Diagenetic Self-Organization: an exercise in replication

C. Summers & C. Thieulot

May 21, 2024

Context

In the spring of 2024 I was approached by Emilia with regards to an article she and her team wished to reproduce. Although recently published, and written by a renowned computational geoscientist, this article did not include link to a code. The article in question is [LHe18] Can we reproduce the results of the paper using only the equations provided in the paper?

The PDEs of the physical problem

The whole geological context is a bit lost on me so I/we approached the project with a pragmatic attitude: We must solve 5 coupled nonlinear advection-diffusion-reaction PDEs in a 1D domain.

The model describes the evolution of the concentrations of five variables:

- Solids: two minerals which dissolve and (re)precipitate from/to the same solutes. They are expressed as proportions of the solid phase, and must accordingly be non-negative, and their sum must be smaller or equal to one. C_C is the proportion of calcite in the solid phase, and C_A is the proportion of aragonite in the solid phase.
- Solutes $\hat{c}_k$, where $k \in \{Ca, CO3\}$. with $\hat{c}_{Ca}$: calcium concentration in pore water, and $\hat{c}_{CO3}$: carbonate concentration in pore water.
- Porosity ϕ

The equations involving our five unknowns C_A, C_C, $\hat{c}_k$ and ϕ are as follows (in order to highlight how nonlinear and coupled these equations are, I have colored the five unknowns throughout):

$$\begin{aligned}
\frac{\partial C_A}{\partial t} &= -U(\phi)\frac{\partial C_A}{\partial x} - Da[(1-C_A)C_A(\Omega_{DA}-\nu_1\Omega_{PA}) + \lambda C_A C_C(\Omega_{PC}-\nu_2\Omega_{DC})] \\
\frac{\partial C_C}{\partial t} &= -U(\phi)\frac{\partial C_C}{\partial x} + Da[(1-C_C)C_C(\Omega_{PC}-\nu_2\Omega_{DC}) + \lambda C_A C_C(\Omega_{DA}-\nu_1\Omega_{PA})] \\
\frac{\partial \hat{c}_k}{\partial t} &= -W(\phi)\frac{\partial \hat{c}_k}{\partial x} + \frac{1}{\phi}\frac{\partial}{\partial x}\left(\phi d_k \frac{\partial \hat{c}_k}{\partial x}\right) + Da\frac{1-\phi}{\phi}(\delta - \hat{c}_k)[C_A(\Omega_{DA}-\nu_1\Omega_{PA}) - \lambda C_C(\Omega_{PC}-\nu_2\Omega_{DC})] \\
\frac{\partial \phi}{\partial t} &= -\frac{\partial}{\partial x}(W(\phi)\phi) + d_\phi \frac{\partial^2\phi}{\partial x^2} + Da(1-\phi)[C_A(\Omega_{DA}-\nu_1\Omega_{PA}) - \lambda C_C(\Omega_{PC}-\nu_2\Omega_{DC})]
\end{aligned}$$

$k=1$ corresponds to $\hat{c}_{CA}$, and and $k=2$ corresponds to $\hat{c}_{C03}$.

We assume that the coordinate $x=0$ corresponds to the top of the domain which is of length L.

The PDEs above are actually dimensionless (the primes have been dropped), with

$$x' = x/X^\star, \quad t' = t/T^\star \quad \hat{c}'_k = \hat{c}_k/\sqrt{K_C}, \quad U = u/S, \quad W = w/S.$$

and with

$$Da = k_2 T^\star = \frac{k_2 D^0_{Ca}}{S^2}, \quad \lambda = k_3/k_2, \quad \nu_1 = k_1/k_2, \quad \nu_2 = k_4/k_3, \quad d_k = D_k/D^0_{Ca}, \quad d_\phi = D_\phi/D^0_{Ca}, \quad \delta = \frac{\rho_s}{\mu_A\sqrt{K_C}}.$$

Da can be interpreted as a Damköhler number[1].

