numpy notes

Created	@April 20, 2025 8:00 PM

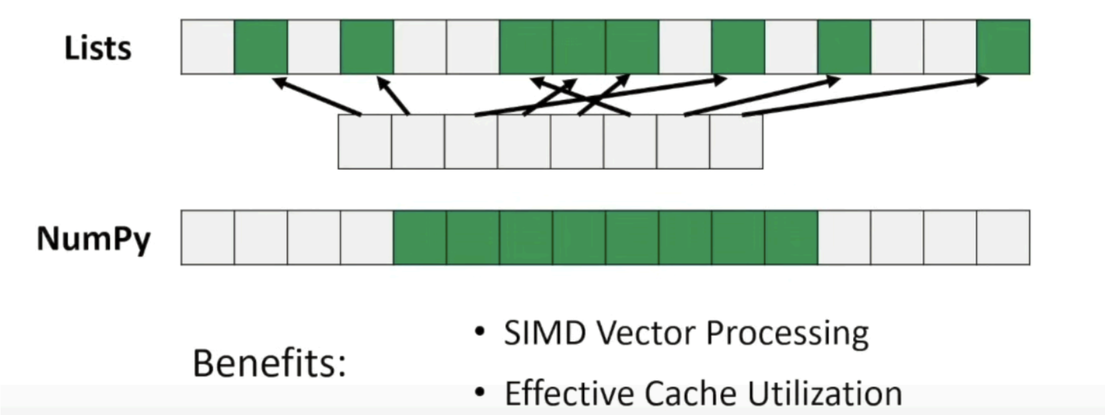
NumPy is a library for working with homogeneous, N-dimensional arrays in Python. Unlike a built-in Python list—which is really an array of pointers to arbitrary Python objects—every NumPy array occupies a single, contiguous block of memory, with each element taking up a fixed number of bytes (e.g. 8 bytes for a 64-bit float). In contrast, a Python int object on a list requires at least:

- 4 bytes for the integer value
- 8 bytes for a pointer to its type
- 8 bytes for reference counting
- 8 bytes for other object metadata

plus the overhead of the list's array of object pointers.

Because NumPy arrays store data in tightly packed, static-type format, they:

1. **Use much less memory per element**,
2. **Eliminate Python-level type checks and pointer indirections** when you iterate or perform arithmetic, and
3. NumPy utilizes contiguous memory, which means they store elements as a single, contiguous block of memory rather than as pointers to separate Python objects scattered around the heap.



Some simple math concepts where numpy is useful -

Trace

- **What it is:** The sum of the diagonal entries of a square matrix.
- **Why it matters:**
  - Invariant under change of basis – stays the same if you rotate your coordinate system.
  - Shows up in the characteristic polynomial (used for finding eigenvalues).
- **Example:**

$$A = \begin{pmatrix} 2 & 5 \\ 1 & 3 \end{pmatrix}, \quad \mathrm{trace}(A) = 2 + 3 = 5$$

Singular Value Decomposition (SVD)

- **What it is:** A factorization of any $m \times n$ matrix A into three parts

$$A = U\,\Sigma\,V^T$$

  where

  - U is an $m \times m$ orthogonal matrix (its columns are left-singular vectors),
  - Σ is an $m \times n$ diagonal matrix of nonnegative **singular values**,
  - V is an $n \times n$ orthogonal matrix (its columns are right-singular vectors).
- **Why it matters:**
  - Gives the "best" low-rank approximations of A (useful in compression, noise reduction).
  - Reveals the directions in which A stretches or squashes space.

- **Intuition:**
  1. V^T rotates your data into a coordinate system where axes are "principal directions."
  2. Σ scales each axis by a singular value (how much A stretches in that direction).
  3. U rotates the result into the final output space.

