HW 7

ATSC 507

Christopher Rodell

```
In [1]: import context
        import numpy as np
        from cr507.utils import plt_set
        import matplotlib.pyplot as plt
        from collections import namedtuple
        from puff_funs import Approximator
```

```
******************************
context imported. Front of path:
/Users/rodell/atsc507
/private/var/folders/hc/bh1xlzfj3_n4c5gz42dbpw400000gn/T/8179651f-a0ff-
4a37-9568-14987a1f4de3
******************************

through /Users/rodell/atsc507/py/hw7/context.py -- pha
through /Users/rodell/atsc507/cr507/__init__.py pha II
```

A 1-D pollutant puff "anomaly" is being advected in the x-direction by a constant wind u. The "anomaly" includes positive and negative concentration deviations about a mean concentration.

$$\frac{\partial P}{\partial t} = -u_0 \frac{\partial P}{\partial x}$$

The 3 finite-difference schemes you will compare are

- (a) FTBS - Forward in time, Backward in space;

$$P_{j,n+1} = P_{j,n} - u_0 \frac{P_{j,n} - P_{j-1,n}}{\Delta x} \Delta t$$

- (b) RK3 - - RK3 centered in space

$$\begin{aligned} P^{*}_{i,n} &= P_{i,n} + \frac{\Delta t}{3}\left[-u_0 \frac{P_{i+1,n} - P_{i-1,n}}{2\Delta x}\right] \\ P^{**}_{i,n} &= P_{i,n} + \frac{\Delta t}{2}\left[-u_0 \frac{P^{*}_{i+1,n} - P^{*}_{i-1,n}}{2\Delta x}\right] \\ P_{i,n+1} &= P_{i,n} + \Delta t\left[-u_0 \frac{P^{**}_{i+1,n} - P^{**}_{i-1,n}}{2\Delta x}\right] \end{aligned}$$

- (c) PPM - Piecewise Parabolic Method where PPM is the scheme used to advect pollutants in the CMAQ model.

  Wasnt able to find a good source defining the PPM method equation ...I copy-pasted your code to make the PPM plot work I wanted to rewrite in python but...didn't happen, sorry...The R scripted (for PPM) and approximator class (for RK3 and FTBS) are at the bottom of this pdf

In [2]:
```
## Create the grid and initial conditions
initialVals={'gridx': 1000 , 'dx':100.  ,'dt':10. , 'u0': 5. , \
     'xx': np.arange(0,1000,1) , 'cmax': 10. }

## Call on approximator class
coeff = Approximator(initialVals)
```

- 1) Calculate and display the Courant number. You can display it in the graph in the question (3) if you would like.

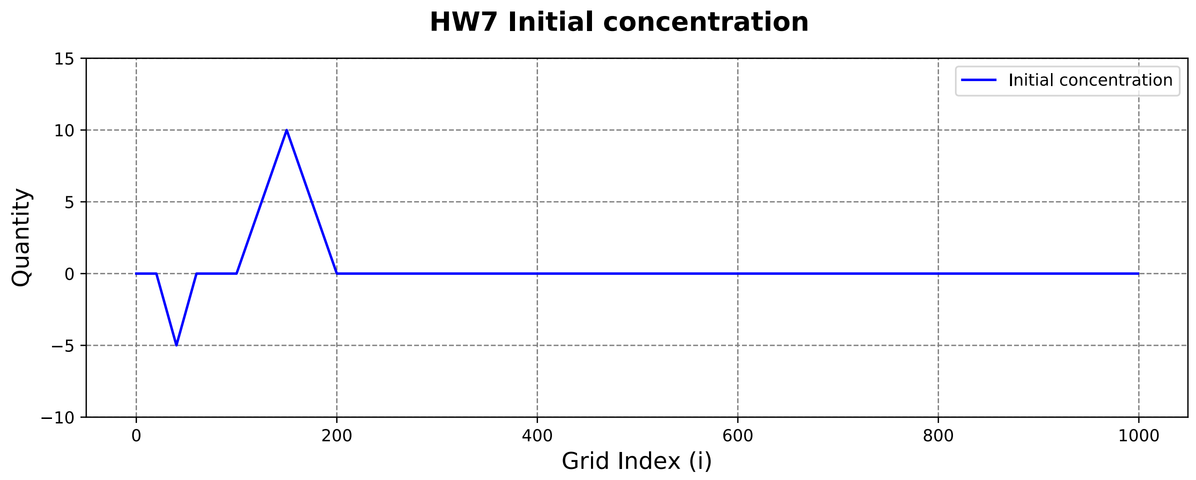
$$C = \frac{u\Delta t}{\Delta x} \leq C_{\max}$$

