Numerical solution of the diffusion equation in 1D: two simple examples

Francesco Piazza
Université d'Orléans,
Centre de Biophysique Moléculaire (CBM), CNRS UPR4301
Francesco.Piazza@cnrs-orleans.fr

February 18, 2019

1 Introduction

This short note describes the implementation of different methods to solve the diffusion equation a 1D finite domain, $x \in \Omega = [0, L]$. The numerical implementation has been performed in Python (codes annexed in appendix) while the analytical solutions have been worked out in the classes with both the Laplace transform method and the method of separation of variables (Fourier series).

2 Explicit finite-difference scheme

In this section we describe the Python code written to solve the diffusion equation in 1D with a simple explicit difference method in the finite domain $x \in \Omega = [0, L]$ (forward differences in time) Discretization: $c(x,t) \to c_{nj}$, with $x = ndx$, $t = jdt$,

$$c_{n,j+1} = rc_{n+1,j} + (1-2r)c_{n,j} + rc_{n-1,j} \tag{1}$$

where $r = Ddt/dx^2$ (D being the diffusion coefficient). This is written in matrix form as follows

$$C_{j+1} = KC_j \tag{2}$$

where C_j is the discretized spatial profile vector at time jdt, i.e. $C_j = (c_{1,j}, c_{2,j}, \ldots, c_{N,j})$, and the matrix K reads

$$K_{nm} = r\delta_{n,m+1} + r\delta_{n,m-1} + (1-2r)\delta_{nm} \tag{3}$$

This algorithm is unstable for $r > 1/2$. The initial condition implemented in the script is that of an initially empty layer, namely

$$c(x,0) = c_0(x) = 0 \tag{4}$$

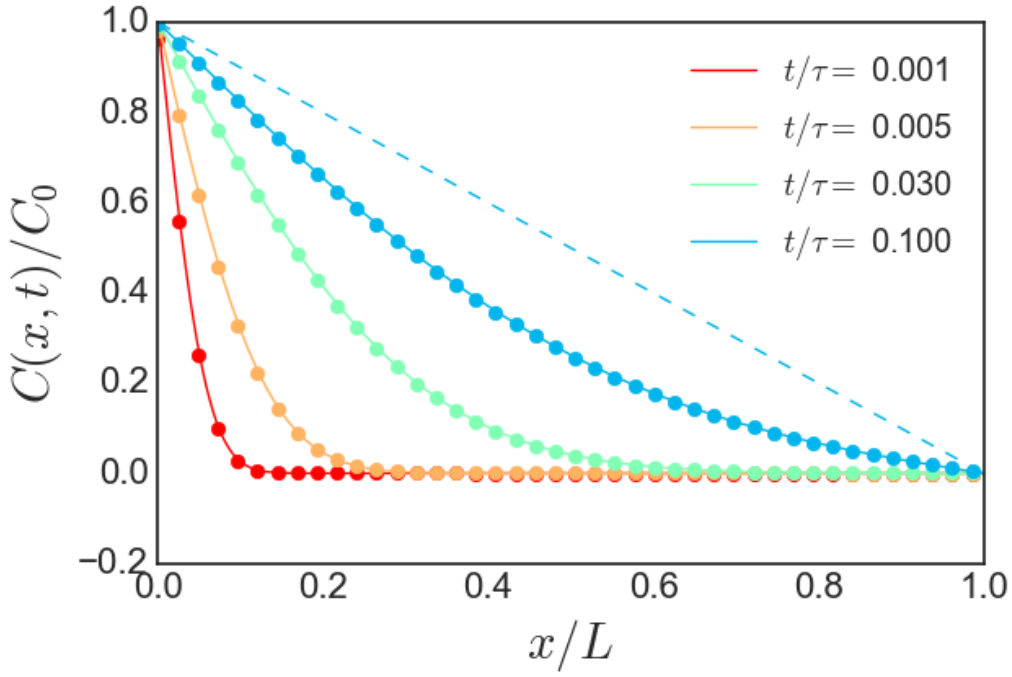

Figure 1: Illustration of the numerical solution of the diffusion equation in $\Omega = [0, L]$ with the explicit-differences scheme with $r = 1/2$ (symbols), initial condition (4), $c(0,t)/C_0 = 1$, $c(L,t) = 0$ and comparison with the analytical solution, Eq. (7), computed by truncating the sum over m to include 100 harmonics (solid lines). The dashed straight line represents the asymptotic profile (8). Parameters are: $r = 1/2$, $N = 500$.

