# Geology 575 Homework 5

Chris McKinney

## Contents



## Setup

### Parameters

| Parameter | Units | Meaning |
|---|---|---|
| $c_a$ | mg/L | $c(x,0) = c_a,\ 0 \leq x \leq L$ |
| $c_b(1)$ | mg/L | $c(0,t) = c_b(1),\ 0 < t \leq t_p$ |
| $c_c(1)$ | mg/L | $c(L,t) = c_c(1),\ 0 < t \leq t_p$ |
| $L$ | meters | Length to model |
| $v$ | m/day | Average linear flow velocity |
| $D$ | $m^2$/d | Dispersion coefficient |
| $t_p$ | days | Length of the plume; forever if $< 0$ |
| $\Delta x$ | meters | Spatial discretization |
| $\Delta t$ | days | Temporal discretization |
| $\theta$ | | Model implicitness |
| $T_L$ | | Number of elements in $T$ |
| $T$ | days | List of times at which to output |

### Fitting Length to Discretization

$\ell$ is the number of "active" nodes (total nodes - boundary nodes).

$$\ell = (\text{ieee\_round } \frac{L}{\Delta x}) - 1$$

### Coefficients

$$
\begin{aligned}
P &= \frac{D\Delta t}{(\Delta x)^2} & R &= \frac{v\Delta t}{2\Delta t} \\
A^* &= -\theta(P+R) & A^{**} &= (1-\theta)(P+R) \\
B^* &= 1+2\theta P & B^{**} &= 1+2(\theta-1)P \\
C^* &= -\theta(P-R) & C^{**} &= (1-\theta)(P-R)
\end{aligned}
$$

### Boundary Conditions

$$
c_b(t) = \begin{cases} 0, & t_p \geq 0 \wedge t > t_p \\ c_b(1), & t > 0 \\ c_a, & t = 0 \end{cases}
$$

$$
c_c(t) = \begin{cases} 0, & t_p \geq 0 \wedge t > t_p \\ c_c(1), & t > 0 \\ c_a, & t = 0 \end{cases}
$$

## General Finite Differences

The 1-D advection-dispersion equation is

$$
\frac{\partial c}{\partial t} = D\frac{\partial^2 c}{\partial x^2} - v\frac{\partial c}{\partial x}
$$

where $D$ is the dispersion coefficient, and $v$ is the average linear flow velocity.

The general equation for finite difference is

$$
\begin{aligned}
\frac{c_i(t+\Delta t) - c_i}{\Delta t} = \theta &\left[ D\frac{c_{i+1}(t+\Delta t) - 2c_i(t+\Delta t) + c_{i-1}(t+\Delta t)}{(\Delta x)^2} - v\frac{c_{i+1}(t+\Delta t) - c_{i-1}(t+\Delta t)}{2\Delta x} \right] \\
&+ (1-\theta)\left[ D\frac{c_{i+1}(t) - 2c_i(t) + c_{i-1}(t)}{(\Delta x)^2} - v\frac{c_{i+1}(t) - c_{i-1}(t)}{2\Delta x} \right]
\end{aligned}
$$

Rearranged,

$$
A^* c_{i-1}(t+\Delta t) + B^* c_i(t+\Delta t) + C^* c_{i+1}(t+\Delta t) = A^{**} c_{i-1}(t) + B^{**} c_i(t) + C^{**} c_{i+1}(t)
$$

Converting to matrices,

$$
\begin{bmatrix}
1 & & & & & & \\
A^* & B^* & C^* & & & & \\
 & A^* & B^* & C^* & & & \\
 & & \ddots & \ddots & \ddots & & \\
 & & & A^* & B^* & C^* & \\
 & & & & A^* & B^* & C^* \\
 & & & & & & 1
\end{bmatrix}
\begin{bmatrix}
c_0(t+\Delta t) \\ c_1(t+\Delta t) \\ \vdots \\ \vdots \\ \vdots \\ c_\ell(t+\Delta t) \\ c_{\ell+1}(t+\Delta t)
\end{bmatrix}
=
\begin{bmatrix}
g_0(t) \\ g_1(t) \\ \vdots \\ \vdots \\ \vdots \\ g_\ell(t) \\ g_{\ell+1}(t)
\end{bmatrix}
$$

where $g_i(t) = A^{**}c_{i-1}(t) + B^{**}c_i(t) + C^{**}c_{i+1}(t)$ for $1 \leq g \leq \ell$ and $g_0(t)$ and $g_{\ell+1}(t)$ are boundary conditions.

