_1 x_1}{k_2} \end{bmatrix} \text{ec(0)}$$

$$\begin{bmatrix} b_0 - i_1 x_1 - i_2 x_2 \\ b_1 - j_0 x_0 - j_2 x_2 \\ b_2 - k_0 x_0 - k_1 x_1 \end{bmatrix} \text{ec(1)}$$

$$\begin{bmatrix} b_0 - i_0 x_0 - i_1 x_1 - i_2 x_2 \\ b_1 - j_0 x_0 - j_1 x_1 - j_2 x_2 \\ b_2 - k_0 x_0 - k_1 x_1 - k_2 x_2 \end{bmatrix} \text{ec(2)}$$

[22]:
```
ec2-ec1
```

[22]: $$\begin{bmatrix} -i_0 x_0 \\ -j_1 x_1 \\ -k_2 x_2 \end{bmatrix}$$

$$\underline{\underline{A}} = \begin{bmatrix} i_0 & i_1 & i_2 \\ j_0 & j_1 & j_2 \\ k_0 & k_1 & k_2 \end{bmatrix}$$

$$\underline{X} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

¿Qué elementos no son comunes?

$$\begin{bmatrix} b_0 - \underline{i_0 x_0} - i_1 x_1 - i_2 x_2 \\ b_1 - j_0 x_0 - \underline{j_1 x_1} - j_2 x_2 \\ b_2 - k_0 x_0 - k_1 x_1 - \underline{k_2 x_2} \end{bmatrix}$$

Método de Jacobi

1.1. Generalización del método a mxm

$$\left[\begin{matrix}\frac{b_0-i_1x_1-i_2x_2}{i_0}\\\frac{b_1-j_0x_0-j_2x_2}{j_1}\\\frac{b_2-k_0x_0-k_1x_1}{k_2}\end{matrix}\right]\text{ec(0)}$$

$$\left[\begin{matrix}b_0-i_1x_1-i_2x_2\\b_1-j_0x_0-j_2x_2\\b_2-k_0x_0-k_1x_1\end{matrix}\right]\text{ec(1)}$$

$$\left[\begin{matrix}b_0-i_0x_0-i_1x_1-i_2x_2\\b_1-j_0x_0-j_1x_1-j_2x_2\\b_2-k_0x_0-k_1x_1-k_2x_2\end{matrix}\right]\text{ec(2)}$$

[22]:
```
ec2-ec1
```

[22]:

$$\left[\begin{matrix}-i_0x_0\\-j_1x_1\\-k_2x_2\end{matrix}\right]$$

Claramente, corresponden a los elementos de la diagonal de la matriz de coeficientes constantes.

Por lo tanto, la propuesta consiste en crear una matriz $\underline{\underline{T}}$ con los elementos de $\underline{\underline{A}}$, pero sin los elementos de la diagonal.

Método de Jacobi

1.1. Generalización del método a mxm

$$\begin{bmatrix} \frac{b_0 - i_1 x_1 - i_2 x_2}{i_0} \\ \frac{b_1 - j_0 x_0 - j_2 x_2}{j_1} \\ \frac{b_2 - k_0 x_0 - k_1 x_1}{k_2} \end{bmatrix} \text{ec(0)}$$

$$\begin{bmatrix} b_0 - i_1 x_1 - i_2 x_2 \\ b_1 - j_0 x_0 - j_2 x_2 \\ b_2 - k_0 x_0 - k_1 x_1 \end{bmatrix} \text{ec(1)}$$

$$\begin{bmatrix} b_0 - i_0 x_0 - i_1 x_1 - i_2 x_2 \\ b_1 - j_0 x_0 - j_1 x_1 - j_2 x_2 \\ b_2 - k_0 x_0 - k_1 x_1 - k_2 x_2 \end{bmatrix} \text{ec(2)}$$

[23]:
```
# Se crea una matriz T con los mismos coeficientes constantes,
# pero sin la diagonal.
T = sp.Matrix([[ 0, i1, i2],
               [j0,  0, j2],
               [k0, k1,  0]])
```