In [3]:
```
print("Courant number  ", coeff.cr)
```

```
Courant number   0.5
```

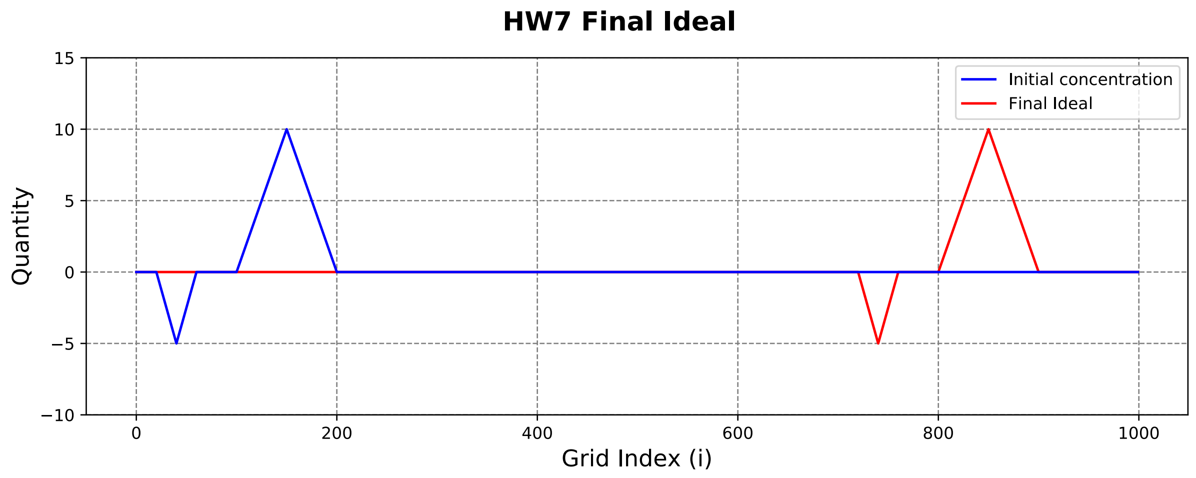
- 2) (a) Create initial concentration anomaly distribution in the x-direction *See def**init**() in Approximator class*
- (b) Plot (using blue colour) the initial concentration distribution on a graph.

In [4]:
```
## Plot With initial concentration in blue
plot = coeff.plot_functions('Initial')
```



3) Also, on the same plot, show (in red) the ideal exact final solution, after the puff anomaly has been advected downwind, as given by *See def**init**() in Approximator class*

In [5]:
```
## Plot With final concentration in red
plot = coeff.plot_functions('Final')
```

