```python
# Groundwater Modeling Coding Assignment #2
# Jim Finnegan
# 1D Transport Equation
# functions for FE, FD, and analytical solution

import numpy as np
from scipy.sparse import diags
from math import exp, sqrt
from scipy.special import erfc


def finite_element(d, r):
    """
    Computes matrix of C/C0, 0<x<200, 0<t<400 using Galerkin FEM
    @param d: float, hydrodynamic dispersion coefficient (m^2/d)
    @param r: float, retardation coefficient (unitless)
    @return C: array, matrix where rows are time steps and columns are C/C0(x)
    """
    # PARAMETERS
    # user inputs
    d = float(d)
    r = float(r)
    # other parameters
    v, L, dx, t, dt = 0.5, 200, 2, 400, 10
    # matrix dimensions
    rows = int(t / dt) + 1
    n_el = int(L / dx)
    cols = n_el + 1
    # initial conditions
    C = np.zeros((rows, cols))
    C[:, 0] = 1      # boundary condition: C/C0 = 1 at x=0

    # CONSTRUCT STIFFNESS AND STORAGE MATRICES
    # element matrices
    alpha = (r * dx) / 6
    lam_1 = d / dx
    lam_2 = v / 2
    Ae = [[lam_1 - lam_2, -lam_1 + lam_2], [-lam_1 - lam_2, lam_1 + lam_2]]     # element stiffness matrix
    Be = [[2 * alpha, alpha], [alpha, 2 * alpha]]                               # element storage matrix
    # global matrices
    A = np.zeros((cols, cols))
    B = np.zeros((cols, cols))
    for i in range(1, cols):
        A[i, i] += Ae[1][1]             # assemble Ae elements
        A[i, i - 1] += Ae[1][0]
        A[i - 1, i] += Ae[0][1]
        A[i - 1, i - 1] += Ae[0][0]
        B[i, i] += Be[1][1]             # assemble Be elements
        B[i, i - 1] += Be[1][0]
        B[i - 1, i] += Be[0][1]
        B[i - 1, i - 1] += Be[0][0]
    LH = (A / 2 + B / dt)
    RH = (-A / 2 + B / dt)

    # TIME STEPPING
    for k in range(1, rows):
        b_f = np.dot(RH, C[k - 1, :])                       # solve RHS
```

```
        b_f[0] = LH[0][0] + LH[0][1] * C[k - 1][1]      # boundary condition
        C[k, :] = np.linalg.solve(LH, b_f)              # solve LHS
    return C


def finite_difference(d, r):
    """
    Computes matrix of C/C0, 0<x<200, 0<t<400 using Crank-Nicholson FDM
    @param d: float, hydrodynamic dispersion coefficient (m^2/d)
    @param r: float, retardation coefficient (unitless)
    @return C: array, matrix where rows are time steps and columns are C/C0(x)
    """
    d = float(d)
    r = float(r)

    # other parameters
    v, L, dx, t, dt = 0.5, 200, 2, 400, 10
    # matrix dimensions
    rows = int(t / dt) + 1
    cols = int(L / dx) + 1

    # initial conditions
    C = np.zeros((rows, cols))
    C[:, 0] = 1  # boundary condition: C/C0 = 1 at x=0

    # simplified variables from central difference derivation
    G = (d * dt) / (2 * r * dx ** 2)
    H = (v * dt) / (4 * r * dx)
    lam_1, lam_2, lam_3, lam_4 = G + H, 2 * G + 1, G - H, 2 * G - 1

    # CENTERED DIFFERENCE SCHEME
    #   Left hand side - k+1
    A_diagonals = [np.ones(cols - 1) * lam_1, np.ones(cols) * -lam_2, np.ones(cols - 1) * lam_3]
    A = diags(A_diagonals, offsets=[-1, 0, 1], shape=(cols, cols)).toarray()
    #   Right hand side - k
    B_diagonals = [np.ones(cols - 1) * -lam_1, np.ones(cols) * lam_4, np.ones(cols - 1) * -lam_3]
    B = diags(B_diagonals, offsets=[-1, 0, 1], shape=(cols, cols)).toarray()

    for k in range(1, rows):
        b = np.dot(B, C[k - 1, :])          # solve RHS
        b[0] = -(1 + lam_1)                 # boundary condition
        C[k, :] = np.linalg.solve(A, b)     # solve LHS

    return C


def analytical(d):
    """
    @param d: float, hydrodynamic dispersion coefficient (m^2/d)
    @return C: array, matrix where rows are time steps and columns are C/C0(x)
    """
    # initial conditions
    # for R = 1
    v = 0.1
    d = float(d)
    L, dx = 200, 2
```

```
    dist = np.linspace(2, L, num=int(L / dx))
    dist = [int(x) for x in dist]
    t, dt = 400, 10
    time = np.linspace(10, t, num=int(t / dt))
    time = [int(t) for t in time]

    # initialize grid for C
    # x is distance (one column is 2 ft), y is time (one row is 10 days)
    C = np.zeros((len(time) + 1, len(dist) + 1))
    C[:, 0] = 1  # boundary condition: C/C0 = 1 at x=0

    # calculate C using analytical solution for all x>0
    for x in range(len(dist)):
        for t in range(len(time)):
            C[t + 1][x + 1] = (1 / 2) * (exp(v * dist[x] / d) * erfc((dist[x] + v * time[t]) / (2 * sqrt(d * time[t])))
                                         + erfc((dist[x] - v * time[t]) / (2 * sqrt(d * time[t]))))

    return C


def analytical_vfive(d):
    """
    @param d: float, hydrodynamic dispersion coefficient (m^2/d)
    @return C: array, matrix where rows are time steps and columns are C/C0(x)
    """
    v = 0.5
    D = float(d)
    L, dx = 200, 2
    dist = np.linspace(2, L, num=int(L / dx))
    dist = [int(x) for x in dist]
    t = 200

    # calculate C using analytical solution for all x>0
    C = np.zeros(len(dist) + 1)
    for x in range(len(dist)):
        try:
            C[x] = (1 / 2) * (exp(v * dist[x] / D) * erfc((dist[x] + v * t) / (2 * sqrt(D * t))) + erfc(
                (dist[x] - v * t) / (2 * sqrt(D * t))))
        except OverflowError:
            C[x] = 0  # set C = 0 if math overflow error

    return C
```