

Juan Francisco Pinto

Taller LU

```
In [3]: load_ext autoreload
In [4]: #----- logging -----#
import logging
from sys import stdout
from time import datetime
logging.basicConfig()
logger=logging.INFO,
format='%(asctime)s|%(levelname)s|%(message)s',
stream=stdout,
datefmt='%m-%d %H:%M:%S',
level=logging.INFO)
logging.info(datetime.now())
[0:-1] 2013[29][INFO] 2013-01-13 20:37:29.940782
```

```
In [5]: !autoreload 2
from arc import eliminacion_gaussiana
[0:-1] 2013[29][INFO] 2013-01-13 20:37:29.994378
```

```
In [6]: A = [[1, 3, 4], [2, 1, 3], [4, 2, 1]]
```

```
In [7]: import numpy as np
```

```
In [8]: #----- gauss_jordan(A: np.ndarray) --> np.ndarray:
*****Realiza la eliminación de Gauss-Jordan
```

```
    ## Parameters
    ``A``: matriz del sistema de ecuaciones lineales. Debe ser de tamaño n-by-(n), donde n es el número de incógnitas.
    ## Return
    ``A``: matriz reducida por filas.
    ***

    if not isinstance(A, np.ndarray):
        logging.debug("convirtiendo A a numpy array.")
        A=np.array(A, dtype=float)
    n=A.shape[0]

    for i in range(0, n): # loop por columnas
        # encontrar pivote
        p=None # default, first element
        for pi in range(i, n):
            if A[pi, i] == 0:
                continue
            if abs(A[pi, i]) < abs(A[p, i]):
                p=pi
        if p is None:
            raise ValueError("No existe solución única.")
        if p != i:
            # swap rows
            logging.debug("Intercambiando filas (%i) y (%i)" % (p,i))
            _aux=A[i,:].copy()
            A[i,:]=A[p,:].copy()
            A[p,:]=_aux

        # --- Eliminación: loop por fila
        for j in range(n): # Gauss-Jordan
            if j == i:
                continue # skip pivot row
            m = A[j, i] / A[i, i]
            A[j, i] = A[j, i] - m * A[i, i]
            # dividir para la diagonal
            A[i, i] = A[i, i] / A[i, i]
        logging.info("%s(A)" % A)

    if A[0, 1, ..., n-1] == 0:
        raise ValueError("No existe solución única.")

    print("%s(A)" % A)
    #----- Sustitución hacia atrás
    #----- Implementación de la inversa
    if solution[n-1] == A[0, 1, ..., n-1] / A[0, 1, ..., n-1]:
        return A
    else:
        for i in range(n-2, -1, -1):
            suma = 0
            for j in range(i+1, n):
                suma += A[i, j] * solution[j]
            solution[i] = (A[i, n] - suma) / A[i, i]
        return A
```

```
In [9]: from pprint import pprint
pprint(A)
gauss_jordan(A)
```

```
[0:-1] 2013[30][INFO] 2013-01-13 20:37:30
[[1, 3, 4], [2, 1, 3], [4, 2, 1]]
[[1, 3, 4], [0, -5, -1], [0, 0, 1]]
[[0, -5, -1], [0, 0, 1], [0, 0, 1]]
[0:-1] 2013[30][INFO]
[[1, 0, 1, 1], [0, 0, 1, 1], [0, 0, 1, 1], [0, 0, 0, 1]]
[[1, 0, 1, 1], [0, 0, 1, 1], [0, 0, 1, 1], [0, 0, 0, 1]]
[[0, 0, 1, 1], [0, 0, 1, 1], [0, 0, 1, 1], [0, 0, 0, 1]]
[[0, 0, 0, 1], [0, 0, 0, 1], [0, 0, 0, 1], [0, 0, 0, 1]]
```

```
Out[8]: array([[1., 0., 0., 0.],
       [-0., 1., 0., 0.],
       [-0., -0., 1., 0.]])
```

```
In [9]: # Poner matriz identidad a la derecha
n = len(A)
A_aug = np.hstack((A, np.eye(n)))
A_aug
```

```
Out[9]: array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.]])
```

```
In [10]: np.vstack((A, np.eye(n)))
```

```
Out[10]: array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

