Université de Montréal

PHY 3075 – Modélisation numérique en physique

Projet 2 - Barrière de confinement plasma

---

par :

Patrice Béchard

20019173

27 février 2017

1 Introduction

La fusion nucléaire est souvent vue comme la solution miracle aux problèmes énergétiques de la planète. Il est possible de fusionner un noyau de deutérium (^2H) avec un noyau de tritium (^3H)pour produire de l'énergie, et ce, en laissant comme produit un noyau d'hélium et un neutron, qui sont des déchets propres. Sa mise en application au niveau pratique est compliquée par le fait qu'il faut amener les atomes à de très grandes températures pour que le processus de fusion se produise, températures auxquelles aucun matériau connu ne peut supporter sans fondre. Il est cependant possible de confiner le plasma d'hydrogène qui réagit dans une partie du réacteur utilisé à l'aide de champs magnétiques de sorte que les parois ne soient pas exposés à ces grandes températures.

Le but de ce projet est de modéliser numériquement le confinement de plasma en tokamak (chambre magnétique toroïdale), phénomène faisant appel à la résolution d'équation différentielles partielles (EDP) de type *diffusion*. La méthode de résolution d'EDP utilisée est la méthode de Crank-Nicolson, algorithme implicite demandant la résolution d'un système tridiagonal d'équation linéaires. Les détails de l'implémentation de l'algorithme sont présentés dans les notes de cours [1].

Une validation du code utilisé pour modéliser le système sera tout d'abord présenté, et suivra ensuite une présentation des résultats issus d'une exploration numérique de divers phénomènes et de leur effet sur la barrière de confinement plasma. Une version du code ainsi que des figures en pleine résolution et en couleur contenues dans ce rapport sont disponible sur le dépot électronique situé à l'adresse suivante : `https://goo.gl/mPOaI6`

2 Validation du code

Pour valider le programme construit, les figures 2.15, 2.16 et 2.17 des notes de cours ont été répliquées[1]. Tout d'abord, la figure 1 présente la variation du flux de chaleur en fonction du gradient de température pour différentes valeurs de coefficient χ_1 associé au transport turbulent.

La figure produite par le programme est identique à la figure 2.15 des notes de cours, signe que notre programme sait bien répliquer cette partie de l'analyse. Plus la constante χ_1 devient grande par rapport à χ_0, plus la relation entre la variation du flux de chaleur et le gradient de température non-monotone.

La seconde figure ayant été répliquée est la figure 2.16 des notes de cours, présentée à la figure 2 du présent document. Elle présente l'évolution dans le temps du profil de température sur le domaine spatial allant de -10 à 10. Les différentes courbes tracées représentent plusieurs temps distincts, soit aux temps 0.05, 0.1, 0.25, 0.5, 1.0, 2.5, 5.0, 10.0, 25.0 (allant du bas vers le haut).

Encore une fois, la figure répliquée est identique à l'originale, signe que le programme fonctionne