[1] https://en.wikipedia.org/wiki/Damkohler_numbers

Hydraulic conductivity

$$K(\phi) = \beta\frac{\phi^3}{(1-\phi)^2}F(\phi) = \beta\frac{\phi^3}{(1-\phi)^2}\left[1-\exp\left(-\frac{10(1-\phi)}{\phi}\right)\right]$$

which translated into the following code snippet:

```
def K(self,phi):
    return self.beta*(phi**3/(1-phi)**2)*(1-np.exp(-10*(1-phi)/phi))
```

Since the porosity ϕ is between 0 and 1 and $K(\phi)$ contains terms which could become infinite within this range, we could ask ourselves:

$$\lim_{\phi\to 0} K(\phi) = ? \qquad \lim_{\phi\to 1} K(\phi) = ?$$

Let us recall the Taylor expansion of the exponential function in zero:

$$\exp(x) \simeq 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \dots$$

When $\phi \to 1$, then $(1-\phi)/\phi \to 0$ and then

$$1-\exp\left(-\frac{10(1-\phi)}{\phi}\right) \sim 1-(1-\frac{10(1-\phi)}{\phi}) = \frac{10(1-\phi)}{\phi}$$

so that

$$\lim_{\phi\to 1} K(\phi) = \beta\frac{\phi^3}{(1-\phi)^2}\cdot\frac{10(1-\phi)}{\phi} = 10\beta\frac{\phi^2}{1-\phi}$$

which is Eq. (16) of the paper.

Also, since When $\phi \to 0$, then $(1-\phi)/\phi \to +\infty$ and then

$$1-\exp\left(-\frac{10(1-\phi)}{\phi}\right) \sim 1-\exp{-\infty} = 1$$

and finally

$$\lim_{\phi\to 0} K(\phi) = 0$$

Velocities

W is the velocity of the pore water

The velocities U and W are given by

$$U(\phi) = 1-\frac{K(\phi^0)}{S}(1-\phi^0)(\frac{\rho_s^0}{\rho_w}-1)+\frac{K(\phi)}{S}(1-\phi)(1-\phi^0)(\frac{\rho_s}{\rho_w}-1) \quad (1)$$

$$W(\phi) = 1-\frac{K(\phi^0)}{S}(1-\phi^0)(\frac{\rho_s^0}{\rho_w}-1)-\frac{K(\phi)}{S}\frac{(1-\phi)^2}{\phi}(1-\phi^0)(\frac{\rho_s}{\rho_w}-1) \quad (2)$$

These translate into

```
def U(self, phi):
    u = 1 - ( 1 / self.sed_rate )*\
          ( self.K(self.phi_0) * ( 1 - self.phi_0 ) - self.K(phi) * ( 1 - phi ) )*\
          ( self.rho_s0 / self.rho_w - 1 )
    return u

def W(self,phi):
    w = 1-( 1 / self.sed_rate )*\
         ( self.K(self.phi_0) * ( 1 - self.phi_0 ) + self.K(phi)*(1-phi)**2/phi)*\
         ( self.rho_s0 / self.rho_w - 1 )
    return w
```

The U velocity contains the term $K(\phi)(1-\phi)$ and the W contains the term $K(\phi)(1-\phi)^2/\phi$. We find

$$\begin{aligned}
\lim_{\phi\to 0} K(\phi)(1-\phi) &= 0 \\
\lim_{\phi\to 1} K(\phi)(1-\phi) &= 10\beta\frac{\phi^2}{1-\phi}(1-\phi) = 10\beta \\
\lim_{\phi\to 0} K(\phi)(1-\phi)^2/\phi &= \beta\frac{\phi^3}{(1-\phi)^2}(1-\phi)^2/\phi = \beta\phi^2 = 0 \\
\lim_{\phi\to 1} K(\phi)(1-\phi)^2/\phi &= 10\beta\frac{\phi^2}{1-\phi}(1-\phi)^2/\phi = 0
\end{aligned}$$

