

Sur les schémas de l'équation “KPP avec mémoire”

Liam Toran

Contents

1 Schémas et Positivité	1
1.1 Pour l'équation différentielle ordinaire	1
1.1.1 Schéma semi-implicite I pour l'EDO	1
1.1.2 Schéma semi-implicite II pour l'EDO	2
1.2 Pour l'équation aux dérivées partielles	2
1.2.1 Schéma semi-implicite I pour l'EDP	2
2 Résolution numérique	3
2.1 Résolution de l'EDO	3
2.1.1 Code de résolution de l'EDO	3
2.1.2 Résultat de la simulation de l'EDO	6
2.2 Résolution de l'EDP en 1D	7
2.2.1 Code de la résolution de l'EDP en 1D	7
2.2.2 Résultat de la simulation de l'EDP en 1D	10

1 Schémas et Positivité

On a le modèle suivant (“KPP avec mémoire”):

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

1.1 Pour l'équation différentielle ordinaire

Sans dépendance spatiale:

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

1.1.1 Schéma semi-implicite I pour l'EDO

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

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

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:

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

1.1.2 Schéma semi-implicite II pour l'EDO

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

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

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:

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

et

$$\boxed{\rho^n < \frac{1}{\Delta t}} \quad (7)$$

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:

$$\boxed{\begin{cases} \mu^{n+1} = \mu^n + K \Delta t \frac{\mu^{n+1} - 2\mu^n + \mu^{n-1}}{\Delta x^2} + \Delta t(C^n(\mu^{n+1} + \rho^{n+1}) - \mu^{n+1} \rho^n) \\ \rho^{n+1} = \rho^n + \Delta t(F_0 \mu^{n+1}) \\ C^{n+1} = C^n - \Delta t(b \rho^{n+1} C^{n+1}) \end{cases}} \quad (8)$$

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 $-\Delta$:

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

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

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

2 Résolution numérique

2.1 Résolution de l'EDO

2.1.1 Code de résolution de l'EDO

```
1000 import matplotlib.pyplot as plt
1001 from scipy.integrate import ode
1002
1003 b=.1 # dtC=-b*rho*C
1004 F0=1 # dtRho = F0*Mu
1005 tf=50 # temps final de la simulation
1006 rho0=0 #rho initial
1007 mu0=0.01 #mu initial
1008 c0=1 #concentration initiale
1009 n=1000 #nombre de pas de temps
1010
1011 #Résolution du schéma explicite
1012 def euler_explicite_edo(b, F0, tf, rho0, mu0, c0, n):
1013     #t0<t1 , temps etudies ,
1014     #rho0, mu0,c0 reels positifs: condition initiale
1015     #n entier(nombre d'iterations)
1016     h=tf/n #pas Deltat
1017     rho=rho0
1018     mu=mu0
1019     c=c0
1020     t=0
1021     Rho=[rho0]
1022     Mu=[mu0]
1023     C=[c0]
1024     T=[t]
1025     for k in range(n):
1026         new_mu = mu + h*(c*(mu+rho)-mu*rho)
1027         new_rho = rho + h*F0*mu
1028         new_c = c - h*b*rho*c
1029         mu=new_mu
1030         rho=new_rho
1031         c=new_c
1032         t=t+h
1033         Mu.append(new_mu)
1034         Rho.append(new_rho)
1035         C.append(new_c)
1036         T.append(t)
1037     return T,Mu,Rho,C
1038
1039 #Résolution du schéma semi- implicite I
1040 def euler_semi_I_edo(b, F0, tf, rho0, mu0, c0, n):
1041     #t0<t1 , temps etudies ,
1042     #rho0, mu0,c0 reels positifs: condition initiale
1043     #n entier(nombre d'iterations)
1044     h=tf/n #pas Deltat
1045     rho=rho0
1046     mu=mu0
1047     c=c0
1048     t=0
1049     Rho=[rho0]
1050     Mu=[mu0]
1051     C=[c0]
```

```

1052 T=[0]
      for k in range(n):
1054     new_mu = (mu + h*c*rho)/(1+h*rho-h*c*(1+h*F0))
        new_rho = rho + h*F0*new_mu
1056     new_c = c/(1 + b*h*new_rho)
        mu=new_mu
1058     rho=new_rho
        c=new_c
1060     t=t+h
        Mu.append(new_mu)
1062     Rho.append(new_rho)
        C.append(new_c)
1064     T.append(t)
      return T,Mu,Rho,C
1066
1067 #Résolution du schéma semi- implicite II
1068 def euler_semi_II_edo(b, F0, tf, rho0, mu0, c0, n):
      #t0<t1 , temps etudies ,
1070     #rho0, mu0,c0 reels positifs: condition initiale
      #n entier(nombre d'iterations)
1072     t=0
      h=tf/n #pas Deltat
1074     rho=rho0
      mu=mu0
1076     c=c0
      Rho=[rho0]
1078     Mu=[mu0]
      C=[c0]
1080     T=[0]
      for k in range(n):
1082         new_mu = (mu + h*c*rho-h*rho*mu)/(1-h*c*(1+h*F0))
            new_rho = rho + h*F0*new_mu
1084             new_c = c/(1 + b*h*new_rho)
                mu=new_mu
1086                 rho=new_rho
                    c=new_c
1088                     t=t+h
                        Mu.append(new_mu)
1090                         Rho.append(new_rho)
                            C.append(new_c)
1092                             T.append(t)
              return T,Mu,Rho,C
1094
1095 #Programmation de la méthode de Newton-Raphson
1096 def newton(f, gradf, newton_steps, x0):
      x=x0
1098     for k in range(newton_steps):
        x=x-f(x)/gradf(x)
1100     return x
1101 #Résolution du schéma implicite
1102 def euler_implicite_edo(b, F0, tf, rho0, mu0, c0, n):
      #t0<t1 , temps étudiés ,
1104     #rho0, mu0,c0 reels positifs: conditions initiale
      #n entier(nombre d'itérations)
1106     t=0
      newton_steps=10 #nombre d'itérations de la méthode de Newton-Raphson pour le
        calcul implicite
1108     h=tf/n #pas deltat
      rho=rho0