Removing the borders because they are known,

$$\begin{bmatrix} B^* & C^* & & & \\ A^* & B^* & C^* & & \\ & \ddots & \ddots & \ddots & \\ & & A^* & B^* & C^* \\ & & & A^* & B^* \end{bmatrix} \begin{bmatrix} c_1(t+\Delta t) \\ c_2(t+\Delta t) \\ \vdots \\ c_{\ell-1}(t+\Delta t) \\ c_\ell(t+\Delta t) \end{bmatrix} = \begin{bmatrix} g_1(t) - A^* c_0(t+\Delta t) \\ g_2(t) \\ \vdots \\ g_{\ell-1}(t) \\ g_\ell(t) - C^* c_{\ell+1}(t+\Delta t) \end{bmatrix}$$

### Suitability of the Thomas Algorithm

The LAPACK subroutine `dgtsv` uses Gaussian elimination since the Thomas algorithm is conditionally stable. Gaussian elimination, however, is $O(n^3)$, whereas the Thomas algorithm is $O(n)$. The Thomas algorithm is stable when the matrix is diagonally dominant.

$$\begin{aligned} |B^*| &\geq |A^* + C^*| \\ 1 + 2\theta P &\geq 2\theta P \\ 1 &\geq 0 \quad \checkmark \end{aligned}$$

Thus the Thomas algorithm is stable for this problem.

## The Thomas Algorithm

$$C'_i(t) = \begin{cases} \frac{C^*}{B^*}, & i = 1 \\ \frac{C^*}{B^* - A^* C'_{i-1}(t)}, & 2 \leq i \leq \ell - 1 \end{cases}$$

$$G'_i(t) = \begin{cases} \frac{g_1(t) - A^* c_b(t+\Delta t)}{B^*}, & i = 1 \\ \frac{g_i(t) - A^* G'_{i-1}(t)}{B^* - A^* C'_{i-1}(t)}, & 2 \leq i \leq \ell - 1 \end{cases}$$

$$c_i(0) = c_a$$

$$c_i(t+\Delta t) = \begin{cases} c_b(t+\Delta t), & i = 0 \\ c_c(t+\Delta t), & i = \ell + 1 \\ \frac{g_\ell(t) - C^* c_c(t+\Delta t) - A^* G'_{\ell-1}(t)}{B^* - A^* C'_{\ell-1}(t)}, & i = \ell \\ G'_i(t) - C'_i(t) c_{i+1}(t+\Delta t), & ell - 1 \geq i \geq 1 \end{cases}$$