Porosity diffusion coefficient

d_ϕ is the dimensionless form, calculated as $d_\phi = D_\phi / D^0_{Ca}$. In the article and in our calculations, it is set to be constant given by:

```
self.d_phi = self.beta*(self.phi_init**3 / ( 1 - self.phi_init ) )*\
             (1 / ( self.b*self.g*self.rho_w*( self.phi_NR - self.phi_inf ) ) )*\
             (1 - np.exp( -10*( 1 - self.phi_init ) / self.phi_init) )*( 1 / self.D_Ca_0 )
```

k components diffusion coefficient

$D_k(\phi)$ the (tortuosity-corrected) diffusion coefficient of component k. In Eq. 6 of the paper we find

$$D_k = \frac{D^0_k}{1 - \log \phi^2}$$

where D^0_k is the diffusion of component k in seawater.

```
def d_c_ca(self, phi):
        # scaled with D_Ca_0
        return 1.0/( 1 - 2*np.log(phi) )

def d_c_co(self, phi):
        # scaled with D_Ca_0
        return self.D_CO/self.D_Ca_0*(1 / ( 1 - 2*np.log(phi) ) )
```

Saturation factors

Ω_{PA} and Ω_{DA} are saturation factors for precipitation and dissolution of aragonite. Ω_{PC} and Ω_{DC} are saturation factors for precipitation and dissolution of calcite.

The under- and oversaturation factors

$$\Omega_{PA} = \left(\frac{\hat{c}_{CA}\hat{c}_{CO3}K_C}{K_A} - 1 \right)^{m'}$$

where m' is a reaction order (different from m in general). It is understood that this reaction is only effective when the system is oversaturated, i.e. $\hat{c}_{CA}\hat{c}_{CO3}K_C/K_A - 1 > 0$.

$$\Omega_{DA} = \left(1 - \frac{\hat{c}_{CA}\hat{c}_{CO3}K_C}{K_A} \right)^{m} \theta(x)$$

where K_A is the solubility of aragonite and m is a reaction order. It is understood that this reaction occurs only when the system is undersaturated, i.e. $1 - \hat{c}_{CA}\hat{c}_{CO3}/K_A > 0$. Finally, the factor $\theta(x)$ is a characteristic function that is zero when x is outside the aragonite dissolution zone (ADZ) and one otherwise. The ADZ is characterized by the position of its top edge x_d and its thickness h_d.

Note that there is a confusion about m and m' in eqs 27 and 45 of the paper, but in practice we set $m = m'$.

Looking at the PDEs we find that Ω_{DA} and Ω_{PA} always occur together so we define

$$\Omega_A = \Omega_{DA} - \nu_1 \Omega_{PA}$$

Likewise we define

$$\Omega_C = \Omega_{PC} - \nu_2 \Omega_{DC}$$

leading to write

```
def Omega_A(self, c_ca, c_co, x):
    sp = c_ca*c_co*self.K_C/self.K_A - 1
    Omega_PA = (max(0.0,sp))**self.m
    sa = 1 - c_ca*c_co*self.K_C/self.K_A
    Omega_DA = (max(0.0,sa))**self.m * heaviside(x,self.ADZ_bot,self.ADZ_top,self.x_scale)
    return Omega_DA - self.nu1*Omega_PA

def Omega_C(self, c_ca, c_co):
    sp = c_ca*c_co - 1
    Omega_PC = (max(0.0,sp))**self.n
    sa = 1 - c_ca*c_co
    Omega_DC = (max(0.0, sa))**self.n
    return Omega_PC - self.nu2*Omega_DC
```

Numerical methods

we solve the coupled PDEs by the Method of Lines, i.e. we will discretise the right hand sides of the equations by means of the Finite Difference Method but will keep the time derivative as they are and we will use an Initial Value Problem integrator, e.g. Runge-Kutta methods.

We have implemented two methods: a simple 1st order Euler method (with a user-chosen constant timestep dt), and the use of the solve_ivp from scipy The code has been designed with modularity in mind so that many functions have been created, each carrying out a single task.

Because sharp transitions and discontinuities are very often source of error in numerical modelling we have replaced the Heaviside function $\theta(x)$ by a smoothed version (the value of 500 controls the smoothness):

```
def heaviside(x,xbot,xtop,xscale):
    val=0.5*(1+np.tanh((x-xtop/xscale)*500)) *0.5*(1+np.tanh((xbot/xscale-x)*500))
    return val
```

