# Sur les schémas de l'équation "KPP avec mémoire"

Liam Toran

## Contents

## 1 Schémas et Positivité

On a le modèle suivant ("KPP avec mémoire"):

$$\begin{cases} \partial_t \mu = K\Delta\mu + C(\mu+\rho) - \mu\rho \\ \partial_t \rho = F_0 \mu \\ \partial_t C = -b\rho C \end{cases} \tag{1}$$

### 1.1 Pour l'équation différentielle ordinaire

Sans dépendance spatiale:

$$\begin{cases} \partial_t \mu = C(\mu+\rho) - \mu\rho \\ \partial_t \rho = F_0 \mu \\ \partial_t C = -b\rho C \end{cases} \tag{2}$$

#### 1.1.1 Schéma semi-implicite I pour l'EDO

Soit le schéma semi-implicite I pour l'EDO:

$$\boxed{\begin{cases} \mu^{n+1} = \mu^n + \Delta t(C^n(\mu^{n+1} + \rho^{n+1}) - \mu^{n+1}\rho^n) \\ \rho^{n+1} = \rho^n + \Delta t(F_0 \mu^{n+1}) \\ C^{n+1} = C^n - \Delta t(b\rho^{n+1}C^{n+1}) \end{cases}} \tag{3}$$

Ce schéma donne:

$$\begin{cases} \mu^{n+1}(1 - \Delta t(C^n(1+\Delta t F_0)) + \rho^n) = \mu^n + \Delta t C^n \rho^n \\ \rho^{n+1} = \rho^n + \Delta t(F_0 \mu^{n+1}) \\ C^{n+1} = C^n \frac{1}{1+\Delta t b \rho^{n+1}} \end{cases}$$

Pour conserver la positivité il suffit que le terme $(1 - \Delta t(C^n(1+\Delta t F_0)) + \rho^n)$ reste positif:
Par exemple:

$$\boxed{C^0 < \frac{1}{\Delta t(1+F_0\Delta t)}} \tag{4}$$

### 1.1.2 Schéma semi-implicite II pour l'EDO

Soit le schéma semi-implicite II pour l'EDO:

$$\boxed{\begin{cases} \mu^{n+1} = \mu^n + \Delta t(C^n(\mu^{n+1} + \rho^{n+1}) - \mu^n \rho^n) \\ \rho^{n+1} = \rho^n + \Delta t(F_0 \mu^{n+1}) \\ C^{n+1} = C^n - \Delta t(b\rho^{n+1} C^{n+1}) \end{cases}} \tag{5}$$

Ce schéma donne:

$$\begin{cases} \mu^{n+1}(1 - \Delta t(C^n(1 + \Delta t F_0))) = \mu^n + \Delta t \rho^n (C^n - \mu^n) \\ \rho^{n+1} = \rho^n + \Delta t(F_0 \mu^{n+1}) \\ C^{n+1} = C^n \frac{1}{1 + \Delta t b \rho^{n+1}} \end{cases}$$

Pour conserver la positivité il suffit que les terme $(1 - \Delta t(C^n(1 + \Delta t F_0)))$ et $\mu^n + \Delta t \rho^n (C^n - \mu^n)$ restent positif:
Par exemple:

$$\boxed{C^0 < \frac{1}{\Delta t(1 + F_0 \Delta t)}} \tag{6}$$

et

$$\boxed{\rho^n < \frac{1}{\Delta t}} \tag{7}$$

On obtient une condition de plus que le schéma semi-implicite I.

## 1.2 Pour l'équation aux dérivées partielles

### 1.2.1 Schéma semi-implicite I pour l'EDP

Soit le schéma semi-implicite I pour l'EDP:

$$\boxed{\begin{cases} \mu_i^{n+1} = \mu_i^n + K\Delta t \frac{\mu_{i+1}^{n+1} - 2\mu_i^{n+1} + \mu_{i-1}^{n+1}}{\Delta x^2} + \Delta t(C_i^n(\mu_i^{n+1} + \rho_i^{n+1}) - \mu_i^{n+1} \rho_i^n) \\ \rho_i^{n+1} = \rho_i^n + \Delta t(F_0 \mu_i^{n+1}) \\ C_i^{n+1} = C_i^n - \Delta t(b\rho_i^{n+1} C_i^{n+1}) \end{cases}} \tag{8}$$

