

Operaciones con matrices en python

Curso de Física Computacional

M. en C. Gustavo Contreras Mayén

Facultad de Ciencias - UNAM

23 de noviembre de 2017



1. Operaciones con matrices
2. Solución de sistemas algebraicos
3. Matrices en banda

1. Operaciones con matrices

1.1 Operaciones básicas

1.2 Determinantes

1.3 La traza

1.4 La matriz inversa

1.5 La matriz transpuesta

2. Solución de sistemas algebraicos

3. Matrices en banda

Operaciones con matrices

Para fines prácticos, los arreglos (**arrays**) de `numpy`, pueden ser utilizados como matrices. Revisaremos algunas operaciones con matrices de `numpy`.

El determinante

El determinante de una matriz se calcula mediante la función `det()`, contenida en el módulo `numpy.linalg`:

Código 1: Cálculo del determinante

```
1
2 from numpy import array, linalg
3
4 a = array([[ 3, -5, 8], [-1, 2, 3],
5            [-5, -6, 2]])
6 print(linalg.det(a))
```

La traza de una matriz

La traza de una matriz se calcula con la función **trace()**, contenida en el módulo `numpy`:

Código 2: Cálculo de la traza

```
1
2 from numpy import array, trace
3
4 a = array([[ 3, -5, 8], [-1, 2, 3],
5            [-5, -6, 2]])
6 print(trace(a))
```

La traza de una matriz - 2

La función `trace()`, permite el uso de un parámetro que calcula la traza con un desplazamiento (`offset`):

```
trace(matriz, offset)
```

Donde el `offset` es un valor entero:

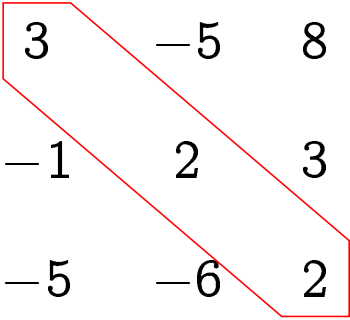
- `offset` = $k > 0$, número de la k -ésima codiagonal superior (por arriba de la diagonal principal).
- `offset` = $k < 0$, número de la k -ésima codiagonal inferior (por abajo de la diagonal principal).

La traza de una matriz - 2

Código 3: Cálculo de la traza con desplazamiento

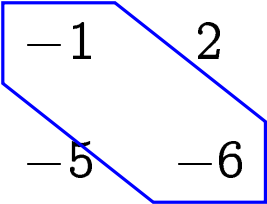
```
1 from numpy import array, trace
2
3 a = array([[ 3, -5, 8], [-1, 2, 3],
4            [-5, -6, 2]])
5 print(trace(a, -1))
6
7 print(trace(a, 1))
```


Desplazamiento de las matrices

$$\begin{pmatrix} 3 & -5 & 8 \\ -1 & 2 & 3 \\ -5 & -6 & 2 \end{pmatrix}$$


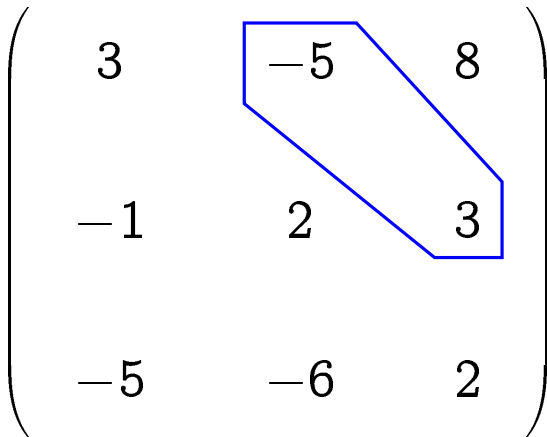
trace (a)

Desplazamiento de las matrices

$$\begin{pmatrix} 3 & -5 & 8 \\ -1 & 2 & 3 \\ -5 & -6 & 2 \end{pmatrix}$$


$\text{trace}(a, -1)$

Desplazamiento del las matrices



A 3x3 matrix is shown with a blue parallelogram highlighting a 2x2 submatrix. The matrix is:

$$\begin{pmatrix} 3 & -5 & 8 \\ -1 & 2 & 3 \\ -5 & -6 & 2 \end{pmatrix}$$

The highlighted submatrix consists of the elements -5 , 8 , 2 , and 3 .