```
In [11]: _m_inv = gauss_jordan(A_aug)
```

```
[0:-1] 2013[30][INFO]
[[1, 0, 0, 0, 1], [0, 1, 0, 0, 1], [0, 0, 1, 0, 1], [0, 0, 0, 1, 1]]
[[1, 0, 0, 0, 1], [0, 0, 1, 0, 1], [0, 0, 0, 1, 1], [0, 0, 0, 0, 1]]
[[0, 0, 1, 0, 1], [0, 0, 0, 1, 1], [0, 0, 0, 0, 1], [0, 0, 0, 0, 1]]
[[0, 0, 0, 1, 1], [0, 0, 0, 0, 1], [0, 0, 0, 0, 1], [0, 0, 0, 0, 1]]
```

```
Out[11]: array([[-0.2, 0.2, 0.2, 0.2],
       [0.4, -0.6, 0.2, 0.2],
       [-0.4, 0.4, -0.2, 0.2]])
```

```
In [12]: _m_inv*(1, 0, 0, 0, 1)
```

```
Out[12]: array([-0.2, 0.2, 0.2,
       [0.4, -0.6, 0.2, 0.2],
       [-0.4, 0.4, -0.2, 0.2]])
```

Comprobando respuestas

```
In [13]: np.linalg.inv(np.array(A))
```

```
Out[13]: array([-0.2, 0.2, 0.2,
       [0.4, -0.6, 0.2, 0.2],
       [-0.4, 0.4, -0.2, 0.2]])
```

Calculo de la matriz inversa del resto de literales

```
In [13]: A = [[1,3,4],[2,1,3],[4,2,1]]
```

```
# Poner matriz identidad a la derecha
n = len(A)
A_aug = np.hstack((A, np.eye(n)))
A_aug
```

```
# implementamos gauss jordan
_m_inv = gauss_jordan(A_aug)
```

```
# mostramos la inversa
_m_inv*(1, 0, 0, 0, 1)
```

```
[0:-1] 2013[30][INFO]
[[1, 3, 4, 1], [2, 1, 3, 1], [4, 2, 1, 1]]
[[1, 3, 4, 1], [0, -5, -1, 1], [0, 0, 1, 1]]
[[0, -5, -1, 1], [0, 0, 1, 1], [0, 0, 1, 1]]
[[0, 0, 1, 1], [0, 0, 1, 1], [0, 0, 1, 1], [0, 0, 0, 1]]
```

```
Out[13]: array([[-0.2, 0.2, 0.2, 0.2],
       [0.4, -0.6, 0.2, 0.2],
       [-0.4, 0.4, -0.2, 0.2]])
```

```
In [14]: A = [[1,2,3],[0,1,4],[5,6,0]]
```

```
# Poner matriz identidad a la derecha
n = len(A)
A_aug = np.hstack((A, np.eye(n)))
A_aug
```

```
# implementamos gauss jordan
_m_inv = gauss_jordan(A_aug)
```

```
# mostramos la inversa
_m_inv*(1, 0, 0, 0, 1)
```

```
[0:-1] 2013[47][52][INFO]
[[1, 2, 3, 1, 0, 0, 0, 1], [0, 1, 4, 0, 1, 0, 0, 1], [0, 0, 0, 5, 0, 1, 0, 1]]
[[1, 2, 3, 1, 0, 0, 0, 1], [0, 0, 0, 5, 0, 1, 0, 1], [0, 0, 0, 0, 1, 0, 1]]
[[0, 0, 0, 5, 0, 1, 0, 1], [0, 0, 0, 0, 1, 0, 1], [0, 0, 0, 0, 0, 1]]
```

```
Out[14]: array([[-0.2, 0.2, 0.2, 0.2],
       [0.4, -0.6, 0.2, 0.2],
       [-0.4, 0.4, -0.2, 0.2]])
```

```
In [15]: A = [[1,2,3],[0,1,4],[5,6,0]]
```

```
# Poner matriz identidad a la derecha
n = len(A)
A_aug = np.hstack((A, np.eye(n)))
A_aug
```

```
# implementamos gauss jordan
_m_inv = gauss_jordan(A_aug)
```

```
# mostramos la inversa
_m_inv*(1, 0, 0, 0, 1)
```

```
[0:-1] 2013[47][52][INFO]
[[1, 2, 3, 1, 0, 0, 0, 1], [0, 1, 4, 0, 1, 0, 0, 1], [0, 0, 0, 5, 0, 1, 0, 1]]
[[1, 2, 3, 1, 0, 0, 0, 1], [0, 0, 0, 5, 0, 1, 0, 1], [0, 0, 0, 0, 1, 0, 1]]
[[0, 0, 0, 5, 0, 1, 0, 1], [0, 0, 0, 0, 1, 0, 1], [0, 0, 0, 0, 0, 1]]
```

