```
/Users/leonslaptop/anaconda3/envs/python39/bin/python /U
Mean Absolute Error (MAE): 0.236885367367657 47

Process finished with exit code 0
```

Code:

```
def upwindEikonal(self, node):
  neighbors = [(1, 0, 0), (-1, 0, 0), (0, 1, 0), (0, -1, 0), (0, 0, 1), (0,
0, -1)]
  r, c, d = self.dmap.shape
  for dx, dy, dz in neighbors:
    x, y, z = node.x + dx, node.y + dy, node.z + dz
    if 0 <= x < r and 0 <= y < c and 0 <= z < d and self.active[x, y, z] ==
1:
      # Compute tentative distances based on the Eikonal equation
      a_dist, b_dist, c_dist = np.inf, np.inf, np.inf
      if 0 <= x - 1 < r:
        a_dist = self.dmap[x - 1, y, z]
      if x + 1 < r:
        a_dist = min(a_dist, self.dmap[x + 1, y, z])
      if 0 <= y - 1 < c:
        b_dist = self.dmap[x, y - 1, z]
      if y + 1 < c:
        b_dist = min(b_dist, self.dmap[x, y + 1, z])
      if 0 <= z - 1 < d:
        c_dist = self.dmap[x, y, z - 1]
      if z + 1 < d:
        c_dist = min(c_dist, self.dmap[x, y, z + 1])

      # Ensure a, b, c are sorted such that a <= b <= c
      a_dist, b_dist, c_dist = sorted([a_dist, b_dist, c_dist])

      # Calculate new distance using the Eikonal equation
      F = 1.0 / self.speed[x, y, z]
      distance = self.solveEikonal(a_dist, b_dist, c_dist, F)

      # Update the distance map and active set if the new distance is smaller
      if distance < self.dmap[x, y, z]:
        self.dmap[x, y, z] = distance
        self.nb.insert(lSNode(x, y, z, distance))  # Add node to heap

def solveEikonal(self, a, b, c, F):
  """
  Solves the Eikonal equation given three smallest distances from neighbors
(a, b, c)
  and the local speed (F) at the node. Returns the updated distance T to the
node.

  Parameters:
  - a, b, c: float, the three smallest neighbor distances, where a <= b <= c
```

```
  - F: float, the local speed at the node

  Returns:
  - T: float, the updated distance to the node
  """

  # Calculate the argument of the square root to check if it's non-negative
  sqrt_arg = 2 * F ** 2 - (a - b) ** 2

  if sqrt_arg >= 0:
    # Safe to take the square root
    T = (a + b + np.sqrt(sqrt_arg)) / 2
  else:
    # Fallback to using only the smallest distance and F if the square root
argument is negative
    T = a + F

  # Further check to ensure T does not exceed c, indicating an issue with the
chosen distances
  if T > c:
    T = a + F

  return T
```