

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

Liam Toran

Contents

1 Schémas et Positivité

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

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

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:

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

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 import numpy as np

1004 b=.1 # dtC=-b*rho*C
1005 F0=1 # dtRho = F0*Mu
1006 tf=50 # temps final de la simulation
1007 rho0=0 #rho initial
1008 mu0=.1 #mu initial
1009 c0=1 #concentration initiale
1010 n=1000 #nombre de pas de temps

1012 #Résolution du schéma explicite
1013 def euler_explicite_edo(b, F0, tf, rho0, mu0, c0, n):
1014     #t0<t1 , temps etudies ,
1015     #rho0, mu0,c0 reels positifs: condition initiale
1016     #n entier(nombre d'iterations)
1017     h=tf/n #pas Deltat
1018     rho=rho0
1019     mu=mu0
1020     c=c0
1021     t=0
1022     Rho=[rho0]
1023     Mu=[mu0]
1024     C=[c0]
1025     T=[t]
1026     for k in range(n):
1027         new_mu = mu + h*(c*(mu+rho)-mu*rho)
1028         new_rho = rho + h*F0*mu
1029         new_c = c - h*b*rho*c
1030         mu=new_mu
1031         rho=new_rho
1032         c=new_c
1033         t=t+h
1034         Mu.append(new_mu)
1035         Rho.append(new_rho)
1036         C.append(new_c)
1037         T.append(t)
1038     return T,Mu,Rho,C

1040 #Résolution du schéma semi- implicite I
1041 def euler_semi_I_edo(b, F0, tf, rho0, mu0, c0, n):
1042     #t0<t1 , temps etudies ,
1043     #rho0, mu0,c0 reels positifs: condition initiale
1044     #n entier(nombre d'iterations)
1045     h=tf/n #pas Deltat
1046     rho=rho0
1047     mu=mu0
1048     c=c0
1049     t=0
1050     Rho=[rho0]
1051     Mu=[mu0]
```

```

1052 C=[c0]
1053 T=[0]
1054 for k in range(n):
1055     new_mu = (mu + h*c*rho)/(1+h*rho-h*c*(1+h*F0))
1056     new_rho = rho + h*F0*new_mu
1057     new_c = c/(1 + b*h*new_rho)
1058     mu=new_mu
1059     rho=new_rho
1060     c=new_c
1061     t=t+h
1062     Mu.append(new_mu)
1063     Rho.append(new_rho)
1064     C.append(new_c)
1065     T.append(t)
1066 return T,Mu,Rho,C

1068 #Résolution du schéma semi- implicite II
1069 def euler_semi_II_edo(b, F0, tf, rho0, mu0, c0, n):
1070     #t0<t1 , temps etudies ,
1071     #rho0, mu0,c0 reels positifs: condition initiale
1072     #n entier(nombre d'iterations)
1073     t=0
1074     h=tf/n #pas Deltat
1075     rho=rho0
1076     mu=mu0
1077     c=c0
1078     Rho=[rho0]
1079     Mu=[mu0]
1080     C=[c0]
1081     T=[0]
1082     for k in range(n):
1083         new_mu = (mu + h*c*rho-h*rho*mu)/(1-h*c*(1+h*F0))
1084         new_rho = rho + h*F0*new_mu
1085         new_c = c/(1 + b*h*new_rho)
1086         mu=new_mu
1087         rho=new_rho
1088         c=new_c
1089         t=t+h
1090         Mu.append(new_mu)
1091         Rho.append(new_rho)
1092         C.append(new_c)
1093         T.append(t)
1094     return T,Mu,Rho,C

