TIPE

Modélisation numérique de l’évolution et des déplacements d’une marée noire dans l’océan

Bastien Schoonaert
16002

Objectif du TIPE

Objectif :

Prévoir les déplacements d'une marée noire pour optimiser son ramassage

- Etude de la **dispersion d'un polluant** à la surface d'un **liquide en mouvement.**

- Plusieurs **phénomènes physiques** régissant cette dispersion

SOMMAIRE

I. Introduction et cadre du problème

- Différentes **méthodes de prévention** actuellement en place

- Quelques **caractéristiques importantes**

⇒ on peut **négliger les variations suivant la verticale** et **l'impact du vent**

Notation : Dans toute la suite, nous noterons : $\boldsymbol{n(M,t)}$ **la concentration de particules au point M à l'instant t**

II. Advection

Équation d'advection :

$$\frac{\partial n}{\partial t} + \vec{A}\,\vec{\nabla} n = 0$$

avec $\vec{A}$ la vitesse d'advection et $\vec{\nabla}$ l'opérateur nabla

- **Simulation** pour l'équation d'advection à une dimension : $\frac{\partial n}{\partial t}(x,t) + A\frac{\partial n}{\partial x}(x,t) = 0$

- Cas 2D : $\frac{\partial n}{\partial t}(x,y,t) + A_x\frac{\partial n}{\partial x}(x,y,t) + A_y\frac{\partial n}{\partial y}(x,y,t) = 0$

II. Advection

• **Simulation de résolution informatique** de l'équation d'advection en la discrétisant par **méthode des différences finies**

Discrétisation : $u_i^n = u(x_i, t_n)$

i = indice de l'intervalle d'espace considéré
n = indice de l'intervalle de temps considéré

II. Advection

Valeurs communes utilisées pour chaque représentation à une dimension

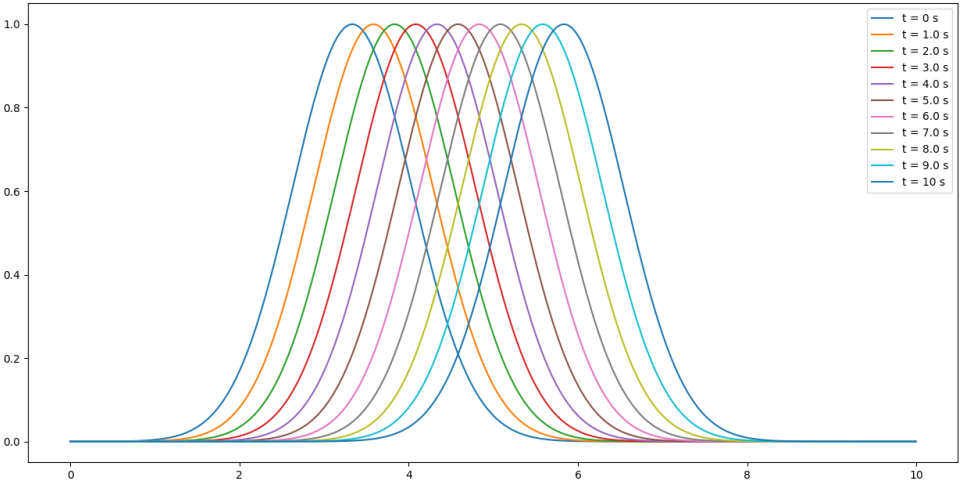
- Vitesse d'advection : $\mathbf{A = 0{,}25\ m.s^{-1}}$
- Longueur du segment : $\mathbf{l = 10\ m}$
- Durée : $\mathbf{t = 10\ s}$
- Nombre d'intervalles de longueur : $\mathbf{nx = 1000}$ $(\Delta x = l/nx)$
- Nombre d'intervalles de temps : $\mathbf{nt = 1000}$ $(\Delta t = t/nt)$

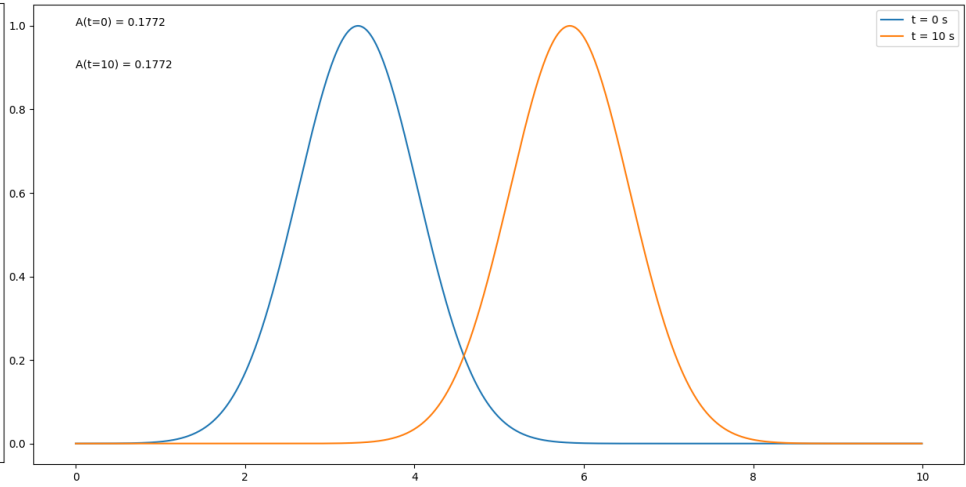
II. Advection

Présentation des résultats : **Abscisse : x** **Ordonnée : n(x,t)** à t fixé

1) **Évolution** au cours du temps
2) **- Courbe bleue :** Répartition **initiale** des particules
   - Courbe orange : Répartition des particules **après un temps T donné** avec valeur de l'aire sous la courbe

Exemples de résultat pour la solution exacte :

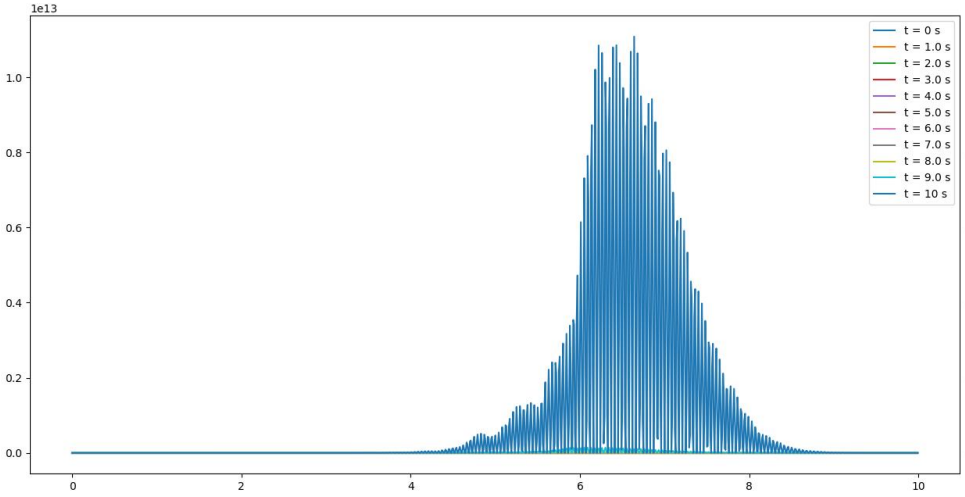


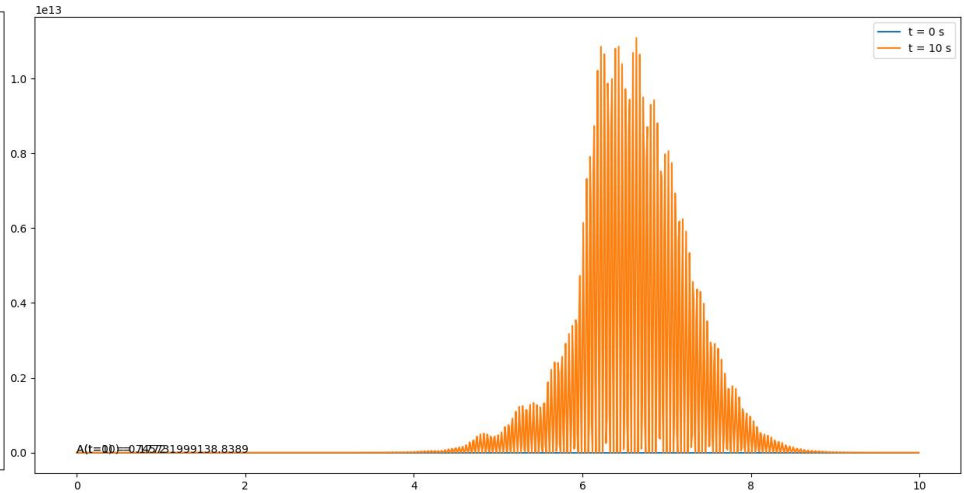


II. Advection

1) Méthode explicite d'ordre 1 :

$$u_i^{n+1} = u_i^n + A\frac{\Delta t}{2\Delta x}\left[u_{i+1}^n - u_{i-1}^n\right]$$

