∗Problem 9. Let $A = \begin{bmatrix} 1 & 0 \\ \times & \times \\ 0 & 0 \end{bmatrix}$. Determine $N(A)$ and $R(A)$. Are they equal? Is this true in general? If this is true in general, prove it; if not, provide a counterexample.

∗Problem 10.

Suppose we want to approximate the derivative of a function $f(x)$ that is sampled on the grid $x_1, \ldots, x_n$ where $x_{i+1} = x_i + \delta$. Then we might do so using the expression

$$\frac{\partial f(x)}{\partial x}|_{x=x_i} \approx \frac{f(x_{i+1}) - f(x_i)}{\delta}$$

When $\delta = 1$, this yields

$$f'(x_i) := \frac{\partial f(x)}{\partial x}|_{x=x_i} \approx f(x_{i+1}) - f(x_i).$$

We can express this relation via a matrix-vector product where

$$\begin{bmatrix} f'(x_1) \\ f'(x_2) \\ \vdots \\ f'(x_n) \end{bmatrix} = D_{nn} \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{bmatrix},$$

where D_{nn} is the so-called first difference matrix between and is given by

$$D_{nn} = \begin{bmatrix} -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & \ddots & \\ & & & -1 & 1 \\ 1 & & & & -1 \end{bmatrix}. \tag{1}$$

Note that setting

$$D_{nn}(n, 1) = 1,$$

corresponds to the approximation

$$f'(x_n) \approx f(x_i) - f(x_n).$$

This is referred to as a periodic differentiation rule which is associated with a periodic boundary conditon because we are assuming that the domain wraps around. Notice that D_{nn} is a circulant matrix and so its eigenvectors can be computed in closed form!

The goal of this exercise is to derive the analog of D_{nn} for 2-D differentiation. To that end, let $f(x, y)$ be a function of two variables. We can approximate its partial derivatives using finite differences as follows:

$$\begin{aligned} \frac{\partial f(x,y)}{\partial x} &\approx \frac{f(x+1,y) - f(x,y)}{(x+1) - x} \\ &= f(x+1,y) - f(x,y) \end{aligned} \tag{2}$$

and

$$\begin{aligned} \frac{\partial f(x,y)}{\partial y} &\approx \frac{f(x,y+1) - f(x,y)}{(y+1) - y} \\ &= f(x,y+1) - f(x,y), \end{aligned} \tag{3}$$

To simplify notation, let us define the $m \times n$ `MATLAB` matrices `FXY`, `DFDX`, and `DFDY` such that

$$\begin{aligned} \texttt{FXY(i,j)} &= f(i,j) \\ \texttt{DFDX(i,j)} &= \frac{\partial f(i,j)}{\partial x} \\ \texttt{DFDY(i,j)} &= \frac{\partial f(i,j)}{\partial y} \end{aligned} \tag{4}$$

and the $mn \times 1$ MATLAB vectors `fxy`, `dfdx`, and `dfdy` such that

$$\begin{aligned} \texttt{fxy} &= \texttt{FXY(:)} \\ \texttt{dfdx} &= \texttt{DFDX(:)} \\ \texttt{dfdy} &= \texttt{DFDY(:)} \end{aligned} \tag{5}$$

With this notation, we can succinctly express equations (2) and (3) as

$$\begin{bmatrix} \texttt{dfdx} \\ \texttt{dfdy} \end{bmatrix} = A * \texttt{fxy} \tag{6}$$

For some $2mn \times mn$ matrix A. We can express A in terms of the first differences matrix.
Find an expression for A in terms of the first difference matrix using only the first difference matrix, the identity matrix, and the Kronecker product.
Hint: try it for m = n = 3 first.
Once you have determined what A is, write a MATLAB function called `hw2p10.m` or a Python function called `hw2p10.py` that takes as input the dimensions m and n of `FXY` and returns the appropriate A matrix where A is a **sparse** matrix.
Your MATLAB function should have signature:

```
function A = hw2p10(m,n)
```

and your Python function should have signature:

```
def hw2p10(m,n):
    # Your code here
    return A
```

The matrix A can be gigantic. For example, suppose $m = 550$ and $n = 430$, then A is a $709,500 \times 236,500$ matrix that would require 1.2 TB of RAM if stored as a full double precision matrix! However, A has only $4mn$ non-zero entries, so it is very *sparse*.
It is not uncommon to have matrices with a large number of zero-valued elements and, because the MATLAB and Python store zeros in the same way it stores any other numeric value, these elements can use memory space unnecessarily and can sometimes require extra computing time.
Sparse matrices provide a way to store data that has a large percentage of zero elements more efficiently. While full matrices internally store every element in memory regardless of value, sparse matrices store only the nonzero elements and their row indices. Using sparse matrices can significantly reduce the amount of memory required for data storage.
See:

`http://www.mathworks.com/help/matlab/ref/spdiags.html`

for how to create sparse banded matrices.
Thus, for example:

```
n = 5
e = ones(n,1);
A = spdiags([e e], [-1 1], n, n);
```

Creates the following matrix:

```
A =

   (2,1)        1
   (1,2)        1
   (3,2)        1
   (2,3)        1
   (4,3)        1
   (3,4)        1
   (5,4)        1
   (4,5)        1
```

where all other entries are then 0. Similarly:

```
 I = spdiags(ones(n,1),0);
```

Creates the identity matrix in sparse format.
The analogous commands in python are:

```
import numpy as np
from scipy.sparse import spdiags
n = 5
e = np.ones(n)
A = spdiags([e, e],[-1,1],n,n)

I = spdiags(e,0,5,5)
```

More information on these commands can be found at:

`http://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.spdiags.html`

We will return to this A matrix in a later homework.