```
Out[15]: array([[-0.2, 0.2, 0.2, 0.2],
       [0.4, -0.6, 0.2, 0.2],
       [-0.4, 0.4, -0.2, 0.2]])
```

```
In [16]: A = [[2,4,6,1],[4,7,5,-6],[2,5,18,10],[6,12,38,16]]
```

```
# Poner matriz identidad a la derecha
n = len(A)
A_aug = np.hstack((A, np.eye(n)))
A_aug
```

```
# implementamos gauss jordan
_m_inv = gauss_jordan(A_aug)
```

```
# mostramos la inversa
_m_inv*(1, 0, 0, 0, 1)
```

```
[0:-1] 2013[50][INFO]
[[2, 4, 6, 1, 0, 0, 0, 1], [4, 7, 5, -6, 0, 0, 0, 1], [2, 5, 18, 10, 0, 0, 0, 1], [6, 12, 38, 16, 0, 0, 0, 1]]
[[2, 4, 6, 1, 0, 0, 0, 1], [0, 0, 0, 1, 0, 0, 0, 1], [0, 0, 0, 0, 1, 0, 0, 1], [0, 0, 0, 0, 0, 1, 0, 1]]
[[0, 0, 0, 1, 0, 0, 0, 1], [0, 0, 0, 0, 1, 0, 0, 1], [0, 0, 0, 0, 0, 1, 0, 1], [0, 0, 0, 0, 0, 0, 1]]
```

```
Out[16]: array([[-0.2, 0.2, 0.2, 0.2],
       [0.4, -0.6, 0.2, 0.2],
       [-0.4, 0.4, -0.2, 0.2],
       [0.8, -0.8, 0.4, -0.4]])
```

```
In [17]: A = [[2,4,6,1],[4,7,5,-6],[2,5,18,10],[6,12,38,16]]
```

```
# Poner matriz identidad a la derecha
n = len(A)
A_aug = np.hstack((A, np.eye(n)))
A_aug
```

```
# implementamos gauss jordan
_m_inv = gauss_jordan(A_aug)
```

```
# mostramos la inversa
_m_inv*(1, 0, 0, 0, 1)
```

```
[0:-1] 2013[50][INFO]
[[2, 4, 6, 1, 0, 0, 0, 1], [4, 7, 5, -6, 0, 0, 0, 1], [2, 5, 18, 10, 0, 0, 0, 1], [6, 12, 38, 16, 0, 0, 0, 1]]
[[2, 4, 6, 1, 0, 0, 0, 1], [0, 0, 0, 1, 0, 0, 0, 1], [0, 0, 0, 0, 1, 0, 0, 1], [0, 0, 0, 0, 0, 1, 0, 1]]
[[0, 0, 0, 1, 0, 0, 0, 1], [0, 0, 0, 0, 1, 0, 0, 1], [0, 0, 0, 0, 0, 1, 0, 1], [0, 0, 0, 0, 0, 0, 1]]
```

```
Out[17]: array([[-0.2, 0.2, 0.2, 0.2],
       [0.4, -0.6, 0.2, 0.2],
       [-0.4, 0.4, -0.2, 0.2],
       [0.8, -0.8, 0.4, -0.4]])
```

```
In [18]: import numpy as np
def lu_factor(A):
    """ Factoriza con pivoteo parcial:
        P = A * L
        Recorta P, L, U
    """
    P = np.array(A, dtype=float, copy=True)
    n, m = P.shape
    if n != m:
        raise ValueError("A debe ser cuadrada (n,n).")
    P = np.eye(n)
    L = np.zeros((n, n))
    U = np.zeros((n, n))

    for k in range(n):
        p = k
        for pi in range(k, n):
            if abs(P[pi, k]) > abs(P[p, k]):
                p=pi
        if p != k:
            # swap rows
            logging.debug("Intercambiando filas (%i) y (%i)" % (p,k))
            _aux=P[:,p].copy()
            P[:,p]=P[:,k].copy()
            P[:,k]=_aux.copy()

        # --- Eliminación: loop por fila
        for j in range(n): # Gauss-Jordan
            if j == k:
                continue # skip pivot row
            m = P[j, k] / P[k, k]
            P[j, k] = P[j, k] - m * P[k, k]
            # dividir para la diagonal
            P[k, k] = P[k, k] / P[k, k]
        logging.info("%s(A)" % P)

    if P[0, 1, ..., n-1] == 0:
        raise ValueError("No existe solución única.")