```

```

1110 mu=mu0
1111 c=c0
1112 Rho=[rho0]
1113 Mu=[mu0]
1114 C=[c0]
1115 T=[0]
1116 for k in range(n):
1117     #Calcul de new_mu par methode de Newton Raphson
1118     #coefficients du polynome d'ordre 3 en new_mu
1119     alpha = -h**4*F0**2*b
1120     beta= -F0*h**2*(b+1+2*rho*b*h)
1121     gamma= -(1+b*h*rho)+b*h**2*F0*mu+h*(c*(1+h*F0)-rho*(1+b*h*rho))
1122     delta=(1+b*h*rho)*mu + h*c*rho
1123     def P(X):
1124         return alpha*X**3+beta*X**2+gamma*X+delta
1125     def gradP(X):
1126         return 3*alpha*X**2+2*beta*X+gamma
1127     new_mu=newton(P,gradP,newton_steps,mu)
1128     new_rho = rho + h*F0*new_mu
1129     new_c = c/(1 + b*h*new_rho)
1130     mu=new_mu
1131     rho=new_rho
1132     c=new_c
1133     t=t+h
1134     Mu.append(new_mu)
1135     Rho.append(new_rho)
1136     C.append(new_c)
1137     T.append(t)
1138 return T,Mu,Rho,C

1140 #Utilisation des libraries python (scipy) pour résoudre l'EDO
1141 def black_box_edo(b, F0, tf, rho0, mu0, c0, n):
1142     def f(t,y,arg1,arg2):
1143         mu=y[0]
1144         rho=y[1]
1145         c=y[2]
1146         return [c*(mu+rho)-mu*rho, F0*mu, -b*rho*c]

1147     r = ode(f).set_integrator('zvode', method='adams')
1148     r.set_initial_value([mu0,rho0,c0],0).set_f_params(F0,b)
1149     dt=tf/(n-1)
1150     Rho=[rho0]
1151     Mu=[mu0]
1152     C=[c0]
1153     t=0
1154     T=[0]
1155     while r.t < tf:
1156         mu,rho,c = r.integrate(r.t+dt)
1157         Mu.append(mu)
1158         Rho.append(rho)
1159         C.append(c)
1160         T.append(r.t)
1161     return T,Mu,Rho,C

1164
1166 T,Mu,Rho,C = black_box_edo(b, F0, tf, rho0, mu0, c0, n)
1167 #Tracé des solutions
1168 plt.subplot(221)

