2. Sparsity

1. Write a code that builds the second order finite difference matrix, called `s`, for the Laplacian an in two dimensions using the command `scipy.sparse.diags`. This data format exploits that most of the entries of the matrix are zero, which are not stored.

2. Select a size of $m = 100$, i.e. 100×100 grid points for the x and y axes. The size of `s` in bytes is given by

   ```
   s.data.nbytes + s.indptr.nbytes + s.indices.nbytes
   ```

   For comparison, we copy the matrix into a dense format with

   ```
   d = np.copy(sparse.todense())
   ```

   This latter format is similar to a $2d$ array in C++. The size in bytes can be obtained by `d.nbytes`. Print both the size of the sparse and dense matrices in megabytes.

3. Calculate a multiplication of the given matrix in both formats with a vector `v` consisting of ones only and print out the computation time. The current time in seconds can be obtained by `time.time()` after importing `import time`.

4. **Optional:** Repeat 2 and 3 with larger m. Depending on you system, maybe $m \approx 120 - 200$. Try to find a system monitor for your computer, which shows you how much working memory and swap memory is used. For $m = 100$ your computer will likely only use working memory. For higher m it will have to use swap memory. Observe the effect on the time of the matrix vector multiplication.

   WARNING: If m becomes large enough to use swap memory, your computer may freeze or become very slow. So save all your work before you try this and exit all other applications!

   Hint: If you just get a `MemoryError`, replace `d` by `d = np.ones((m*m, m*m))`