[24]:
```
T
```

[24]:

$$\begin{bmatrix} 0 & i_1 & i_2 \\ j_0 & 0 & j_2 \\ k_0 & k_1 & 0 \end{bmatrix}$$

Método de Jacobi

1.1. Generalización del método a mxm

$$\begin{bmatrix} \frac{b_0-i_1x_1-i_2x_2}{i_0} \\ \frac{b_1-j_0x_0-j_2x_2}{j_1} \\ \frac{b_2-k_0x_0-k_1x_1}{k_2} \end{bmatrix} \text{ec(0)}$$

$$\begin{bmatrix} b_0 - i_1x_1 - i_2x_2 \\ b_1 - j_0x_0 - j_2x_2 \\ b_2 - k_0x_0 - k_1x_1 \end{bmatrix} \text{ec(1)}$$

$$\begin{bmatrix} b_0 - i_0x_0 - i_1x_1 - i_2x_2 \\ b_1 - j_0x_0 - j_1x_1 - j_2x_2 \\ b_2 - k_0x_0 - k_1x_1 - k_2x_2 \end{bmatrix} \text{ec(2)}$$

Ahora, en vez de usar:

$\underline{B} - \underline{\underline{A}}.\underline{X}$ ec(2)

Se usa: $\underline{B} - \underline{\underline{T}}.\underline{X}$ ec(3)

[25]:
```
ec3 = B - T*X
```

[26]:
```
ec3
```

[26]:

$$\begin{bmatrix} b_0 - i_1x_1 - i_2x_2 \\ b_1 - j_0x_0 - j_2x_2 \\ b_2 - k_0x_0 - k_1x_1 \end{bmatrix}$$

Método de Jacobi

1.1. Generalización del método a mxm

$$\begin{bmatrix} \frac{b_0 - i_1 x_1 - i_2 x_2}{i_0} \\ \frac{b_1 - j_0 x_0 - j_2 x_2}{j_1} \\ \frac{b_2 - k_0 x_0 - k_1 x_1}{k_2} \end{bmatrix} \text{ec(0)}$$

$$\begin{bmatrix} b_0 - i_1 x_1 - i_2 x_2 \\ b_1 - j_0 x_0 - j_2 x_2 \\ b_2 - k_0 x_0 - k_1 x_1 \end{bmatrix} \text{ec(1)}$$

$$\begin{bmatrix} b_0 - i_0 x_0 - i_1 x_1 - i_2 x_2 \\ b_1 - j_0 x_0 - j_1 x_1 - j_2 x_2 \\ b_2 - k_0 x_0 - k_1 x_1 - k_2 x_2 \end{bmatrix} \text{ec(2)}$$

Ahora, en vez de usar:

$\underline{B} - \underline{\underline{A}}.\ \underline{X}$ ec(2)

Se usa: $\underline{B} - \underline{\underline{T}}.\ \underline{X}$ ec(3)

Se verifica que la ecuación 3 con la nueva propuesta sea igual a la ecuación 1:

[27]:
```
ec3-ec1
```

[27]:
$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Método de Jacobi

1.1. Generalización del método a mxm

$$\begin{bmatrix} \frac{b_0 - i_1 x_1 - i_2 x_2}{i_0} \\ \frac{b_1 - j_0 x_0 - j_2 x_2}{j_1} \\ \frac{b_2 - k_0 x_0 - k_1 x_1}{k_2} \end{bmatrix} \text{ec(0)}$$

$$\begin{bmatrix} b_0 - i_1 x_1 - i_2 x_2 \\ b_1 - j_0 x_0 - j_2 x_2 \\ b_2 - k_0 x_0 - k_1 x_1 \end{bmatrix} \text{ec(1)}$$

$$\begin{bmatrix} b_0 - i_0 x_0 - i_1 x_1 - i_2 x_2 \\ b_1 - j_0 x_0 - j_1 x_1 - j_2 x_2 \\ b_2 - k_0 x_0 - k_1 x_1 - k_2 x_2 \end{bmatrix} \text{ec(2)}$$

Ahora sí son iguales. Solo falta multiplicar cada fila por $1/i_0$, $1/j_1$ y $1/k_2$, correpondientemente, para llegar a la ec(0).

