1 Matrix-Vector Product for a Sparse Matrix

Consider the matrix $A \in \mathbb{R}^{n \times n}$ given by

$$A = \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -1 & 2 & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{pmatrix}$$

and implement the matrix-vector product $A \cdot x$ for some vector $x \in \mathbb{R}^n$ in three different ways:

1. Implement the matrix A as a `numpy.ndarray` and define a function `dense(x)` which computes and outputs the numpy matrix-vector product `A@x`. Print the number of Gbytes which are needed to store the matrix (you can use the attribute `A.nbytes`).
2. Implement a function `matfree(x)` which outputs the matrix-vector product $A \cdot x$ without using the matrix A explicitly.
3. The "correct" way in Python: Use the modul `scipy.sparse` to implement the matrix A in CSR (compressed sparse row) format and define a function `sparse(x)` which computes and outputs the matrix-vector product using this CSR object. Print the number of Gbytes which are needed to store the matrix in CSR format (have a look at: `A.data`, `A.indptr`, `A.indices`).

Play around with the dimension n and measure the time which is needed in each way to compute the matrix-vector product for a random input vector `x = numpy.random.rand(n)`.

Hint: In the IPython shell you can simply use the *magic function* `%timeit` to measure the time for a certain operation. For example, you can type `%timeit dense(x)`.

Solution:

```
import numpy as np
import scipy.sparse as sparse
import timeit
import sys


runs = 500

# dense with numpy
n = 3500
A = 2 * np.eye(n) - np.eye(n , k=1) - np.eye(n , k=-1)
x = np.random.rand(n)

#-----------------------------------------------------------------#
#    NUMPY DENSE
#-----------------------------------------------------------------#
def dense():
    return A@x
print("---- Numpy Dense ----")
print("Memory:",np.round(A.nbytes * 10**-9, decimals=4), "Gbytes")
#%timeit dense(x)
print("Time", timeit.timeit("dense()", setup="from __main__ import dense",number=runs))
```

```
#-----------------------------------------------------------------#
#   MATRIX FREE
#-----------------------------------------------------------------#
def matfree():
    n = len(x)
    y = np.zeros(n)
    y[0] = 2 * x[0] - x[1]
    y[-1] = -x[-2] + 2*x[-1]
    for i in range(1,n-1):
        y[i] = - x[i-1] + 2*x[i] - x[i+1]
    return y
print("\n---- As a Function ----")
print("Memory:",np.round(sys.getsizeof(matfree) * 10**-9, decimals=4), "Gbytes")
#%timeit matfree(x)
print("Time:", timeit.timeit("matfree()", setup="from __main__ import matfree",number=runs))


#-----------------------------------------------------------------#
#   SCIPY SPARSE
#-----------------------------------------------------------------#
A = -3*sparse.eye(n,k=0) + sparse.eye(n , k=1) + sparse.eye(n , k=-1)
def sparse():
    return A.dot(x)
print("\n---- Scipy Sparse ----")
print("Memory:", np.round((A.data.nbytes + A.indptr.nbytes + A.indices.nbytes)* 10**-9,
    decimals=4), "Gbytes")
print("Time:", timeit.timeit("sparse()", setup="from __main__ import sparse",number=runs))
```