As the equations require many constant parameters, we created a class to contain all the functions and parameter variables. This way the functions need only take the solution variables and the 'self' keyword as arguments, rather than passing lengthy lists of parameters to each function. Parameter variables are set in the constructor function '__init__(self)' then accessed within the function using 'self.var_name'. We also applied jit (just in time) compiling to the entire class, using the Numba package [2]. This produces compiled binaries for python functions at the first time they are encountered in the run and then reuses these binaries for all subsequent function calls, resulting in dramatically improved performance. For this code, speed-up varies between a factor of roughly 3-10, depending on the mode the code runs in. The addition of jit to the class is relatively simple, the main addition is that the type of all parameter values must be explicitly specified and passed to the jit decorator as a list of 2-element tuples specifying the variable name and its type:

```
from numba.experimental import jitclass

# for the jit-compiling, we need to specify the type of all parameters in the class
spec = [
    ('g', float64),
    ('K_C', float64),
    ('K_A', float64),
    ('ADZ_bot', float64),
    ('ADZ_top', float64),
    .
    ..
    ...
    ]
# This class contains the functions and parameters that are used to calculate
# the RHS of eqns 40-43 of L'Heureux (2018)
@jitclass(spec)
class LHeureux:

    # define all params as instance vars
    # this way we can easily modify them at the instance level
    # i.e. for parameter searches
    def __init__(self):
        # physical constants
        self.g = 9.81*100              # gravitational acceleration (cm/s^2)
        # model parameters
        self.K_C = 10**-6.37           # Calcite solubility (M^2)
        self.K_A = 10**-6.19           # Aragonite solubility (M^2)
        self.ADZ_top = 50              # top of the Aragonite dissolution zone (cm)
        self.ADZ_bot = 150
        .
        ..
        ...
```

$$
\begin{aligned}
\frac{\partial C_A}{\partial t} &= -U\frac{\partial C_A}{\partial x} + \mathcal{R}_{\mathrm{Aragonite}} \\
\frac{\partial C_C}{\partial t} &= -U\frac{\partial C_C}{\partial x} + \mathcal{R}_{\mathrm{Calcite}} \\
\frac{\partial \hat{c}_k}{\partial t} &= -W\frac{\partial \hat{c}_k}{\partial x} + \frac{1}{\phi}\frac{\partial}{\partial x}\left(\phi d_k \frac{\partial \hat{c}_k}{\partial x}\right) + \mathcal{R}_k \\
\frac{\partial \phi}{\partial t} &= -\frac{\partial}{\partial x}(W\phi) + \mathcal{R}_\phi
\end{aligned} \tag{3}
$$

[2] https://numba.readthedocs.io/en/stable/

The $\mathcal{R}$ rhs terms are implemented via five functions:

```
#Aragonite
def R_AR(self, AR, CA, c_ca, c_co, x):
    return - self.Da*( ( 1 - AR ) * AR * self.Omega_A(c_ca, c_co, x) +\
                          self.lamb * AR * CA * self.Omega_C(c_ca, c_co) )

#Calcite
def R_CA(self, AR, CA, c_ca, c_co, x):
    return self.Da*( self.lamb * ( 1 - CA ) * CA * self.Omega_C(c_ca, c_co) +\
                          AR * CA * self.Omega_A(c_ca, c_co, x) )

# Ca ions
def R_c_ca(self, AR, CA, c_ca, c_co, phi, x):
    return self.Da*( ( 1 - phi ) / phi ) * (self.delta - c_ca)*\
                ( AR * self.Omega_A(c_ca, c_co, x) - self.lamb * CA * self.Omega_C(c_ca, c_co) )

# CO3 ions
def R_c_co(self, AR, CA, c_ca, c_co, phi, x):
    return self.Da*( ( 1 - phi ) / phi ) * (self.delta - c_co)*\
                ( AR * self.Omega_A(c_ca, c_co, x) - self.lamb * CA * self.Omega_C(c_ca, c_co) )

# porosity
def R_phi(self, AR, CA, c_ca, c_co, phi, x):
    return self.Da*( 1 - phi )*\
                ( AR * self.Omega_A(c_ca, c_co, x) - self.lamb * CA * self.Omega_C(c_ca, c_co) )
```