Dirichlet BC on the frontier $\partial\Omega$ $x = 0 \cup x = L$, corresponding to $n = 0$ ($x = 0$) and $n = N + 1$ ($x = L$),

$$c(0,t) = C_0 \tag{5}$$
$$c(L,t) = 0 \tag{6}$$

This corresponds to a simple non-equilibrium problem of diffusion through a layer (e.g. the *stratum corneum* of the skin) of thickness L, with constant concentration at the interface with the left medium (e.g. air) and a sink at the interface with the righ medium (e.g. the viable epidermis, the medium where the diffusant should be delivered). The analytical solution to this problem reads (see Fig. 8)

$$c(x,t) = c_\infty(x) - \frac{2}{\pi}\sum_{m=1}^{\infty}\frac{e^{-m^2\pi^2 t/\tau}}{m}\sin\left(\frac{m\pi x}{L}\right) \tag{7}$$

where $c_\infty(x)$ is the stationary profile

$$c_\infty(x) = C_0\left(1 - \frac{x}{L}\right) \tag{8}$$

3 Implicit schemes: the Crank-Nicholson method

In this section we describe the Python code written to solve the diffusion equation in 1D with the Crank-Nicholson method in the finite domain $x \in \Omega = [0, L]$. Discretization: $c(x,t) \to c_{nj}$, with $x = ndx$, $t = jdt$,

$$-rc_{n+1,j+1} + 2(1+r)c_{n,j+1} - rc_{n-1,j+1} = rc_{n+1,j} + 2(1-r)c_{n,j} + rc_{n-1,j} \tag{9}$$

where $r = Ddt/dx^2$ (D being the diffusion coefficient). This is written in matrix form as follows

$$R^+ C_{j+1} = R^- C_j \tag{10}$$

where C_j is the discretized spatial profile vector at time jdt, i.e. $C_j = (c_{1,j}, c_{2,j}, \ldots, c_{N,j})$, and the matrices $R^\pm$ read

$$R^+_{nm} = -r\delta_{n,m+1} - r\delta_{n,m-1} + 2(1+r)\delta_{nm} \tag{11}$$

$$R^-_{nm} = r\delta_{n,m+1} + r\delta_{n,m-1} + 2(1-r)\delta_{nm} \tag{12}$$

The initial condition implemented in the script

$$c(x,0) = c_0(x) = \sin(\pi x/L). \tag{13}$$

Dirichlet BC on the frontier $\partial\Omega$ $x = 0 \cup x = L$, corresponding to $n = 0$ ($x = 0$) and $n = N + 1$ ($x = L$),

$$c(0,t) = \phi(t) \tag{14}$$

$$c(L,t) = \psi(t) \tag{15}$$

$$c(0,t) \to c_{0,j} = \phi_j \qquad c(L,t) \to c_{N+1,j} = \psi_j$$

Here we consider constant BCs, $\phi(t) = C_0, \psi(t) = C_L$, which leads to the analytical solution

$$c(x,t) = c_\infty(x) + \sin\left(\frac{\pi x}{L}\right) e^{-\pi^2 t/\tau} + \frac{2}{\pi}\sum_{m=1}^{\infty} \frac{C_L(-1)^m - C_0}{m} \sin\left(\frac{m\pi x}{L}\right) e^{-m^2\pi^2 t/\tau} \tag{16}$$