Ce schéma donne:

$$\begin{cases} (1 + \frac{K\Delta t}{\Delta x^2} A - \Delta t(C^n(1 + \Delta t F_0)) + \rho^n)\mu^{n+1} = \mu^n + \Delta t C^n \rho^n \\ \rho^{n+1} = \rho^n + \Delta t(F_0 \mu^{n+1}) \\ C^{n+1} = C^n \frac{1}{1 + \Delta t b \rho^{n+1}} \end{cases}$$

où $A$ est la matrice de $-\Delta$ :

$$A = \begin{bmatrix} 2 & -1 & & 0 \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 2 \end{bmatrix} \tag{9}$$

$A$ étant symétrique définie positive, afin de préserver la positivité, on obtient la même condition (suffisante) que pour l'EDO:

$$\boxed{C^0 < \frac{1}{\Delta t(1 + F_0 \Delta t)}} \tag{10}$$

# 2 Résolution numérique

## 2.1 Résolution de l'EDO

### 2.1.1 Code de résolution de l'EDO

```python
import matplotlib.pyplot as plt
from scipy.integrate import ode
import numpy as np

b=.1 # dtC=-b*rho*C
F0=1 # dtRho = Fo*Mu
tf=50 # temps final de la simulation
rho0=0 #rho initial
mu0=.1 #mu initial
c0=1 #concentration initiale
n=1000 #nombre de pas de temps

#Résolution du schéma éxplicite
def euler_explicite_edo(b, F0, tf, rho0, mu0, c0, n):
  #t0<t1 , temps etudies ,
  #rho0, mu0,c0 reels positifs: condition initiale
  #n entier(nombre d'iterations)
  h=tf/n #pas Deltat
  rho=rho0
  mu=mu0
  c=c0
  t=0
  Rho=[rho0]
  Mu=[mu0]
  C=[c0]
  T=[t]
  for k in range(n):
    new_mu = mu + h*(c*(mu+rho)-mu*rho)
    new_rho = rho + h*F0*mu
    new_c = c - h*b*rho*c
    mu=new_mu
    rho=new_rho
    c=new_c
    t=t+h
    Mu.append(new_mu)
    Rho.append(new_rho)
    C.append(new_c)
    T.append(t)
  return T,Mu,Rho,C

#Résolution du schéma semi- implicite I
def euler_semi_I_edo(b, F0, tf, rho0, mu0, c0, n):
  #t0<t1 , temps etudies ,
  #rho0, mu0,c0 reels positifs: condition initiale
  #n entier(nombre d'iterations)
  h=tf/n #pas Deltat
  rho=rho0
  mu=mu0
  c=c0
  t=0
  Rho=[rho0]
  Mu=[mu0]
```

```python
  C=[c0]
  T=[0]
  for k in range(n):
    new_mu = (mu + h*c*rho)/(1+h*rho-h*c*(1+h*F0))
    new_rho = rho + h*F0*new_mu
    new_c = c/(1 + b*h*new_rho)
    mu=new_mu
    rho=new_rho
    c=new_c
    t=t+h
    Mu.append(new_mu)
    Rho.append(new_rho)
    C.append(new_c)
    T.append(t)
  return T,Mu,Rho,C

#Résolution du schéma semi- implicite II
def euler_semi_II_edo(b, F0, tf, rho0, mu0, c0, n):
  #t0<t1 , temps etudies,
  #rho0, mu0,c0 reels positifs: condition initiale
  #n entier(nombre d'iterations)
  t=0
  h=tf/n #pas Deltat
  rho=rho0
  mu=mu0
  c=c0
  Rho=[rho0]
  Mu=[mu0]
  C=[c0]
  T=[0]
  for k in range(n):
    new_mu = (mu + h*c*rho-h*rho*mu)/(1-h*c*(1+h*F0))
    new_rho = rho + h*F0*new_mu