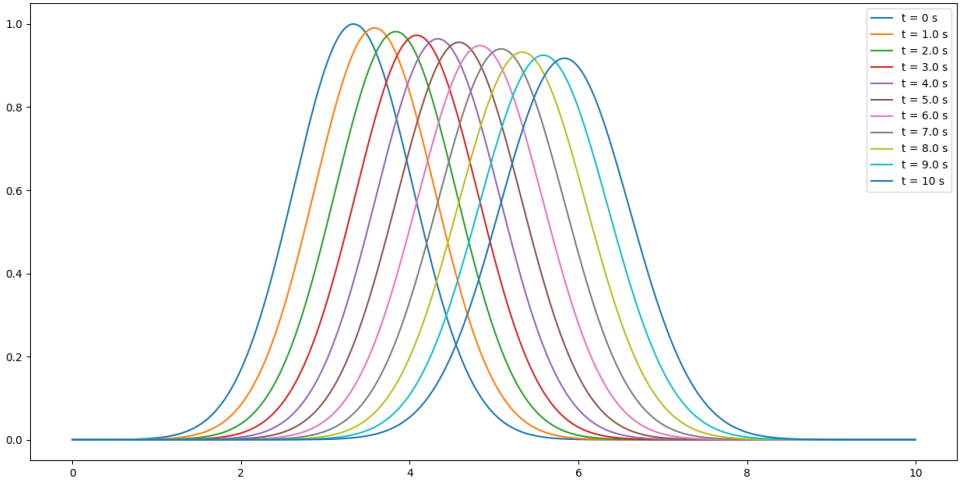


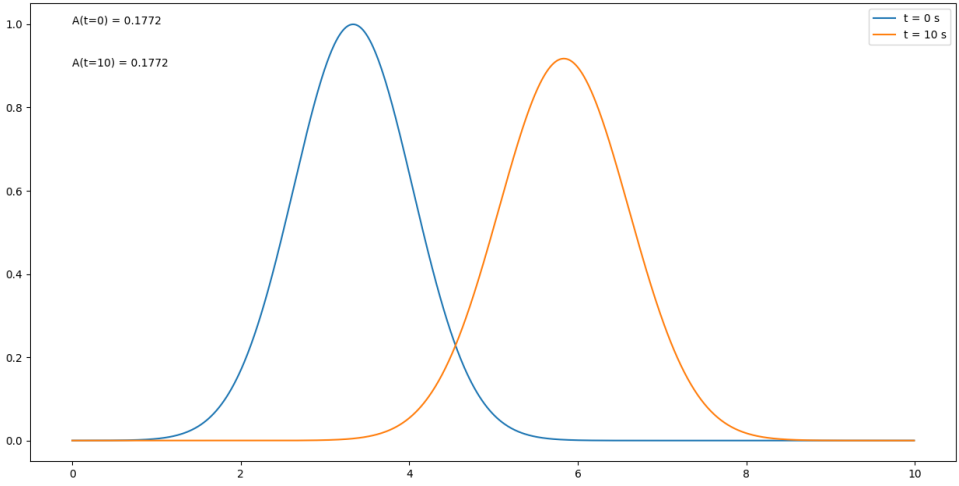


II. Advection

2) Méthode de Lax-Friedrichs :

$$u_i^{n+1} = \frac{1}{2}\left(u_{i+1}^n + u_{i-1}^n\right) - A\frac{\Delta t}{2\Delta x}\left[u_{i+1}^n - u_{i-1}^n\right]$$

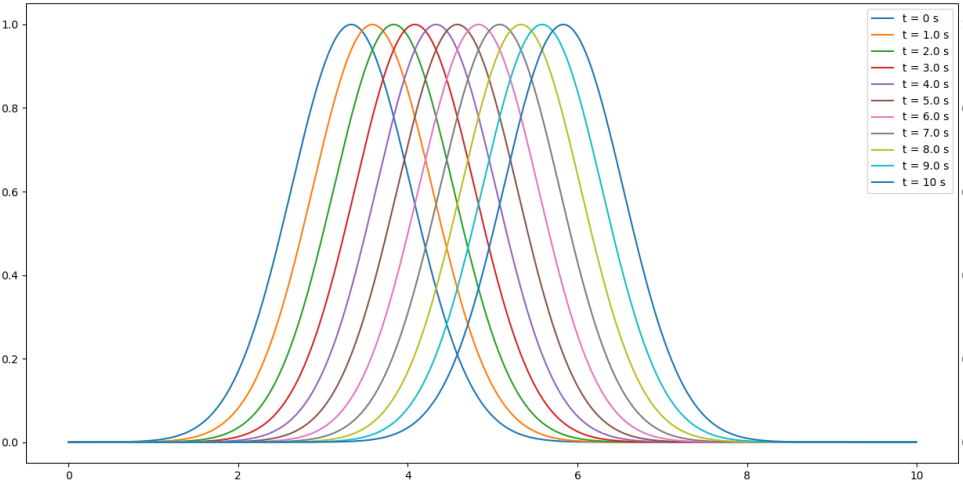


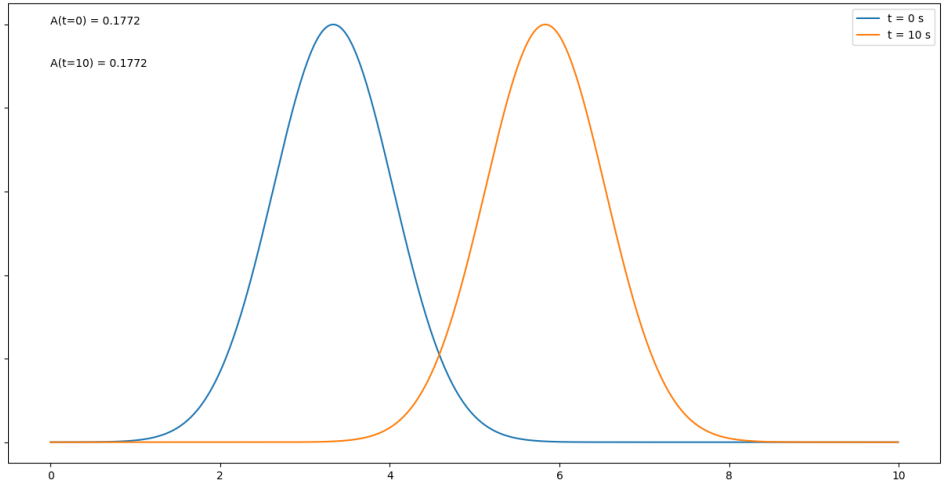


II. Advection

3) Méthode de Lax-Wendroff :

$$u_i^{n+1} = u_i^n + A\frac{\Delta t}{2\Delta x}[u_{i+1}^n - u_{i-1}^n] + A^2\frac{\Delta t^2}{2\Delta x^2}[u_{i+1}^n - 2u_i^n + u_{i-1}^n]$$

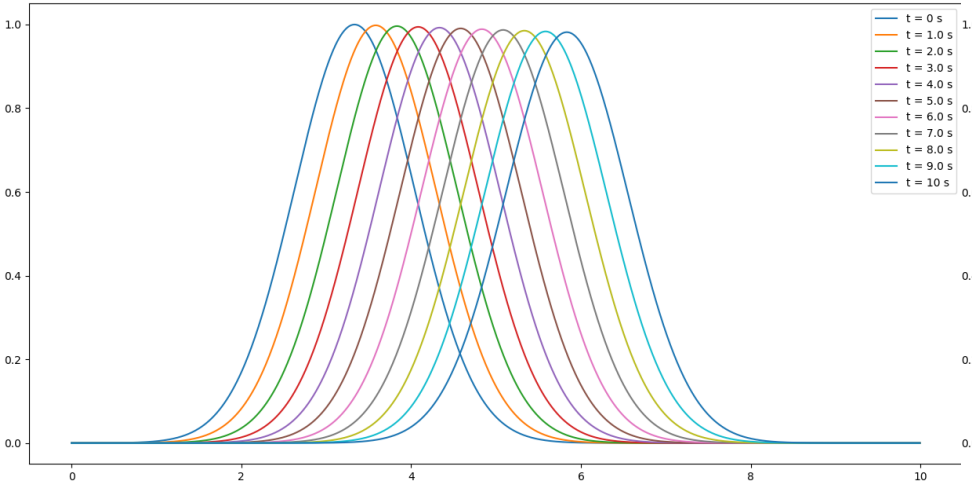


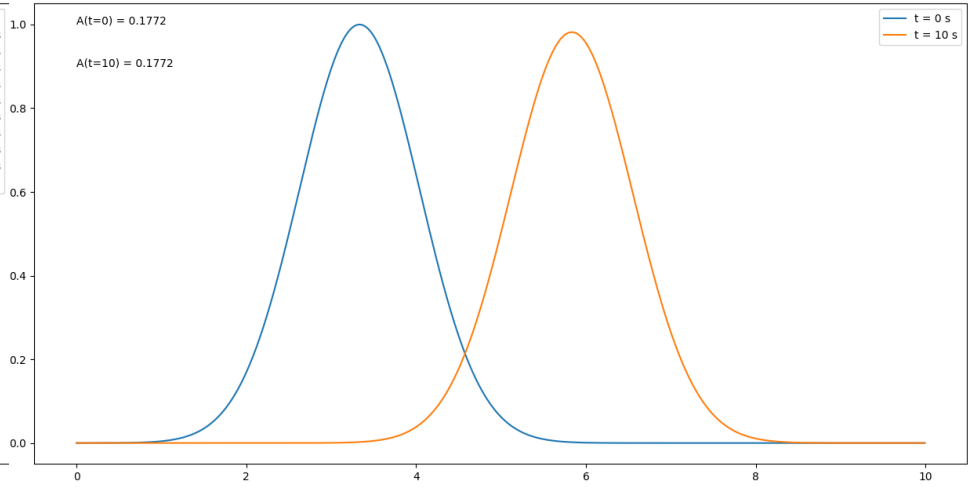


II. Advection

4) Méthode Upwind :

$$u_i^{n+1} = u_i^n + A\frac{\Delta t}{2\Delta x}[u_i^n - u_{i-1}^n]$$

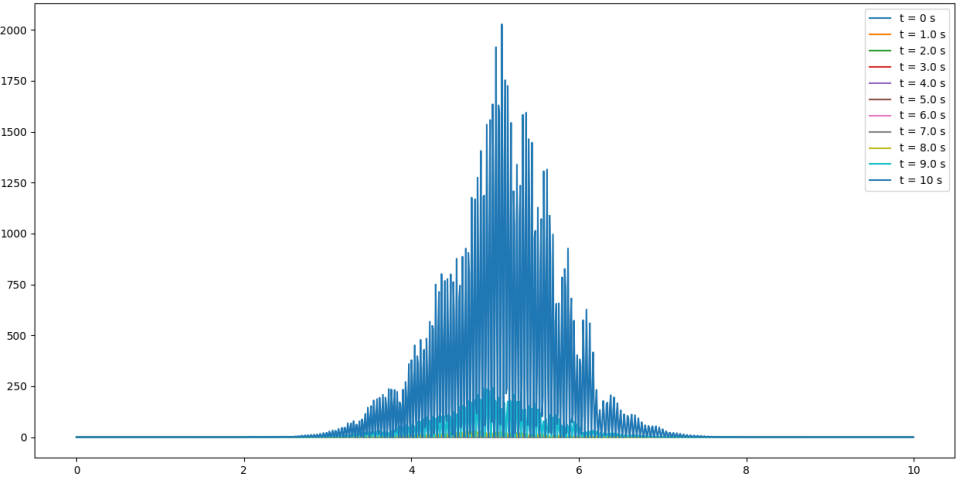


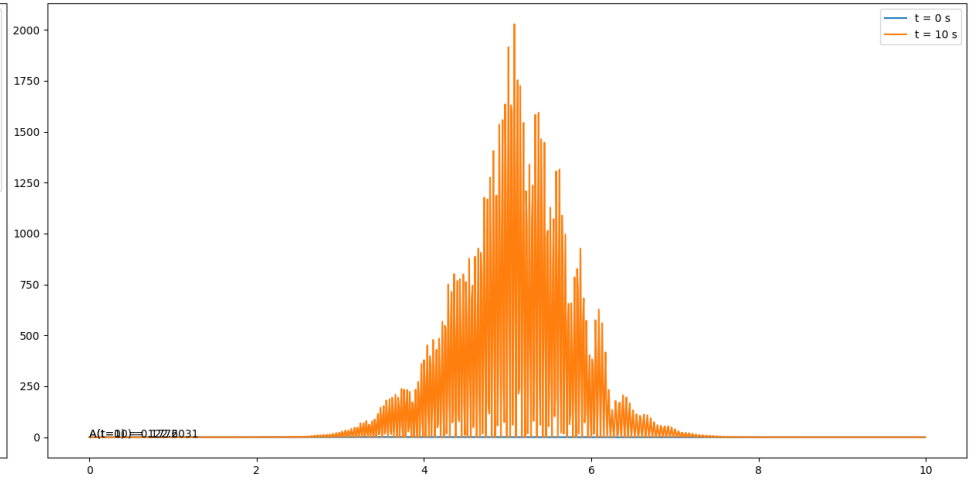


II. Advection

5) Discrétisation d'ordre 1 en temps et 4 en espace :