¿Cómo se hace esto a partir de una operación matricial simple?

[28]:
```
# Se crea una matriz D con la diagonal de A.
D = sp.Matrix([
               [i0,  0,  0],
               [ 0, j1,  0],
               [ 0,  0, k2],
                           ])
```

Método de Jacobi

1.1. Generalización del método a mxm

$$\left[\begin{matrix}\frac{b_0-i_1x_1-i_2x_2}{i_0}\\\frac{b_1-j_0x_0-j_2x_2}{j_1}\\\frac{b_2-k_0x_0-k_1x_1}{k_2}\end{matrix}\right] \text{ec(0)}$$

$$\left[\begin{matrix}b_0-i_1x_1-i_2x_2\\b_1-j_0x_0-j_2x_2\\b_2-k_0x_0-k_1x_1\end{matrix}\right] \text{ec(1)}$$

$$\left[\begin{matrix}b_0-i_0x_0-i_1x_1-i_2x_2\\b_1-j_0x_0-j_1x_1-j_2x_2\\b_2-k_0x_0-k_1x_1-k_2x_2\end{matrix}\right] \text{ec(2)}$$

Solo falta multiplicar cada fila por $1/i_0$, $1/j_1$ y $1/k_2$

```
[29]: D
```

[29]:
$$\left[\begin{matrix}i_0 & 0 & 0\\0 & j_1 & 0\\0 & 0 & k_2\end{matrix}\right]$$

Nótese que $\underline{\underline{A}} = \underline{\underline{D}} + \underline{\underline{T}}$, es decir, una forma de descomponer la matriz.

Método de Jacobi

1.1. Generalización del método a mxm

$$\begin{bmatrix} \frac{b_0-i_1x_1-i_2x_2}{i_0} \\ \frac{b_1-j_0x_0-j_2x_2}{j_1} \\ \frac{b_2-k_0x_0-k_1x_1}{k_2} \end{bmatrix} \text{ec}(0)$$

$$\begin{bmatrix} b_0 - i_1x_1 - i_2x_2 \\ b_1 - j_0x_0 - j_2x_2 \\ b_2 - k_0x_0 - k_1x_1 \end{bmatrix} \text{ec}(1)$$

$$\begin{bmatrix} b_0 - i_0x_0 - i_1x_1 - i_2x_2 \\ b_1 - j_0x_0 - j_1x_1 - j_2x_2 \\ b_2 - k_0x_0 - k_1x_1 - k_2x_2 \end{bmatrix} \text{ec}(2)$$

Solo falta multiplicar cada fila por $1/i_0$, $1/j_1$ y $1/k_2$

[30]:
```
# Se comprueba lo anterior.
A-(D+T)
```

[30]:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Método de Jacobi

1.1. Generalización del método a mxm

$$\begin{bmatrix} \frac{b_0 - i_1 x_1 - i_2 x_2}{i_0} \\ \frac{b_1 - j_0 x_0 - j_2 x_2}{j_1} \\ \frac{b_2 - k_0 x_0 - k_1 x_1}{k_2} \end{bmatrix} \text{ec(0)}$$

$$\begin{bmatrix} b_0 - i_1 x_1 - i_2 x_2 \\ b_1 - j_0 x_0 - j_2 x_2 \\ b_2 - k_0 x_0 - k_1 x_1 \end{bmatrix} \text{ec(1)}$$

$$\begin{bmatrix} b_0 - i_0 x_0 - i_1 x_1 - i_2 x_2 \\ b_1 - j_0 x_0 - j_1 x_1 - j_2 x_2 \\ b_2 - k_0 x_0 - k_1 x_1 - k_2 x_2 \end{bmatrix} \text{ec(2)}$$

Solo falta multiplicar cada fila por $1/i_0$, $1/j_1$ y $1/k_2$

Luego, $\underline{\underline{D}}^{-1}$ es:

[31]:
```
D**-1
```

[31]:

$$\begin{bmatrix} \frac{1}{i_0} & 0 & 0 \\ 0 & \frac{1}{j_1} & 0 \\ 0 & 0 & \frac{1}{k_2} \end{bmatrix}$$

Que tiene un muy bajo costo computacional, e incluso no es necesario realizar la operación con el concepto de inversa, sino con una rutina particular.