where $c_\infty(x)$ is the stationary profile

$$c_\infty(x) = C_0 - (C_0 - C_L)\frac{x}{L} \tag{17}$$

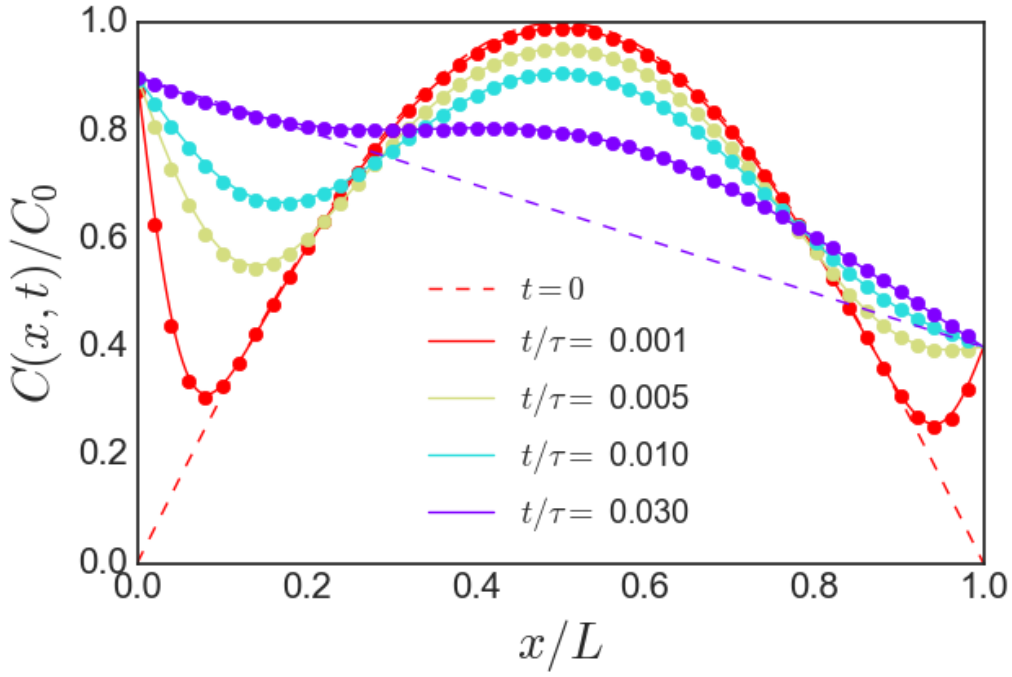

Figure 2: Illustration of the numerical solution of the diffusion equation in $\Omega = [0, L]$ with the Crank Nicholson method with $r = 1$ (symbols), initial condition (13), $c(0,t)/C_0 = 0.9$, $c(L,t)/C_0 = 0.4$ and comparison with the analytical solution, Eq. (16), computed by truncating the sum over m to include 100 harmonics (solid lines). The dashed straight line represents the asymptotic profile (17). Parameters are: $r =$, $N = 1000$.

Analytical solution computed as in Carlslaw & Jaeger, Conduction of Heat in Solids (Oxford Science Publications), 2nd Edition, page 99.

The inclusion of the BC leads to the following linear system

$$R^+ C_{j+1} = R^- C_j + V \tag{18}$$

where

$$V_n = \begin{cases} r(\phi_{j+1} + \phi_j) & \text{for} \quad n = 1 \\ 0 & \text{for} \quad 1 < n < N \\ r(\psi_{j+1} + \psi_j) & \text{for} \quad n = N \end{cases} \tag{19}$$

Non-dimensional units in the code: x/L, t/τ, c/C_0 with $\tau = L^2/D$.