Eigenvalues & Eigenvectors

- **What they are:** For a square matrix A, a nonzero vector v and scalar λ satisfying

$$A\,v = \lambda\,v.$$

  - v is an **eigenvector**, the direction unchanged by A.
  - λ is the **eigenvalue**, the factor by which A stretches v.
- **Why they matter:**
  - Describe fundamental modes of a linear transformation (e.g. principal axes).
  - Used in stability analysis, PCA, differential equations.
- **Example:**

$$A = \begin{pmatrix} 4 & 0 \\ 0 & 2 \end{pmatrix} \Rightarrow \lambda_1 = 4,\ v_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}; \quad \lambda_2 = 2,\ v_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Frobenius Norm

- **What it is:** A way to measure the "size" of a matrix, defined as

$$\|A\|_F = \sqrt{\sum_{i,j} A_{ij}^2}\,.$$

- **Why it matters:**
  - Generalizes the Euclidean length of a vector to matrices.
  - Simple to compute and useful for measuring error (e.g. $\|A - B\|_F$).
- **Intuition:** Treat all entries of A as coordinates in a big vector, then take its usual length.

Example -

For the blue part -

a[2:4 , 0:2]

For the green part -

a[[0,1,2,3] , [1,2,3,4]] # [row] , [column]

For the red part -

a[[0,4,5] , 3:]

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

```
>>> np.info(np.ndarray.dtype)
```

Aggregate Functions

```
>>> a.sum() #Array-wise sum
>>> a.min() #Array-wise minimum value
>>> b.max(axis=0) #Maximum value of an array row
>>> b.cumsum(axis=1) #Cumulative sum of the elements
>>> a.mean() #Mean
>>> np.median(b) #Median
>>> np.corrcoef(a) #Correlation coefficient
>>> np.std(b) #Standard deviation
```

Boolean Indexing

```
>>> a[a<2] #Select elements from a less than 2
 array([1])
```

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]] #Select elements (1,0),(0,1),(1,2) and (0,0)
 array([ 4. , 2. , 6. , 1.5])
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]] #Select a subset of the matrix's rows and columns
 array([[ 4. ,5. , 6. , 4. ],
        [ 1.5, 2. , 3. , 1.5],
        [ 4. , 5. , 6. , 4. ],
        [ 1.5, 2. , 3. , 1.5]])
```

Transposing Array

```
>>> i = np.transpose(b) #Permute array dimensions
>>> i.T #Permute array dimensions
```

Changing Array Shape

```
>>> b.ravel() #Flatten the array
>>> g.reshape(3,-2) #Reshape, but don't change data
```

Adding/Removing Elements

```
>>> h.resize((2,6)) #Return a new array with shape (2,6)
>>> np.append(h,g) #Append items to an array
>>> np.insert(a, 1, 5) #Insert items in an array
>>> np.delete(a,[1]) #Delete items from an array
```

Combining Arrays

```
>>> np.concatenate((a,d),axis=0) #Concatenate arrays
 array([ 1, 2, 3, 10, 15, 20])
>>> np.vstack((a,b)) #Stack arrays vertically (row-wise)
 array([[ 1. , 2. , 3. ],
        [ 1.5, 2. , 3. ],
        [ 4. , 5. , 6. ]])
>>> np.r_[e,f] #Stack arrays vertically (row-wise)
>>> np.hstack((e,f)) #Stack arrays horizontally (column-wise)
 array([[ 7., 7., 1., 0.],
        [ 7., 7., 0., 1.]])
```

```
>>> np.column_stack((a,d)) #Create stacked column-wise arrays
 array([[ 1, 10],
        [ 2, 15],
        [ 3, 20]])
>>> np.c_[a,d] #Create stacked column-wise arrays
```

Splitting Arrays

```
>>> np.hsplit(a,3) #Split the array horizontally at the 3rd index
 [array([1]),array([2]),array([3])]
>>> np.vsplit(c,2) #Split the array vertically at the 2nd index
 [array([[[ 1.5, 2. , 1. ],
          [ 4. , 5. , 6. ]]]),
 array([[[ 3., 2., 3.],
         [ 4., 5., 6.]]])]
```

```
>>> h = a.view() #Create a view of the array with the same data
>>> np.copy(a) #Create a copy of the array
>>> h = a.copy() #Create a deep copy of the array
```