$$u_i^{n+1} = u_i^n - A\frac{\Delta t}{12\Delta x}\left(u_{i-2}^n - 8u_{i-1}^n + 8u_{i+1}^n - u_{i+2}^n\right)$$





II. Advection

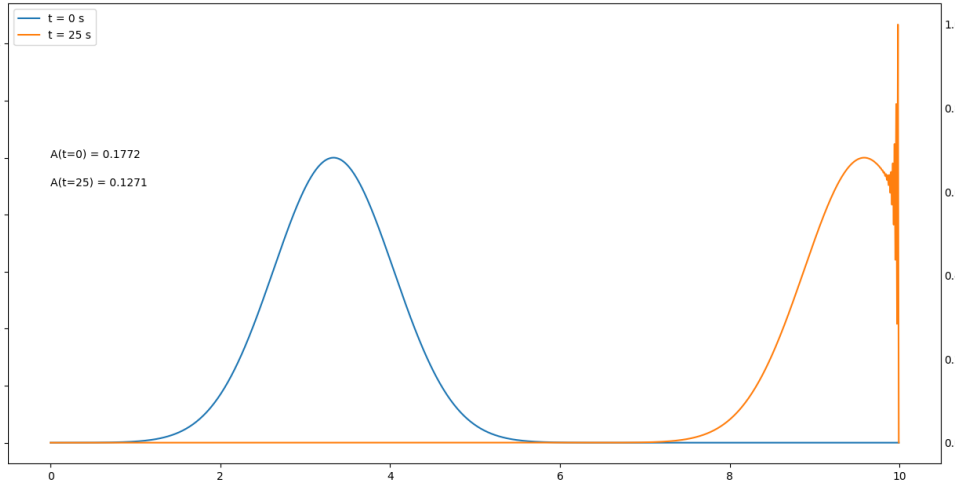
Conclusion préliminaire

2 méthodes pertinentes :

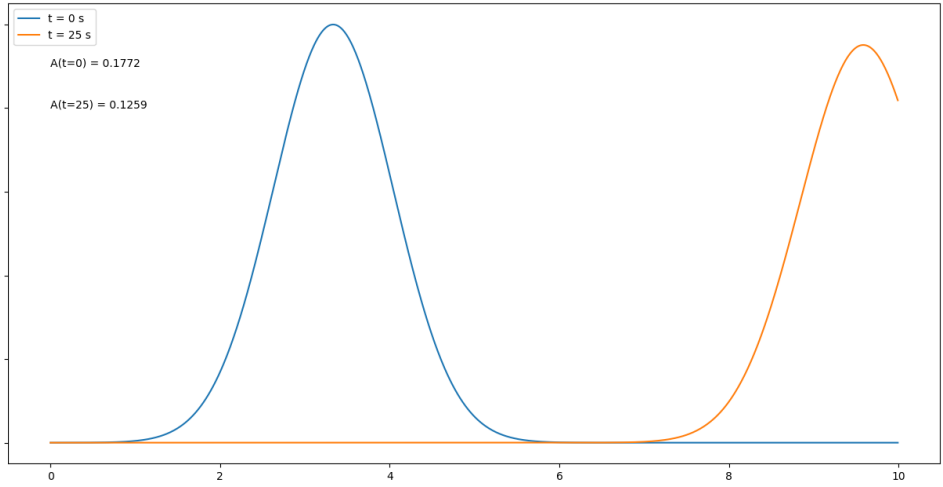
- Méthode de Lax-Wendroff
- Méthode Upwind

⇒ test pour **t = 25 s**

Méthode de Lax-Wendroff



Méthode Upwind



Conclusion pour l'équation d'advection à une dimension

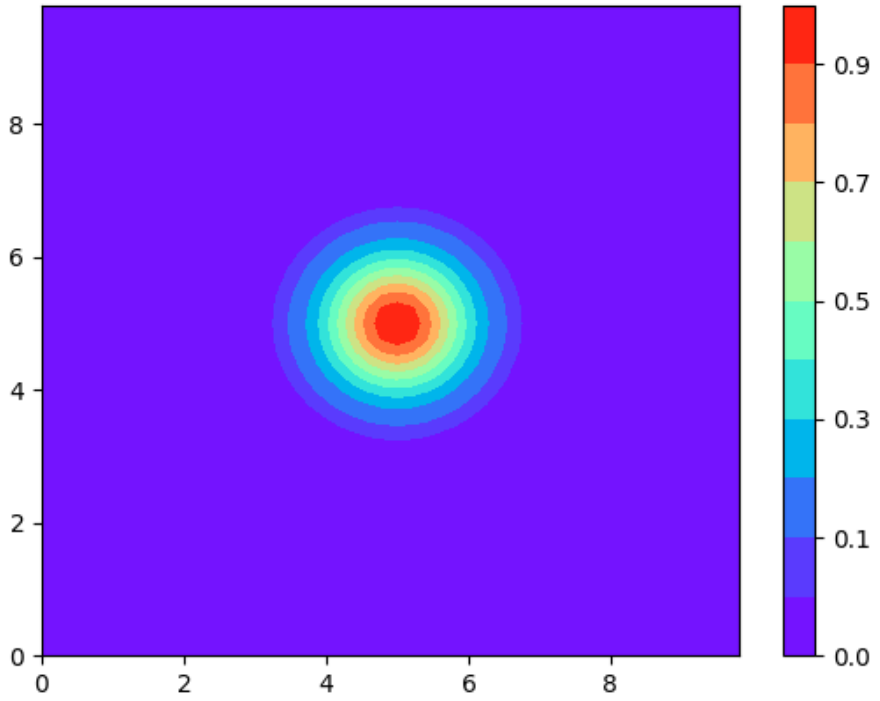
- La **méthode Upwind** semble donc être la plus pertinente pour toute simulation à une dimension

- **Critique** sur la simulation

II. Advection

Équation d'advection à deux dimensions

• Les 4 premières méthodes précédentes ont été **adaptées et testées dans ce cas,** avec la répartition initiale suivante :

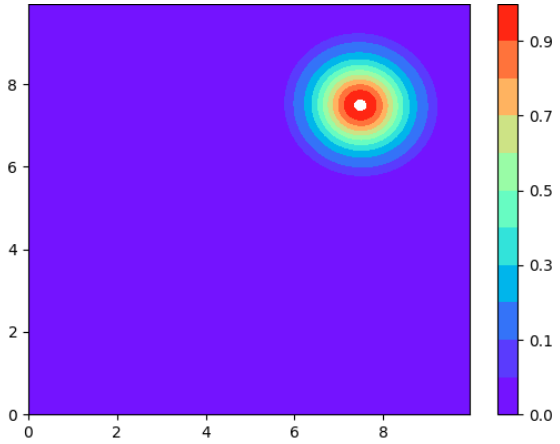


Valeurs utilisées :

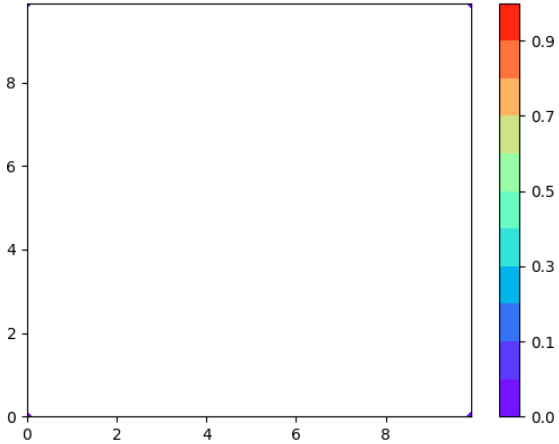
- **A = 0,25 m.s^{-1}**
- **l = 10 m**
- **t = 5 s**
- **nx = 200**
- **nt = 500**