    new_c = c/(1 + b*h*new_rho)
    mu=new_mu
    rho=new_rho
    c=new_c
    t=t+h
    Mu.append(new_mu)
    Rho.append(new_rho)
    C.append(new_c)
    T.append(t)
  return T,Mu,Rho,C

#Programmation de la méthode de Newton-Raphson
def newton(f,gradf,newton_steps,x0):
  x=x0
  for k in range(newton_steps):
    x=x-f(x)/gradf(x)
  return x
#Résolution du schéma implicite
def euler_implicite_edo(b, F0, tf, rho0, mu0, c0, n):
  #t0<t1 , temps etudiés,
  #rho0, mu0,c0 reels positifs: conditions initiale
  #n entier(nombre d'itérations)
  t=0
  newton_steps=10 #nombre d'itérations de la méthode de Newton-Raphson pour le calcul implicite
  h=tf/n #pas deltat
```

```python
  rho=rho0
  mu=mu0
  c=c0
  Rho=[rho0]
  Mu=[mu0]
  C=[c0]
  T=[0]
  for k in range(n):
    #Calcul de new_mu par methode de Newton Raphson
    #coefficients du polynome d'ordre 3 en new_mu
    alpha = -h**4*F0**2*b
    beta= -F0*h**2*(b+1+2*rho*b*h)
    gamma= -(1+b*h*rho)+b*h**2*F0*mu+h*(c*(1+h*F0)-rho*(1+b*h*rho))
    delta=(1+b*h*rho)*mu + h*c*rho
    def P(X):
      return alpha*X**3+beta*X**2+gamma*X+delta
    def gradP(X):
      return 3*alpha*X**2+2*beta*X+gamma
    new_mu=newton(P,gradP,newton_steps,mu)
    new_rho = rho + h*F0*new_mu
    new_c = c/(1 + b*h*new_rho)
    mu=new_mu
    rho=new_rho
    c=new_c
    t=t+h
    Mu.append(new_mu)
    Rho.append(new_rho)
    C.append(new_c)
    T.append(t)
  return T,Mu,Rho,C

#Utilisation des libraries python (scipy) pour résoudre l'EDO
def black_box_edo(b, F0, tf, rho0, mu0, c0, n):
    def f(t,y,arg1,arg2):
        mu=y[0]
        rho=y[1]
        c=y[2]
        return [c*(mu+rho)-mu*rho, F0*mu, -b*rho*c]

    r = ode(f).set_integrator('zvode', method='adams')
    r.set_initial_value([mu0,rho0,c0],0).set_f_params(F0,b)
    dt=tf/(n-1)
    Rho=[rho0]
    Mu=[mu0]
    C=[c0]
    t=0
    T=[0]
    while r.t < tf:
        mu,rho,c = r.integrate(r.t+dt)
        Mu.append(mu)
        Rho.append(rho)
        C.append(c)
        T.append(r.t)
    return T,Mu,Rho,C


T,Mu,Rho,C = black_box_edo(b, F0, tf, rho0, mu0, c0, n)
rho_inf_theorique = .5*(rho0+np.sqrt(rho0**2 + 4*F0*(mu0+(c0/b))))
print(rho_inf_theorique)
```

```
rho_inf= Rho[n-1]
print(rho_inf)
A=[np.log(rho_inf-x) for x in Rho]
B=[-b*y*rho_inf+np.log((F0)/(b*rho_inf)) for y in T]
#Tracé des solutions
plt.subplot(221)
plt.plot(T,Mu)
plt.ylabel('mu')
plt.xlabel('t')
plt.subplot(222)
plt.plot(T,Rho)
plt.ylabel('rho')
plt.subplot(223)
plt.plot(T,C)
plt.ylabel('C')
plt.subplot(224)
plt.plot(T,A)
plt.plot(T,B)
plt.ylabel('log(rho_inf-rho)+b*rho_inf*t')
plt.show()
```