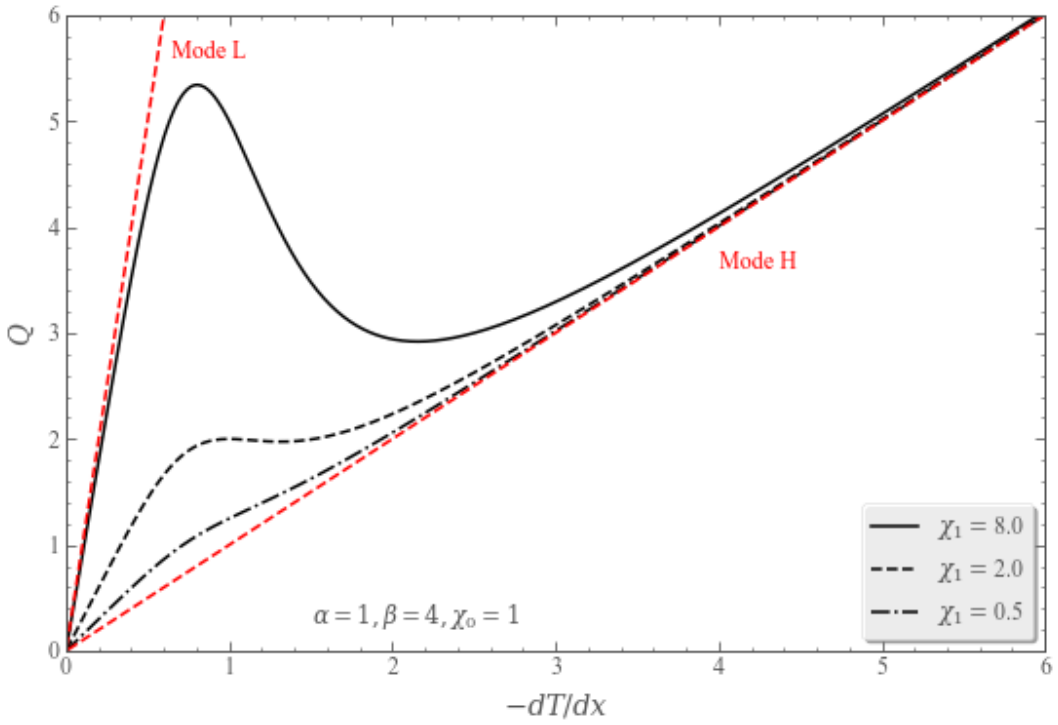

FIGURE 1 – Variation du flux de chaleur en fonction du gradient de température dans le cas d ?un transport turbulent en 1D.

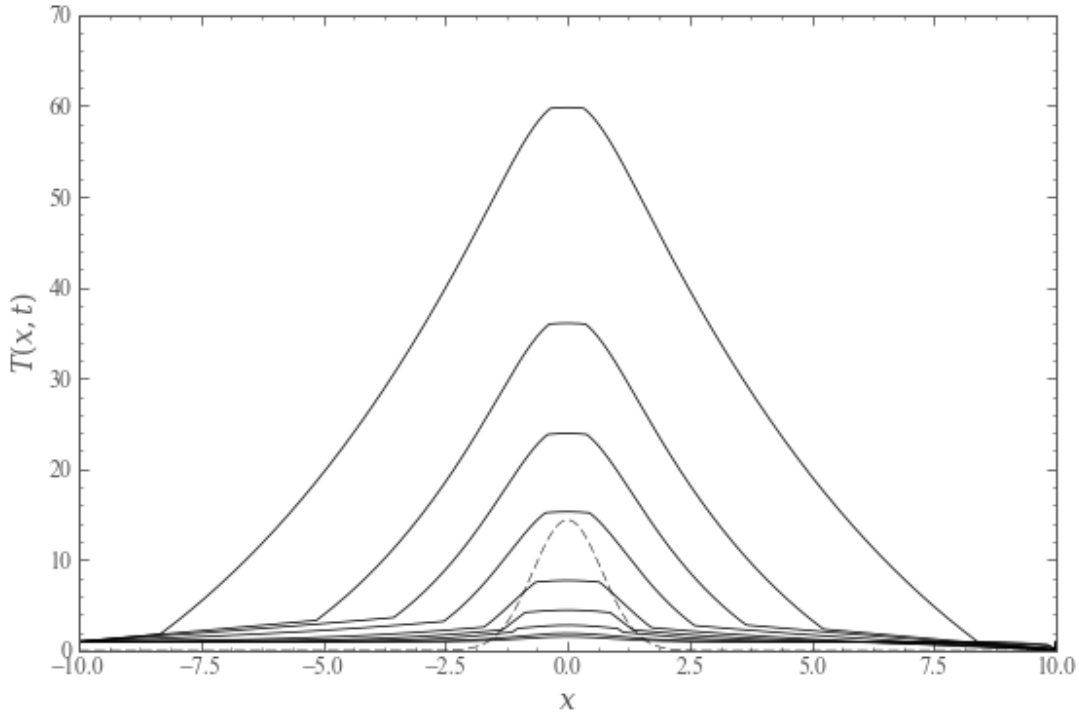

FIGURE 2 – Évolution du profil de température pour une solution de l'éq. (2.112) des notes de cours avec valeurs de paramètres $\chi_0 = 1$, $\chi_1 = 8$, $\alpha = 1$, $\beta = 4$ et $H_0 = 14.4$. Le trait en tirets montre le profil Gaussien du terme source. Cette solution numérique par Crank-Nicolson utilise des pas $\Delta x = 0.02$ et $\Delta t = 5 \times 10^{-4}$.

comme voulu. Il est possible de voir deux discontinuités de part et d'autre du centre du domaine, soit une au sommet et une autre lorsque le profil de température descend vers de faibles valeurs. C'est le bris de pente près du centre qui est à l'origine de l'établissement de la barrière de confinement.