II. Advection

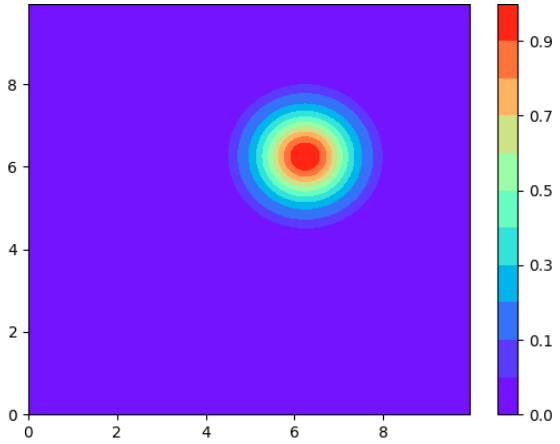
Méthode explicite



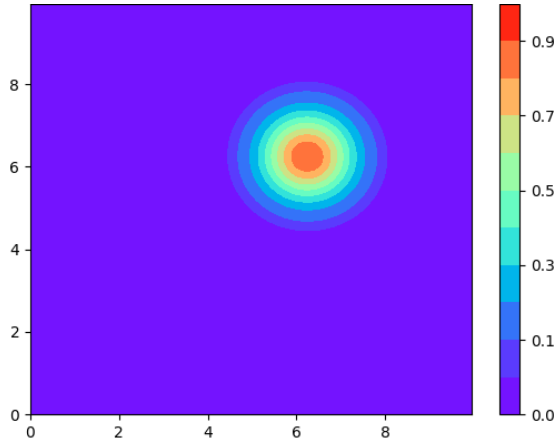
Méthode de Lax-Friedrichs



Méthode de Lax-Wendroff



Méthode Upwind



II. Advection

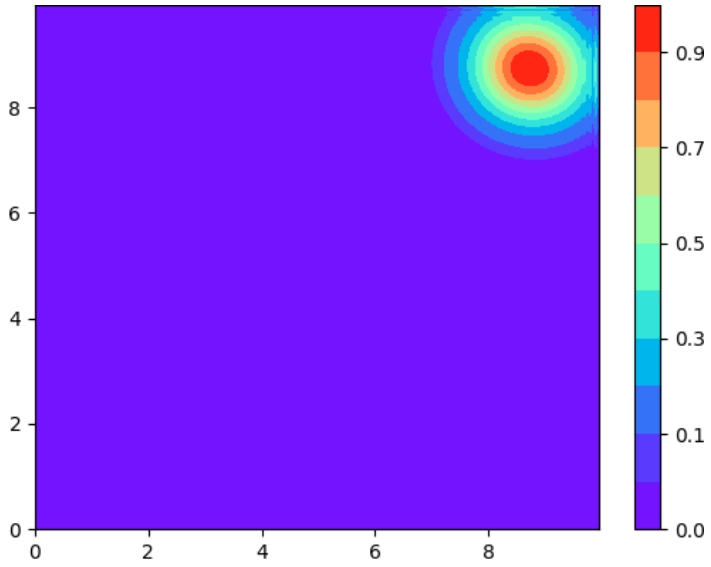
Conclusion préliminaire

2 méthodes pertinentes :

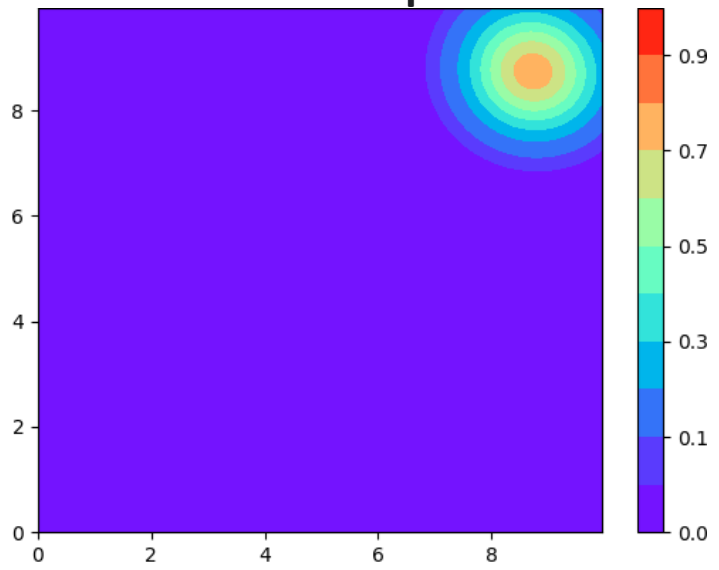
- Méthode de Lax-Wendroff
- Méthode Upwind

⇒ test pour **t = 15 s**

Méthode de Lax-Wendroff



Méthode Upwind



Conclusion pour l'équation d'advection à deux dimensions

- La **méthode Upwind** semble donc être à nouveau la plus pertinente pour toute simulation à deux dimensions

- **Critique** sur la simulation

III. Diffusion

Équation de diffusion :

$$\frac{\partial n}{\partial t} - \mathrm{D}\Delta n = 0$$

avec **D** le coefficient de diffusion et **Δ** l'opérateur laplacien

- **Simulation** pour l'équation d'advection à une dimension : $\frac{\partial n}{\partial t}(x,t) - D\frac{\partial^2 n}{\partial x^2}(x,t) = 0$

- Cas 2D : $\frac{\partial n}{\partial t}(x,y,t) - D\frac{\partial^2 n}{\partial x^2}(x,y,t) - D\frac{\partial^2 n}{\partial y^2}(x,y,t) = 0$

III. Diffusion

Valeurs communes utilisées pour chaque représentation à une dimension

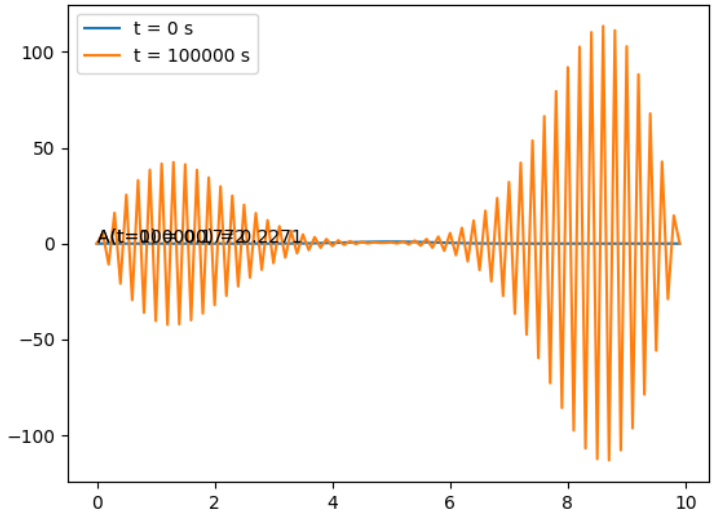
- Coefficient de diffusion : $D = 1.10^{-5}\ m^2.s^{-1}$ (ordre de grandeur)
- Longueur du segment : $l = 10\ m$
- Durée : $t = 100\ 000\ s$

III. Diffusion

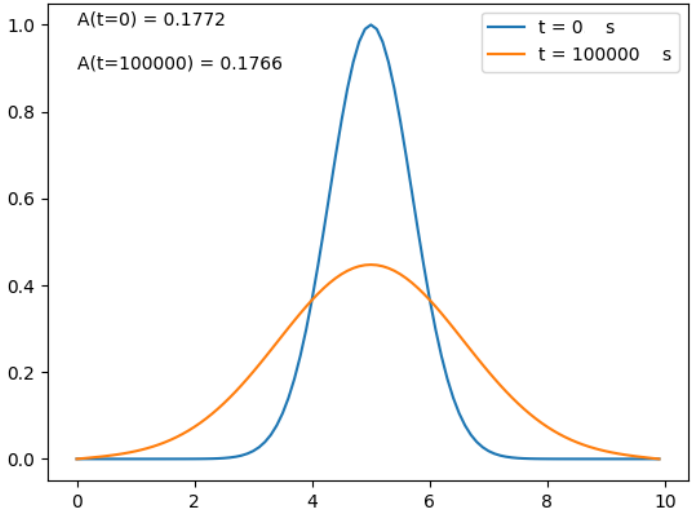
- Méthodes précédentes utilisables en adaptant avec la **dérivée d'ordre 2**
- Exemple avec la **méthode explicite :** $u_i^{n+1} = u_i^n + D\frac{\Delta t}{(\Delta x)^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n)$

Condition de **stabilité :** $\boldsymbol{c = D\frac{\Delta t}{(\Delta x)^2} < 0,5}$

Exemple 1 : nx = 100, nt = 180 ⇒ c = 0,55

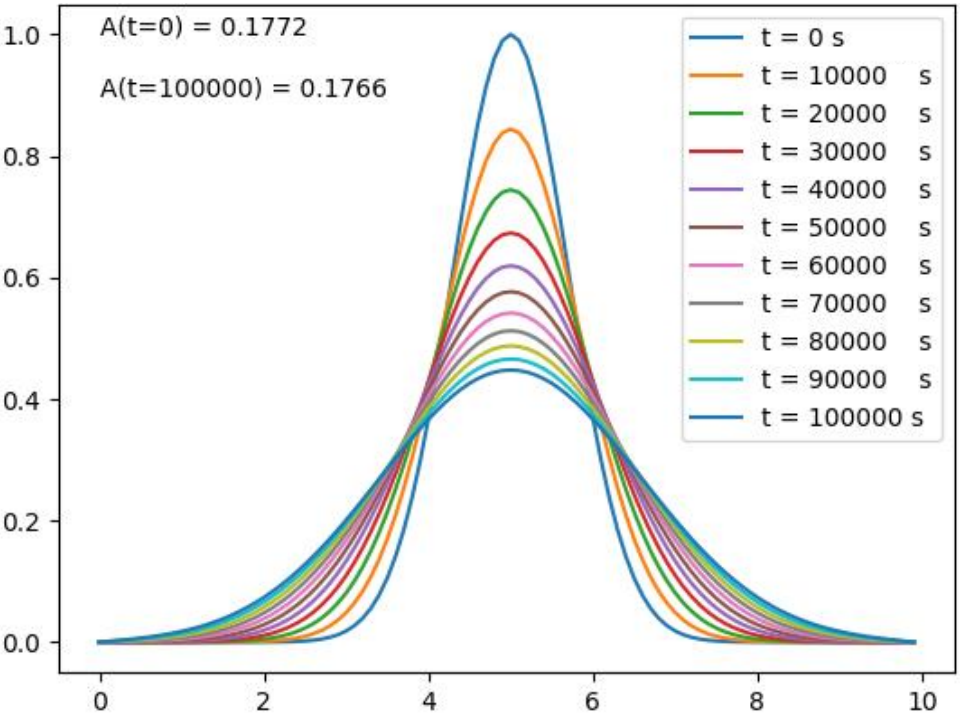


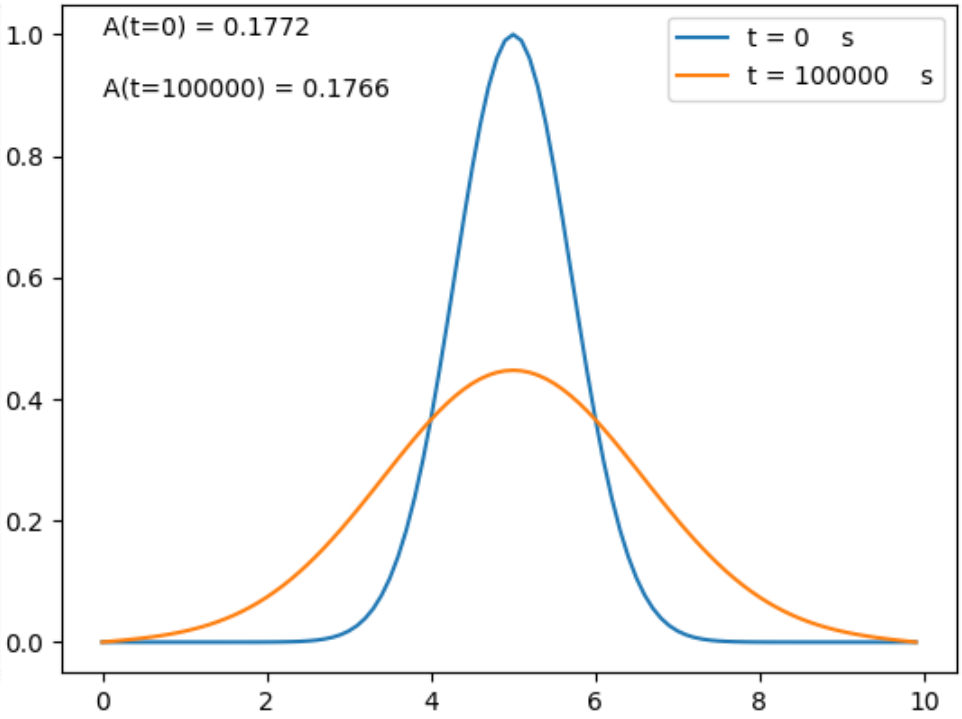
Exemple 2 : nx = 100, nt = 200 ⇒ c = 0,5