## Code

This is basically just a transcription of the math above into Haskell with memoization to speed up the recursive functions.

```haskell
-- Implementation of the math on the board

-- Output --
putHeader :: Int -> IO ()
putConcentrations :: Int -> Int -> IO ()
putAllConcentrations :: Int -> IO()
```

```
putHeader i
    | i == -1 = do
        putStr ((show 0.0) ++ " ")
        putHeader 0
    | i == ell + 1 = do
        putStrLn (show (dx * (fromIntegral i)))
        return ()
    | otherwise = do
        putStr ((show (dx * (fromIntegral i))) ++ " ")
        putHeader (i + 1)
putConcentrations ti i
    | i == -1 = do
        putStr ((show (dt * (fromIntegral ti))) ++ " ")
        putConcentrations ti 0
    | i == ell + 1 = do
        putStrLn (show (c ti i))
        return ()
    | otherwise = do
        putStr ((show (c ti i)) ++ " ")
        putConcentrations ti (i + 1)
putAllConcentrations i
    | i == -1 = do
        putHeader (-1)
        putAllConcentrations 0
    | i == t_len = return ()
    | otherwise = do
        putConcentrations (round ((t_list !! i) / dt)) (-1)
        putAllConcentrations (i + 1)
main = putAllConcentrations (-1)

-- Parameters --
c_a :: Double -- Initial condition
c_b_1 :: Double -- Left boundary
c_c_1 :: Double -- Right boundary
len :: Double -- Length to model
v :: Double -- Average linear velocity
dispers :: Double -- Dispersion coefficient
t_p :: Double -- Plume duration
dx :: Double -- Spatial discretization
dt :: Double -- Temporal discretization
theta :: Double -- Implicitness
t_len :: Int -- Number of output times
t_list :: [Double] -- Output times

#ifndef PARAM_CA
#define PARAM_CA (0.0)
#endif
c_a = PARAM_CA
#ifndef PARAM_CB
#define PARAM_CB (100.0)
#endif
c_b_1 = PARAM_CB
#ifndef PARAM_CC
```

```
#define PARAM_CC (0.0)
#endif
c_c_1 = PARAM_CC
#ifndef PARAM_L
#define PARAM_L (50.0)
#endif
len = PARAM_L
#ifndef PARAM_V
#define PARAM_V (5.0)
#endif
v = PARAM_V
#ifndef PARAM_D
#define PARAM_D (8.0)
#endif
dispers = PARAM_D
#ifndef PARAM_TP
#define PARAM_TP (-1.0)
#endif
t_p = PARAM_TP
#ifndef PARAM_DX
#define PARAM_DX (1.0)
#endif
dx = PARAM_DX
#ifndef PARAM_DT
#define PARAM_DT (0.05)
#endif
dt = PARAM_DT
#ifndef PARAM_THETA
#define PARAM_THETA (0.5)
#endif
theta = PARAM_THETA
#ifndef PARAM_TL
#define PARAM_TL (0)
#endif
t_len = PARAM_TL