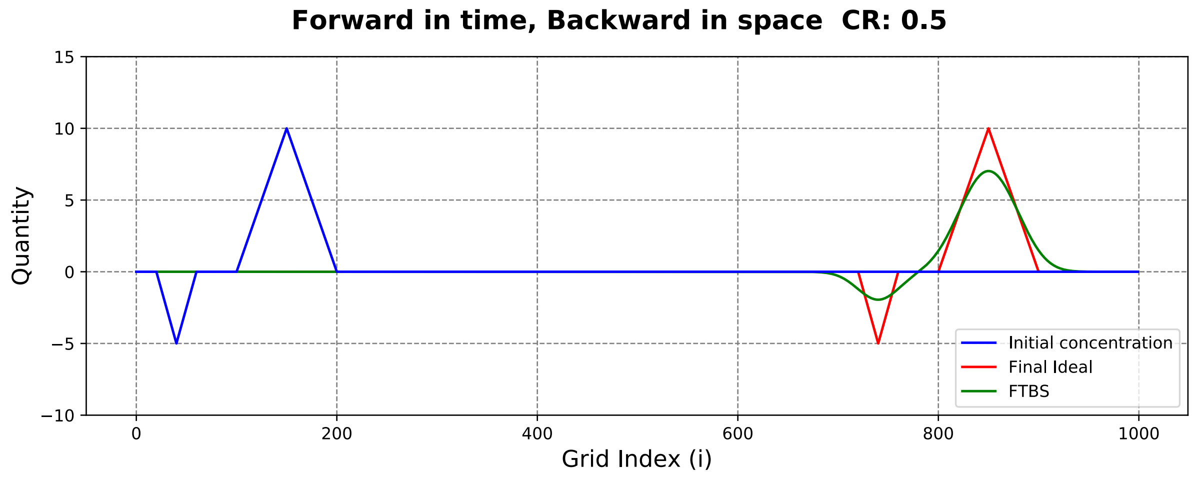


4) Advect the concentration puff anomaly for the following number of time steps and plot (in green) the resulting concentration on the same graph, using

Plot Forward in time backward in space

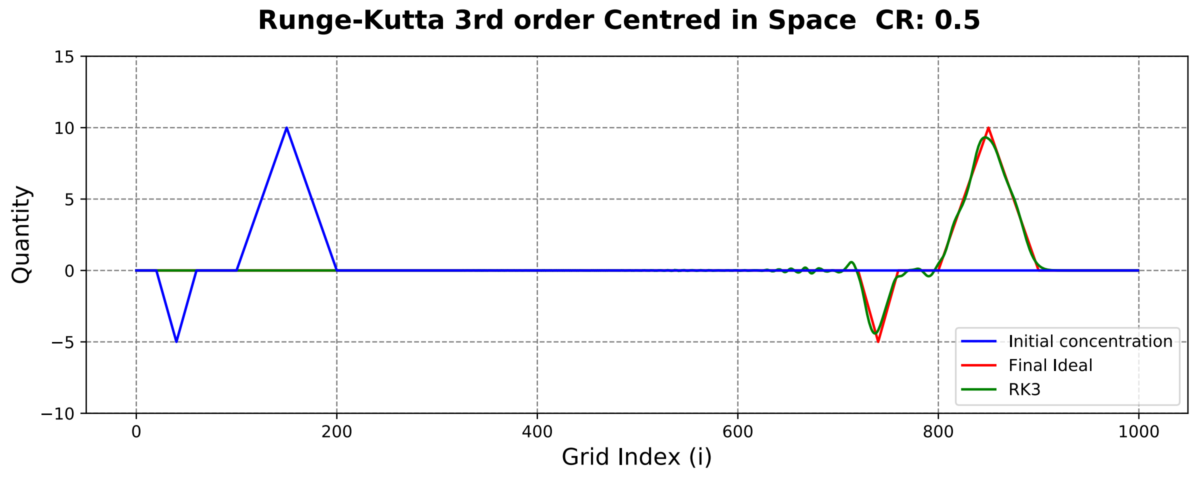
In [6]:
```
## Plot Forward in time backward in space
plot = coeff.plot_functions('FTBS')
```

```
(1400, 1000)
(1400, 1000)
```



