Dense versus sparse matrices

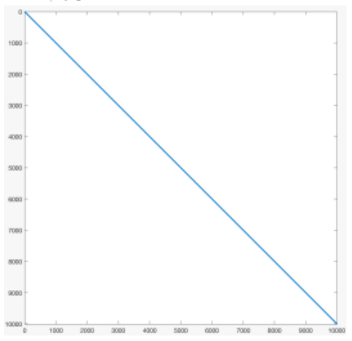
Dense versus sparse matrices

```
>> n = 10^4;
A = zeros(n,n);
for i = 1:n-1
  A(i,i+1) = i;
end
A(1:10,1:10)

ans =

     0     1     0     0     0     0     0     0     0     0
     0     0     2     0     0     0     0     0     0     0
     0     0     0     3     0     0     0     0     0     0
     0     0     0     0     4     0     0     0     0     0
     0     0     0     0     0     5     0     0     0     0
     0     0     0     0     0     0     6     0     0     0
     0     0     0     0     0     0     0     7     0     0
     0     0     0     0     0     0     0     0     8     0
     0     0     0     0     0     0     0     0     0     9
     0     0     0     0     0     0     0     0     0     0
```

```
>> whos A
  Name          Size                    Bytes  Class

  A         10000x10000             800000000  double
```

even though the matrix is almost all 0s, it uses lots of memory

```
>> spy(A)
```



```
>> B = sparse(n,n)

B =

   All zero sparse: 10000×10000

>> for i = 1:n-1
     B(i,i+1) = i;
   end
>> B(1:10,1:10)

ans =

   (1,2)        1
   (2,3)        2
   (3,4)        3
   (4,5)        4
   (5,6)        5
   (6,7)        6
   (7,8)        7
   (8,9)        8
   (9,10)       9
```

```
>> whos B
  Name          Size               Bytes  Class     Attributes

  B         10000x10000           240024  double    sparse
```

Matlab only stores the nonzero matrix elements, saving memory — and allowing for much faster calculations

full converts sparse → dense

```
>> full(B(1:10,1:10))

ans =

     0     1     0     0     0     0     0     0     0     0
     0     0     2     0     0     0     0     0     0     0
     0     0     0     3     0     0     0     0     0     0
     0     0     0     0     4     0     0     0     0     0
     0     0     0     0     0     5     0     0     0     0
     0     0     0     0     0     0     6     0     0     0
     0     0     0     0     0     0     0     7     0     0
     0     0     0     0     0     0     0     0     8     0
     0     0     0     0     0     0     0     0     0     9
     0     0     0     0     0     0     0     0     0     0
```

Python

```
n = 10 ** 4
A = sparse.lil_matrix((n,n))
for i in range(n-1):
  A[i,i+1] = i + 1
A = A.tocsc()  # Matlab uses compressed sparse column format for sparse matrices
print(A[:10,:10].toarray())

print(A.toarray().nbytes)  # dense matrix memory
print(A.data.nbytes + A.indptr.nbytes + A.indices.nbytes) # sparse matrix memory

[[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 2. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 3. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 4. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 5. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 6. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 7. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 8. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 9.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
800000000
159992
```

Other useful sparse matrix commands

spdiags:

```
>> full(spdiags((0:9)',0,10,10))

ans =

     0     0     0     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0     0     0
```

```
>> full(spdiags((0:9)',1,10,10))

ans =

     0     1     0     0     0     0     0     0     0     0
     0     0     2     0     0     0     0     0     0     0
```

```
>> full(spdiags((0:9)',0,10,10))

ans =

     0     0     0     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0     0     0
     0     0     2     0     0     0     0     0     0     0
     0     0     0     3     0     0     0     0     0     0
     0     0     0     0     4     0     0     0     0     0
     0     0     0     0     0     5     0     0     0     0
     0     0     0     0     0     0     6     0     0     0
     0     0     0     0     0     0     0     7     0     0
     0     0     0     0     0     0     0     0     8     0
     0     0     0     0     0     0     0     0     0     9
```

```
>> full(spdiags((0:9)',1,10,10))

ans =

     0     1     0     0     0     0     0     0     0     0
     0     0     2     0     0     0     0     0     0     0
     0     0     0     3     0     0     0     0     0     0
     0     0     0     0     4     0     0     0     0     0
     0     0     0     0     0     5     0     0     0     0
     0     0     0     0     0     0     6     0     0     0
     0     0     0     0     0     0     0     7     0     0
     0     0     0     0     0     0     0     0     8     0
     0     0     0     0     0     0     0     0     0     9
     0     0     0     0     0     0     0     0     0     0
```

```
>> full(spdiags((0:9)',-1,10,10))

ans =

     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0     0     0
     0     0     2     0     0     0     0     0     0     0
     0     0     0     3     0     0     0     0     0     0
     0     0     0     0     4     0     0     0     0     0
     0     0     0     0     0     5     0     0     0     0
     0     0     0     0     0     0     6     0     0     0
     0     0     0     0     0     0     0     7     0     0
     0     0     0     0     0     0     0     0     8     0
```

Python

```python
from scipy.sparse import spdiags
import numpy as np

spdiags(range(10),1,10,10).toarray()

array([[0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 2, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 3, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 4, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 5, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 6, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 7, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 8, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 9],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
```

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

sparse:

```
>> zeros(3,4)

ans =

     0     0     0     0
     0     0     0     0
     0     0     0     0
```

```
>> sparse(3,4)

ans =

   All zero sparse: 3×4
```

```python
import scipy

scipy.sparse.csr_matrix((3,4))

<3x4 sparse matrix of type '<class 'numpy.float64'>'
	with 0 stored elements in Compressed Sparse Row format>
```

```
>> A = sparse(1:9, 2:10, 1:9, 10, 10)

A =

   (1,2)        1
   (2,3)        2
   (3,4)        3
   (4,5)        4
   (5,6)        5
   (6,7)        6
   (7,8)        7
   (8,9)        8
   (9,10)       9
```

row indices   col indices   values

$$\text{sparse}([i_1, i_2, i_3], [j_1, j_2, j_3], [s_1, s_2, s_3])$$

$$\updownarrow$$

$$a_{i_1, j_1} = s_1, \quad a_{i_2 j_2} = s_2, \quad a_{i_3 j_3} = s_3$$

```
>> full(A)

ans =

     0     1     0     0     0     0     0     0     0     0
     0     0     2     0     0     0     0     0     0     0
     0     0     0     3     0     0     0     0     0     0
     0     0     0     0     4     0     0     0     0     0
     0     0     0     0     0     5     0     0     0     0
     0     0     0     0     0     0     6     0     0     0
     0     0     0     0     0     0     0     7     0     0
     0     0     0     0     0     0     0     0     8     0
     0     0     0     0     0     0     0     0     0     9
     0     0     0     0     0     0     0     0     0     0
```

Python

```python
from scipy import sparse
import numpy as np

n = 10
rows = np.arange(n-1) # np.arange() is like range()
cols = rows + 1       #  but ndarrays are nicer than lists
data = range(1,n)
A = sparse.csr_matrix((data, (rows,cols)), shape=(n,n))
A.toarray()

array([[0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 2, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 3, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 4, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 5, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 6, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 7, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 8, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 9],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]], dtype=int64)
```