1096 #Programmation de la méthode de Newton-Raphson
1097 def newton(f, gradf, newton_steps, x0):
1098     x=x0
1099     for k in range(newton_steps):
1100         x=x-f(x)/gradf(x)
1101     return x
1102 #Résolution du schéma implicite
1103 def euler_implicite_edo(b, F0, tf, rho0, mu0, c0, n):
1104     #t0<t1 , temps étudiés ,
1105     #rho0, mu0,c0 reels positifs: conditions initiale
1106     #n entier(nombre d'itérations)
1107     t=0
1108     newton_steps=10 #nombre d'itérations de la méthode de Newton-Raphson pour le
1109         calcul implicite
1110     h=tf/n #pas deltat

```

```

1110 rho=rho0
1111 mu=mu0
1112 c=c0
1113 Rho=[rho0]
1114 Mu=[mu0]
1115 C=[c0]
1116 T=[0]
1117 for k in range(n):
1118     #Calcul de new_mu par methode de Newton Raphson
1119     #coefficients du polynome d'ordre 3 en new_mu
1120     alpha = -h**4*F0**2*b
1121     beta= -F0*h**2*(b+1+2*rho*b*h)
1122     gamma= -(1+b*h*rho)+b*h**2*F0*mu+h*(c*(1+h*F0)-rho*(1+b*h*rho))
1123     delta=(1+b*h*rho)*mu + h*c*rho
1124     def P(X):
1125         return alpha*X**3+beta*X**2+gamma*X+delta
1126     def gradP(X):
1127         return 3*alpha*X**2+2*beta*X+gamma
1128     new_mu=newton(P,gradP,newton_steps,mu)
1129     new_rho = rho + h*F0*new_mu
1130     new_c = c/(1 + b*h*new_rho)
1131     mu=new_mu
1132     rho=new_rho
1133     c=new_c
1134     t=t+h
1135     Mu.append(new_mu)
1136     Rho.append(new_rho)
1137     C.append(new_c)
1138     T.append(t)
1139 return T,Mu,Rho,C
1140
1141 #Utilisation des libraries python (scipy) pour résoudre l'EDO
1142 def black_box_edo(b, F0, tf, rho0, mu0, c0, n):
1143     def f(t,y,arg1,arg2):
1144         mu=y[0]
1145         rho=y[1]
1146         c=y[2]
1147         return [c*(mu+rho)-mu*rho, F0*mu, -b*rho*c]
1148
1149     r = ode(f).set_integrator('zvode', method='adams')
1150     r.set_initial_value([mu0,rho0,c0],0).set_f_params(F0,b)
1151     dt=tf/(n-1)
1152     Rho=[rho0]
1153     Mu=[mu0]
1154     C=[c0]
1155     t=0
1156     T=[0]
1157     while r.t < tf:
1158         mu,rho,c = r.integrate(r.t+dt)
1159         Mu.append(mu)
1160         Rho.append(rho)
1161         C.append(c)
1162         T.append(r.t)
1163     return T,Mu,Rho,C
1164
1165 T,Mu,Rho,C = black_box_edo(b, F0, tf, rho0, mu0, c0, n)
1166 rho_inf_theorique = .5*(rho0+np.sqrt(rho0**2 + 4*F0*(mu0+(c0/b))))
1167 print(rho_inf_theorique)

```