La dernière figure à répliquer pour compléter la validation du code est la figure 2.17 des notes de cours, qui est présentée à a figure 3 de ce rapport. Elle montre le comportement du gradient de température $dT(x,t)/dx$ ainsi que de la conductivité thermique totale $\chi(x,t)$ sur une partie du domaine pour les mêmes temps qu'à la figure précédente.

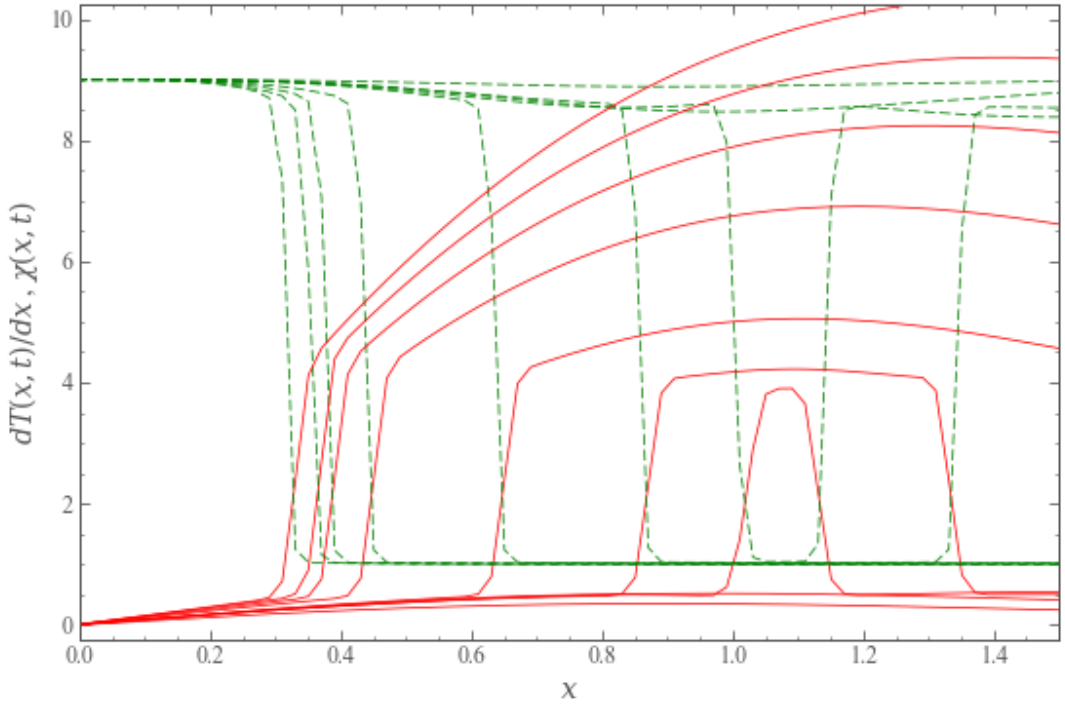

FIGURE 3 – Évolution spatiotemporelle du gradient de température (ligne pleine) et conductivité thermique totale (ligne pointillée) près du centre du domaine, pour la solution de la figure 2. Les courbes sont tracées dans une petite partie du domaine près du centre, s'étendant de 0 à 1.5 et aux mêmes pas de temps.

Cette figure est encore une fois répliquée exactement, signe que notre programme exécute comme voulu. La variation rapide du gradient de température se produit lorsque cette valeur dépasse $1/\alpha = 1$ dans notre cas.

Maintenant que la vérification du code a été effectuée, une analyse plus poussée de certains phénomènes sera abordée dans la prochaine section.

3 Résultats et discussion

Cette partie du rapport sera consacrée à l'exploration numérique de certains aspects présentés à la section 2.9.4 des notes de cours.