III. Diffusion

Résultat pour la méthode explicite avec **nx = 100, nt = 200**





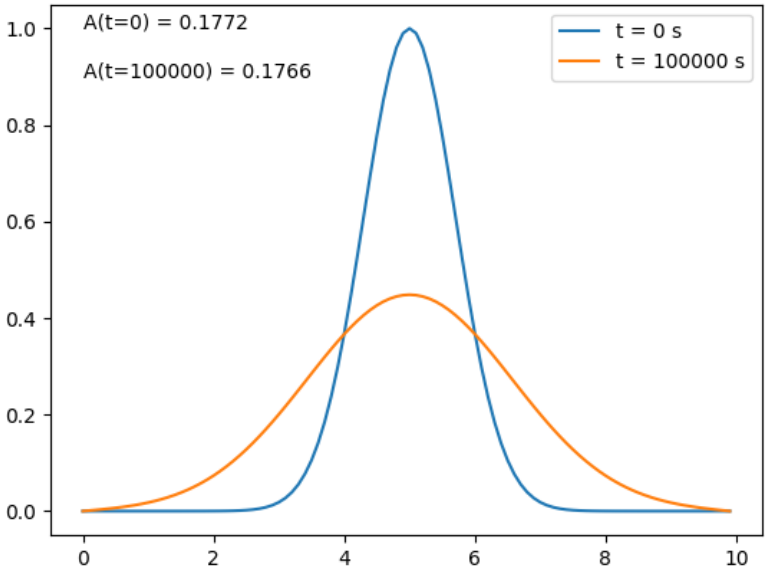
III. Diffusion

Méthode de Crank-Nicolson

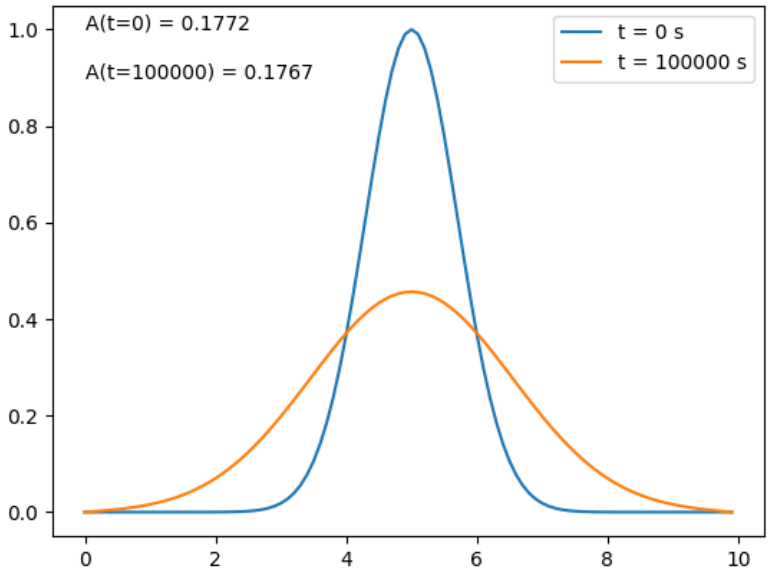
$$\frac{U_i^{n+1} - U_i^n}{\Delta t} = \frac{D}{2}\frac{\left(U_{i+1}^{n+1} - 2U_i^{n+1} + U_{i-1}^{n+1}\right) + \left(U_{i+1}^{n} - 2U_i^{n} + U_{i-1}^{n}\right)}{(\Delta x)^2}$$

Schéma **implicite** … mais on peut obtenir la forme matricielle : $AU^{n+1} = BU^n$
et **méthode avantageuse**

Exemple 1 : nx = 100, nt = 200



Exemple 2 : nx = 100, nt = 20



IV. Advection et diffusion

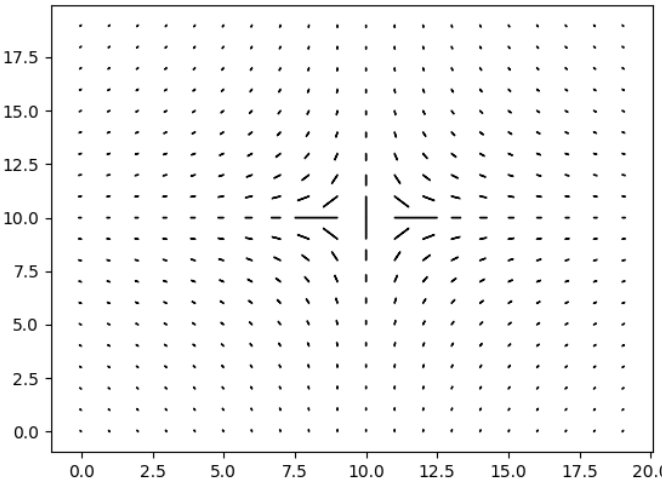
- Réunion des résultats précédents et étude des deux phénomènes **en même temps** et en deux dimensions

- Ajout de **courants marins** dans les modélisations

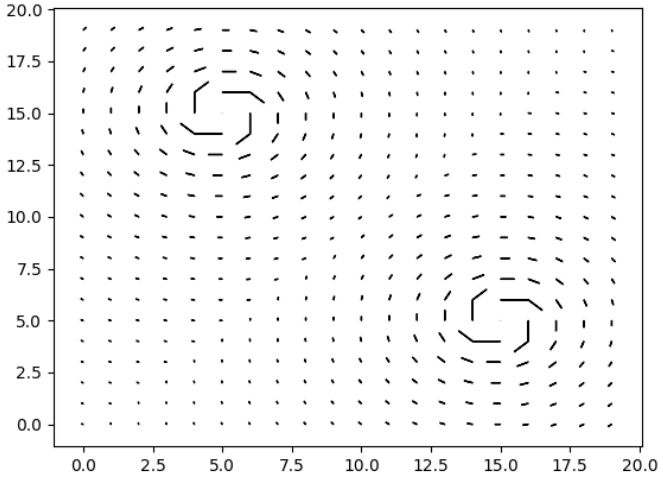
- **Taille** de la zone d'étude **augmentée**

IV. Advection et diffusion

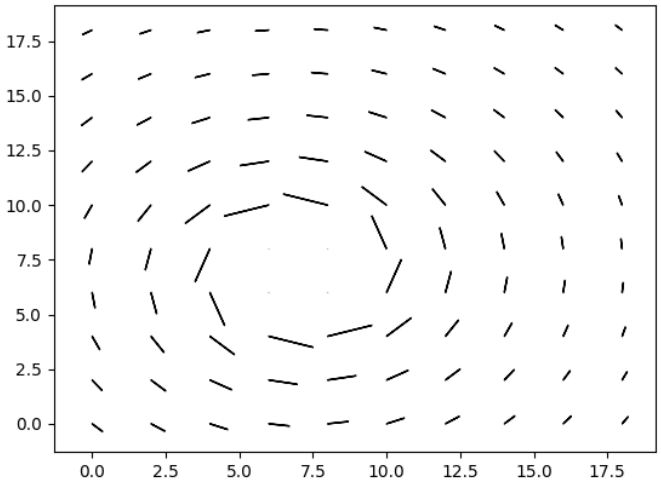
Exemples de courants marins ajoutés



Champ 1



Champ 2



Champ 3

IV. Advection et diffusion

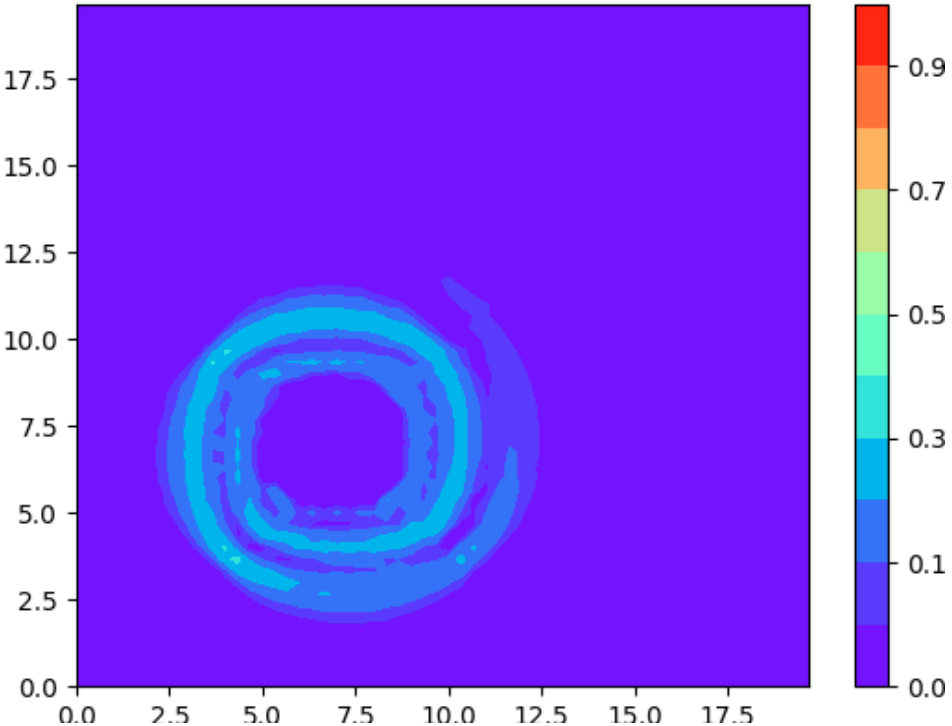
Valeurs communes utilisées pour les prochaines représentations

