```python
# Thanks to Andrew Loeppky and Ruth Moore for sharing their python codes for ppm.
# I also copied the two import commands, which would apear first in the full program

import numpy as np
import matplotlib.pyplot as plt

# What follows is just the ppm advection code, defined as a python function.
# It still needs to be inserted into your own program to set up the grids,
# arrays, initial conditions C, and the loop over all timesteps.
# I added the in-line comments from my R code to Andrew's & Ruth's python code

# Note that a significant difference between R and python is that I started my
# R arrays at index = 1, while Andrew started his python arrays at index = 0.

# Note that I found a bug in my R code, which I think was copied as a bug into the
# codes by Andrew and Ruth.  I think I fixed it here, but let me know if errors.

# For definitions of variables that are not given in this code fragment, please
# see our cmaq homework assignment for ppm advection.
# R. Stull, 7 Apr 2024.  Univ. of British Columbia.



def ppm(C, u, delt):

    # Create values for key constants
    sixth = 1 / 6
    two3rds = 2 / 3
    oneoverdelx = 1 / delx

    # Allocate the vectors
    imax = len(C)  # avoid using global variable imax by reassigning
    dc = np.ones(imax)  # nominal difference in concentration across a cell
    clfirst = np.ones(imax)  # first guess of conc at left edge of cell i
    cr = np.ones(imax)  # conc at right edge of cell i
    cl = np.ones(imax)  # conc at left edge of cell i
    c6 = np.ones(imax)  # corresponds to parabola parameter a6 of CW eq.(1.4)
    FL = np.ones(imax)  # pollutant flux into the left side of a grid cell
    twoFR = np.ones(imax)  # pollutant flux into the right side of a grid cell

    # To guarantee that solution is monotonic, check that the left edge of cell i
    # (which is between cells i and i-1) should not have a concentration lower
    # or higher than the concentrations in those two neighboring cells
    # Namely, is clfirst between c[i] and c[i-1]. If not, then fix.
    for i in range(1, imax - 1):     # for each interior grid point i
        del_cl = C[i] - C[i - 1]     # concentration difference with cell at left
        del_cr = C[i + 1] - C[i]     # concentration difference with cell at right
        dc[i] = 0.5 * (del_cl + del_cr)  # 1st guess of avg conc difference across cell i

        if (del_cl * del_cr) > 0.0:  # then revise average difference across cell i
            dc[i] = np.sign(dc[i]) * min(abs(dc[i]), 2 * abs(del_cl), 2 * abs(del_cr))
        else:
            dc[i] = 0                # for the special case of constant conc across cell

    # First guess for concentration at left edge of each cell, using revised dc value
    for i in range(1, imax - 1):      # for each interior grid point i
        clfirst[i] = 0.5 * (C[i] + C[i - 1]) - sixth * (dc[i] - dc[i - 1])

    # find parameters for the piecewise-continuous parabola in cell i
    for i in range(1, imax - 1):      # for each interior grid point i
        # conc at the right edge (cr) of cell i equals concen at left edge of cell i+1
        cr[i] = clfirst[i + 1]         # concentration at right edge of cell i
        cl[i] = clfirst[i]             # concentration at left edge of cell i

        # Check whether cell i is an extremum (is a peak or valley in the conc plot)
```

```
        if ((cr[i] - C[i]) * (C[i] - cl[i])) > 0:    # then not extremum
            # Find the two coefficients of the parabola: dc and c6:
            dc[i] = cr[i] - cl[i]       # updated concen diff. between right & left edges
            c6[i] = 6 * (C[i] - 0.5 * (cl[i] + cr[i]))

            if (dc[i] * c6[i]) > (dc[i] * dc[i]):  # then adjust for overshoot at left edge
                cl[i] = 3.0 * C[i] - 2.0 * cr[i]
            elif (-dc[i] * dc[i]) > (dc[i] * c6[i]):  # then adjust for overshoot at right
                cr[i] = 3.0 * C[i] - 2.0 * cl[i]

        # if extremum, assume constant value instead of a parabola
        else:                   # For an extremum, don't use a parabola.
            cl[i] = C[i]        # Instead, assume concen is constant across the cell,
            cr[i] = cl[i]       # Thus, left and right concentrations equal average conc.

        # second guess of coefficients for the parabola, from CW eq. (1.5)
        dc[i] = cr[i] - cl[i]
        c6[i] = 6.0 * (C[i] - 0.5 * (cl[i] + cr[i]))

    # end of grid-point (i) loop

    # Initialize to 0 the fluxes into the left and right sides of every cell i
    # assign fluxes from left and right sides
    FL = np.zeros(imax + 1)
    FR = np.zeros(imax + 1)

    # Next, use parabolic fits within each cell to calculate the fluxes betweeen cells

    # At left side of whole domain (i = 0), assume constant flux. Use FR[0] = FR[1]
    # wind from left, do BCs
    if u > 0.0:                  # if wind enters left boundary of domain
        y4 = u * delt            # distance traversed by wind during delt
        x4 = y4 * oneoverdelx    # Courant number is fraction of grid cell traversed
        # Find the flux leaving the right side of left boundary cell
        # Assume flux is parabolic in x
        FR[0] = y4 * (cr[1] - 0.5 * x4 * (dc[1] - c6[1] * (1.0 - two3rds * x4)))

    # In interior of whole domain, use parabola eqs. CW (1.12) to find the fluxes
    # do the upwinding given local wind direction
    for i in range(1, imax - 1):   # for each interior grid point i
        if u < 0:                  # for wind from right to left
            y4 = -u * delt         # distance traversed by wind during delt
            x4 = y4 * oneoverdelx  # Courant number is fraction of grid cell traversed
            # Assume parabolic in x
            FL[i] = y4 * (cl[i] + 0.5 * x4 * (dc[i] + c6[i] * (1.0 - two3rds * x4)))

        if u > 0.0:                 # for wind from left to right
            y4 = u * delt           # distance traversed by wind during delt
            x4 = y4 * oneoverdelx   # Courant number is fraction of grid cell traversed
            # Assume parabolic in x
            FR[i] = y4 * (cr[i] - 0.5 * x4 * (dc[i] - c6[i] * (1.0 - two3rds * x4)))

      # end of loop over all interior grid cells

    # At right side of whole domain(i = imax), assume const.flux.Use FL[imax] = FL[imax-1]
        if u < 0:                  # if wind enters right boundary of domain
            y4 = -u * delt         # distance traversed by wind during delt
            x4 = y4 * oneoverdelx  # Courant number is fraction of grid cell traversed
            FL[imax] = y4 * (
                cl[imax - 1]
                + 0.5 * x4 * (dc[imax - 1] + c6[imax - 1] * (1.0 - two3rds * x4))
            )

# For a realistic case, you would want to impose the actual fluxes at the boundaries.
# But for our simple HW,impose boundary conditions of 0 pollutant flux entering the domain.
```

```
131     if u > 0.0:
132         FR[0] = 0.0
133     if u < 0.0:
134         FL[imax] = 0.0
135 
136 
137 # Update the concentrations in each grid cell.  *** This is the forecast equation.***
138     # apply the forecast eqn to interior and get new concentration C
139     for i in range(1, (imax - 1)):
140         C[i] = C[i] + oneoverdelx * (FR[i - 1] - FR[i] + FL[i + 1] - FL[i])
141 
142     return C
143 
```