```

rho_inf= Rho[n-1]
1170 print(rho_inf)
A=[np.log(rho_inf-x) for x in Rho]
1172 B=[-b*y*rho_inf+np.log((F0)/(b*rho_inf)) for y in T]
#Tracé des solutions
1174 plt.subplot(221)
plt.plot(T,Mu)
1176 plt.ylabel('mu')
plt.xlabel('t')
1178 plt.subplot(222)
plt.plot(T,Rho)
1180 plt.ylabel('rho')
plt.subplot(223)
1182 plt.plot(T,C)
plt.ylabel('C')
1184 plt.subplot(224)
plt.plot(T,A)
1186 plt.plot(T,B)
plt.ylabel('log(rho_inf-rho)+b*rho_inf*t')
1188 plt.show()

```

edo.py

2.1.2 Résultat de la simulation de l'EDO

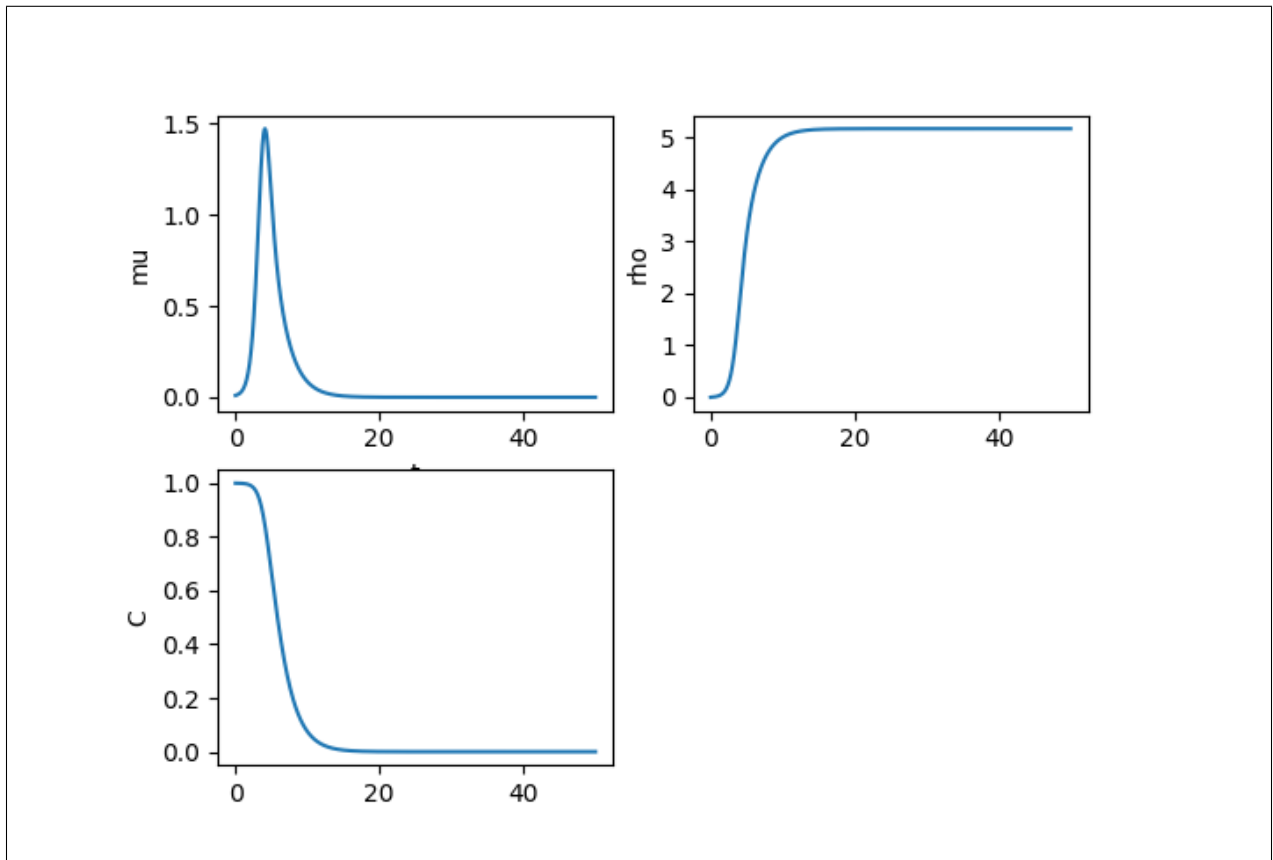


Figure 1: Résolution du schéma implicite pour l'EDO

2.2 Résolution de l'EDP en 1D

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