- Vitesse d'advection : $\mathbf{D = 1.10^{-5}\ m^2.s^{-1}}$
- Longueur du segment : $\mathbf{l = 20\ m}$
- Durée : $\mathbf{t = 40\ s}$
- Nombre d'intervalles de longueur : $\mathbf{nx = 60}$ $(\Delta x = \mathrm{l}/\mathrm{nx})$
- Nombre d'intervalles de temps : $\mathbf{nt = 10000}$ $(\Delta t = \mathrm{t}/\mathrm{nt})$
- Champ choisi : **champ 3**

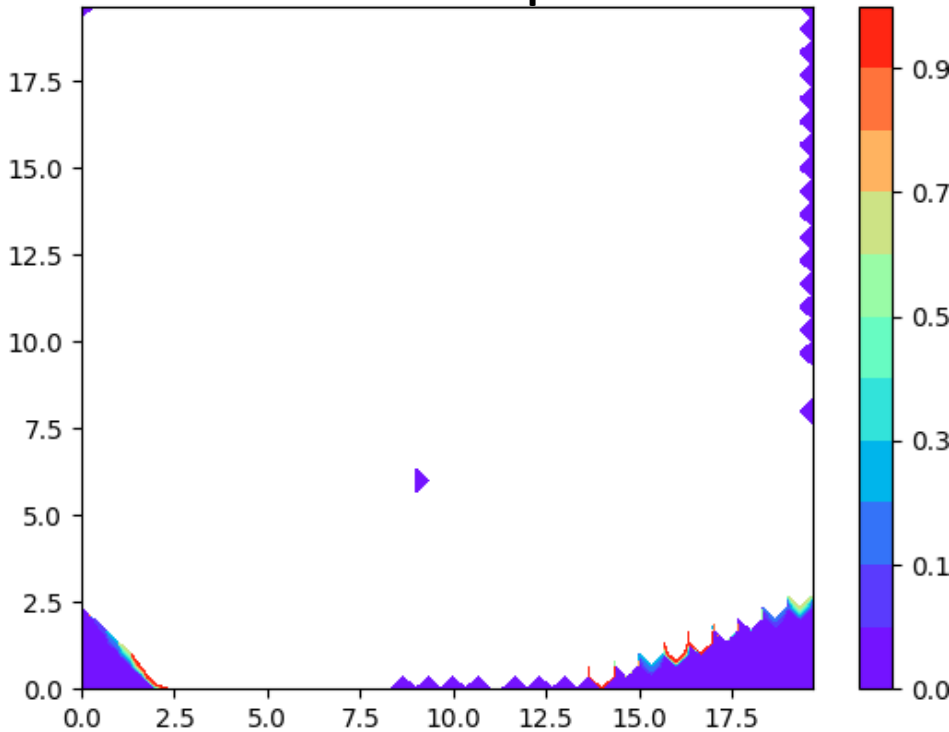
IV. Advection et diffusion

Répartition finale de la simulation de résolution avec la méthode de Lax-Wendroff ou la méthode Upwind pour l'advection et la méthode explicite pour la diffusion **combinées**

Méthode de Lax-Wendroff

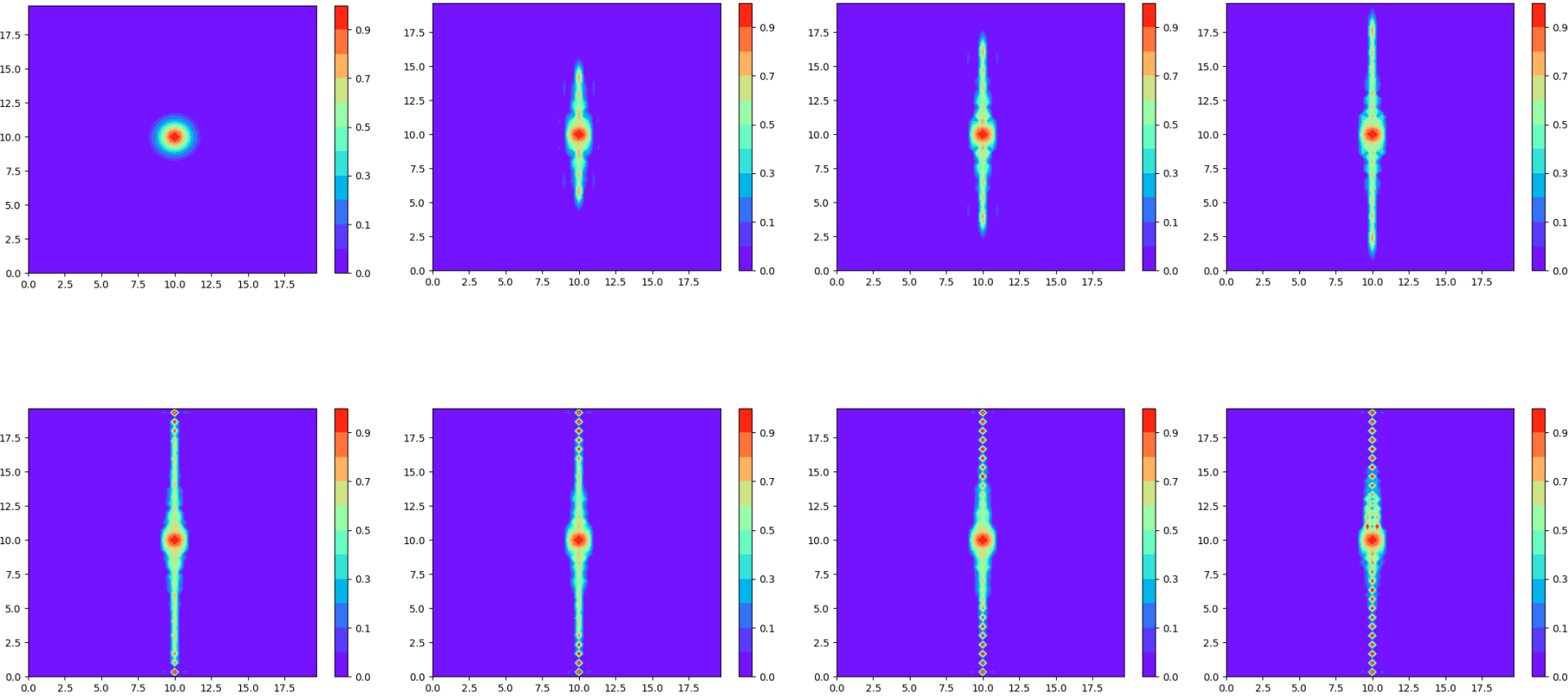


Méthode Upwind



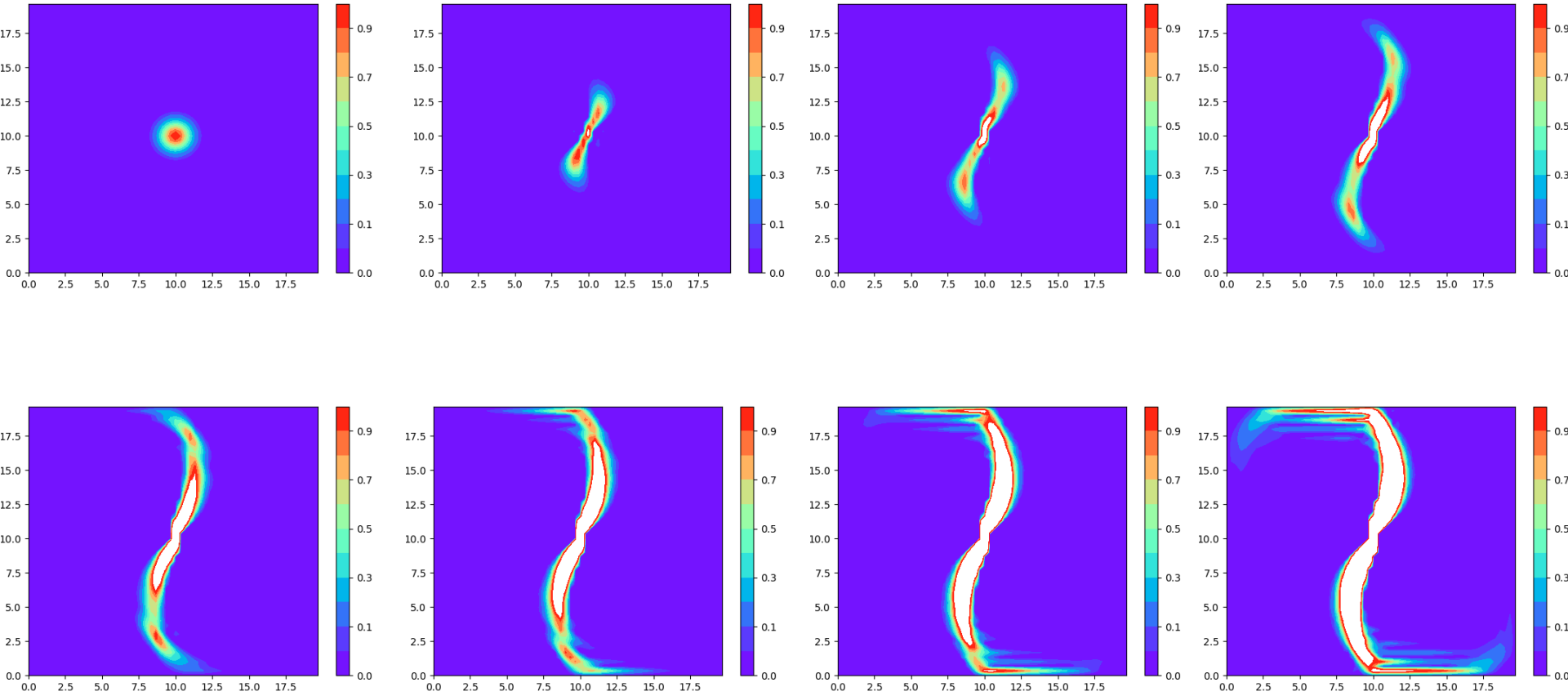
V. Application et conclusion

Observation détaillée des résultats avec la méthode de Lax-Wendroff pour t = 100 s (champ 1)