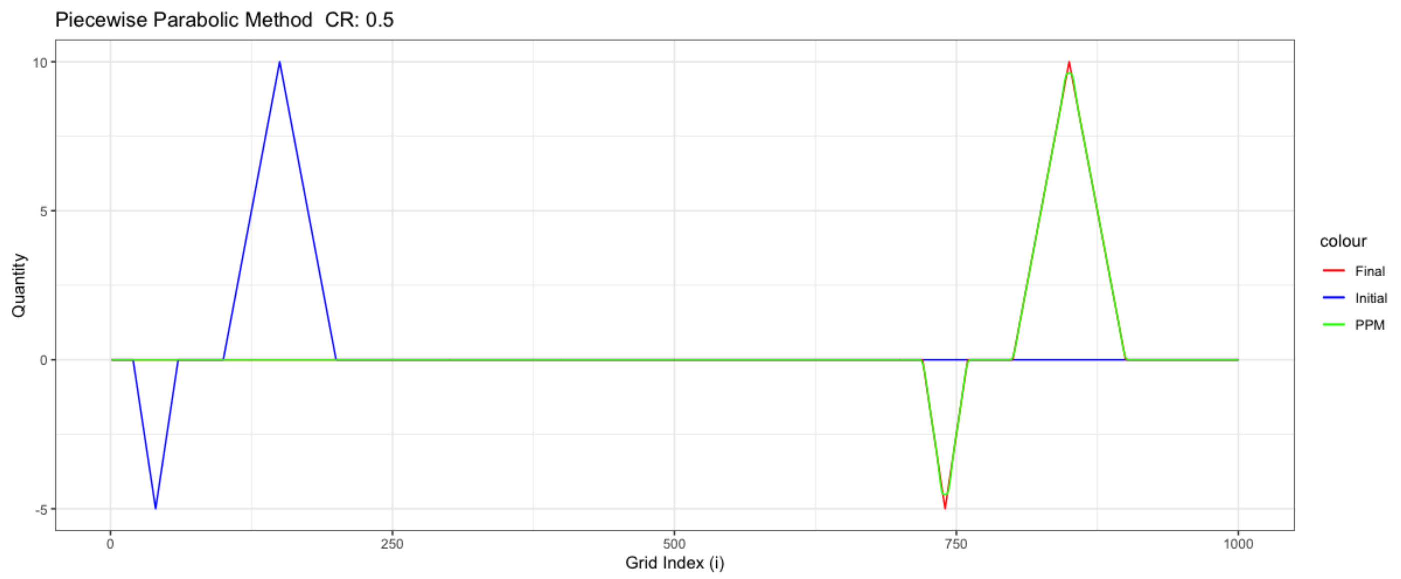
Plot RK3 centered in space solution

In [7]:
```
## Plot RK3 centered in space solution
plot = coeff.plot_functions('RK3')
```



Plot PPM centered in space solution

PNG made using hw7_ppm.R



- 7) Discuss and compare the results of these three advection schemes. Of the three schemes, the PPM performed best. The PPM scheme was meant to handle a sharp curve/gradient associated with pollutant advection. The forward in time centered in space performed the worst and had significant damping or reduction in the concentration levels. RK3 faired well but showed signs of instability around the sharp curves/gradients.

```python
import context
import numpy as np
from cr507.utils import plt_set
import matplotlib.pyplot as plt
from collections import namedtuple


class Approximator:

    #############################################
    # initialize condtions
    #############################################
    def __init__(self, valueDict):
        """
        Create the grid and initial conditions
        """
        ##  Defined conditions from dictonary
        self.__dict__.update(valueDict)

        ## Define number of time steps
        nsteps = (self.gridx - 300) / (self.u0 * self.dt / self.dx)
        nsteps = np.arange(0,nsteps)
        self.nsteps = nsteps

        ## Calculate the Courant number
        cr = self.u0 * self.dt / self.dx
        self.cr = cr

        ## Create initial concentration anomaly
        #  distribution in the x-direction
        conc = np.zeros(self.gridx)
        conc[100:151] = np.linspace(0.,self.cmax,51)
        conc[150:201] = np.linspace(self.cmax, 0.,51)
        conc[20:41] = np.linspace(0., -0.5 * self.cmax, 21)
        conc[40:61] = np.linspace(-0.5 * self.cmax, 0., 21)
        self.Pj = np.array(conc)

        ## Define the ideal exact final solution
        cideal = np.zeros(self.gridx)
        cideal[800:851] = np.linspace(0., self.cmax,51)
        cideal[850:901]  = np.linspace(self.cmax, 0., 51)
        cideal[720:741]  = np.linspace(0., -0.5 * self.cmax, 21)
        cideal[740:761]  = np.linspace(-0.5 * self.cmax, 0., 21)
        self.cideal = np.array(cideal)

    #############################################
    # spatial discretization methods
    #############################################
    def centdif(self):
        """
        Centered difference spatial approximation
        """
        # print(self.Pj[50],"centdif start")
        Pj = -self.u0 * ((np.roll(self.Pj,-1) - np.roll(self.Pj,1)) / (2 *
self.dx))

        # print(Pj[50],"centdif end")
        return Pj

    def backdif(self):
        """
        Backward difference spatial approximation
        """
        # print(self.Pj[50],"backdif start")
        Pj = -self.u0 * ((self.Pj - np.roll(self.Pj,1)) / (self.dx))

        # print(Pj[50],"backdif end")
        return Pj

    #############################################
    # time discretization methods
    #############################################
    def forward(self):
        Pj_OG = self.Pj

        Pjn_1 = []
        for n in range(len(self.nsteps)):
            Pj = self.Pj
            Pn = Pj + self.dt * self.backdif()
            self.Pj = Pn

            Pjn_1.append(Pn)

        Pjn_1 = np.array(Pjn_1)
        print(Pjn_1.shape)
        self.Pj = Pj_OG

        return Pjn_1



    def rk3(self):
        """
        Runge-Kutta 3rd order Centred in Space
        """
        Pj_OG = self.Pj

        Pjn_1 = []

        for n in range(len(self.nsteps)):

            Pj = self.Pj
            # print(Pj[50], "Pj var")
            P_str = Pj + (self.dt/3) * self.centdif()
            # print(P_str[50], 'P_str')

            self.Pj = P_str
            # print(self.Pj[50], 'self Pj should be Pjstr')

            P_str_str  = Pj + (self.dt/2) * self.centdif()
            # print(P_str_str[50], 'P_str_str')

            self.Pj = P_str_str
            # print(self.Pj[50], 'self Pj should be Pj_str_str')

            Pn  = Pj + self.dt * self.centdif()
            Pn = np.array(Pn)
            # print(Pn[50], "Pn pre append")
            Pjn_1.append(Pn)