The solution for all variables is stored in a single array, 5*nnx long, with the first nnx elements corresponding to the AR values, the next nnx giving CA and so on. The full ordering of the variables in X is AR, CA, c_ca, c_co, phi. This structure is designed for use with the ivp_solve function, which expects a single array for the solution variable values. ivp_solve also expects a single function which calculates the RHS, so we also combine the RHS functions in a master function:

```
def X_RHS(self, t, X, nnx, x, h):

dAR_dt   = self.RHS_AR(  X[0:nnx], X[nnx:2*nnx], X[2*nnx:3*nnx], X[3*nnx:4*nnx], X[4*nnx:5*nnx], x,
    h)
dCA_dt   = self.RHS_CA(  X[0:nnx], X[nnx:2*nnx], X[2*nnx:3*nnx], X[3*nnx:4*nnx], X[4*nnx:5*nnx], x,
    h)
dc_ca_dt = self.RHS_c_ca(X[0:nnx], X[nnx:2*nnx], X[2*nnx:3*nnx], X[3*nnx:4*nnx], X[4*nnx:5*nnx], x,
    h)
dc_co_dt = self.RHS_c_co(X[0:nnx], X[nnx:2*nnx], X[2*nnx:3*nnx], X[3*nnx:4*nnx], X[4*nnx:5*nnx], x,
    h)
dphi_dt  = self.RHS_phi( X[0:nnx], X[nnx:2*nnx], X[2*nnx:3*nnx], X[3*nnx:4*nnx], X[4*nnx:5*nnx], x,
    h)

return np.concatenate((dAR_dt, dCA_dt, dc_ca_dt, dc_co_dt, dphi_dt))
```

Initial values

The initial conditions are chosen as spatially homogeneous constants:

$$C_A(x,0) = C_{A,init} \tag{4}$$
$$C_C(x,0) = C_{C,init} \tag{5}$$
$$\hat{c}_k(x,0) = \hat{c}_{k,init} \tag{6}$$
$$\phi(x,0) = \phi_{init} \tag{7}$$

for $x \neq 0$. This is Eq. 36 in the paper.

Boundary conditions

In the paper we find "At the bottom boundary $x = L$ we will assume that the system has no diffusive flux". We are not too sure what to make of this, but the author provides the following:

At $x = 0$ (top of the domain):

$$C_A(0,t) = C_A^0 \tag{8}$$
$$C_C(0,t) = C_C^0 \tag{9}$$
$$\hat{c}_k(0,t) = \hat{c}_k^0 \tag{10}$$
$$\phi(0,t) = \phi^0 \tag{11}$$

At $x = L$ (bottom of the domain):

$$\frac{\partial \hat{c}_k(x,t)}{\partial x}|_L = 0$$

$$\frac{\partial \phi(x,t)}{\partial x}|_L = 0$$

we need boundary conditions there because of the second-order derivative in these equations (diffusion term).

Upwinding

The equations for C_A and C_C are pure advection equations. We have observed that in the absence of any stabilisation the computed fields showcase over- and undershoots, which are unphysical and which also find their way into the rhs of the other PDEs.

We have then decided to use the Fiadeiro & Veronis method [FV77; Wri92], which works as follows for the (steady state) advection-diffusion equation (see also p315 of [Bou97]):

$$\kappa \frac{T_{i+1} - 2T_i + T_{i-1}}{h^2} - u \frac{(1-\sigma)T_{i+1} + 2\sigma T_i - (1+\sigma)T_{i-1}}{2h} = 0$$

which is a blend of backward (upstream) and central differences. The amount of blending is dictated by the value of the parameter σ, defined as

$$\sigma = \coth \frac{uh}{2\kappa} - \frac{2\kappa}{uh} = \coth Pe - \frac{1}{Pe}$$

The parameter σ has the property that $\sigma \to 0$ when $Pe \to 0$ and $\sigma \to 1$ when $Pe \to \infty$. The equation above can also be rewritten:

$$\kappa \frac{T_{i+1} - 2T_i + T_{i-1}}{h^2} - u \frac{1T_{i+1} - T_{i-1}}{2h} + \frac{u\sigma h}{2} \frac{T_{i+1} - 2T_i + T_{i-1}}{h^2} = 0$$

which makes the action of this stabilisation term more obvious: it is a diffusion term whose diffusion coefficient goes away when $h \to 0$.