edo.py

### 2.1.2 Résultat de la simulation de l'EDO

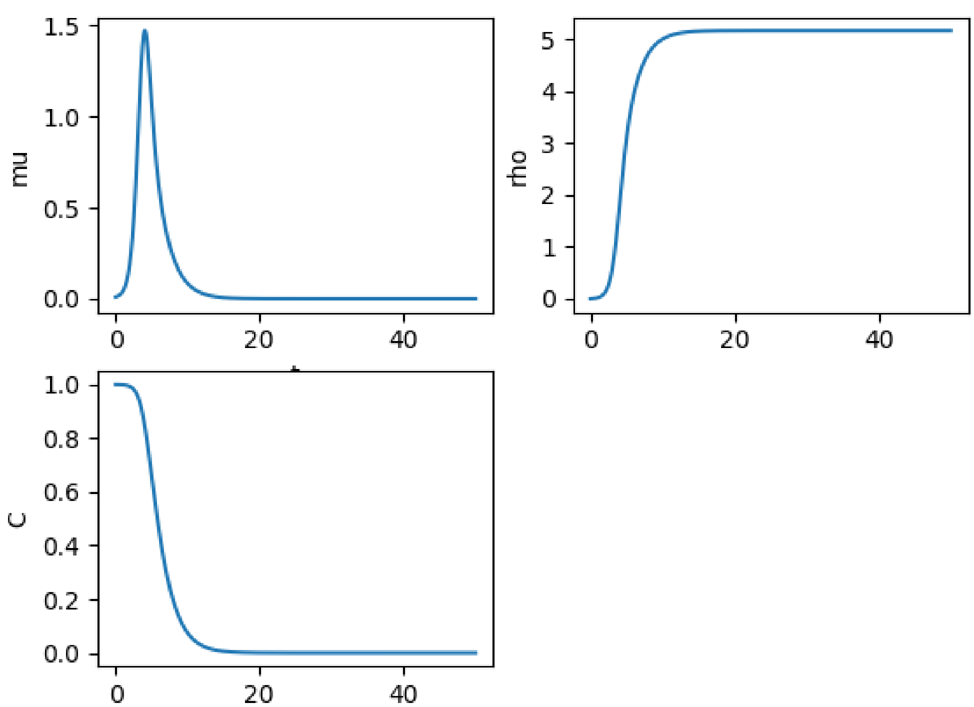


Figure 1: Résolution du schéma implicite pour l'EDO

## 2.2 Résolution de l'EDP en 1D

### 2.2.1 Code de la résolution de l'EDP en 1D

```python
# %load edp_1d.py
import matplotlib.pyplot as plt
import numpy as np
import scipy.sparse as sp
from scipy.sparse.linalg.dsolve import spsolve
import matplotlib.animation as animation

#Coéfficients physiques
K=.4 #coefficient diffusion
b=.2 # dtC=-b*rho*C
F0=1 # dtRho = Fo*Mu

#Paramêtres numériques
n_t=5001 #nombre de pas de temps
tf=25 # temps final de la simulation
xf = 100 #longeur de la simulation
n_x =500 #nombres de points de la simulation

#Données initiales
rho0=np.zeros(n_x) #rho initial
mu0=np.zeros(n_x) #mu initial
mu0[(n_x//2):(n_x//2 +10)]=.01
c0=np.zeros(n_x)+1 #concentration initiale

def edp_1d_explicite(K, b, F0, rho0, mu0, c0, n_t , tf, xf, n_x):
  dt=tf/(n_t-1)
  dx=xf/(n_x-1)
  X=np.linspace(0,xf,n_x)
  T=np.linspace(0,tf,n_t)
  Mu=np.zeros((n_t,n_x))
  Rho=np.zeros((n_t,n_x))
  C=np.zeros((n_t,n_x))
  Mu[0]=mu0
  Rho[0]=rho0
  C[0]=c0
  #Résolution du schema éxplicite
  for n in range(0,n_t-1):
    RHS=np.zeros(n_x)
    alpha=-C[n]*dt*(1+dt*F0)+dt*Rho[n]+1
    RHS[1:-1]= dt*((K/(dx**2))*(Mu[n,:-2]-2*Mu[n,1:-1]+Mu[n,2:])+C[n,1:-1]*Rho[n,1:-1])
    RHS[0]= dt*((K/(dx**2))*(-2*Mu[n,0]+Mu[n,1])+C[n,0]*Rho[n,0])
    RHS[-1]=dt*((K/(dx**2))*(-2*Mu[n,-1]+Mu[n,-2])+C[n,-1]*Rho[n,-1])
    Mu[n+1]=(1/alpha)*(Mu[n]+RHS)
    Rho[n+1]=Rho[n]+dt*F0*Mu[n+1]
    C[n+1]=C[n]/(1 + b*dt*Rho[n])
  return X,T,Mu,Rho,C

def edp_1d_semi_implicite_I(K, b, F0, rho0, mu0, c0, n_t , tf, xf, n_x):
  #Détermination des paramêtres numeriques deltat et deltax
  dt=tf/(n_t-1)
  dx=xf/(n_x-1)
  #Représentation de l'éspace et du temps
  X=np.linspace(0,xf,n_x)
  T=np.linspace(0,tf,n_t)
```

```
  #Initialisation
  Mu=np.zeros((n_t,n_x))
  Rho=np.zeros((n_t,n_x))
  C=np.zeros((n_t,n_x))
  Mu[0]=mu0
  Rho[0]=rho0
  C[0]=c0
  #Résolution du schéma implicite-explicite I
  for n in range(0,n_t-1):
    alpha=-C[n]*dt*(1+dt*F0)+dt*Rho[n]+1
    A=np.diag(-np.ones(n_x-1),-1)+np.diag(2*np.ones(n_x),0)+np.diag(-np.ones(n_x-1)
    ,1)
    A=A*K*dt/(dx**2)
    A+=np.diag(alpha,0)
        A= csc_matrix(A)
    Mu[n+1]= spsolve(A, Mu[n]+dt*C[n]*Rho[n])
    Rho[n+1]=Rho[n]+dt*F0*Mu[n+1]
    C[n+1]=C[n]/(1 + b*dt*Rho[n])
  return X,T,Mu,Rho,C

def edp_1d_semi_implicite_II(K, b, F0, rho0, mu0, c0, n_t , tf, xf, n_x):
  #Détermination des paramètres numériques deltat et deltax
  dt=tf/(n_t-1)
  dx=xf/(n_x-1)
  #Représentation de l'éspace et du temps
  X=np.linspace(0,xf,n_x)