Le premier sujet abordé est celui de la convergence du confinement du plasma vers le centre du réacteur à mesure que le temps avance. Il est possible de voir sur la figure 3 que plus le temps avance, plus la barrière de confinement se rapproche du centre (à x=0). Il est possible de calculer la vitesse de convergence de ce confinement en remarquant que le coefficient de conductivité thermique total $\chi(T)$ chute drastiquement suite au bris de pente au sommet du profil de température. En gardant en mémoire la position où la chute soudaine se produit en fonction du temps écoulé, il est possible de mettre en graphique cette relation. Le tout a été exécuté pour des différentes valeurs d'amplitude du profil gaussien du terme source H_0, soit avec $H_0 \in \{10.0, 14.4, 20.0\}$ en gardant le coefficient χ_1 associé au transport turbulent à 8, ainsi qu'en faisant varier cette dernière valeur ($\chi_1 \in \{4, 8, 12, 16\}$) en conservant $H_0 = 14.4$ constant. Les figures 6 et 7 présentent ces diverses configurations en graphique *log-log*, auquel une courbe de tendance a été ajustée.

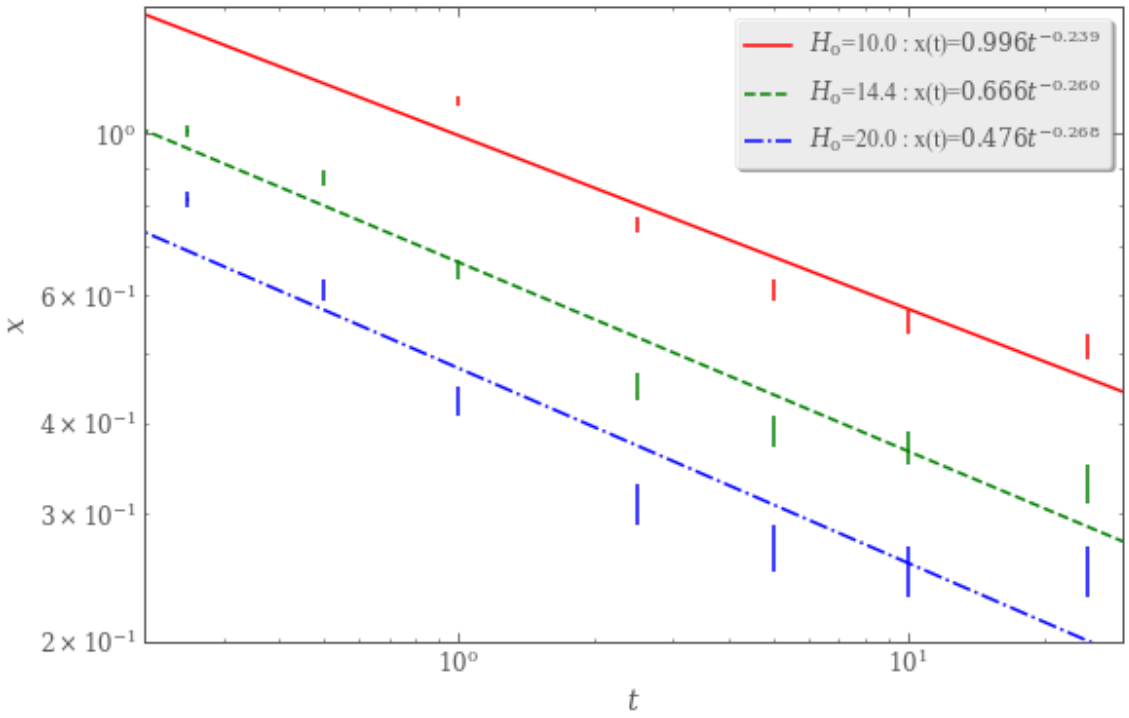

FIGURE 4 – Position du bris de pente en fonction du temps pour des simulations à χ_1 constant à 8 en faisant varier l'amplitude du profil gaussien H_0.