#ifndef PARAM_T_LIST
#define PARAM_T_LIST ([])
#endif
t_list = PARAM_T_LIST

-- Discretization --
ell :: Int
ell = (round (len / dx)) - 1

-- Coefficients --
p :: Double
r :: Double
a_star :: Double
a_2star :: Double
b_star :: Double
b_2star :: Double
c_star :: Double
c_2star :: Double
```

```
p = (dispers*dt) / (dx*dx)
r = (v*dt) / (2*dx)
a_star = -theta * (p+r)
a_2star = (1-theta) * (p+r)
b_star = 1 + 2*theta*p
b_2star = 1 + 2*(theta-1)*p
c_star = -theta * (p-r)
c_2star = (1-theta) * (p-r)

-- Boundary Conditions --
c_b :: Int -> Double
c_c :: Int -> Double
c_b ti
    | t_p >= 0 && dt*(fromIntegral ti) > t_p = 0
    | ti > 0 = c_b_1
    | ti == 0 = c_a
c_c ti
    | t_p >= 0 && dt*(fromIntegral ti) > t_p = 0
    | ti > 0 = c_c_1
    | ti == 0 = c_a

-- Memoization --
memo :: (Int -> Int -> a) -> [[a]]
memo f = map (\x -> map (f x) [0..]) [0..]

-- Forward propagation --
c_prime_slow :: Int -> Int -> Double
c_prime_store :: [[Double]]
c_prime :: Int -> Int -> Double
g_prime_slow :: Int -> Int -> Double
g_prime_store :: [[Double]]
g_prime :: Int -> Int -> Double
g :: Int -> Int -> Double
primed :: Int -> Int -> Double

c_prime_slow ti i
    | i == 1 = c_star / b_star
    | 2 <= i && i <= ell - 1 = c_star / (primed ti i)
c_prime_store = memo c_prime_slow
c_prime ti i = c_prime_store !! ti !! i
g_prime_slow ti i
    | i == 1 = ((g ti 1) - a_star*(c_b (ti+1))) / b_star
    | 2 <= i && i <= ell - 1
        = ((g ti i) - a_star*(g_prime ti (i-1))) / (primed ti i)
g_prime_store = memo g_prime_slow
g_prime ti i = g_prime_store !! ti !! i
g ti i = a_2star*(c ti (i-1)) + b_2star*(c ti i) + c_2star*(c ti (i+1))
primed ti i = b_star - a_star*(c_prime ti (i-1))

-- Concentrations --
c_slow :: Int -> Int -> Double
c_slow_positive :: Int -> Int -> Double
c_store :: [[Double]]
c :: Int -> Int -> Double
```

```
c_slow ti i
    | ti == 0 && 0 <= i && i <= ell + 1 = c_a
    | ti > 0 && i == ell
        = ((g (ti-1) ell) - c_star*(c_c ti) - a_star*(g_prime (ti-1) (ell-1)))
            / (primed (ti-1) i)
    | ti > 0 && ell - 1 >= i && i >= 1
        = (g_prime (ti-1) i) - (c_prime (ti-1) i)*(c ti (i+1))
    | i == 0 = c_b ti
    | i == ell + 1 = c_c ti
c_slow_positive ti i = (max 0.0 (c_slow ti i)) -- Negative c is senseless
c_store = memo c_slow_positive
c ti i = c_store !! ti !! i
```

