A sparse $m \times n$ zero matrix is created with `sparse.coo_matrix((m, n))`. To create a sparse $n \times n$ identity matrix in Python, use `sparse.eye(n)`. We can also create a sparse diagonal matrix (with different offsets) using

In [ ]:
```
diagonals = [[1, 2, 3, 4], [1, 2, 3], [1, 2]]
B = sparse.diags(diagonals, offsets=[0,-1,2])
B.todense()
```

Out[ ]:
```
matrix([[1., 0., 1., 0.],
        [1., 2., 0., 2.],
        [0., 2., 3., 0.],
        [0., 0., 3., 4.]])
```

A useful function for creating a random sparse matrix is `sparse.rand()`. It generates a sparse matrix of a given shape and density with uniformly distributed values.

In [ ]:
```
matrix = sparse.rand(3, 4, density=0.25, format='csr',
↪  random_state=42)
matrix
```

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

6.3. Transpose, addition, and norm

Transpose. In VMLS we denote the transpose of an $m \times n$ matrix A as A^T. In Python, the transpose of `A` is given by `np.transpose(A)` or simply `A.T`.

In [ ]:
```
H = np.array([[0,1,-2,1], [2,-1,3,0]])
H.T
```

Out[ ]:
```
array([[ 0,  2],
       [ 1, -1],
       [-2,  3],
       [ 1,  0]])
```

In [ ]:
```
np.transpose(H)
```

Out[ ]:
```
array([[ 0,  2],
       [ 1, -1],
       [-2,  3],
       [ 1,  0]])
```

Addition, subtraction, and scalar multiplication. In Python, addition and subtraction of matrices, and scalar-matrix multiplication, both follow standard and mathematical notation.

In [ ]:
```
U = np.array([[0,4], [7,0], [3,1]])
V = np.array([[1,2], [2,3], [0,4]])
U + V
```

Out[ ]:
```
array([[1, 6],
       [9, 3],
       [3, 5]])
```

In [ ]:
```
2.2*U
```

Out[ ]:
```
array([[ 0. ,  8.8],
       [15.4,  0. ],
       [ 6.6,  2.2]])
```

(We can also multiply a matrix on the right by a scalar.) Python supports some operations that are not standard mathematical ones. For example, in Python you can add or subtract a constant from a matrix, which carries out the operation on each entry.

Elementwise operations. The syntax for elementwise vector operations described on page 11 carries over naturally to matrices.

Matrix norm. In VMLS we use $\|A\|$ to denote the norm of an $m \times n$ matrix,

$$\|A\| = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} A_{ij}^2 \right)^{\frac{1}{2}}$$

In standard mathematical notation, this is more often written as $\|A\|_F$, where F stands for the name Frobenius. In standard mathematical notation, $\|A\|$ usually refers to another norm of a matrix, which is beyond the scope of topics in VMLS. In Python, `np.linalg.norm(A)` gives the norm used in VMLS.

In [ ]:
```
A = np.array([[2,3,-1], [0,-1,4]])
np.linalg.norm(A)
```

Out[ ]:
```
5.5677643628300215
```

In [ ]:
```
np.linalg.norm(A[:])
```

```
Out[ ]: 5.5677643628300215
```

Triangle inequality. Let's check that the triangle inequality $\|A+B\| \leq \|A\| + \|B\|$ holds, for two specific matrices.

```
In [ ]: A = np.array([[-1,0], [2,2]])
        B = np.array([[3,1], [-3,2]])
        print(np.linalg.norm(A + B))
        print(np.linalg.norm(A) + np.linalg.norm(B))
```

```
4.69041575982343
7.795831523312719
```

Alternatively, we can write our own code to find the norm of `A`:

```
In [ ]: A = np.array([[1,2,3], [4,5,6], [7,8,9], [10,11,12]])
        m,n = A.shape
        print(A.shape)
        sum_of_sq = 0
        for i in range(m):
            for j in range(n):
                sum_of_sq = sum_of_sq + A[i,j]**2
        matrix_norm = np.sqrt(sum_of_sq)
        print(matrix_norm)
        print(np.linalg.norm(A))
```

```
(4, 3)
25.495097567963924
25.495097567963924
```

6.4. Matrix-vector multiplication

In Python, matrix-vector multiplication has the natural syntax `y = A @ x`. Alternatively, we can use the numpy function `np.matmul(A,x)`.

```
In [ ]: A = np.array([[0,2,-1],[-2,1,1]])
        x = np.array([2,1,-1])
        A @ x
```

```
Out[ ]: array([ 3, -4])
```