 Transform a matrix to reduced row echelon form

**import** numpy **as** np

'''Function to transform a matrix to reduced row echelon form'''

**def** rref(A):

tol = 1***e***-14 ## Orignally given 1e-16, but seems to be wrong

#A = B.copy()

rows, cols = A.shape­

r = 0

pivots\_pos = []

row\_exchanges = np.arange(rows)

**for** c **in** range(cols):

## Find the pivot row:

pivot = np.argmax (np.abs (A[r:rows,c])) + r

m = np.abs(A[pivot, c])

**if** m <= tol:

## Skip column c, making sure the approximately zero terms are

## actually zero.

A[r:rows, c] = np.zeros(rows-r)

**else**:

## keep track of bound variables

pivots\_pos.append((r,c))

**if** pivot != r:

## Swap current row and pivot row

A[[pivot, r], c:cols] = A[[r, pivot], c:cols]

row\_exchanges[[pivot,r]] = row\_exchanges[[r,pivot]]

## Normalize pivot row

A[r, c:cols] = A[r, c:cols] / A[r, c];

## Eliminate the current column

v = A[r, c:cols]

## Above (before row r):

**if** r > 0:

ridx\_above = np.arange(r)

A[ridx\_above, c:cols] = A[ridx\_above, c:cols] - np.outer(v, A[ridx\_above, c]).T

## Below (after row r):

**if** r < rows-1:

ridx\_below = np.arange(r+1,rows)

A[ridx\_below, c:cols] = A[ridx\_below, c:cols] - np.outer(v, A[ridx\_below, c]).T

r += 1

## Check if done

**if** r == rows:

**break**;

**return** A