```

```

plt.plot(T,Mu)
1170 plt.ylabel('mu')
plt.xlabel('t')
1172 plt.subplot(222)
plt.plot(T,Rho)
1174 plt.ylabel('rho')
plt.subplot(223)
1176 plt.plot(T,C)
plt.ylabel('C')
1178 plt.show()

```

edo.py

2.1.2 Résultat de la simulation de l'EDO

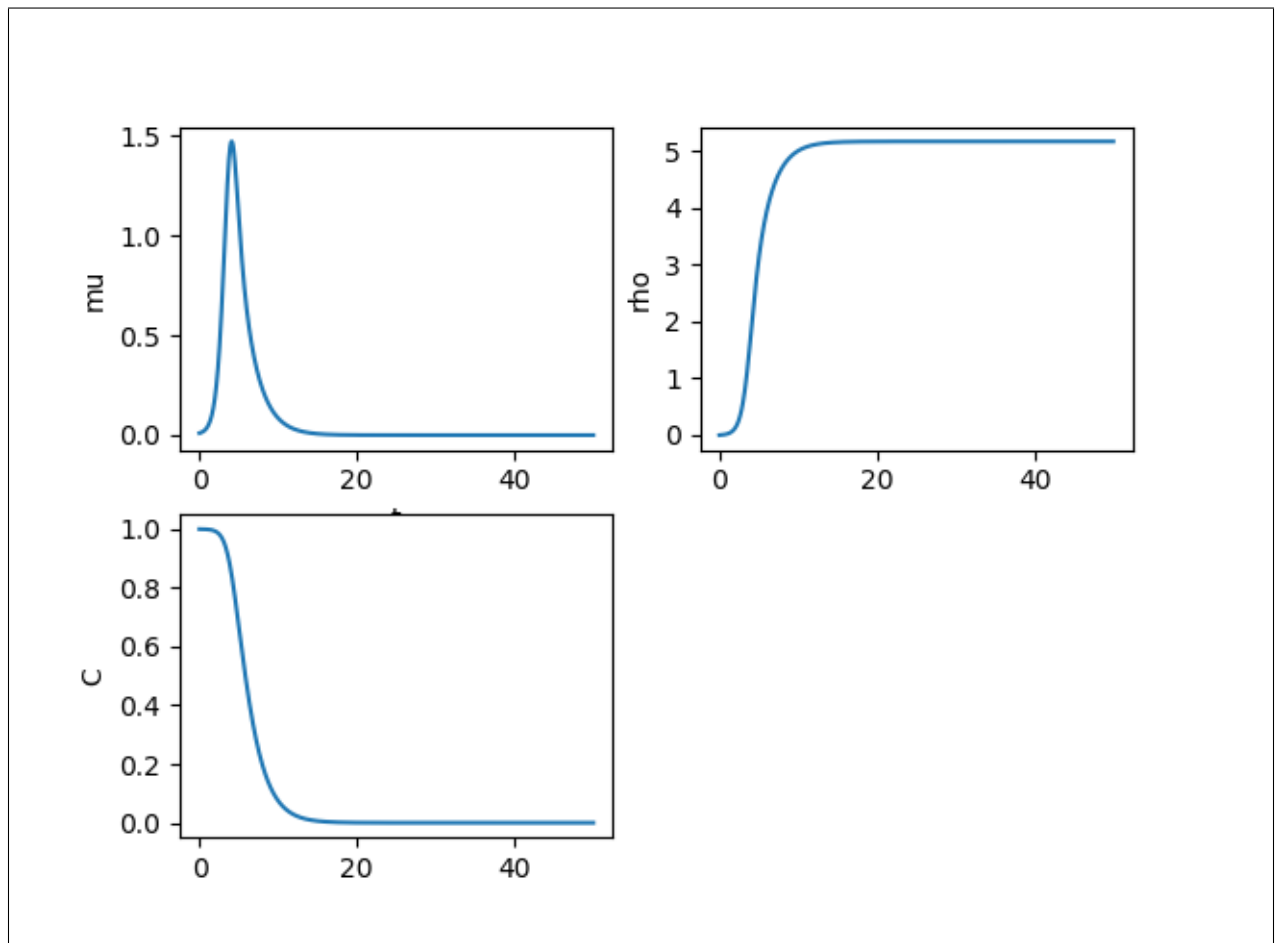


Figure 1: Résolution du schéma implicite pour l'EDO

2.2 Résolution de l'EDP en 1D

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

```
1000 import matplotlib.pyplot as plt
1001 import numpy as np
1002 import scipy.sparse as sp
1003 from scipy.sparse.linalg.dsolve import spsolve
1004 import matplotlib.animation as animation

1006 #Coefficients physiques
1007 K=.2 #coefficient diffusion
1008 b=.2 # dtC=-b*rho*C
1009 F0=1 # dtRho = Fo*Mu
1010
1011 #Paramètres numériques
1012 n_t=501 #nombre de pas de temps
1013 tf=30 # temps final de la simulation
1014 xf = 100 #longueur de lasimulation
1015 n_x =600 #nombres de points de la simulation
1016
1017 #Données initiales
1018 rho0=np.zeros(n_x) #rho initial
1019 mu0=np.zeros(n_x) #mu initial
1020 mu0[(n_x//2):(n_x//2 +10)]=.01
1021 c0=np.zeros(n_x)+1 #concentration initiale
1022
1023 def edp_1d_explicite(K, b, F0, rho0, mu0, c0, n_t , tf, xf, n_x):
1024     dt=tf/(n_t-1)
1025     dx=xf/(n_x-1)
1026     X=np.linspace(0,xf,n_x)
1027     T=np.linspace(0,tf,n_t)
1028     Mu=np.zeros((n_t,n_x))
1029     Rho=np.zeros((n_t,n_x))
1030     C=np.zeros((n_t,n_x))
1031     Mu[0]=mu0
1032     Rho[0]=rho0
1033     C[0]=c0
1034     #Résolution du schema explicite
1035     for n in range(0,n_t-1):
1036         RHS=np.zeros(n_x)
1037         alpha=-C[n]*dt*(1+dt*F0)+dt*Rho[n]+1
1038         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])
1039         RHS[0]= dt*((K/(dx**2))*(-2*Mu[n,0]+Mu[n,1])+C[n,0]*Rho[n,0])
1040         RHS[-1]=dt*((K/(dx**2))*(-2*Mu[n,-1]+Mu[n,-2])+C[n,-1]*Rho[n,-1])
1041         Mu[n+1]=(1/alpha)*(Mu[n]+RHS)
1042         Rho[n+1]=Rho[n]+dt*F0*Mu[n+1]
1043         C[n+1]=C[n]/(1 + b*dt*Rho[n])
1044     return X,T,Mu,Rho,C

1046 def edp_1d_semi_implicite_I(K, b, F0, rho0, mu0, c0, n_t , tf, xf, n_x):
1047     #Détermination des paramètres numeriques deltat et deltax
