Práctica 2 – Matemática Aplicada. Grupo 1^oB

Métodos numéricos de resolución de sistemas de ecuaciones lineales.

Índice

Partes de un vector. Partes de una matriz.

Si tenemos un vector $x = \{x_0, x_1, x_2, \ldots, x_{n-1}\}$ con n componentes, en Python podemos hacer referencia a sus componentes con `x[0]`, `x[1]`, `x[2]`, ..., `x[n-1]`.

Si queremos referenciar a una parte del vector podemos usar dos puntos, de forma que `x[0:m]` es un vector con los elementos `x[0]`, ..., `x[m-1]`. El siguiente ejemplo obtiene todas las componentes de un vector en tres partes distintas.

Ejemplo 1.

```
>>> from numpy import array
>>> x=array([0,1,2,3,4,5,6])
>>> x[0:3]
>>> x[3]
>>> x[4:7]
```

En general si tenemos el vector $x = \{x_0, x_1, x_2, \ldots, x_{n-1}\}$, podemos expresar todas las componentes del vector como `x[0:i]`, `x[i]`, `x[i+1,n]`.

El siguiente ejemplo muestra todas las componentes de un vector `x`, y para cada componente del vector, muestra también las componentes anteriores y posteriores.

Ejemplo 2.

```
from numpy import array
x=array([0,1,2,3,4,5,6])
N=len(x)
for i in range(0,N):
    print(x[0:i],x[i],x[i+1:N])
```

En una matriz A, las filas de la matriz se pueden representar por `A[i]`, así `A[0]` es la primera fila de la matriz A.

Como cada una de las filas de una matriz es, en Python, un vector, podemos obtener partes de esos vectores como hemos visto en los ejemplos anteriores. Así `A[0,0:3]` es un vector con las tres primeras columnas de la primera fila de la matriz A.

Ejercicio 3. *Considere la matriz* $A = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 7 & 8 & 9 & 0 \\ 3 & 1 & 3 & 2 \\ 4 & 6 & 8 & 9 \end{pmatrix}$.

Obtenga como partes de A un vector con las componentes $(7, 8, 9)$ de la segunda fila. Idem $(1, 3, 2)$ de la tercera fila. Obtenga un vector con las dos últimas columnas de la última fila.

Ejemplo método de Jacobi.

Para ilustrar el método de Jacobi comenzamos mostrando un ejemplo. Supongamos el siguiente sistema lineal de ecuaciones

$$\begin{cases} 2x & + & y & + & z & = 7, \\ x & + & 3y & + & z & = 10, \\ x & + & 2y & + & 3z & = 14. \end{cases}$$

Despejamos la primera incógnita de la primera ecuación, la segunda incógnita de la segunda ecuación, y la tercera incógnita de la tercera ecuación, y obtenemos

$$\begin{cases} x & = & (7 - y - z)/2, \\ y & = & (10 - x - z)/3, \\ z & = & (14 - 2y - 3z)/3. \end{cases}$$

Con estas ecuaciones, y partiendo de una iteración inicial (x_0, y_0, z_0) podemos plantear el siguiente método iterativo

$$\begin{cases} x_{n+1} &= (7 - y_n - z_n)/2, \\ y_{n+1} &= (10 - x_n - z_n)/3, \\ z_{n+1} &= (14 - 2y_n - 3z_n)/3. \end{cases}$$

Implementación en Python del método de Jacobi.

Consideramos el sistema $A \cdot x = b$ con n ecuaciones $(0,\ldots, n-1)$ y n incógnitas $(x_0, \ldots, x_{n-1})$. En la notación empleada en Python, la ecuación i-ésima es

```
A[i,0] x[0] + A[i,1] x[1] + ...

  + A[i,i-1] x[i-1] + A[i,i] x[i] + A[i,i+1] x[i+1] + ...

  + A[i,n-1] x[n-1] = b[i]
```

En esta ecuación despejaremos `x[i]`, para lo cual expresaremos en Python los términos

```
A[i,0] x[0] + A[i,1] x[1] + ...  + A[i,i-1] x[i-1]

A[i,i+1] x[i+1] + ...  + A[i,n-1] x[n-1]
```

Ejercicio 4.

1. *Usando el operador* `:` *escriba los elementos* `x[0], x[1], ..., x[i-1]` *como una parte del vector* `x`.
2. *Usando el operador* `:` *escriba los elementos* `A[i,0], A[i,1], ..., A[i,i-1]` *como una parte de la matriz* `A`.
3. *Escriba la suma* `A[i,0] x[0] + A[i,1] x[1] + ... + A[i,i-1] x[i-1]` *mediante un producto escalar, usando la función* `dot` *y los apartados anteriores.*

Ejercicio 5. *Escriba la suma* `A[i,i+1] x[i+1] + ... + A[i,n-1] x[n-1]` *mediante un producto escalar.*

Ejemplo 6 (`www.ugr.es/local/anpalom`).
`Jacobi0.py`

Ejercicio 7. *En el programa* `Jacobi0.py`, *complete la linea 20,*

```
x[i]= (b[i]- dot( , )-dot( , ))/A[i,i]
```

para así poder implementar el método de Jacobi.

Sustitución regresiva.

La ecuación i-ésima de un sistema triangular superior es

```
A[i,i] x[i] + A[i,i+1] x[i+1] + ...  + A[i,n-1] x[n-1] = b[i]
```

Si despejamos `x[i]` tenemos una expresión como la que sigue.

```
x[i]=(b[i] - ..... )/A[i,i]
```

Ejercicio 8. *Exprese el valor de* `x[i]` *mediante un producto escalar.*

Ejemplo 9 (`www.ugr.es/local/anpalom`).
`SustRegresiva0.py`

Ejercicio 10. *En el programa* `SustRegresiva0.py`, *complete la linea 17*

```
x[i]=(b[i]-dot( , ))/A[i,i]
```

para así poder tener una implementación del método de sustitución regresiva.

Método de Gauss.

Suponemos que hemos hecho ceros bajo `A[0,0]`, `A[1,1]`,...,`A[k-1,k-1]`, y nos disponemos a hacer ceros bajo `A[k,k]`. Concretamente, hacemos cero en la posición `A[i,k]`.

Para esto, debemos restar a la fila `i` un múltiplo de la fila `k`.

(...)

Ejercicio 11. *Suponga que se efecta en la matriz* `A` *la operación de filas*

$$Fila\ i = Fila\ i \text{ - } \texttt{li} * Fila\ k$$

donde `li` *es un múltiplo que se quiere determinar.*

1. *Plantée (en función del* `li` *aún desconocido) el valor que se obtiene en* `A[i,k]` *al efectuar la operación de fila descrita.*
2. *Sabiendo que el resultado debe ser 0, halle la expresión que permite calcular* `li`*.*

Ejemplo 12 (`www.ugr.es/local/anpalom`)**.**
`Gauss0.py`

Ejercicio 13. *En el programa* `Gauss0.py`*,*

1. *complete las lineas 18, 20 y 21, definiendo* `li`*,* `A[i,:]` *y* `b[i]`*.*
2. *explique qué resultado debe obtenerse al calcular*

```
x=dot(linalg.inv(A),b)
print(dot(A0,x)-b )
```

y por qué ese resultado indica que el código permite obtener un sistema equivalente al dado inicialmente.