`trace(a, 1)`

La matriz inversa

La matriz inversa se calcula con la función `inv()`, del módulo `numpy`:

Código 4: Cálculo de la matriz inversa

```
1
2 from numpy import array, linalg
3
4 a = array([[ 3, -5, 8], [-1, 2, 3],
5            [-5, -6, 2]])
6 print(linalg.inv(a))
```

La matriz transpuesta

La matriz transpuesta se obtiene con la función **transpose()**, contenida en el módulo `numpy`:

Código 5: Cálculo de la matriz transpuesta

```
1
2 from numpy import array, transpose
3
4 a = array([[ 3, -5, 8], [-1, 2, 3],
5            [-5, -6, 2]])
6 print(transpose(a))
```

Las matrices como objetos de numpy

La librería `numpy` permite manejar las matrices (**arrays**) como objetos.

Estos objetos de matriz tienen métodos integrados para el cálculo del determinante y de la matriz inversa.

Las matrices como objetos

El convertir el **array** expresamente a una matrix, podemos manipularla como un objeto de `numpy`

Código 6: Cálculo de la matriz transpuesta

```
1
2 from numpy import matrix
3
4 a = array([[ 3, -5, 8], [-1, 2, 3],
5            [-5, -6, 2]])
6 a = matrix(a)
7
8 print(a.T)
9
10 print(a.I)
```

1. Operaciones con matrices

2. Solución de sistemas algebraicos

2.1 Sistemas algebraicos con `python`

2.2 La función `scipy.linalg.solve`

2.3 Factorización LU

3. Matrices en banda

Solución de sistemas algebraicos

Un sistema de ecuaciones algebraico se puede expresar en términos de un arreglo matricial

$$\begin{array}{l} a_1 x + b_1 y + c_1 z = d_1 \\ a_2 x + b_2 y + c_2 z = d_2 \\ a_3 x + b_3 y + c_3 z = d_3 \end{array} \implies \begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix}$$

Solución con herramientas

Existen métodos y funciones con `python` para resolver sistemas de ecuaciones algebraicas.

Tomemos en cuenta que:

- 1 La solución de sistemas matriciales es computacionalmente una tarea intensa.

Solución con herramientas

Existen métodos y funciones con `python` para resolver sistemas de ecuaciones algebraicas.

Tomemos en cuenta que:

- 1 La solución de sistemas matriciales es computacionalmente una tarea intensa.
- 2 Veremos algunos métodos para resolver estas ecuaciones.

Solución con herramientas

Existen métodos y funciones con `python` para resolver sistemas de ecuaciones algebraicas.

Tomemos en cuenta que:

- 1 La solución de sistemas matriciales es computacionalmente una tarea intensa.
- 2 Veremos algunos métodos para resolver estas ecuaciones.
- 3 Estos métodos están contenidos en el módulo `scipy.linalg`.

Ejemplo

Revisemos algunos métodos usando el siguiente sistema:

$$4x - 5y + 8z = 4$$

$$2x - 8y + 7z = 0$$

$$-5x - 8y = 4$$

La función `scipy.linalg.solve`

Esta función utiliza la matriz de coeficientes y el vector del lado derecho de la igualdad como argumentos, devuelve un vector con la solución:

La función `scipy.linalg.solve`

Código 7: Uso de la función `solve`

```
1 from scipy import linalg as slin
2 from numpy import array
3
4 cm = array([[4, -5, 8], [2, -8, 7],
5             [-5, 8, 0]])
6 rhs = array([4, 0, -5])
7
8 solucion = slin.solve(cm, rhs)
9 print(solucion)
```

Consideración importante

La matriz de coeficientes debe de ser *no singular*, es decir, el determinante de esa matriz debe de ser distinto de cero:

Código 8: El código se interrumpe

```
1 cm2 = array([[4, -5, 8], [8, -10, 16], [-5, 8, 0]])
2 rhs2 = array([4, 8, -5])
3
4 print(slin.det(cm2))
5
6 solucion2 = slin.solve(cm2, rhs2)
7
8 print(solucion2)
```


Mensaje de error

```
solucion2 = slin.solve(cm2,rhs2)
```

```
File "/home/gustavo/anaconda3/lib/  
python3.5/site-packages/scipy/  
linalg/basic.py", line 219, in solve  
raise LinAlgError('Matrix is singular.
```

LinAlgError: Matrix is singular.

Sistemas algebraicos grandes

Los sistemas algebraicos de ecuaciones muy grandes implican una carga computacional elevada para resolverlo.