1048     dt=tf/(n_t-1)
1049     dx=xf/(n_x-1)
1050     #Représentation de l'espace et du temps
1051     X=np.linspace(0,xf,n_x)
1052     T=np.linspace(0,tf,n_t)
1053     #Initialisation
```



```

1054 Mu=np.zeros((n_t,n_x))
1055 Rho=np.zeros((n_t,n_x))
1056 C=np.zeros((n_t,n_x))
1057 Mu[0]=mu0
1058 Rho[0]=rho0
1059 C[0]=c0
1060 #Résolution du schéma implicite-explicite I
1061 for n in range(0,n_t-1):
1062     alpha=-C[n]*dt*(1+dt*F0)+dt*Rho[n]+1
1063     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)
1064     A=A*K*dt/(dx**2)
1065     A+=np.diag(alpha,0)
1066     Mu[n+1]=spsolve(A, Mu[n]+dt*C[n]*Rho[n])
1067     Rho[n+1]=Rho[n]+dt*F0*Mt[n+1]
1068     C[n+1]=C[n]/(1 + b*dt*Rho[n])
1069 return X,T,Mu,Rho,C
1070
1071 def edp_1d_semi_implicite_II(K, b, F0, rho0, mu0, c0, n_t , tf, xf, n_x):
1072     #Détermination des paramètres numériques deltat et deltax
1073     dt=tf/(n_t-1)
1074     dx=xf/(n_x-1)
1075     #Représentation de l'espace et du temps
1076     X=np.linspace(0,xf,n_x)
1077     T=np.linspace(0,tf,n_t)
1078     #Initialisation
1079     Mu=np.zeros((n_t,n_x))
1080     Rho=np.zeros((n_t,n_x))
1081     C=np.zeros((n_t,n_x))
1082     Mu[0]=mu0
1083     Rho[0]=rho0
1084     C[0]=c0
1085     #Résolution du schéma implicite-explicite II
1086     for n in range(0,n_t-1):
1087         #Matrice du Laplacien
1088         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)
1089         A=A*K*dt/(dx**2) #Laplacien Numerique
1090         #Ajout des termes implicites
1091         alpha=-C[n]*dt*(1+dt*F0)+1
1092         A+=np.diag(alpha,0)
1093         #Résolution du système implicite
1094         Mu[n+1]=spsolve(A, Mu[n]+dt*C[n]*Rho[n]-dt*Mt[n]*Rho[n])
1095         Rho[n+1]=Rho[n]+dt*F0*Mt[n+1]
1096         C[n+1]=C[n]/(1 + b*dt*Rho[n])
1097     return X,T,Mu,Rho,C
1098
1099 X,T,Mu,Rho,C= edp_1d_semi_implicite_I(K, b, F0, rho0, mu0, c0, n_t , tf, xf, n_x)
1100
1101 #Valeur de rho a l'infini
1102 rho_inf = Rho[n_t-1,(n_x//2)]
1103 print(rho_inf)
1104
1105 def speed(X,Rho):
1106     #Position du front
1107     argmed=np.zeros(n_t)
1108     for i in range(n_t):