  T=np.linspace(0,tf,n_t)
  #Initialisation
  Mu=np.zeros((n_t,n_x))
  Rho=np.zeros((n_t,n_x))
  C=np.zeros((n_t,n_x))
  Mu[0]=mu0
  Rho[0]=rho0
  C[0]=c0
  #Résolution du schéma implicite-explicite II
  for n in range(0,n_t-1):
    #Matrice du Laplacien
    A=np.diag(-np.ones(n_x-1),-1)+np.diag(2*np.ones(n_x),0)+np.diag(-np.ones(n_x-1)
    ,1)
    A=A*K*dt/(dx**2) #Laplacien Numerique
    #Ajout des termes implicites
    alpha=-C[n]*dt*(1+dt*F0)+1
    A+=np.diag(alpha,0)
        A= csc_matrix(A)
    #Résolution du système implicite
    Mu[n+1]= spsolve(A, Mu[n]+dt*C[n]*Rho[n]-dt*Mu[n]*Rho[n])
    Rho[n+1]=Rho[n]+dt*F0*Mu[n+1]
    C[n+1]=C[n]/(1 + b*dt*Rho[n])
  return X,T,Mu,Rho,C

X,T,Mu,Rho,C= edp_1d_semi_implicite_I(K, b, F0, rho0, mu0, c0, n_t , tf, xf, n_x)

#Valeur de rho a l'infini
rho_inf = Rho[n_t-1,(n_x//2)]
print(rho_inf)


def speed(X,Rho):
    #Position du front
```

```python
    argmed=np.zeros(n_t)
    for i in range(n_t):
        argmed[i]= X[(n_x//2)+np.min(np.where(np.append(Rho[i,(n_x//2):],[0])<rho_inf/2))]
    #Vitesse du front
    S = (argmed[(n_t//2)+1:]-argmed[(n_t//2):-1])*((n_t-1)/tf)
    s= np.average(S)
    return s
s=0
s = speed(X,Rho)
print('La vitesse de propagation de la simulation est s=',s)
s_theorique = np.sqrt(K*((18*F0+4)+np.sqrt(((18*F0+4)**2)+108*(1+4*F0)*(F0**2)))/(2*(1+4*F0)))
#Attention, ceci est pour C0=1
print('La vitesse théorique de propagation est s_theorique=', s_theorique)

#Animation
fig = plt.figure()

ax = plt.axes(xlim=(0, xf), ylim=(0, rho_inf+1))
line, = ax.plot([], [], lw=2)
line2, = ax.plot([], [], lw=2)
line3, = ax.plot([], [], lw=2)
line4, = ax.plot([], [], lw=2)
time_text = ax.text(0.02, 0.92, '', transform=ax.transAxes)
legend_text = ax.text(0.80, 0.82, '', transform=ax.transAxes)

def init():
    line.set_data([], [])
    line2.set_data([], [])
    line3.set_data([], [])
    line4.set_data([], [])
    time_text.set_text('')
    legend_text.set_text('')
    return line,line2,line3,line4, time_text, legend_text


def animate(i):
    line.set_data(X, C[i])
    line2.set_data(X, Rho[i])
    line3.set_data(X, Mu[i])
    #line4.set_data(50+((i*s)*tf/(n_t-1)),np.linspace(0,rho_inf+1,10))
    time_text.set_text('time = {0:.1f}\n K={1}, b={2}, F0={3} '.format(T[i],K,b,F0))
    legend_text.set_text('Rho=Orange \nMu=Green \nC=Blue\ns={0:.3f}'.format(s))
    return line,line2, line3,line4, time_text, legend_text


anim = animation.FuncAnimation(fig, animate, init_func=init,
                               frames=(n_t-1), interval=(tf*200)/(n_t-1), blit=True)



#anim.save('EDP_1D.gif',writer='imagemagick', fps=30)
plt.show()
```