Python code for the explicit-differences scheme

```
'''
#  Script to solve the diffusion equation in 1D, explicit
#  finite differences
#
#      c_i(t+dt) = (1-a)*c_(t) + a*[c_{i-1}(t) + c_{i-1}(t)]
#
#      where
#               a = D*dt/dx**2
#
#      Initial condition
#      c_i(0) = c0_i
#
#      Domain of solution D=[0,L]. Dirichlet BC at x=0 and x=L
#
#      c(0,t) = cB0
#      c(L,t) = 0.
#
#      This is a simple model to describe steady diffusion through
#      the outermost skin layer (Stratum Corneum)
#
#      Analitycal solution computed as in Carlslaw & Jaeger,
#      page 99
#
#      Non-dimensional units :  x/L, t/tau, c/c0 with tau = L**2/D
#
#      F. Piazza, University of Orleans, February 2019
#============================================================'''

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
plt.style.use('seaborn-white')

cB0 = 1
cBL = 0.

'''============================================================'''
def c0(x):
    return 0.

''' Analytic solution, time in units of tau, x in units of L  '''
def ca(x,t):
    c = cB0*(1.- x)
    for k in range(1,100):
```

```
        c = c - 2*np.sin(np.pi*k*x)/k* \
                np.exp(-k**2*np.pi**2*t)/np.pi
    return c

def ca_steady(x):
    return cB0*(1.- x)

'''================================================================
   The time map is implemented for a=0.5. This amounts to taking
   D*dt = dx**2/2, which fixes the time step from the definition
   of the spatial mesh and the value of the diffusion
   coefficient D '''

N = 500               # number of points #
NI = N - 1            # number of intervals #
xmin = 0.
xmax = 1.
timesteps =   1000000 # units of tau #
dx = (xmax-xmin)/NI
r = 0.5
dt = r * dx**2        # dt/tau = 0.5 * (dx/L)**2 #
x = np.linspace(xmin,xmax,N)
c0v = np.zeros(N)

''' The evolution matrix, that is, the discretized Laplacian '''
K  = r*(np.diag(np.ones(N-1),1) + np.diag(np.ones(N-1),-1)) + \
     (1-2*r)*np.eye(N)

U0 = np.zeros(N)
U0[0]   = r*cB0     # BC at x=0 #
U0[N-1] = r*cBL     # BC at x=L #

cv = c0v
ik = 0
tv = []
NS = 5
cvs = np.zeros((N,timesteps))
t_samp = np.array([0.001,0.005,0.03,0.1,0.5])
for nt in range(1,timesteps):
    cv = np.matmul(K,cv) + U0
    if dt*nt > t_samp[ik]:
        print ('sample at t/tau =',ik, dt*nt)
        cvs[:,ik] = cv
        tv.append(dt*nt)
        if ik == NS-1:
            break
```

```
        ik += 1

'''#############################################################'''
'''  (NICE) PLOTS '''

mev = 12
mark_size=6
plt.figure(figsize=(6.5,4.5))
colors = cm.rainbow(np.linspace(0, 1, ik+1))
for m in range(ik):
    plt.plot(x,cvs[:,m],'o',markevery=mev,markersize=mark_size,
        mec=colors[ik-m],color=colors[ik-m])
    plt.plot(x,ca(x,tv[m]),color=colors[ik-m],
        label='$t/\\tau =$ %5.3f' % tv[m])
plt.plot(x,ca_steady(x),'--',color=colors[ik-m])
plt.legend(loc='best')
plt.xlabel('$x/L$',fontsize=22)
plt.ylabel('$C(x,t)/C_0$',fontsize=22)
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.tight_layout()
plt.savefig('diffusion_through_skin_layer.png')
plt.show()
```