Les notes de cours proposent une relation de la vitesse en fonction du temps allant en $\sqrt{t}$ en première approximation. Cependant, les graphiques obtenus une légère courbe lorsque tracée en *log-log*, signe que la dépendance de la position du bris de pente ne dépend pas seulement du temps élevé à une certaine

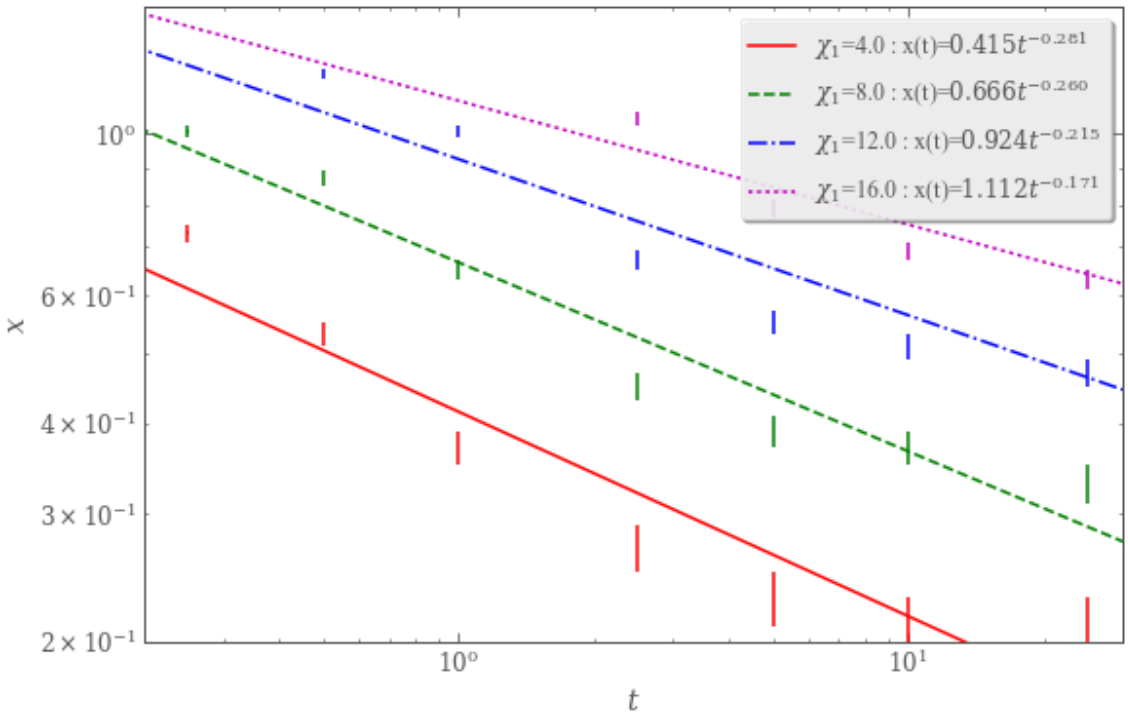

FIGURE 5 – Position du bris de pente en fonction du temps pour des simulations à H_0 constant à 14.4 en faisant varier le coefficient χ_1 associé au transport turbulent H_0.

puissance. Il est tout de même possible d'approximer cette tendance comme étant linéaire en *log-log* pour trouver approximativement la relation entre la position du bris de pente et le temps écoulé. Pour le cas où χ_1 est constant, la puissance à laquelle t est élevée semble dépendre très faiblement de l'amplitude du profil gaussien du terme source H_0, se situant constamment aux alentours de -0.25. Cependant, dans le cas où c'est plutôt H_0 qui est laissé constant, la puissance de t varie de façon plus importante, passant d'environ -0.17 à -0.28 pour des valeurs de χ_1 allant de 4 à 16. La vitesse de convergence de la barrière de confinement dépend donc plus du coefficient associé au transport turbulent qu'elle ne dépend de l'amplitude du profil gaussien du terme source. En dérivant les expressions obtenues, une relation moyenne de la vitesse de convergence est approximativement $v(t) \approx ct^{-\frac{5}{4}}$. Cette relation a un sens physique plus approprié que la relation en $\sqrt{t}$, puisque les valeurs de la vitesse vont tendre vers zéro, soit vers une solution à l'équilibre de la barrière de confinement, alors que dans l'autre cas, celles-ci tendaient vers l'infini, signe qu'aucune solution à l'équilibre n'existe, ce qui n'est évidemment pas le cas.