```

```

1110     argmed[i] = X[(n_x//2)+np.min(np.where(np.append(Rho[i,(n_x//2):],[0])<rho_inf/2)
1111     )]
1112     #Vitesse du front
1113     S = (argmed[(n_t//2)+1:] - argmed[(n_t//2):-1]) * ((n_t-1)/tf)
1114     s = np.average(S)
1115     return s
1116 s=0
1117 s = speed(X,Rho)
1118 print(s)
1119
1120 #Animation
1121 fig = plt.figure()
1122
1123 ax = plt.axes(xlim=(0, xf), ylim=(0, rho_inf+1))
1124 line , = ax.plot([], [], lw=2)
1125 line2 , = ax.plot([], [], lw=2)
1126 line3 , = ax.plot([], [], lw=2)
1127 line4 , = ax.plot([], [], lw=2)
1128 time_text = ax.text(0.02, 0.92, '', transform=ax.transAxes)
1129 legend_text = ax.text(0.80, 0.82, '', transform=ax.transAxes)
1130
1131 def init():
1132     line.set_data([], [])
1133     line2.set_data([], [])
1134     line3.set_data([], [])
1135     line4.set_data([], [])
1136     time_text.set_text('')
1137     legend_text.set_text('')
1138     return line, line2, line3, line4, time_text, legend_text
1139
1140 def animate(i):
1141     line.set_data(X, C[i])
1142     line2.set_data(X, Rho[i])
1143     line3.set_data(X, Mu[i])
1144     #line4.set_data(50+((i*s)*tf/(n_t-1)), np.linspace(0, rho_inf+1, 10))
1145     time_text.set_text('time = {0:.1f}\n K={1}, b={2}, F0={3}'.format(T[i], K, b, F0))
1146     legend_text.set_text('Rho=Orange \nMu=Green \nC=Blue\ns={0:.3f}'.format(s))
1147     return line, line2, line3, line4, time_text, legend_text
1148
1149 anim = animation.FuncAnimation(fig, animate, init_func=init,
1150                               frames=(n_t-1), interval=(tf*200)/(n_t-1), blit=True)
1151
1152
1153 #anim.save('EDP_1D.gif', writer='imagemagick', fps=30)
1154 plt.show()