Método de Jacobi

1.1. Generalización del método a mxm

$$\left[\begin{matrix}\frac{b_0-i_1x_1-i_2x_2}{i_0}\\\frac{b_1-j_0x_0-j_2x_2}{j_1}\\\frac{b_2-k_0x_0-k_1x_1}{k_2}\end{matrix}\right]\text{ec(0)}$$

$$\left[\begin{matrix}b_0-i_1x_1-i_2x_2\\b_1-j_0x_0-j_2x_2\\b_2-k_0x_0-k_1x_1\end{matrix}\right]\text{ec(1)}$$

$$\left[\begin{matrix}b_0-i_0x_0-i_1x_1-i_2x_2\\b_1-j_0x_0-j_1x_1-j_2x_2\\b_2-k_0x_0-k_1x_1-k_2x_2\end{matrix}\right]\text{ec(2)}$$

Solo falta multiplicar cada fila por $1/i_0$, $1/j_1$ y $1/k_2$

Se verifica si con esta expresión se llega a la ecuación 0:

$$\underline{\underline{D}}^{-1}(\underline{B}-\underline{\underline{T}}.\underline{X})$$

[32]:
```
# Se verifica lo anterior.
ec4 = D**-1*(B - T*X)
```

Método de Jacobi

1.1. Generalización del método a mxm

$$\left[\begin{matrix}\frac{b_0-i_1x_1-i_2x_2}{i_0}\\ \frac{b_1-j_0x_0-j_2x_2}{j_1}\\ \frac{b_2-k_0x_0-k_1x_1}{k_2}\end{matrix}\right] \text{ec(0)}$$

$$\left[\begin{matrix}b_0 - i_1x_1 - i_2x_2\\ b_1 - j_0x_0 - j_2x_2\\ b_2 - k_0x_0 - k_1x_1\end{matrix}\right] \text{ec(1)}$$

$$\left[\begin{matrix}b_0 - i_0x_0 - i_1x_1 - i_2x_2\\ b_1 - j_0x_0 - j_1x_1 - j_2x_2\\ b_2 - k_0x_0 - k_1x_1 - k_2x_2\end{matrix}\right] \text{ec(2)}$$

Solo falta multiplicar cada fila por $1/i_0$, $1/j_1$ y $1/k_2$

Se verifica si con esta expresión se llega a la ecuación 0:

$$\underline{\underline{D}}^{-1}(\underline{B} - \underline{\underline{T}}.\underline{X})$$

[33]:
```
ec4
```

[33]:

$$\left[\begin{matrix}\frac{b_0-i_1x_1-i_2x_2}{i_0}\\ \frac{b_1-j_0x_0-j_2x_2}{j_1}\\ \frac{b_2-k_0x_0-k_1x_1}{k_2}\end{matrix}\right]$$

Método de Jacobi

1.1. Generalización del método a mxm

$$\left[\begin{matrix}\frac{b_{0}-i_{1}x_{1}-i_{2}x_{2}}{i_{0}}\\\frac{b_{1}-j_{0}x_{0}-j_{2}x_{2}}{j_{1}}\\\frac{b_{2}-k_{0}x_{0}-k_{1}x_{1}}{k_{2}}\end{matrix}\right] \text{ec(0)}$$

$$\left[\begin{matrix}b_{0} - i_{1} x_{1} - i_{2} x_{2}\\b_{1} - j_{0} x_{0} - j_{2} x_{2}\\b_{2} - k_{0} x_{0} - k_{1} x_{1}\end{matrix}\right] \text{ec(1)}$$

$$\left[\begin{matrix}b_{0} - i_{0} x_{0} - i_{1} x_{1} - i_{2} x_{2}\\b_{1} - j_{0} x_{0} - j_{1} x_{1} - j_{2} x_{2}\\b_{2} - k_{0} x_{0} - k_{1} x_{1} - k_{2} x_{2}\end{matrix}\right] \text{ec(2)}$$

