```
# ================================
# 6) Use the HPPM method from CMAQ
# CW refers to the paper by Colella and Woodward.

# 1-D domain covers grid points i = 1 to imax.  But 1 and imax are boundary-
# condition cells.  The main interior computation is for i = 2 to (imax-1).

# Pre-calculate some constants
sixth = 1.0/6.0
two3rds = 2.0/3.0
oneoverdelx = 1.0 / delx

# Allocate the vectors
dc = numeric(imax)       # nominal difference in concentration across a cell
clfirst = numeric(imax)  # first guess of conc at left edge of cell i
cr = numeric(imax)       # conc at right edge of cell i
cl = numeric(imax)       # conc at left edge of cell i
c6 = numeric(imax)       # this corresponds to parabola parameter a6 of CW eq.(1.4)
FL = numeric(imax)       # pollutant flux into the left side of a grid cell
FR = numeric(imax)       # pollutant flux into the right side of a grid cell


# Iterate forward in time
for (n in 1:nsteps) {                   # for each time step n


    # 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 2:(imax - 1)) {           # for each interior grid point i
        del_cl = conc[i] - conc[i-1]    # concentration difference with cell at left
        del_cr = conc[i+1] - conc[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] = sign(dc[i]) * min( abs(dc[i]) , 2*abs(del_cl) , 2*abs(del_cr) )
        } else {dc[i]=0.0}              # for the special case of constant conc across cell
    }                                   # end of grid-point (i) loop

    # First guess for concentration at left edge of each cell, using revised dc value
    for (i in 2:(imax - 1)) {           # for each interior grid point i
        clfirst[i] = 0.5*(conc[i]+conc[i-1]) - sixth*(dc[i]-dc[i-1])
    }                                   # end of grid-point (i) loop

    # find parameters for the piecewise-continuous parabola in cell i
    for (i in 2:(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]-conc[i]) * (conc[i] - cl[i]) )   > 0.0) {   # then not extremum

            # Find the two coefficients of the parabola: dc and c6:
            dc[i] = cr[i] - cl[i]       # updated concen diff. between right and left edges
            c6[i] = 6*( conc[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*conc[i] - 2.0*cr[i]
            } else if ((-dc[i]*dc[i]) > (dc[i]*c6[i])) {  # then adjust for overshoot at right
                cr[i] = 3.0*conc[i] - 2.0*cl[i]
            }                           # end of block of "not extremum" calculations

```

```
        } else {                        # For an extremum, don't use a parabola.
            cl[i] = conc[i]             # Instead, assume concen is constant across the cell,
            cr[i] = cl[i]               # Thus, left and right concentrations equal average conc.
        }                               # end of grid-point (i) loop

        # second guess of coefficients for the parabola, from CW eq. (1.5)
        dc[i] = cr[i] - cl[i]
        c6[i] = 6.0*(conc[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 cell i
    FL <- rep(0.0, imax)
    FR <- rep(0.0, imax)


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

    # At left side of whole domain (i = 1), assume constant flux. Use FR[1] = FR[2]
    if (u > 0.0) {                      # if wind enters left boundary of domain
        y = u*delt                      # distance traversed by wind during delt
        x = y*oneoverdelx               # Courant number is fraction of grid cell traversed
        # Find the flux leaving the right side of left boundary cell
        FR[1] = y*( cr[2] - 0.5*x*(dc[2] - c6[2]*(1.0 - two3rds*x))  )    # parabolic in x
    }

    # In interior of whole domain, use parabola eqs. CW (1.12) to find the fluxes
    for (i in 2:(imax-1)) {             # for each interior grid point i

        if (u < 0.0) {                  # for wind from right to left
            y = -u*delt                 # distance traversed by wind during delt
            x = y*oneoverdelx           # Courant number is fraction of grid cell traversed
            FL[i] = y*( cl[i] + 0.5*x*(dc[i] + c6[i]*(1.0 - two3rds*x))  )   # parabolic in x
        }

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

    }                                   # 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 < 0.0) {                      # if wind enters right boundary of domain
        y = -u*delt                     # distance traversed by wind during delt
        x = y*oneoverdelx               # Courant number is fraction of grid cell traversed
        FL[imax] = y*( cl[imax-1] + 0.5*x*(dc[imax-1] + c6[imax-1]*(1.0 - two3rds*x))  )
    }


    # For a realistic case, you would want to impose the actual fluxes at the boundaries.
    # But for our simple HW, impose boundry conditions of zero pollutant flux entering the domain.
    if (u > 0.0) FR[1] = 0.0
    if (u < 0.0) FL[1] = 0.0


    # Update the concentrations in each grid cell.  *** This is the forecast equation.***

    for (i in 2:(imax-1)) {             # for each interior grid point i
        conc[i] <- conc[i] + oneoverdelx* (FR[i-1] - FR[i] + FL[i+1] - FL[i])  # CW eq. 1.13
    }                                   # end of loop over all interior grid cells i

}                                       # end of loop over all time iterations n
```