```
1000 # %load edp_1d.py
1001 import matplotlib.pyplot as plt
1002 import numpy as np
1003 import scipy.sparse as sp
1004 from scipy.sparse.linalg.dsolve import spsolve
1005 import matplotlib.animation as animation
1006
1007 # Coefficients physiques
1008 K=.4 #coefficient diffusion
1009 b=.2 # dtC=-b*rho*C
1010 F0=1 # dtRho = Fo*MU
1011
1012 # Paramètres numériques
1013 n_t=5001 #nombre de pas de temps
1014 t_f=25 # temps final de la simulation
1015 x_f = 100 #longueur de la simulation
1016 n_x =500 #nombres de points de la simulation
1017
1018 #Données initiales
1019 rho0=np.zeros(n_x) #rho initial
1020 mu0=np.zeros(n_x) #mu initial
1021 mu0[(n_x//2):(n_x//2 +10)]=.01
1022 c0=np.zeros(n_x)+1 #concentration initiale
1023
1024 def edp_1d_explicite(K, b, F0, rho0, mu0, c0, n_t, t_f, x_f, n_x):
1025     dt=t_f/(n_t-1)
1026     dx=x_f/(n_x-1)
1027     X=np.linspace(0, x_f, n_x)
1028     T=np.linspace(0, t_f, n_t)
1029     MU=np.zeros((n_t, n_x))
1030     Rho=np.zeros((n_t, n_x))
1031     C=np.zeros((n_t, n_x))
1032     MU[0]=mu0
1033     Rho[0]=rho0
1034     C[0]=c0
1035     #Résolution du schema explicite
1036     for n in range(0, n_t-1):
1037         RHS=np.zeros(n_x)
1038         alpha=-C[n]*dt*(1+dt*F0)+dt*Rho[n]+1
1039         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])
1040         RHS[0]= dt*((K/(dx**2))*(-2*MU[n, 0]+MU[n, 1])+C[n, 0]*Rho[n, 0])
1041         RHS[-1]=dt*((K/(dx**2))*(-2*MU[n, -1]+MU[n, -2])+C[n, -1]*Rho[n, -1])
1042         MU[n+1]=(1/alpha)*(MU[n]+RHS)
1043         Rho[n+1]=Rho[n]+dt*F0*MU[n+1]
1044         C[n+1]=C[n]/(1 + b*dt*Rho[n])
1045     return X, T, MU, Rho, C
1046
1047 def edp_1d_semi_implicite_I(K, b, F0, rho0, mu0, c0, n_t, t_f, x_f, n_x):
1048     #Détermination des paramètres numériques deltat et deltax
1049     dt=t_f/(n_t-1)
1050     dx=x_f/(n_x-1)
1051     #Représentation de l'espace et du temps
1052     X=np.linspace(0, x_f, n_x)
1053     T=np.linspace(0, t_f, n_t)
```