Solo falta multiplicar cada fila por $1/i_0$, $1/j_1$ y $1/k_2$

Se verifica si con esta expresión se llega a la ecuación 0:

$$\underline{\underline{D}}^{-1}(\underline{B} - \underline{\underline{T}}.\underline{X})$$

[34]:
```python
# Se verifica si ec(4)=ec(0)
ec4-ec0
```

[34]:

$$\left[\begin{matrix}0\\0\\0\end{matrix}\right]$$

Método de Jacobi

1.1. Generalización del método a mxm

Por lo tanto,

$$\begin{bmatrix} x_0^{(m+1)} \\ x_1^{(m+1)} \\ x_2^{(m+1)} \end{bmatrix} = \begin{bmatrix} \frac{1}{i_0} & 0 & 0 \\ 0 & \frac{1}{j_1} & 0 \\ 0 & 0 & \frac{1}{k_2} \end{bmatrix} \left(\begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} - \begin{bmatrix} 0 & i_1 & i_2 \\ j_0 & 0 & j_2 \\ k_0 & k_1 & 0 \end{bmatrix} \begin{bmatrix} x_0^{(m)} \\ x_1^{(m)} \\ x_2^{(m)} \end{bmatrix} \right)$$

Con lo que se llega a la expresión buscada:

$$\begin{bmatrix} x_0^{(m+1)} \\ x_1^{(m+1)} \\ x_2^{(m+1)} \end{bmatrix} = \begin{bmatrix} \frac{b_0 - i_1 x_1^{(m)} - i_2 x_2^{(m)}}{i_0} \\ \frac{b_1 - j_0 x_0^{(m)} - j_2 x_2^{(m)}}{j_1} \\ \frac{b_2 - k_0 x_0^{(m)} - k_1 x_1^{(m)}}{k_2} \end{bmatrix}$$

Método de Jacobi

1.1. Generalización del método a mxm

La expresión general del método es:

$$\underline{X}^{m+1} = \underline{\underline{D}}^{-1}(\underline{B} - \underline{\underline{T}}.\underline{X}^{m})$$

en donde, $\underline{\underline{A}} = \underline{\underline{D}} + \underline{\underline{T}}$

$\underline{\underline{D}}$ es una matriz diagonal con los elementos de la diagonal de la matriz $\underline{\underline{A}}$.

$$\boxed{\underline{X}^{m+1} = \underline{\underline{D}}^{-1}(\underline{B} - \underline{\underline{T}}.\underline{X}^{m})}$$

Método de Jacobi

1.3. Solución numérica con Numpy

Se realiza una propuesta de solución de sistemas de ecuaciones usando el método de Jacobi con la librería Numpy de Python, únicamente usando las operaciones matriciales básicas.

[103]:
```
# Importación de librerías
import numpy as np
```

Inicialmente se crea una rutica para el caso particular de un sistema de $3x3$, pero en pasos posteriores se sistematiza para nxn.

[104]:
```
# Se define la matriz coeficientes constantes A como una lista de listas.
A = [[ 3, -1, -1],
     [-1,  3,  1],
     [ 2,  1,  4]]
# Se define el vector B como una lista.
B = [1, 3, 7]
```

Método de Jacobi

1.3. Solución numérica con Numpy

Se convierten los datos de entrada de listas a `np.array()` :

[105]:
```
A = np.array(A)
B = np.array(B)
```

Se define el tamaño del sistema, de las operaciones y los ciclos:

[106]:
```
n = len(A)
```

Matriz $\underline{\underline{D}}$ (con la diagonal de $\underline{\underline{A}}$)