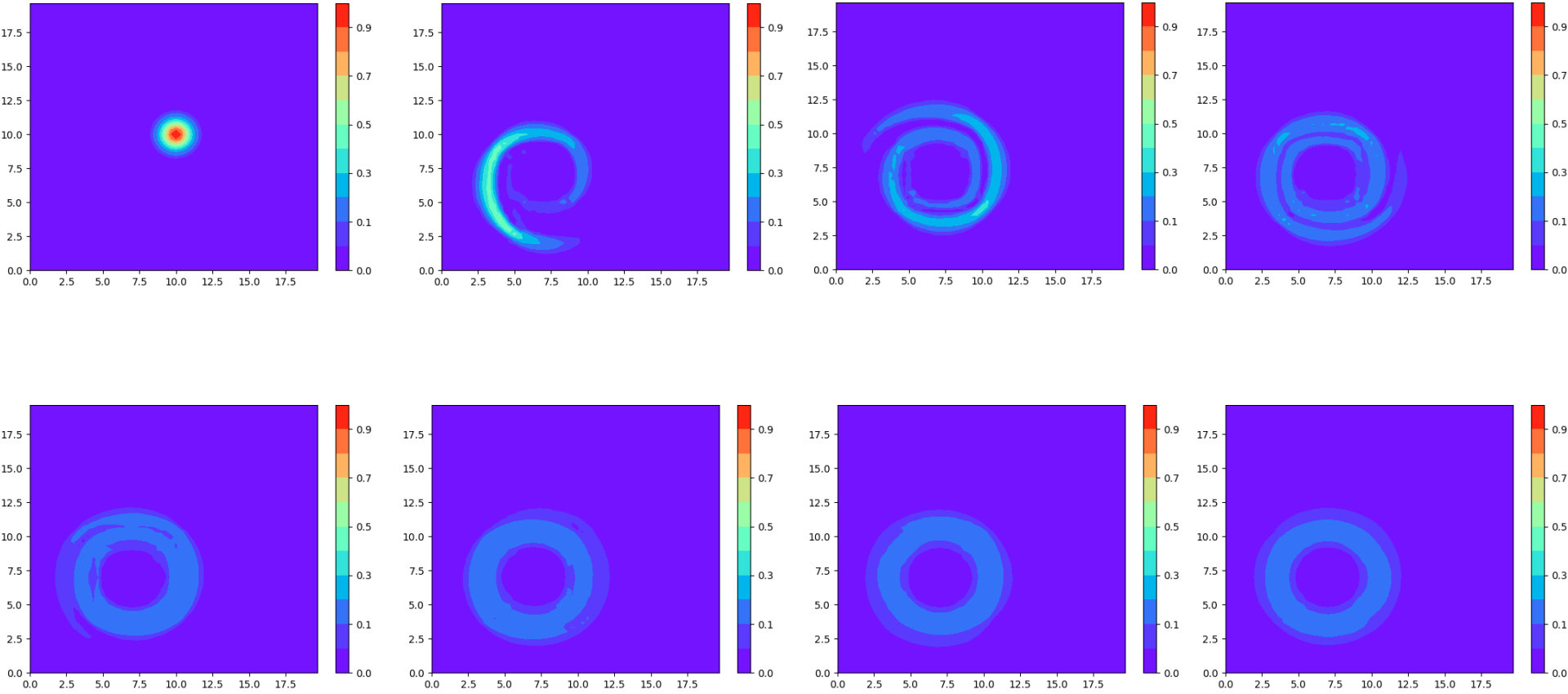
V. Application et conclusion

Observation détaillée des résultats avec la méthode de Lax-Wendroff pour *t* = 100 s (champ 2)



V. Application et conclusion

Observation détaillée des résultats avec la méthode de Lax-Wendroff pour t = 100 s (champ 3)



V. Application et conclusion

CONCLUSION

ANNEXE

Démonstration méthode de Crank-Nicolson

- Schéma implicite : $\frac{U_i^{n+1} - U_i^n}{\Delta t} = \frac{D}{2}\frac{\left(U_{i+1}^{n+1} - 2U_i^{n+1} + U_{i-1}^{n+1}\right) + \left(U_{i+1}^n - 2U_i^n + U_{i-1}^n\right)}{(\Delta x)^2}$

- Il faut résoudre le système : $-\frac{\alpha}{2}U_{j-1}^{n+1} + (1+\alpha)U_j^{n+1} - \frac{\alpha}{2}U_{j+1}^{n+1} = \frac{\alpha}{2}U_{j-1}^n + (1-\alpha)U_j^n + \frac{\alpha}{2}U_{j+1}^n$

  avec $\alpha = \frac{D\Delta t}{(\Delta x)^2}$

- Système tridiagonal : $A_{j,j-1}U_{j-1}^{n+1} + A_{j,j}U_j^{n+1} + A_{j,j+1}U_{j+1}^{n+1} = B_{j,j-1}U_{j-1}^n + B_{j,j}U_j^n + B_{j,j+1}U_{j+1}^n$

D'où la forme matricielle $AU^{n+1} = BU^n$

ANNEXE

```python
## IMPORTATIONS
from math import *
import numpy as np
import matplotlib.pyplot as plt
import os

os.chdir("nom du répertoire où enregistrer les résultats")

## REPRÉSENTATIONS
# Aire sous une courbe
def integrate(liste):
    n = len(liste)
    a = 0
    for i in range(n):
        a += liste[i]/n
    return a

## Advection en 1 dimension (renvoie la répartition initiale et la répartition après un certain temps OU l'évolution de la répartition à différents instants t)
def advection_1D(k,v,l,nx,t,nt,nc):
    dx = l/nx # Pas en longueur
    dt = t/nt # Pas en temps
    m = (nx//2)*dx/1.5
    A = v*(dt/dx)

    abscisse_x=[] # Création liste de positions
    for i in range(nx):
        abscisse_x.append(i*dx)

    abscisse_t=[] # Création liste de temps
    for i in range(nt):
        abscisse_t.append(i*dt)
```

ANNEXE

```
32
33      init = [] # Répartition initiale sous la forme d'une courbe de Gauss
34      for i in range(nx):
35          init.append(exp(-(i*dx-m)**2))
36
37      liste1 = [0]*nx
38      liste2 = init
39      listeTOT = [] # Ensemble des listes : [temps][position]
40      listeTOT.append(init) # Condition initiale pour t = 0
41
42      if k == 1: # Explicite ordre 2
43          for z in range (1,nt+1):
44              for i in range(1,len(liste2)-1):
45                  liste1[i] = max(0,liste2[i] - A*(liste2[i+1]-liste2[i-1]))
46          liste2 = liste1
47          listeTOT.append(liste1)
48          liste1 = [0]*nx
49
50      if k == 2: # Lax-Friedrich
51          for z in range (1,nt+1):
52              for i in range(1,len(liste2)-1):
53                  liste1[i] = max(0,0.5*(liste2[i+1]+liste2[i-1]) - A*(liste2[i+1]-liste2[i-1])*0.5)
54          liste2 = liste1
55          listeTOT.append(liste1)
56          liste1 = [0]*nx
57
58      if k == 3: # Lax-Wendroff
59          for z in range (1,nt+1):
60              for i in range(1,len(liste2)-1):
61                  liste1[i] = max(0,liste2[i] - A*(liste2[i+1]-liste2[i-1])*0.5 + ((A)**2)*(liste2[i+1]-2*liste2[i]+liste2[i-1])*0.5)
62          liste2 = liste1
63          listeTOT.append(liste1)
64          liste1 = [0]*nx
```

ANNEXE

```
    if k == 4: # Upwind
        for z in range (1,nt+1):
            for i in range(1,len(liste2)):
                liste1[i] = max(0,liste2[i] - A*(liste2[i]-liste2[i-1]))
        liste2 = liste1
        listeTOT.append(liste1)
        liste1 = [0]*nx

    if k == 5: # Explicite ordre 4
        for z in range (1,nt+1):
            for i in range(2,len(liste2)-2):
                liste1[i] = max(0,liste2[i] - A*(liste2[i-2]-8*liste2[i-1]+8*liste2[i+1]-liste2[i+2])/12)
        liste2 = liste1
        listeTOT.append(liste1)
        liste1 = [0]*nx

    if k == 6: # Courbe initiale translatée (sert de référence)
        for z in range (1,nt+1):
            for i in range(len(liste2)):
                liste1[i]=  max(0,exp(-((-v*(z*dt-i*dx/v)-m)**2)))
        liste2 = liste1
        listeTOT.append(liste1)
        liste1 = [0]*nx

    plt.plot(abscisse_x,listeTOT[0],label = "t = " + str(0) + " s")
    pas = nt//nc
    # for j in range (1,nc): # Affichage d'une succession de courbes à différents instants t
    #     plt.plot(abscisse_x,listeTOT[j*pas],label = "t = " + str(round(j*t/nc,3)) + " s")
    #     plt.pause(0.01)
    plt.plot(abscisse_x,listeTOT[-1],label = "t = " + str(t) + " s")
    plt.text(0,1,'A(t=' + str(0) + ') = ' + str(round(integrate(listeTOT[0]),4)))    # Vérification de la constance de l'aire sous la courbe
    plt.text(0,0.9,'A(t=' + str(t) + ') = ' + str(round(integrate(listeTOT[-1]),4))) # Vérification de la constance de l'aire sous la courbe
    plt.legend()
```

ANNEXE

```
100
101 ## Advection en 2 dimensions (renvoie la répartition initial et la répartition après un certain temps)
102 def advection_2D(k,v,l,nx,t,nt):
103     dx = l/nx # Pas en longueur
104     dt = t/nt # Pas en temps
105     m = (nx//2)*dx
106     A = v*(dt/dx)
107 
108     abscisse_x = [] # Création liste de positions
109     for i in range (nx):
110         abscisse_x.append(i*dx)
111 
112     tab = np.zeros((nt,nx,nx)) # Espace de résolution sous forme d'un carré
113 
114     for i in range (nx): # Répartition initiale sous la forme d'une courbe de Gauss
115         for j in range (nx):
116             x = i*dx - m
117             y = j*dx - m
118             tab[0,i,j] = exp(-(x**2 + y**2))
119 
120     if k == 1: # Explicite ordre 2
121         for p in range (0,nt-1):
122             for i in range (1,nx-1):
123                 for j in range (1,nx-1):
124                     tab[p+1,i,j] = max(0,tab[p,i,j] - A*(tab[p,i+1,j] + tab[p,i,j+1] - tab[p,i-1,j] -tab[p,i,j-1]))
125 
126     if k == 2: # Lax-Friedrich
127         for p in range (0,nt-1):
128             for i in range (1,nx-1):
129                 for j in range (1,nx-1):
130                     tab[p+1,i,j] = max(0,0.5*(tab[p,i+1,j] + tab[p,i-1,j] + tab[p,i,j+1] + tab[p,i,j-1]) - A*(tab[p,i+1,j] - tab[p,i-1,j] +
    tab[p,i,j+1] - tab[p,i,j-1])*0.5)
131
```