```

1054 #Initialisation
1055 Mu=np.zeros((n_t,n_x))
1056 Rho=np.zeros((n_t,n_x))
1057 C=np.zeros((n_t,n_x))
1058 Mu[0]=mu0
1059 Rho[0]=rho0
1060 C[0]=c0
1061 #Résolution du schéma implicite-explicite I
1062 for n in range(0,n_t-1):
1063     alpha=-C[n]*dt*(1+dt*F0)+dt*Rho[n]+1
1064     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)
1065     A=A*K*dt/(dx**2)
1066     A+=np.diag(alpha,0)
1067     A= csc_matrix(A)
1068     Mu[n+1]= spsolve(A, Mu[n]+dt*C[n]*Rho[n])
1069     Rho[n+1]=Rho[n]+dt*F0*MU[n+1]
1070     C[n+1]=C[n]/(1 + b*dt*Rho[n])
1071 return X,T,Mu,Rho,C
1072
1073 def edp_1d_semi_implicite_II(K, b, F0, rho0, mu0, c0, n_t , tf, xf, n_x):
1074     #Détermination des paramètres numériques deltat et deltax
1075     dt=tf/(n_t-1)
1076     dx=xf/(n_x-1)
1077     #Représentation de l'espace et du temps
1078     X=np.linspace(0,xf,n_x)
1079     T=np.linspace(0,tf,n_t)
1080     #Initialisation
1081     Mu=np.zeros((n_t,n_x))
1082     Rho=np.zeros((n_t,n_x))
1083     C=np.zeros((n_t,n_x))
1084     Mu[0]=mu0
1085     Rho[0]=rho0
1086     C[0]=c0
1087     #Résolution du schéma implicite-explicite II
1088     for n in range(0,n_t-1):
1089         #Matrice du Laplacien
1090         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)
1091         A=A*K*dt/(dx**2) #Laplacien Numerique
1092         #Ajout des termes implicites
1093         alpha=-C[n]*dt*(1+dt*F0)+1
1094         A+=np.diag(alpha,0)
1095         A= csc_matrix(A)
1096         #Résolution du système implicite
1097         Mu[n+1]= spsolve(A, Mu[n]+dt*C[n]*Rho[n]-dt*MU[n]*Rho[n])
1098         Rho[n+1]=Rho[n]+dt*F0*MU[n+1]
1099         C[n+1]=C[n]/(1 + b*dt*Rho[n])
1100     return X,T,Mu,Rho,C
1101
1102 X,T,Mu,Rho,C= edp_1d_semi_implicite_I(K, b, F0, rho0, mu0, c0, n_t , tf, xf, n_x)
1103
1104 #Valeur de rho a l'infini
1105 rho_inf = Rho[n_t-1,(n_x//2)]
1106 print(rho_inf)
1107
1108 def speed(X,Rho):
1109     #Position du front

```