```

```
                self.Pj = Pn
                # print(self.Pj[50], "self Pj or Pn")

        Pjn_1 = np.array(Pjn_1)
        self.Pj = Pj_OG

        return Pjn_1

    def plot_functions(self, method):
        if method == 'Initial':
            fig, ax = plt.subplots(1,1, figsize=(12,4))
            fig.suptitle('HW7 Initial concentration', \
                fontsize= plt_set.title_size, fontweight="bold")
            ax.plot(self.xx, self.Pj, color = 'blue', \
                 label = "Initial concentration", zorder = 9)
            ax.set_xlabel('Grid Index (i)', fontsize = plt_set.label)
            ax.set_ylabel('Quantity', fontsize = plt_set.label)
            ax.xaxis.grid(color='gray', linestyle='dashed')
            ax.yaxis.grid(color='gray', linestyle='dashed')
            ax.set_ylim(-10,15)
            ax.legend()
            plt.show()

        elif method == 'Final':
            fig, ax = plt.subplots(1,1, figsize=(12,4))
            fig.suptitle('HW7 Final Ideal', \
                fontsize= plt_set.title_size, fontweight="bold")
            ax.plot(self.xx, self.Pj, color = 'blue', \
                label = "Initial concentration", zorder = 9)
            ax.plot(self.xx,self.cideal, color = 'red', \
                label = "Final Ideal", zorder = 8)
            ax.set_xlabel('Grid Index (i)', fontsize = plt_set.label)
            ax.set_ylabel('Quantity', fontsize = plt_set.label)
            ax.xaxis.grid(color='gray', linestyle='dashed')
            ax.yaxis.grid(color='gray', linestyle='dashed')
            ax.set_ylim(-10,15)
            ax.legend()
            plt.show()

        elif method == 'RK3':
            fig, ax = plt.subplots(1,1, figsize=(12,4))
            fig.suptitle("Runge-Kutta 3rd order Centred in Space  CR: 0.5", \
                fontsize= plt_set.title_size, fontweight="bold")
            ax.plot(self.xx, self.Pj, color = 'blue', \
                 label = "Initial concentration", zorder = 10)
            ax.plot(self.xx,self.cideal, color = 'red', \
                 label = "Final Ideal", zorder = 8)
            Prk3 = self.rk3()
            ax.plot(self.xx,Prk3.T[:,-1], color = 'green', \
            label = "RK3", zorder = 9)
            ax.set_xlabel('Grid Index (i)', fontsize = plt_set.label)
            ax.set_ylabel('Quantity', fontsize = plt_set.label)
            ax.xaxis.grid(color='gray', linestyle='dashed')
            ax.yaxis.grid(color='gray', linestyle='dashed')
            ax.set_ylim(-10,15)
            ax.legend()
            plt.show()

        elif method == 'FTBS':
            fig, ax = plt.subplots(1,1, figsize=(12,4))
            fig.suptitle("Forward in time, Backward in space  CR: 0.5", \
                 fontsize= plt_set.title_size, fontweight="bold")
            ax.plot(self.xx, self.Pj, color = 'blue',  \
                label = "Initial concentration", zorder = 10)
            ax.plot(self.xx,self.cideal, color = 'red', \
                label = "Final Ideal", zorder = 8)
            Ftbs = self.forward()
            print(Ftbs.shape)
            ax.plot(self.xx,Ftbs.T[:,-1], color = 'green', \
                label = "FTBS", zorder = 9)
            ax.set_xlabel('Grid Index (i)', fontsize = plt_set.label)
            ax.set_ylabel('Quantity', fontsize = plt_set.label)
            ax.xaxis.grid(color='gray', linestyle='dashed')
            ax.yaxis.grid(color='gray', linestyle='dashed')
            ax.set_ylim(-10,15)
            ax.legend()
            plt.show()

        else:
            pass


        return
```

~~~
# install.packages("magrittr") # package installations are only needed the first time you use it
# install.packages("dplyr")    # alternative installation of the %>%
library(magrittr) # needs to be run every time you start R and want to use %>%
library(dplyr)    # alternatively, this also loads %>%
# Part 6 - PPM scheme advection

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
library(ggplot2)
library(tidyverse)
~~~

```{r}
# Create the grid and initial conditions
imax = 1000             # number of grid points in x-direction
delx = 100.             # horizontal grid spacing (m)
delt = 10.              # time increment (s)
u = 5.                  # horizontal wind speed (m/s)
```

```{r}
# Create initial concentration anomaly distribution in the x-direction
conc <- rep(0.0, imax)   # initial concentration of background is zero
cmax = 10.0                         # max initial concentration
conc[100:150] <- seq(0., cmax, len = 51)        # insert left side of triangle
conc[150:200] <- seq(cmax, 0., len = 51)        # insert right side of triangle
conc[20:40] <- seq(0., -0.5*cmax, len = 21)    # insert left side of triangle
conc[40:60] <- seq(-0.5*cmax, 0., len = 21)    # insert right side of triangle
conc_orig <- conc
```

```{r}
# create ideal solution
cideal <- rep(0.0, imax)   # initial concentration of ideal background is zero
cideal[800:850] <- seq(0., cmax, len = 51)    # insert left side of triangle
cideal[850:900] <- seq(cmax, 0., len = 51)    # insert right side of triangle
cideal[720:740] <- seq(0., -0.5*cmax, len = 21)    # insert left side of triangle
cideal[740:760] <- seq(-0.5*cmax, 0., len = 21)    # insert right side of triangle
```

```{r}
nsteps = (imax - 300) / (u * delt / delx)
xvals = seq(1,1000)
```