ANNEXE

```
    if k == 3: # Lax-Wendroff
        for p in range (0,nt-1):
            for i in range (1,nx-1):
                for j in range (1,nx-1):
                    tab[p+1,i,j] = max(0,tab[p,i,j] - A*(tab[p,i+1,j] + tab[p,i,j+1] - tab[p,i-1,j] - tab[p,i,j-1])*0.5 + ((A)**2)*(tab[p,i+1,j] + tab[p,i,j+1] - 4*tab[p,i,j] + tab[p,i-1,j] + tab[p,i,j-1])*0.5)

    if k == 4: # Upwind
        for p in range (0,nt-1):
            for i in range (1,nx):
                for j in range (1,nx):
                    tab[p+1,i,j] = max(0,tab[p,i,j] - A*(2*tab[p,i,j] - tab[p,i-1,j] - tab[p,i,j-1]))

    print (A)                    # Schémas théoriquement stable si A < 1

    plt.contourf(abscisse_x,abscisse_x,tab[-1],[0,0.05,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1], cmap = 'rainbow')
    plt.clim(0,1)
    plt.colorbar()
    plt.show()

## Diffusion en 1 dimension (renvoie la répartition initiale et la répartition après un certain temps OU l'évolution de la répartition à différents instants t)
def tridiag(n,cdiag,ctrih,ctrib):
    M = [[0 for _ in range (n)] for _ in range (n)]
    M[n-1][n-1] = cdiag
    for i in range (n-1):
        M[i][i] = cdiag
        M[i][i+1] = ctrih
        M[i+1][i] = ctrib
    return np.array(M)
```

ANNEXE

```
160
161 def diffusion_1D(k,D,l,nx,t,nt,nc):
162     dx = l/nx # Pas en longueur
163     dt = t/nt # Pas en temps
164     m = (nx//2)*dx
165     c = D*dt/(dx**2)
166     print (c)
167
168     abscisse_x = [] # Création liste de positions
169     for i in range (nx):
170         abscisse_x.append(i*dx)
171
172     init = [] # Répartition initiale sous la forme d'une courbe de Gauss
173     for i in range (nx):
174         init.append(exp(-(i*dx-m)**2))
175
176     if k == 1:  # Explicite ordre 2
177         tab = [[0 for _ in range (nx)] for _ in range (nt)]
178         tab[0] = init
179
180         for i in range (nt-1):
181             for k in range (1,nx-1):
182                 tab[i+1][k] = tab[i][k] + c*(tab[i][k+1] - 2*tab[i][k] + tab[i][k-1])
183
184         plt.plot(abscisse_x,tab[0],label = "t = " + str(0) + " s")
185         pas = nt//nc
186         # for j in range (1,nc): # Affichage d'une succession de courbes à différents instants t
187         #     plt.plot(abscisse_x,tab[j*pas],label = "t = " + str(round(j*t/nc)) + " s")
188         plt.plot(abscisse_x,tab[-1],label = "t = " + str(t) + " s")
189         plt.text(0,1,'A(t=' + str(0) + ') = ' + str(round(integrate(tab[0]),4)))      # Vérification de la constance de l'aire sous la courbe
190         plt.text(0,0.9,'A(t=' + str(t) + ') = ' + str(round(integrate(tab[-1]),4)))   # Vérification de la constance de l'aire sous la courbe
191         plt.legend()
192
```

ANNEXE

```
    if k == 2 : # Méthode de Crank-Nicolson
        A = tridiag(nx-2,1+c,-c/2,-c/2)
        B = tridiag(nx-2,1-c,c/2,c/2)
        b = np.zeros(nx-2)
        u = init
        bb = B.dot(u[1:-1])
        pas = nt//nc

        plt.plot(abscisse_x,u,label = "t = " + str(0) + " s")
        plt.text(0,1,'A(t=' + str(0) + ') = ' + str(round(integrate(init),4)))      # Vérification de la constance de l'aire sous la courbe

        for i in range (1,nt):
            # if (i%pas == 0): # Affichage d'une succession de courbes à différents instants t
                # plt.plot(abscisse_x,u,label = "t = " + str(round(i*dt)) + " s")
            u[1:-1] = np.linalg.solve(A,bb)
            bb = B.dot(u[1:-1]) + b

        plt.plot(abscisse_x,u,label = "t = " + str(round(t)) + " s")
        plt.text(0,0.9,'A(t=' + str(t) + ') = ' + str(round(integrate(u),4)))       # Vérification de la constance de l'aire sous la courbe
        plt.legend()

## Champs (= courants marins)
def champ(i,x,y):
    if i == 1 : # Champ 1
        x -= 10
        y -= 10
        r = (x**2 + y**2)**0.5
        if r < 1:
            return (0,0)
        return (x/(r**2),-y/(r**2))

```

ANNEXE

```
    if i == 2 : # Champ 2
        if x > y :
            x -= 15
            y -= 5
            r = (x**2 + y**2)**0.5
            if r < 1:
                return (0,0)
            return (y/(r**2),-x/(r**2))
        else :
            x -= 5
            y -= 15
            r = (x**2 + y**2)**0.5
            if r < 1:
                return (0,0)
            return (y/(r**2),-x/(r**2))

    if i == 3 : # Champ 3
        x -= 7
        y -= 7
        r = (x**2+y**2)**0.5
        if r < 2:
            return (0,0)
        return (-y*5/(r**2),x*5/(r**2))

## Advection et diffusion en 2 dimensions avec courants marins
def advection_diffusion_2D(k,kc,nc,D,l,nx,t,nt):
    dx = l/nx # Pas en longueur
    dt = t/nt # Pas en temps
    m = (nx//2)*dx
    c = D*dt/(dx**2)

```

ANNEXE

```
255     abscisse_x=[]  # Création liste de positions
256     for i in range(nx):
257         abscisse_x.append(i*dx)
258 
259     abscisse_t=[] # Création liste de temps
260     for i in range(nt):
261         abscisse_t.append(i*dt)
262 
263     tab = np.zeros((nt,nx,nx)) # Espace de résolution sous forme d'un carré
264 
265     for i in range(nx): # Répartition initiale sous la forme d'une courbe de Gauss
266         for j in range(nx):
267             x = i*dx - m
268             y = j*dx - m
269             tab[0,i,j] = exp(-(x**2 + y**2))
270 
271     if k == 3: # Lax-Wendroff + explicite
272             for t in range (0,nt-1):
273                 for i in range (1,nx-1):
274                     for j in range (1,nx-1):
275                         vx,vy = champ(kc,i*dx,j*dx)
276                         Ax = vx * dt/dx
277                         Ay = vy * dt/dx
278                         tab[t+1,i,j] = max(0,tab[t,i,j] - Ax*(tab[t,i+1,j]-tab[t,i-1,j])*0.5 - Ay*(tab[t,i,j+1]-tab[t,i,j-1])*0.5 +
    Ax**2*(tab[t,i+1,j] - 2*tab[t,i,j] + tab[t,i-1,j])*0.5 + Ay**2*(tab[t,i,j+1] - 2*tab[t,i,j] + tab[t,i,j-1])*0.5 + c*(tab[t,i+1,j] + tab[t,i,j+1] -
    4*tab[t,i,j] + tab[t,i-1,j] + tab[t,i,j-1]))
279 
```

ANNEXE

```
    if k == 4: # Upwind + explicite
            for t in range (0,nt-1):
                for i in range (1,nx-1):
                    for j in range (1,nx-1):
                        vx,vy = champ(kc,i*dx,j*dx)
                        Ax = vx * dt/dx
                        Ay = vy * dt/dx
                        tab[t+1,i,j] = max(0,tab[t,i,j] - Ax*(tab[t,i,j]-tab[t,i-1,j]) - Ay*(tab[t,i,j]-tab[t,i,j-1]) + c*(tab[t,i+1,j] + tab[t,i,j+1]
- 4*tab[t,i,j] + tab[t,i-1,j] + tab[t,i,j-1]))

    pas = nt//nc
    for i in range(nc-1): # Évolution de la répartition dans le plan au cours du temps
        plt.clf()
        plt.contourf(abscisse_x,abscisse_x,tab[i*pas],[0,0.05,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1],cmap = 'rainbow')
        plt.clim(0, 1)
        plt.colorbar()
        plt.pause(0.01)
        # plt.savefig("test{}".format(i)) # Pour enregistrer chaque répartition affichée

## Variables et affichage
vitesse = ? # Vitesse d'advection
coefficient_de_diffusion = ? # Ordre de grandeur
longueur = ?
nombre_de_points_longueur = ?
temps = ?
nombre_de_points_temps = ?
nombre_de_captures = ?
k_méthode = ? # Choix de la méthode choisie pour les différents programmes (Advection : 1) Explicite, 2) L-F, 3) L-W, 4) Upwind, 5) Explicite ordre 4,
6) Référence ; Diffusion : 1) Explicite, 2) C-N)
k_champ = ? # Choix du champ choisi pour advection_diffusion2D

```

ANNEXE

```
# Affichage des champs
for i in range (0,20):
    for j in range (0,20):
        (fx,fy) = champ(?,i,j) # Champ affiché à choisir
        plt.arrow(i,j,fx,fy)

advection_1D(k_méthode,vitesse,longueur,nombre_de_points_longueur,temps,nombre_de_points_temps,nombre_de_captures)
advection_2D(k_méthode,vitesse,longueur,nombre_de_points_longueur,temps,nombre_de_points_temps)
diffusion_1D(k_méthode,coefficient_de_diffusion,longueur,nombre_de_points_longueur,temps,nombre_de_points_temps,nombre_de_captures)
advection_diffusion_2D(k_méthode,k_champ,nombre_de_captures,coefficient_de_diffusion,longueur,nombre_de_points_longueur,temps,nombre_de_points_temps)

plt.show()
```