Existen varios métodos especializados para resolver eficientemente esos grandes sistemas de ecuaciones.

Factorización LU

Si el vector del lado derecho cambia, pero la matriz de coeficientes no cambia, entonces esta matriz puede descomponerse usando la factorización LU.

Esta descomposición LU se puede utilizar para resolver el sistema para cualquier vector del lado derecho.

Esto ahorra tiempo, porque el proceso de factorización LU es el que implica la mayor carga de recursos.

Por lo que al no tener que rehacerlo cada vez, se ahorra tiempo y recursos computacionales.

Factorización LU

Para utilizar la factorización LU:

- 1 Primero, usemos la función `slin.lu_factor()` en la matriz de coeficientes, y asignamos el resultado en una nueva variable.

Factorización LU

Para utilizar la factorización LU:

- 1 Primero, usemos la función `slin.lu_factor()` en la matriz de coeficientes, y asignamos el resultado en una nueva variable.
- 2 Segundo, usamos la función `slin.lu_solve()` con la factorización obtenida y los el vector de consantes `rhs` como argumentos.

Código 9: Uso de scipy.linalg.lu

```
1 from scipy import linalg as slin
2 from numpy import array
3
4 cm = array([[ 4, -5, 8], [ 2, -8, 7]
5             ], [-5, 8, 0])
6
7 rhs = array([ 4, 0, -5])
8
9 #nota: las funciones llevan guion
10      bajo
11 lu = slin.lu-factor(cm)
12
13 solucion = slin.lu-solve(lu,rhs)
14
15 print(solucion)
```

Ventaja con la factorización LU

Una vez que se logra la factorización LU, podemos usar cualquier vector del lado derecho, sin necesidad de que se calcule nuevamente la factorización.

Código 10: La misma factorización cambiando el vector b

```
1 b2 =array([4, -3, 9])
2
3 #nota: lleva guion bajo
4 solucion-b2 = slin.lu-solve(lu, b2)
5
6 print('\nSolucion con el vector b2')
7 print(solucion-b2)
8
9 b3 = array([-2, -3, -12])
10
11 #nota: lleva guion bajo
12 solucion-b3 = slin.lu-solve(lu, b3)
13
14 print('\nSolucion con el vector b3')
15 print(solucion-b32)
```

1. Operaciones con matrices

2. Solución de sistemas algebraicos

3. Matrices en banda

3.1 Resolviendo matrices en banda

3.2 Representación de la matriz en banda

3.3 Arreglo de las codiagonales

3.4 Función `scipy.linalg.solve_banded()`

Matrices en banda

Muchas matrices grandes tanto en ciencias como en ingeniería son de tal naturaleza, que los elementos de la misma están agrupados (*en banda*).

Lo que significa que sus valores no nulos se encuentran a lo largo de las diagonales de la matriz.

Se han desarrollado métodos para resolver eficientemente este tipo de matrices.

Las matrices en banda tampoco requieren tanta memoria para almacenar valores intermedios, ya que muchos de las entradas son cero.

Considera la siguiente matriz en banda

$$\begin{bmatrix} 1 & -5 & 3 & 0 & 0 & 0 & 0 & 0 \\ 3 & 2 & -3 & 5 & 0 & 0 & 0 & 0 \\ 0 & -2 & 1 & 5 & 9 & 0 & 0 & 0 \\ 0 & 0 & 9 & -1 & 5 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -3 & -2 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 1 & -6 \\ 0 & 0 & 0 & 0 & 0 & -3 & 2 & 7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 9 & 1 \end{bmatrix}$$

Nueva representación de la matriz

La matriz en banda se puede representar por una matriz no cuadrada del tipo

Nueva representación de la matriz

La matriz en banda se puede representar por una matriz no cuadrada del tipo

$$\begin{bmatrix} 0 & 0 & 3 & 5 & 9 & 4 & -2 & -6 \\ 0 & -5 & -3 & 5 & 5 & -3 & 1 & 7 \\ 1 & 2 & 1 & -1 & 2 & 0 & 2 & 1 \\ 3 & -2 & 9 & 0 & 2 & -3 & 9 & 0 \end{bmatrix}$$

Diagonales y codiagonales

The image shows an 8x8 matrix with three diagonals highlighted in different colors:

- Blue diagonal (Main Diagonal):** Elements are 1, 3, 0, 0, 0, 0, 0, 9.
- Red diagonal (First Super-Diagonal):** Elements are -5, 2, -2, 0, 0, 0, -3, 0.
- Green diagonal (Second Super-Diagonal):** Elements are 3, -3, 1, -1, 0, 0, 1, 0.