edp_1d.py

### 2.2.2 Résultat de la simulation de l'EDP en 1D

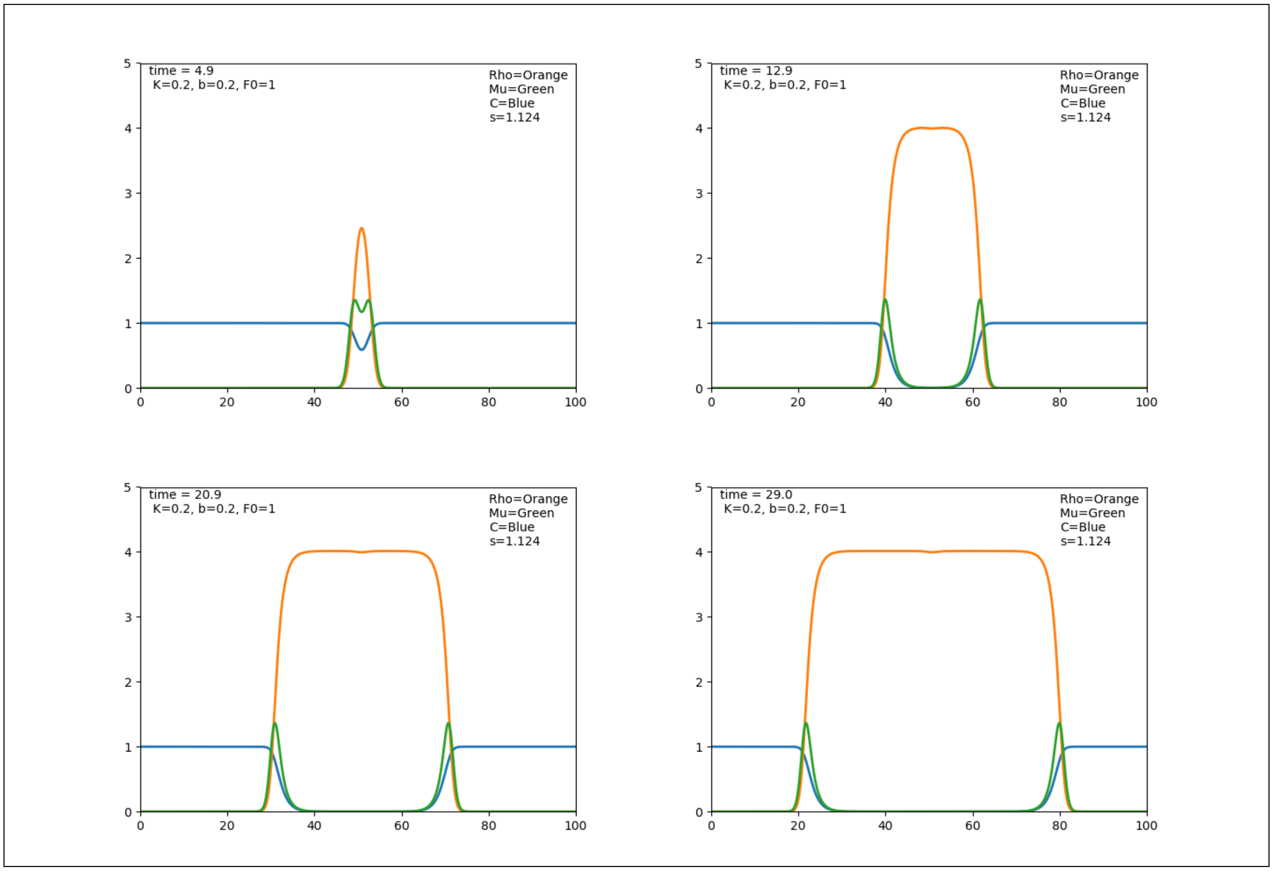


Figure 2: Résolution du schéma semi implicite I pour l'EDP en 1D