

## 8.1: Sparse Matrix Algebra

When matrices are sparse, ordinary approaches to matrix arithmetic are wasteful, since a lot of time is spent adding/subtracting or multiplying by zero. For instance, consider the tridiagonal matrix discussed above. To perform the matrix-vector product  $\mathbf{H}\vec{\psi}$  in the usual way, for each row  $i$  we must compute

$$\left(\mathbf{H}\vec{\psi}\right)_i = \sum_{j=0}^{N-1} H_{ij}\psi_j \quad (8.1.1)$$

$$= \cdots + (0 \cdot \psi_{i-2}) + (H_{i,i-1} \cdot \psi_{i-1}) + (H_{ii} \cdot \psi_i) + (H_{i,i+1} \cdot \psi_{i+1}) + (0 \cdot \psi_{i+2}) + \cdots \quad (8.1.2)$$

The sum involves  $O(N)$  arithmetic operations, so the overall runtime for all  $N$  rows is  $O(N^2)$ . Clearly, however, most of this time is spent multiplying zero elements of  $\mathbf{H}$  with elements of  $\vec{\psi}$ , which doesn't contribute to the final result. If we could omit these terms from each sum, the overall runtime for the product would be only  $O(N)$ .

However, such savings cannot be achieved with the array data structures we have been using so far. To calculate the matrix-vector product efficiently, the processor needs to know which elements on each row of  $\mathbf{H}$  are non-zero, so that it can skip the rest. However, arrays do not provide a fast way to identify which elements are non-zero; the only way to find out is to search the storage blocks one by one, which takes  $O(N)$  time on each row. That would wipe out the runtime savings.

Scipy provides special data structures for storing sparse matrices. Unlike ordinary arrays, these data structures are designed so that zero elements are omitted, which not only saves memory, but also allows certain matrix operations to be greatly sped up. Unlike arrays, which can represent not just matrices (2D arrays) but also vectors (1D arrays) and higher-rank tensors (arrays with dimension higher than 2), these sparse data structures are specifically restricted to matrices.

But here's an important catch: there is no single sparse matrix data structure that is ideal for every scenario. Instead, there are multiple **sparse matrix formats**, and each format is only effective for a certain subset of matrix operations, or for certain kinds of sparse matrices. Therefore, we need to know how the different sparse matrix formats are implemented, as well as their benefits and limitations. Of the [many sparse matrix formats offered by Scipy](#), we will discuss four: List of Lists (LIL), Diagonal Storage (DIA), Compressed Sparse Row (CSR), and Compressed Sparse Column (CSC).

This page titled [8.1: Sparse Matrix Algebra](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by [Y. D. Chong](#) via [source content](#) that was edited to the style and standards of the LibreTexts platform.