# Plots

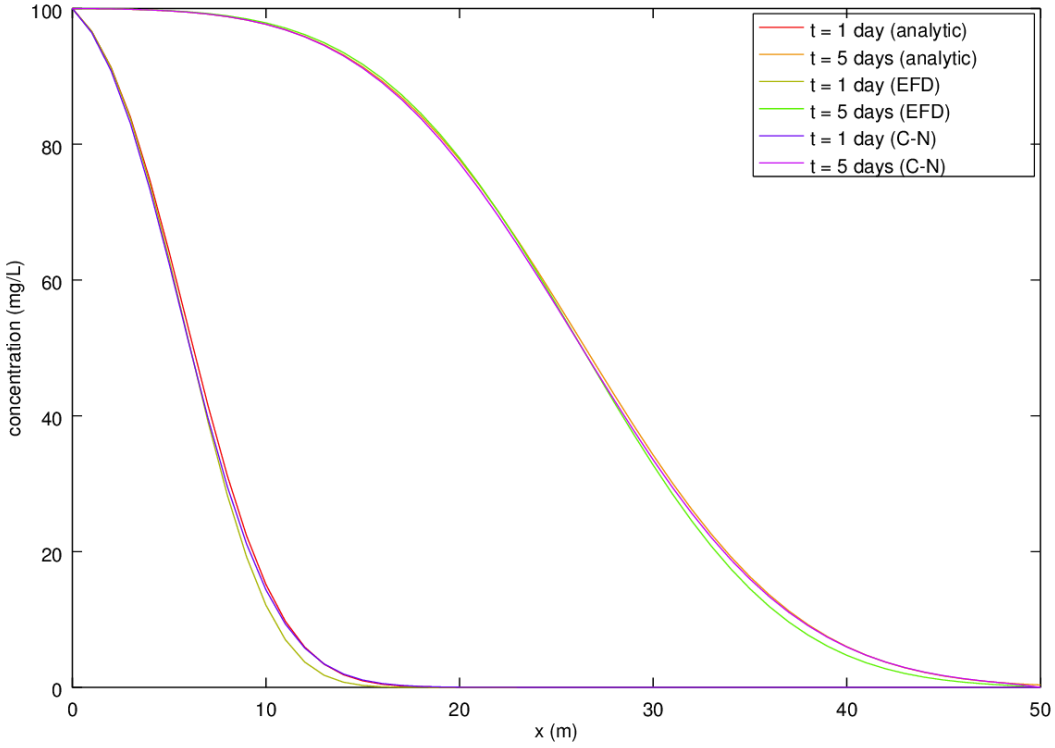


Figure 1: Analytic, EFD, and Crank-Nicolson with $D = 8.0\text{m}^2/\text{day}$

## Crank-Nicolson Data

```
(1)  00.00 00.00 01.00 02.00 03.00 04.00 05.00 06.00 07.00 08.00 09.00 10.00 11.00 ...
(2)  01.00 100.0 96.44 90.82 83.06 73.39 62.40 50.89 39.72 29.63 21.11 14.36 09.34 ...
(3)  05.00 100.0 99.98 99.94 99.88 99.78 99.65 99.45 99.18 98.80 98.31 97.67 96.85 ...
```

```
(1)  12.00 13.00 14.00 15.00 16.00 17.00 18.00 19.00 20.00 21.00 22.00 23.00 24.00 ...
(2)  05.80 03.45 01.97 01.08 00.57 00.29 00.14 00.07 00.03 00.01 00.01 00.00 00.00 ...
(3)  95.82 94.56 93.03 91.21 89.08 86.61 83.81 80.68 77.22 73.45 69.42 65.16 60.71 ...

(1)  25.00 26.00 27.00 28.00 29.00 30.00 31.00 32.00 33.00 34.00 35.00 36.00 37.00 ...
(2)  00.00 00.00 00.00 00.00 00.00 00.00 00.00 00.00 00.00 00.00 00.00 00.00 00.00 ...
(3)  56.14 51.50 46.86 42.28 37.81 33.52 29.44 25.62 22.09 18.86 15.95 13.36 11.07 ...

(1)  38.00 39.00 40.00 41.00 42.00 43.00 44.00 45.00 46.00 47.00 48.00 49.00 50.00
(2)  00.00 00.00 00.00 00.00 00.00 00.00 00.00 00.00 00.00 00.00 00.00 00.00 00.00
(3)  09.09 07.39 05.95 04.74 03.73 02.92 02.25 01.72 01.30 00.96 00.67 00.39 00.00
```