Plot 1
This has only the original concentration and the ideal solution

```
```

~~~
```{r}
plot(xvals, cideal, col = 'red', type = "l")
lines(xvals, conc, col = 'blue')
legend(400, 10, legend=c("cideal", "conc orig"),
       col=c("red", "blue"), lty=1, cex=0.8)
# plot <- ggplot() +
#   geom_line(aes(x = xvals, y = conc), color ="blue") +
#   geom_line(aes(x = xvals, y = cideal), color ="red") +
#   theme_bw() +
#   xlab("grid index (i)") +
#   ylab("quantity")+
#   ggtitle("PPM plot")
~~~
PPM scheme code

Here I use the code that was provided for the homework.

```{r}
#### USING THE CODE PROVIDED
# ================================
# 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
```

```r
    }                                   # 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
```

Make PPM plot
```{r}
df <- data.frame("Initial Concentration" = conc_orig,
                 "Final Ideal" = cideal,
                 "PPM" = conc,
                 "Grid Index" = xvals)
```

```{r}
df %>% ggplot(aes(x = Grid.Index)) +
  geom_line(aes(y = Initial.Concentration, color = "Initial")) +
  geom_line(aes(y = Final.Ideal, color ="Final")) +
  geom_line(aes(y = PPM, color ="PPM")) +
  theme_bw() +
  xlab("Grid Index (i)") +
  ylab("Quantity")+
  scale_color_manual(values = c("Initial" = 'blue',
                                "Final" = "red",
                                "PPM" = "green")) +
  ggtitle("Piecewise Parabolic Method  CR: 0.5")


```