    print("%s(A)" % P)
    #----- Sustitución hacia atrás
    #----- Implementación de la inversa
    if solution[n-1] == P[0, 1, ..., n-1] / P[0, 1, ..., n-1]:
        return P
    else:
        for i in range(n-2, -1, -1):
            suma = 0
            for j in range(i+1, n):
                suma += P[i, j] * solution[j]
            solution[i] = (P[i, n] - suma) / P[i, i]
        return P
```

```
In [19]: A = [[1, 3, 4], [2, 1, 3], [4, 2, 1]]
```

```
# Poner matriz identidad a la derecha
n = len(A)
A_aug = np.hstack((A, np.eye(n)))
A_aug
```

```
# implementamos gauss jordan
_m_inv = gauss_jordan(A_aug)
```

```
# mostramos la inversa
_m_inv*(1, 0, 0, 0, 1)
```

```
[0:-1] 2013[50][INFO]
[[1, 3, 4, 1], [2, 1, 3, 1], [4, 2, 1, 1]]
[[1, 3, 4, 1], [0, -5, -1, 1], [0, 0, 1, 1]]
[[0, -5, -1, 1], [0, 0, 1, 1], [0, 0, 1, 1]]
[[0, 0, 1, 1], [0, 0, 1, 1], [0, 0, 1, 1], [0, 0, 0, 1]]
```

```
Out[19]: array([[-0.2, 0.2, 0.2, 0.2],
       [0.4, -0.6, 0.2, 0.2],
       [-0.4, 0.4, -0.2, 0.2]])
```

```
In [20]: A = [[1,2,3],[0,1,4],[5,6,0]]
```

```
# Poner matriz identidad a la derecha
n = len(A)
A_aug = np.hstack((A, np.eye(n)))
A_aug
```

```
# implementamos gauss jordan
_m_inv = gauss_jordan(A_aug)
```

```
# mostramos la inversa
_m_inv*(1, 0, 0, 0, 1)
```

```
[0:-1] 2013[50][INFO]
[[1, 2, 3, 1, 0, 0, 0, 1], [0, 1, 4, 0, 1, 0, 0, 1], [0, 0, 0, 5, 0, 1, 0, 1]]
[[1, 2, 3, 1, 0, 0, 0, 1], [0, 0, 0, 5, 0, 1, 0, 1], [0, 0, 0, 0, 1, 0, 1]]
[[0, 0, 0, 5, 0, 1, 0, 1], [0, 0, 0, 0, 1, 0, 1], [0, 0, 0, 0, 0, 1]]
```

```
Out[20]: array([[-0.2, 0.2, 0.2, 0.2],
       [0.4, -0.6, 0.2, 0.2],
       [-0.4, 0.4, -0.2, 0.2]])
```

Calcular la descomposición LU para estas matrices y encuentre la solución para estos vectores de valores independientes b

```
In [20]: import numpy as np
def lu_factor(A):
    """ Factoriza con pivoteo parcial:
        P = A * L
        Recorta P, L, U
    """
    P = np.array(A, dtype=float, copy=True)
    n, m = P.shape
    if n != m:
        raise ValueError("A debe ser cuadrada (n,n).")
    P = np.eye(n)
    L = np.zeros((n, n))
    U = np.zeros((n, n))

    for k in range(n):
        p = k
        for pi in range(k, n):
            if abs(P[pi, k]) > abs(P[p, k]):
                p=pi
        if p != k:
            # swap rows
            logging.debug("Intercambiando filas (%i) y (%i)" % (p,k))
            _aux=P[:,p].copy()
            P[:,p]=P[:,k].copy()
            P[:,k]=_aux.copy()

        # --- Eliminación: loop por fila
        for j in range(n): # Gauss-Jordan
            if j == k:
                continue # skip pivot row
            m = P[j, k] / P[k, k]
            P[j, k] = P[j, k] - m * P[k, k]
            # dividir para la diagonal
            P[k, k] = P[k, k] / P[k, k]
        logging.info("%s(A)" % P)

    if P[0, 1, ..., n-1] == 0:
        raise ValueError("No existe solución única.")

    print("%s(A)" % P)
    #----- Sustitución hacia atrás
    #----- Implementación de la inversa
    if solution[n-1] == P[0, 1, ..., n-1] / P[0, 1, ..., n-1]:
        return P
    else:
        for i in range(n-2, -1, -1):
            suma = 0
            for j in range(i+1, n):
                suma += P[i, j] * solution[j]
            solution[i] = (P[i, n] - suma) / P[i, i]
        return P
```