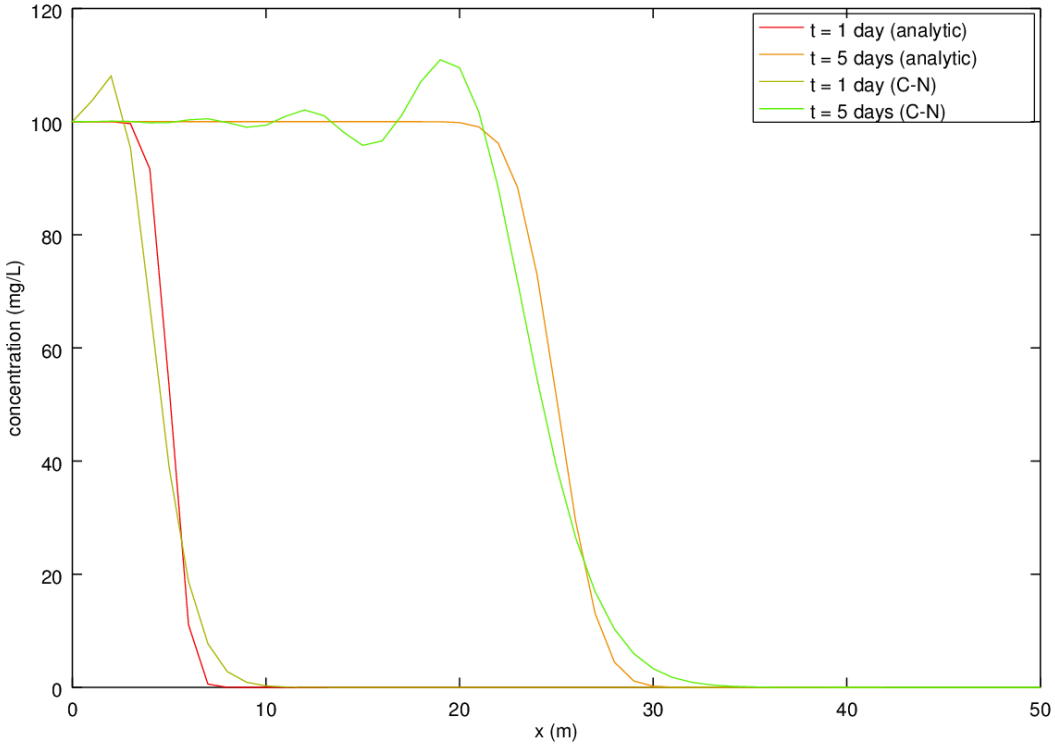


Figure 2: Analytic and Crank-Nicolson with $D = 0.3\text{m}^2/\text{day}$

This model has a Péclet number of

$$\text{Pe} = \frac{v\Delta x}{D} = \frac{(5\text{ m/day})(1\text{ m})}{0.3\text{ m}^2/\text{day}} = 16.67$$

This is far above the acceptable value, and as such, there are large oscillations.

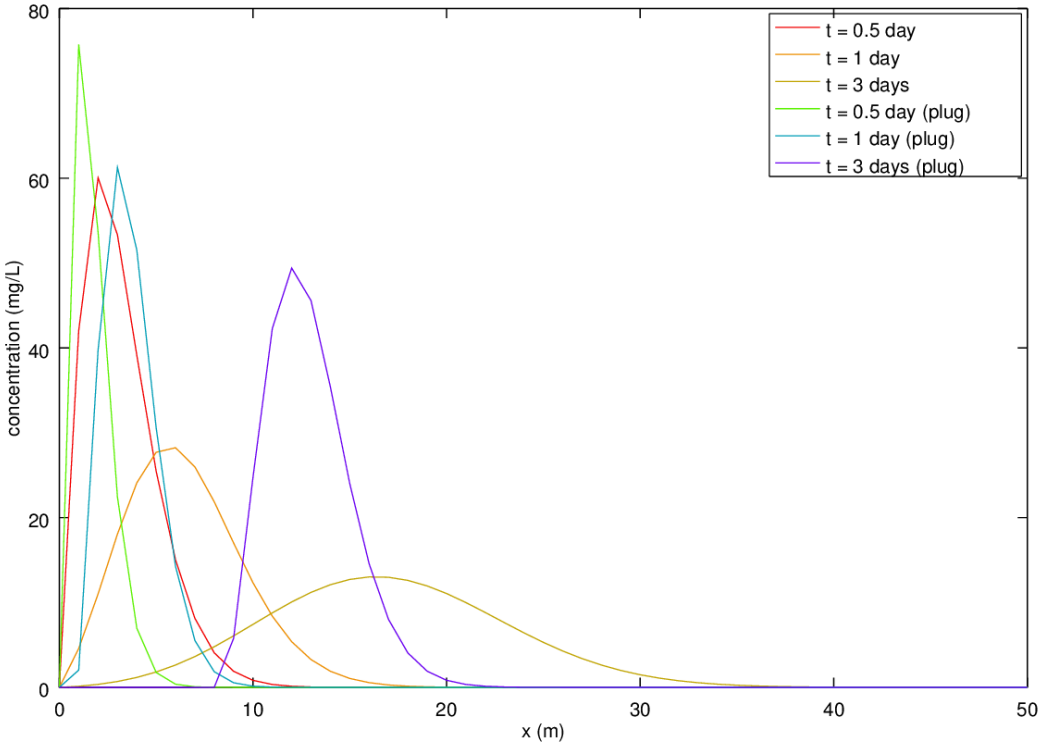


Figure 3: Crank-Nicolson Plume, $D = 8.0\text{m}^2/\text{day}$ & plug flow