Common NumPy Vectorization Patterns

Element-wise Operations

```
import numpy as np

# Element-wise maximum/minimum
a = np.array([1, 5, 3, 8, 2])
b = np.array([4, 2, 6, 1, 9])

max_elements = np.maximum(a, b)
min_elements = np.minimum(a, b)

print("a:", a)
print("b:", b)
print("Maximum:", max_elements)
print("Minimum:", min_elements)

# Element-wise comparison
c = np.array([10, 20, 30])
d = np.array([15, 20, 25])
print("\nEqual elements:", c == d)
print("Greater elements:", c > d)
```

Conditional Operations

```
data = np.array([-2, -1, 0, 1, 2])

# np.where: conditional selection
result = np.where(data > 0, data * 2, data)
print("If positive multiply by 2, else keep:", result)

# Multiple conditions
result2 = np.where(data > 0, "positive",
                   np.where(data < 0, "negative", "zero"))
print("Label values:", result2)

# Clipping values
clipped = np.clip(data, -1, 1)
print("\nClipped to [-1, 1]:", clipped)

# Replace values
arr = np.array([1, 2, -1, 4, -1, 6])
arr[arr == -1] = 0
print("Replace -1 with 0:", arr)
```

Distance and Norm Calculations

```
# Euclidean distance from origin
points = np.array([[3, 4],
                   [5, 12],
                   [8, 15]])

# Method 1: Manual calculation
distances = np.sqrt(np.sum(points**2, axis=1))
print("Distances (manual):", distances)

# Method 2: Using linalg.norm
distances_2 = np.linalg.norm(points, axis=1)
print("Distances (linalg):", distances_2)
```

```
# Pairwise distances between points
def pairwise_distances(X):
    # Broadcasting trick
    diff = X[:, np.newaxis, :] - X[np.newaxis, :, :]
    return np.sqrt(np.sum(diff**2, axis=2))

dist_matrix = pairwise_distances(points)
print("\nPairwise distances:\n", dist_matrix)
```

Aggregation Patterns

```
# Statistics along axes
data = np.array([[1, 2, 3],
                 [4, 5, 6],
                 [7, 8, 9]])

print("Overall mean:", np.mean(data))
print("Mean per column:", np.mean(data, axis=0))
print("Mean per row:", np.mean(data, axis=1))

# Multiple statistics
print("\nStd deviation:", np.std(data))
print("Median:", np.median(data))
print("Sum:", np.sum(data))
print("Product:", np.prod(data))

# Cumulative operations
arr = np.array([1, 2, 3, 4, 5])
print("\nCumulative sum:", np.cumsum(arr))
print("Cumulative product:", np.cumprod(arr))
```

Broadcasting Patterns

```
# Broadcasting for efficient computation
# Add row vector to each row of matrix
matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])
row_vector = np.array([10, 20, 30])

result = matrix + row_vector
print("Matrix + row vector:\n", result)

# Add column vector to each column
col_vector = np.array([[100], [200], [300]])
result2 = matrix + col_vector
print("\nMatrix + column vector:\n", result2)

# Outer product using broadcasting
a = np.array([1, 2, 3])
b = np.array([10, 20, 30, 40])
outer = a[:, np.newaxis] * b
print("\nOuter product:\n", outer)
```