```

1112     argmed=np.zeros(n_t)
1113     for i in range(n_t):
1114         argmed[i]= X[(n_x//2)+np.min(np.where(np.append(Rho[i,(n_x//2):],[0])<rho_inf/2)
1115         )]
1116     #Vitesse du front
1117     S = (argmed[(n_t//2)+1:] - argmed[(n_t//2):-1]) * ((n_t-1)/tf)
1118     s= np.average(S)
1119     return s
1120 s=0
1121 s = speed(X,Rho)
1122 print('La vitesse de propagation de la simulation est s=',s)
1123 s_theorique = np.sqrt(K*((18*F0+4)+np.sqrt(((18*F0+4)**2)+108*(1+4*F0)*(F0**2))))
1124     /(2*(1+4*F0)))
1125 #Attention, ceci est pour C0=1
1126 print('La vitesse théorique de propagation est s_theorique=', s_theorique)
1127
1128 #Animation
1129 fig = plt.figure()
1130
1131 ax = plt.axes(xlim=(0, xf), ylim=(0, rho_inf+1))
1132 line , = ax.plot([], [], lw=2)
1133 line2 , = ax.plot([], [], lw=2)
1134 line3 , = ax.plot([], [], lw=2)
1135 line4 , = ax.plot([], [], lw=2)
1136 time_text = ax.text(0.02, 0.92, '', transform=ax.transAxes)
1137 legend_text = ax.text(0.80, 0.82, '', transform=ax.transAxes)
1138
1139 def init():
1140     line.set_data([], [])
1141     line2.set_data([], [])
1142     line3.set_data([], [])
1143     line4.set_data([], [])
1144     time_text.set_text('')
1145     legend_text.set_text('')
1146     return line, line2, line3, line4, time_text, legend_text
1147
1148 def animate(i):
1149     line.set_data(X, C[i])
1150     line2.set_data(X, Rho[i])
1151     line3.set_data(X, Mu[i])
1152     #line4.set_data(50+((i*s)*tf/(n_t-1)),np.linspace(0,rho_inf+1,10))
1153     time_text.set_text('time = {0:.1f}\n K={1}, b={2}, F0={3}'.format(T[i],K,b,F0))
1154     legend_text.set_text('Rho=Orange \nMu=Green \nC=Blue\ns={0:.3f}'.format(s))
1155     return line, line2, line3, line4, time_text, legend_text
1156
1157 anim = animation.FuncAnimation(fig, animate, init_func=init,
1158                               frames=(n_t-1), interval=(tf*200)/(n_t-1), blit=True)
1159
1160 #anim.save('EDP_1D.gif',writer='imagemagick', fps=30)
1161 plt.show()

```

edp_1d.py

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

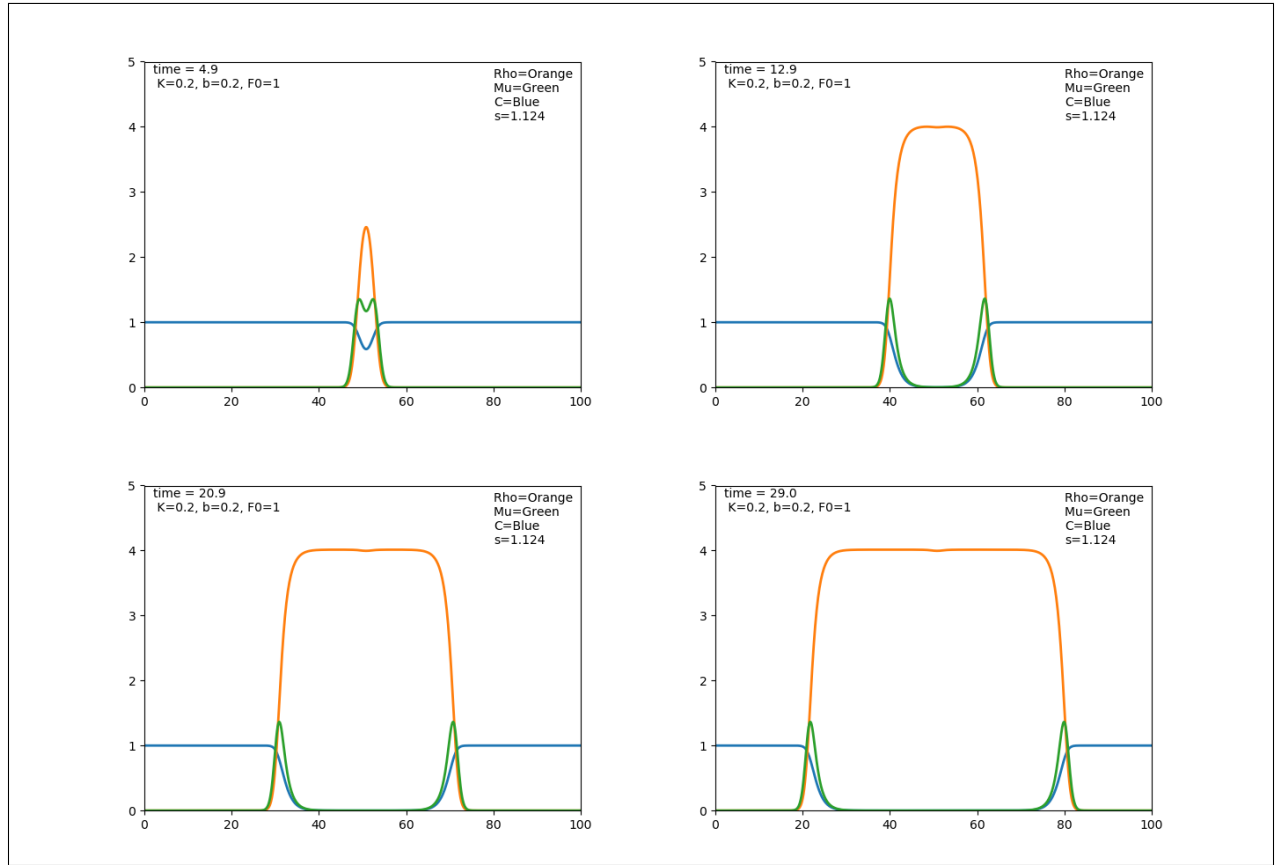


Figure 2: Résolution du schéma semi implicite I pour l'EDP en 1D