The full matrix is:

$$\begin{pmatrix} 1 & -5 & 3 & 0 & 0 & 0 & 0 & 0 \\ 3 & 2 & -3 & 5 & 0 & 0 & 0 & 0 \\ 0 & -2 & 1 & 5 & 9 & 0 & 0 & 0 \\ 0 & 0 & 9 & -1 & 5 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -3 & -2 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 1 & -6 \\ 0 & 0 & 0 & 0 & 0 & -3 & 2 & 7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 9 & 1 \end{pmatrix}$$

Arreglo de las codiagonales

$$\begin{pmatrix} 0 & 0 & 3 & 5 & 9 & 4 & -2 & -6 \\ 0 & -5 & -3 & 5 & 5 & -3 & 1 & 7 \\ 1 & 2 & 1 & -1 & 2 & 0 & 2 & 1 \\ 3 & -2 & 9 & 0 & 2 & -3 & 9 & 0 \end{pmatrix}$$

- Las codiagonales superiores tienen ceros a la izquierda.
- Las codiagonales inferiores tienen ceros al final.

Resolviendo matrices en banda

Para resolver un sistema de ecuaciones con una matriz en banda, usaremos la función `scipy.linalg.solve_banded()`.

Resolviendo matrices en banda

Para resolver un sistema de ecuaciones con una matriz en banda, usaremos la función

`scipy.linalg.solve_banded()`.

La sintaxis para esta función es la siguiente

`slin.solve_banded((l,u), cm, rhs)`

Resolviendo matrices en banda

```
slin.solve_banded((l,u), cm, rhs)
```

Donde:

- `(l, u)` es una tupla, donde l es el número de codiagonales inferiores no nulas, y u es el número de codiagonales superiores no nulas.

Resolviendo matrices en banda

```
slin.solve_banded((l,u), cm, rhs)
```

Donde:

- `(l, u)` es una tupla, donde l es el número de codiagonales inferiores no nulas, y u es el número de codiagonales superiores no nulas.
- `cm` es la matriz en banda de coeficientes.

Resolviendo matrices en banda

```
slin.solve_banded((l,u), cm, rhs)
```

Donde:

- `(l, u)` es una tupla, donde l es el número de codiagonales inferiores no nulas, y u es el número de codiagonales superiores no nulas.
- `cm` es la matriz en banda de coeficientes.
- `rhs` es el vector de constantes del lado derecho.

Ejercicio 1 - Matriz en banda

Resuelve el siguiente sistema algebraico

$$\begin{bmatrix} 1 & -5 & 3 & 0 & 0 & 0 & 0 & 0 \\ 3 & 2 & -3 & 5 & 0 & 0 & 0 & 0 \\ 0 & -2 & 1 & 5 & 9 & 0 & 0 & 0 \\ 0 & 0 & 9 & -1 & 5 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -3 & -2 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 1 & -6 \\ 0 & 0 & 0 & 0 & 0 & -3 & 2 & 7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 9 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \\ -3 \\ 2 \\ 0 \\ 7 \\ 8 \\ 1 \end{bmatrix}$$

Solución al sistema

$$a = 200.639937$$

$$b = -25.491352$$

$$c = -107.698899$$

$$d = -174.206761$$

$$e = 102.750000$$

$$f = 70.833333$$

$$g = -3.500000$$

$$h = 32.500000$$

Ejercicio 2 - Matriz tridiagonal simétrica

Usando la función `solve_banded()`, resuelve el sistema $Ax = b$ (que ya habíamos trabajado), donde

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix} \quad b = \begin{bmatrix} 5 \\ -5 \\ 4 \\ -5 \\ 5 \end{bmatrix}$$

Solución al sistema tridiagonal simétrico

Cuando usamos las funciones **LUdescomp3** y **LUsoluc3** para este sistema tridiagonal simétrico, la solución obtenida es:

Solución al sistema tridiagonal simétrico

Cuando usamos las funciones **LUdescomp3** y **LUsoluc3** para este sistema tridiagonal simétrico, la solución obtenida es:

$$x = [2. \quad -1. \quad 1. \quad -1. \quad 2.]$$

Que debe de ser la misma solución con la función **solve_banded()**