[107]:
```
np.diag(np.diag(A))
```

[107]:
```
array([[3, 0, 0],
       [0, 3, 0],
       [0, 0, 4]])
```

[108]:
```
D = np.diag(np.diag(A))
```

Método de Jacobi

1.3. Solución numérica con Numpy

Matriz $\underline{\underline{T}}$ (con los elementos que no son de la diagonal de $\underline{\underline{A}}$)

[109]:
```
A-D
```

[109]:
```
array([[ 0, -1, -1],
       [-1,  0,  1],
       [ 2,  1,  0]])
```

[110]:
```
T = A-D
```

Método de Jacobi

1.3. Solución numérica con Numpy

Matriz $\underline{\underline{D}}^{-1}$

[111]:
```
1/np.diag(A) # Esto no es lo mismo que la inversa matricial.
```

[111]:
```
array([0.33333333, 0.33333333, 0.25      ])
```

[112]:
```
np.diag(1/np.diag(A))
```

[112]:
```
array([[0.33333333, 0.        , 0.        ],
       [0.        , 0.33333333, 0.        ],
       [0.        , 0.        , 0.25      ]])
```

[113]:
```
Dinv = np.diag(1/np.diag(A))
```

Método de Jacobi

1.3. Solución numérica con Numpy

Vector inicial de solución X_0

[114]:
```
np.zeros(n)
```

[114]:
```
array([0., 0., 0.])
```

[115]:
```
X0 = np.zeros(n)
```

Método de Jacobi

1.3. Solución numérica con Numpy

[108]:
```
D = np.diag(np.diag(A))
```

[110]:
```
T = A-D
```

[113]:
```
Dinv = np.diag(1/np.diag(A))
```

[115]:
```
X0 = np.zeros(n)
```

Iteraciones de la solución

Se realiza iterativamente este cálculo:

$$\underline{X}^{m+1} = \underline{\underline{D}}^{-1}(\underline{B} - \underline{\underline{T}}.\underline{X}^{m})$$

a partir de:

$$\underline{X_0} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Método de Jacobi

1.3. Solución numérica con Numpy

[116]:
```
TX0 = T@X0
B_TX0 = B-TX0
X1 = Dinv@B_TX0
```

[117]:
```
X1
```

[117]:
```
array([0.33333333, 1.        , 1.75      ])
```

[118]:
```
TX1 = T@X1
B_TX1 = B-TX1
X2 = Dinv@B_TX1
```

[119]:
```
X2
```

[119]:
```
array([1.25      , 0.52777778, 1.33333333])
```

[120]:
```
TX2 = T@X2
B_TX2 = B-TX2
X3 = Dinv@B_TX2
```

[121]:
```
X3
```

[121]:
```
array([0.9537037 , 0.97222222, 0.99305556])
```

Método de Jacobi

1.3. Solución numérica con Numpy

En este caso sabemos que la solución es `[1, 1, 1]`, y realmente se empieza a acercar con la tercera iteración.

Una propuesta es crear un ciclo en `m` pasos, donde `m` puede ser un número como 10 o 100, y así, se garantiza muchos recálculos. ¿serán suficientes?

Se prueba con 5 ciclos:

[122]:
```
X0 = np.zeros(n)  # Se reinicia la solución.
for i in range(5):
    TX0 = T@X0
    B_TX0 = B-TX0
    X1 = Dinv@B_TX0
    X0 = X1
```

[123]:
```
X0
```

[123]:
```
array([1.00565844, 0.98611111, 1.00906636])
```

Método de Jacobi

1.3. Solución numérica con Numpy

En este caso sabemos que la solución es `[1, 1, 1]`, y realmente se empieza a acercar con la tercera iteración.

Una propuesta es crear un ciclo en `m` pasos, donde `m` puede ser un número como 10 o 100, y así, se garantiza muchos recálculos. ¿serán suficientes?

Se prueba con 17 ciclos:

[124]:
```
X0 = np.zeros(n)  # Se reinicia la solución.
for i in range(17):
    TX0 = T@X0
    B_TX0 = B-TX0
    X1 = Dinv@B_TX0
    X0 = X1
```

[125]:
```
X0
```

[125]:
```
array([1.        , 0.99999999, 1.00000001])
```

Método de Jacobi

1.3. Solución numérica con Numpy

En este caso sabemos que la solución es `[1, 1, 1]`, y realmente se empieza a acercar con la tercera iteración.