Thus, if $\sigma = 0$ (diffusion dominated), then pure central differencing is obtained, and if $\sigma = 1$ (advection dominated), then pure backward differencing results. Interestingly enough, this blended or weighted scheme is second-order accurate even as it switches to backward differencing [FV77]. When $\kappa \to 0$ (no physical diffusion at all - as is the case for C_A and C_C) then σ tends to ± 1, and more precisely: $\sigma = sign(u)$, which allows for a simple and elegant implementation.

1st PDE

$$\frac{\partial C_A}{\partial t} = -U(\phi)\frac{\partial C_A}{\partial x} - Da[(1 - C_A)C_A(\Omega_{DA} - \nu_1 \Omega_{PA}) + \lambda C_A C_C(\Omega_{PC} - \nu_2 \Omega_{DC})]$$

Turning to the Aragonite PDE, we must now produce a FD approximation of the advection term. For all the nodes inside the domain, use use a centered approximation:

$$-U \frac{\partial C_A}{\partial x} \to -U_i \frac{C_A|_{i+1} - C_A|_{i-1}}{2h}$$

For the node at the top (i.e. $i = 0$), we are prescribing the value of C_A, i.e. the derivative is then zero so then

$$-U \frac{\partial C_A}{\partial x} \to 0$$

For the node at the bottom, no boundary condition is imposed so we use a backward approximation:

$$-U \frac{\partial C_A}{\partial x} \to -U_i \frac{C_A|_i - C_A|_{i-1}}{h}$$

```
def RHS_AR(self, AR, CA, c_ca, c_co, phi, x, h):
    dAR_dt = np.zeros(len(x))
    u = self.U(phi)
    for i in range(0,len(x)):
        # x = 0 BC, Dirichlet
        if (i==0):
            dAR_dt[i]= 0
        # x = Lx, no prescribed BC
        elif (i==len(x)-1):
            dAR_dt[i]= -u[i]*(AR[i]-AR[i-1])/h + self.R_AR(AR[i],CA[i],c_ca[i],c_co[i],x[i])
        else:
            dAR_dt[i]= -u[i]*(AR[i+1]-AR[i-1])/(2*h) + self.R_AR(AR[i],CA[i],c_ca[i],c_co[i], x[i])
    return dAR_dt
```

2nd PDE

$$\frac{\partial C_C}{\partial t} = -U(\phi)\frac{\partial C_C}{\partial x} + Da[(1-C_C)C_C(\Omega_{PC}-\nu_2\Omega_{DC}) + \lambda C_A C_C(\Omega_{DA}-\nu_1\Omega_{PA})]$$

The structure of the 2nd PDE is identical to the first one so we adopt the same approch.

```
def RHS_CA(self, AR, CA, c_ca, c_co, phi, x, h):
    dCA_dt = np.zeros(len(x))
    u = self.U(phi)
    for i in range(0,len(x)):
        # x = 0 BC, Dirichlet
        if (i==0):
            dCA_dt[i]= 0
        # x = Lx, no prescribed BC
        elif (i==len(x)-1):
            dCA_dt[i]= -u[i]*( CA[i]-CA[i-1] ) / h + self.R_CA(AR[i], CA[i], c_ca[i], c_co[i], x[i])
        else:
            dCA_dt[i]= -u[i]*( CA[i+1]-CA[i-1])/(2*h)+self.R_CA(AR[i],CA[i],c_ca[i],c_co[i],x[i])
    return dCA_dt
```

3rd & 4th PDE

$$\frac{\partial \hat{c}_k}{\partial t} = -W(\phi)\frac{\partial \hat{c}_k}{\partial x} + \frac{1}{\phi}\frac{\partial}{\partial x}\left(\phi d_k \frac{\partial \hat{c}_k}{\partial x}\right) + Da\frac{1-\phi}{\phi}(\delta - \hat{c}_k)[C_A(\Omega_{DA}-\nu_1\Omega_{PA}) - \lambda C_C(\Omega_{PC}-\nu_2\Omega_{DC})]$$

The advection term is treated in the same way as above, but we here have to discretise a diffusion term with a non constant diffusion coefficient. The standard procedure in such cases is as follows:

$$\frac{\partial}{\partial x}\left(k\frac{\partial f}{\partial x}\right) \simeq \frac{k_{i+1/2}\frac{f_{i+1}-f_i}{h} - k_{i-1/2}\frac{f_i - f_{i-1}}{h}}{h}$$

where $k_{i\pm1/2}$ is evaluated between the points to maintain the second order accuracy.