```
In [21]: A = [[1, 3, 4], [2, 1, 3], [4, 2, 1]]
```

```
# Poner matriz identidad a la derecha
n = len(A)
A_aug = np.hstack((A, np.eye(n)))
A_aug
```

```
# implementamos gauss jordan
_m_inv = gauss_jordan(A_aug)
```

```
# mostramos la inversa
_m_inv*(1, 0, 0, 0, 1)
```

```
[0:-1] 2013[50][INFO]
[[1, 3, 4, 1], [2, 1, 3, 1], [4, 2, 1, 1]]
[[1, 3, 4, 1], [0, -5, -1, 1], [0, 0, 1, 1]]
[[0, -5, -1, 1], [0, 0, 1, 1], [0, 0, 1, 1]]
[[0, 0, 1, 1], [0, 0, 1, 1], [0, 0, 1, 1], [0, 0, 0, 1]]
```

```
Out[21]: array([[-0.2, 0.2, 0.2, 0.2],
       [0.4, -0.6, 0.2, 0.2],
       [-0.4, 0.4, -0.2, 0.2]])
```

```
In [22]: A = [[1,2,3],[0,1,4],[5,6,0]]
```

```
# Poner matriz identidad a la derecha
n = len(A)
A_aug = np.hstack((A, np.eye(n)))
A_aug
```

```
# implementamos gauss jordan
_m_inv = gauss_jordan(A_aug)
```

```
# mostramos la inversa
_m_inv*(1, 0, 0, 0, 1)
```

```
[0:-1] 2013[50][INFO]
[[1, 2, 3, 1, 0, 0, 0, 1], [0, 1, 4, 0, 1, 0, 0, 1], [0, 0, 0, 5, 0, 1, 0, 1]]
[[1, 2, 3, 1, 0, 0, 0, 1], [0, 0, 0, 5, 0, 1, 0, 1], [0, 0, 0, 0, 1, 0, 1]]
[[0, 0, 0, 5, 0, 1, 0, 1], [0, 0, 0, 0, 1, 0, 1], [0, 0, 0, 0, 0, 1]]
```

```
Out[22]: array([[-0.2, 0.2, 0.2, 0.2],
       [0.4, -0.6, 0.2, 0.2],
       [-0.4, 0.4, -0.2, 0.2]])
```

```
In [23]: A = [[2,4,6,1],[4,7,5,-6],[2,5,18,10],[6,12,38,16]]
```

```
# Poner matriz identidad a la derecha
n = len(A)
A_aug = np.hstack((A, np.eye(n)))
A_aug
```

```
# implementamos gauss jordan
_m_inv = gauss_jordan(A_aug)
```

```
# mostramos la inversa
_m_inv*(1, 0, 0, 0, 1)
```

```
[0:-1] 2013[47][52][INFO]
[[2, 4, 6, 1, 0, 0, 0, 1], [4, 7, 5, -6, 0, 0, 0, 1], [2, 5, 18, 10, 0, 0, 0, 1], [6, 12, 38, 16, 0, 0, 0, 1]]
[[2, 4, 6, 1, 0, 0, 0, 1], [0, 0, 0, 1, 0, 0, 0, 1], [0, 0, 0, 0, 1, 0, 0, 1], [0, 0, 0, 0, 0, 1, 0, 1]]
[[0, 0, 0, 1, 0, 0, 0, 1], [0, 0, 0, 0, 1, 0, 0, 1], [0, 0, 0, 0, 0, 1, 0, 1], [0, 0, 0, 0, 0, 0, 1]]
```

```
Out[23]: array([[-0.2, 0.2, 0.2, 0.2],
       [0.4, -0.6, 0.2, 0.2],
       [-0.4, 0.4, -0.2, 0.2],
       [0.8, -0.8, 0.4, -0.4]])
```

```
In [24]: A = [[2,4,6,1],[4,7,5,-6],[2,5,18,10],[6,12,38,16]]
```

```
# Poner matriz identidad a la derecha
n = len(A)
A_aug = np.hstack((A, np.eye(n)))
A_aug
```

```
# implementamos gauss jordan
_m_inv = gauss_jordan(A_aug)
```

```
# mostramos la inversa
_m_inv*(1, 0, 0, 0, 1)
```

```
[0:-1] 2013[47][52][INFO]
[[2, 4, 6, 1, 0, 0, 0, 1], [4, 7, 5, -6, 0, 0, 0, 1], [2, 5, 18, 10
```

{(-7,1)
(-8,1)
(-1,1)
(-9,1)}