Una propuesta es crear un ciclo en `m` pasos, donde `m` puede ser un número como 10 o 100, y así, se garantiza muchos recálculos. ¿serán suficientes?

Se prueba con 18 ciclos:

[126]:
```
X0 = np.zeros(n)  # Se reinicia la solución.
for i in range(18):
    TX0 = T@X0
    B_TX0 = B-TX0
    X1 = Dinv@B_TX0
    X0 = X1
```

[127]:
```
X0
```

[127]:
```
array([1., 1., 1.])
```

Método de Jacobi

1.3. Solución numérica con Numpy

En este caso, claramente fue suficiente y la respuesta convergió muy bien. Pero se hace necesario un concepto más general para definir la cantidad de pasos de iteración a realizar para cualquier sistema.

En **Khoury R., Harder, D. W. (2016)** se propone comparar el valor la distancia entre dos puntos soluciones en pasos sucesivos, de tal manera que, cuando se llegue a un valor de tolerancia definido, paren las iteraciones. Este criterio, tiene sentido en la medida de que cada vez más la solución de una iteración a otra se va a parecer más.

Método de Jacobi

1.3. Solución numérica con Numpy

Criterio de convergencia

Suponiendo que de un paso a otro se obtuvieron las soluciones `X1` y `X2`:

```
[128]: X1 = np.array([0.9537037 , 0.97222222, 0.99305556])
       X2 = np.array([1.25       , 0.52777778, 1.33333333])
```

Teniendo en cuenta que la distancia entre un punto $\underline{A} = (x_1, y_1, z_1)$ y un punto $\underline{B} = (x_2, y_2, z_2)$ es:

$$dist_{AB} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Método de Jacobi

1.3. Solución numérica con Numpy

Criterio de convergencia

$$dist_{AB} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

```
[129]: X2-X1
```

```
[129]: array([ 0.2962963 , -0.44444444,  0.34027777])
```

```
[130]: (X2-X1)**2
```

```
[130]: array([0.0877915 , 0.19753086, 0.11578896])
```

```
[131]: sum((X2-X1)**2)
```

```
[131]: 0.4011131839677666
```

Método de Jacobi

1.3. Solución numérica con Numpy

Criterio de convergencia

$$dist_{AB} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Finalmente, la distancia entre los dos puntos es:

```
[132]: sum((X2-X1)**2)**0.5
```

```
[132]: 0.6333334969798903
```

Método de Jacobi

1.3. Solución numérica con Numpy

Criterio de convergencia

Se implementa el criterio de convergencia mediante un ciclo `while True` - `break`:

[133]:

```
X0 = np.zeros(n)              # Se reinicia la solución.
toler = 0.0000000000000001   # Definición de la tolerancia.
paso = 0

while True:
    TX0 = T@X0
    B_TX0 = B-TX0
    X1 = Dinv@B_TX0
    paso += 1
    # Criterio de convergencia
    dis_puntos = sum((X1-X0)**2)**0.5
    if dis_puntos < toler:
        print(f"Se hicieron {paso} iteraciones")
        break
    X0 = X1
```

```
Se hicieron 36 iteraciones
```

Método de Jacobi

1.3. Solución numérica con Numpy

Criterio de convergencia

[134]:
```
X0
```

[134]:
```
array([1., 1., 1.])
```

Método de Jacobi

1.3. Solución numérica con Numpy

Se implementan los resultados anteriores como una función:

[135]:
```
def np_metodo_jacobi(A, B):
    A = np.array(A)
    B = np.array(B)
    n = len(A)
    # Se definen las matrices constantes del método.
    D = np.diag(np.diag(A))
    Dinv = np.diag(1/np.diag(A))
    T = A-D
    # Se define X0, para la primera iteración.
    X0 = np.zeros(n)

    toler = 0.0000000000000001
    paso = 0

    while True:
        TX0 = T@X0
        B_TX0 = B-TX0
        X1 = Dinv@B_TX0
        paso += 1
        # Criterio de convergencia
        dis_puntos = sum((X1-X0)**2)**0.5
        if dis_puntos < toler:
            print(f"Se hicieron {paso} iteraciones")
            break
        X0 = X1

    X = X0
    return X
```

Método de Jacobi

1.3. Solución numérica con Numpy

Probando con el ejemplo inicial:

[136]:
```
np_metodo_jacobi(A, B)
```

```
Se hicieron 36 iteraciones
```

[136]:
```
array([1., 1., 1.])
```

Ejemplo 2:

[137]:
```
A1 = [[11, -9],[11, 13]]
B1 = [99, 286]
```

[138]:
```
np_metodo_jacobi(A1, B1)
```

```
Se hicieron 198 iteraciones
```

[138]:
```
array([15.95454545,  8.5       ])
```

Método de Jacobi

1.3. Solución numérica con Numpy

Ejemplo 3:

```
[139]: A2 = [[11, 13],[11, -9]]
       B2 = [286, 99]
```

```
[140]: np_metodo_jacobi(A2, B2) # Si se corre, se debe reiniciar suspender manualmente.
```

```
<ipython-input-135-7e3b59ebc933>:21: RuntimeWarning: overflow encountered in squ
are
  dis_puntos = sum((X1-X0)**2)**0.5
<ipython-input-135-7e3b59ebc933>:16: RuntimeWarning: overflow encountered in mat
mul
```

"El error de desbordamiento implica que una operación produce un valor fuera del rango definido para el tipo de datos correspondiente. Para numpy double, ese rango es (-1.79769313486e + 308, 1.79769313486e + 308)"

Fuente: stackoverflow.com

Método de Jacobi

1.3. Solución numérica con Numpy

Con lo anterior, se evidencia que la función nunca arroja un resultado, por lo que se debe establecer un límite en la cantidad de repeticiones, de tal manera que, se identifiquen los sistemas que **no convergen**.

Se plantea la siguiente función, a la cual aún le faltan criterios de mejoramiento de la solución:

Ver: [15-np_metodo_jacobi.py]

Método de Jacobi

1.4. Discusión final acerca de la convergencia de las soluciones

Observe en los ejemplos 7 y 8 que, el problema $\underline{\underline{A}}_6 \cdot \underline{X} = \underline{B}_6$ tiene la misma solución que $\underline{\underline{A}}_7 \cdot \underline{X} = \underline{B}_7$, pero en un caso el método converge a una respuesta y en el otro no. ¿Qué hacer?

En **Chapra (2015)** se expone que una condición **suficiente** pero **no necesaria** para garantizar la convergencia es que el elemento de la diagonal de la matriz de coeficientes constantes debe ser mayor que todo elemento fuera de la diagonal de cada fila. Esto, generalizado es para un sistema de n ecuaciones:

$$|a_{ii}| > \sum_{\substack{j=1 \\ i \neq j}}^{n} |a_{ij}|$$

De donde surge el concepto de matriz dominante.

Método de Jacobi

1.4. Discusión final acerca de la convergencia de las soluciones

Es claro que el pivoteo parcial es una forma de buscar matrices en lo posible dominantes, también sirve para evitar los problemas de ceros (o valores cercanos) en la diagonal. Sin embargo, para este método se debe complementar con una estrategia que se implementaba implícitamente en el método de eliminación de Gauss-Jordan: **normalización respecto al pivote** (o elemento de la diagonal), con el objetivo de evitar divisiones con números muy grandes o muy pequeños. Y también existe una técnica adicional que se denomina **relajación**.

Referencias

Chapra, S. C., & Canale, R. P. (2015). *Métodos Numéricos para ingenieros* (7.ª ed.). México D.F.: Mc Graw Hill.

Khoury R., Harder, D. W. (2016). *Numerical Methods and Modelling for Engineering*. Springer.