In our case we then have (I have removed the k subscript):

$$\frac{\partial}{\partial x}\left(\phi d\frac{\partial \hat{c}}{\partial x}\right) \simeq \frac{\phi_{i+1/2}d_{i+1/2}\frac{c_{i+1}-c_i}{h} - \phi_{i-1/2}d_{i-1/2}\frac{c_i-c_{i-1}}{h}}{h}$$

The value of c is prescribed at $x=0$ but only its derivative is prescribed at the bottom ($x=L$):

$$\frac{\partial \hat{c}(x,t)}{\partial x}|_L = 0$$

i.e.

$$\frac{c_i - c_{i-1}}{h} = 0 \qquad \text{for} \quad i = nnx-1$$

We then write the expression above fully backward:

$$\frac{\partial}{\partial x}\left(\phi d\frac{\partial \hat{c}}{\partial x}\right) \simeq \frac{\phi_{i-1/2}d_{i-1/2}\frac{c_i-c_{i-1}}{h} - \phi_{i-3/2}d_{i-3/2}\frac{c_{i-1}-c_{i-2}}{h}}{h}$$

and we see that the b.c. kills the first term, so we are left with

$$\frac{\partial}{\partial x}\left(\phi d\frac{\partial \hat{c}}{\partial x}\right) \simeq \frac{-\phi_{i-3/2}d_{i-3/2}\frac{c_{i-1}-c_{i-2}}{h}}{h} = -\phi_{i-3/2}d_{i-3/2}\frac{c_{i-1}-c_{i-2}}{h^2}$$

and we see that the b.c. kills the first term, so we are left with

```
...
```

5th PDE

$$\frac{\partial \phi}{\partial t} = -\frac{\partial}{\partial x}(W\phi) + d_\phi\frac{\partial^2\phi}{\partial x^2} + Da(1-\phi)[C_A(\Omega_{DA}-\nu_1\Omega_{PA}) - \lambda C_C(\Omega_{PC}-\nu_2\Omega_{DC})]$$

$$\frac{\partial}{\partial x}(W(\phi)\phi) \simeq \frac{W(\phi_{i+1})\phi_{i+1} - W(\phi_{i-1})\phi_{i-1}}{2h}$$

$$\frac{\partial^2\phi}{\partial x^2} \simeq \frac{\phi_{i+1}-\phi_{i-1}}{2h}$$

The value of ϕ is prescribed at $x = 0$ but only its derivative is prescribed at the bottom ($x = L$):

$$\frac{\partial \phi(x,t)}{\partial x}|_L = 0$$

i.e.

$$\frac{\phi_i - \phi_{i-1}}{h} = 0 \qquad \text{for} \quad i = nnx - 1$$

The fully backward diffusion term on the right side is

$$\frac{\phi_{i-2} - 2\phi_{i-1} + \phi_i}{h^2}$$

which means that given the boundary condition it becomes

$$\frac{\phi_{i-2} - \phi_{i-1}}{h^2}$$

References

[FV77] Manuel E Fiadeiro and George Veronis. "On weighted-mean schemes for the finite-difference approximation to the advection-diffusion equation". In: *Tellus* 29.6 (1977), pp. 512–522. DOI: 10.3402/tellusa.v29i6.11385.

[Wri92] Daniel G Wright. "Finite difference approximations to the advection-diffusion equation". In: *Tellus A* 44.3 (1992), pp. 261–269. DOI: 10.1034/j.1600-0870.1992.t01-2-00005.x.

[Bou97] Bernard P Boudreau. *Diagenetic models and their implementation.* Vol. 505. 1997. ISBN: 978-3-642-64399-6. DOI: 10.1007/978-3-642-60421-8.

[LHe18] Ivan L'Heureux. "Diagenetic self-organization and stochastic resonance in a model of limestone-marl sequences". In: *Geofluids* 2018 (2018), pp. 1–18. DOI: 10.1155/2018/4968315.