Python code for the Crank-Nicholson scheme

```
'''
#  Script to solve the diffusion equation in 1D, finite domain,
#  Crank-Nicholson method. Discretization:
#  $c(x,t) \to c_{nj}$, with $x = n dx$, $t = j dt$,
#
#      \[
#      -r c_{n+1,j+1} + 2(1+r) c_{n,j+1} - r c_{n-1,j+1} =
#       r c_{n+1,j}   + 2(1-r) c_{n,j}   + r c_{n-1,j}
#      \]
#
#  where $r = D*dt/dx**2$ ($D$ being the diffusion coefficient).
#
#  Initial condition $c(x,0) = c_0(x) = \sin(\pi x/L)$
#  Domain of solution $x\in\Omega=[0,L]$, discretized as
#  $x_n = ndx$,  with $n=1,2,\dots,N$.
#
#  Dirichlet BC on the frontier $\partial \Omega$ $x=0 \cup x=L$,
#  corresponding to $n=0 (x=0)$ and $n=N+1 (x=L)$,
#
#      \begin{eqnarray}
#      \label{e:BC}
#      c(0,t) = \phi(t) \\
#      c(L,t) = \psi(t)
#      \end{eqnarray}
#
#      \[
#      c(0,t) \to c_{0,j}   = \phi_j  \qquad
#      c(L,t) \to c_{N+1,j} = \psi_j
#      \]
#
#  Here we consider constant BCs, $\phi(t) = C_0,\psi(t) = C_L$.
#
#  Inclusion of the BC leads to the following linear system
#
#      \begin{equation}
#      \label{e:LSBC}
#      R^+ C_{j+1} = R^- C_{j} + V
#      \end{equation}
#
#  where
#
#      \begin{equation}
#      \label{e:VBC}
#      V_n =
```

```
#       \begin{cases}
#       r (\phi_{j+1} + \phi_j) & \text{for} n = 0 \\
#       0                       & \text{for} 0<n<N \\
#       r (\psi_{j+1} + \psi_j) & \text{for} n = N \\
#       \end{cases}
#       \end{equation}
#
#  Analitycal solution computed as in Carlslaw & Jaeger,
#  Conduction of Heat in Solids
#  (Oxford Science Publications)
#  2nd Edition, page 99.
#
#  Non-dimensional units :  x/L, t/tau, c/cB0 with tau = L**2/D
#
#  F. Piazza, University of Orleans, February 2019
#================================================================'''

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
plt.style.use('seaborn-white')

cB0 = 0.9      # Dirichlet bounbdary condition at x = 0 #
cBL = 0.4      # Dirichlet bounbdary condition at x = L #

'''================================================================'''
''' The initial condition '''
def c0(x):
    return np.sin(np.pi*x)

''' The analytical solution  '''
def ca(x,t):
    c = cB0 + (cBL-cB0)*x + np.sin(np.pi*x)*np.exp(-np.pi**2*t)
    for k in range(1,100):
        c = c + 2*(cBL*(-1)**k-cB0)/k*np.sin(np.pi*k*x)* \
                np.exp(-k**2*np.pi**2*t)/np.pi
    return c

''' The stationary profile '''
def ca_steady(x):
    return cB0 + (cBL-cB0)*x

'''================================================================
    The time map is implemented for $r = 1$  This amounts to taking
    D*dt = dx**2/2, which fixes the time step from the definition
    of the spatial mesh and the value of the diffusion
```

```
    coefficient D '''

N = 500               # number of points       #
NI = N - 1            # number of intervals    #
xmin = 0.
xmax = 1.
timesteps =   100000 # units of tau           #
dx = (xmax-xmin)/NI
r = 1
dt = r * dx**2        # dt/tau = r * (dx/L)**2 #
x = np.linspace(xmin,xmax,N)
c0v = c0(x)

''' The evolution matrices '''

RP  = -r*(np.diag(np.ones(N-1),1) + np.diag(np.ones(N-1),-1)) + \
       2*(1. + r)*np.eye(N)

RM  = r*(np.diag(np.ones(N-1),1) + np.diag(np.ones(N-1),-1)) + \
      2*(1. - r)*np.eye(N)

RPI = np.linalg.inv(RP)
K   = np.matmul(RPI,RM)

U0 = np.zeros(N)
U0[0]   = 2*r*cB0       # BC at x=0 #
U0[N-1] = 2*r*cBL       # BC at x=L #
K0 = np.matmul(RPI,U0)

cv = c0v
ik = 0
tv = []
NS = 4
cvs = np.zeros((N,timesteps))
t_samp = np.array([0.001,0.005,0.01,0.03])
       # np.logspace(-3,-1,NS) #

for j in range(1,timesteps):
    cv = np.matmul(K,cv) + K0
    if dt*j > t_samp[ik]:
        print (' sample at t/tau =',ik, dt*j)
        cvs[:,ik] = cv
        tv.append(dt*j)
        if ik == NS-1:
            break
        ik += 1
```

```
137
138 '''############################################################'''
139 '''  (NICE) PLOTS '''
140
141 mev = 10
142 mark_size=6
143 plt.figure(figsize=(6.5,4.5))
144 colors = cm.rainbow(np.linspace(0, 1, ik+1))
145 plt.plot(x,c0v,'r--',color=colors[-1],label='$t=0$')
146 for m in range(ik+1):
147     plt.plot(x,cvs[:,m],'o',markevery=mev,ms=mark_size,
148     mec=colors[ik-m],color=colors[ik-m])
149     plt.plot(x,ca(x,tv[m]),color=colors[ik-m],
150     label='$t/\\tau =$ %5.3f' % tv[m])
151 plt.plot(x,ca_steady(x),'--',color=colors[ik-m])
152 plt.legend(loc='best')
153 plt.xlabel('$x/L$',fontsize=22)
154 plt.ylabel('$C(x,t)/C_0$',fontsize=22)
155 plt.xticks(fontsize=16)
156 plt.yticks(fontsize=16)
157 plt.tight_layout()
158 plt.savefig('Diffusion1D_Crank_Nicolson_Dirichlet.png')
159 plt.show()
```