```

edp_1d.py

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

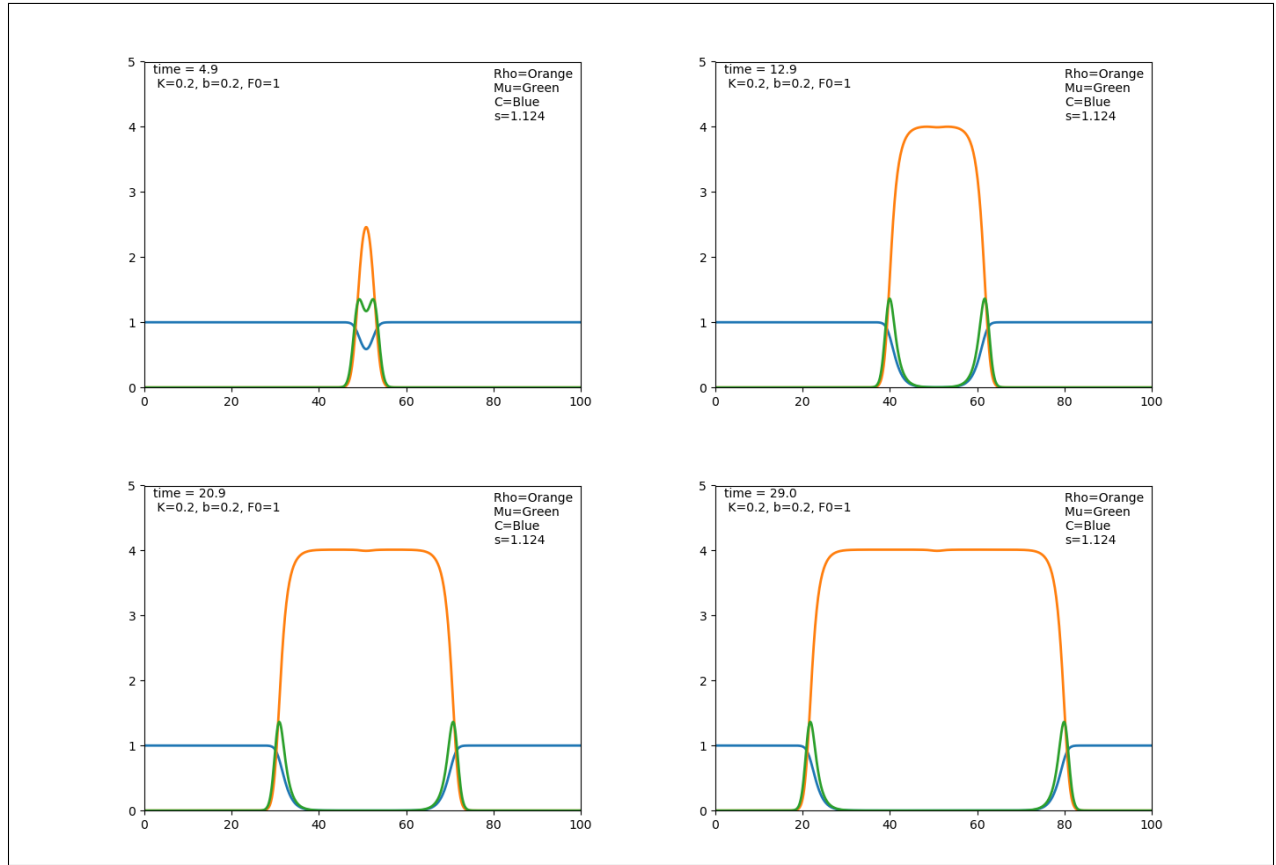


Figure 2: Résolution du schéma semi implicite I pour l'EDP en 1D