L'effet de la forme du terme source sur la vitesse de convergence de la barrière de confinement a aussi été étudié. En gardant l'amplitude du terme source H_0 à 14.4, diverses simulations ont été effectuées en modifiant la largeur de la gaussienne (associée au paramètre σ) pour des valeurs de $\sigma \in \{0.25, 0.5, 1.0, 2.0\}$. De plus, d'autres simulations ont été menées avec une fonction marche comme terme source, étant nulle partout sauf au centre, où son amplitude était encore une fois de H_0 et sa largeur totale était

$\in \{0.5, 1.0, 2.0, 4.0\}$. Les figures **??** et **??** présentent la position du bris de pente en fonction du temps pour ces termes sources, comme le faisaient les figures 6 et 7.

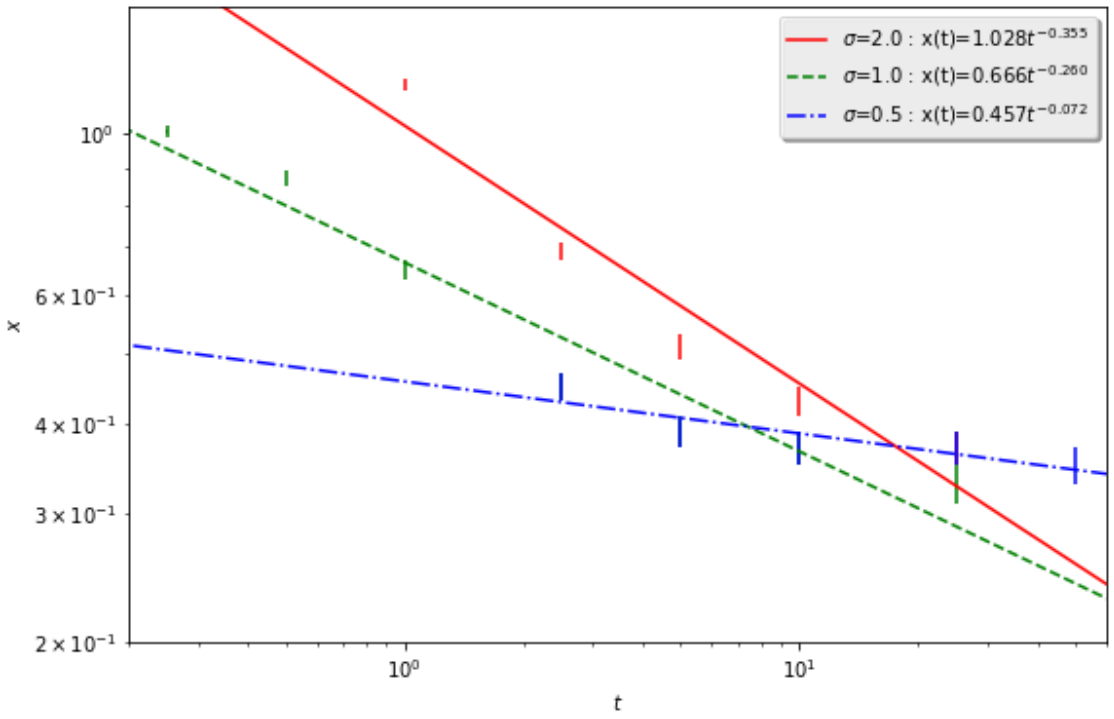

FIGURE 6 – Position du bris de pente en fonction du temps pour des simulations à H_0 constant à 14.4 en faisant varier la largeur du profil gaussien σ.

La première figure montre une forte dépendance de la vitesse de convergence de la fonction barrière par-rapport à la largeur du profil gaussien du terme source. Cela dit, pour les temps faibles, aucune caractéristique de confinement n'était observable lorsque σ était de 0.5, alors ces points n'ont pas été tracés. Les points restants sont donc la queue de la distribution et ont tendance à avoir un ralentissement plus grand que les points lorsque les temps étaient peu élevés. Il y a donc un biais qui a été introduit à cet endroit, puisque la position ne varie pas simplement selon une puissance en fonction du temps. La comparaison entre les courbes à $\sigma = 1$ et $\sigma = 2$ montre cependant encore une disparité entre les puissances auxquelles t est élevé, signe que la vitesse de convergence dépend bel et bien de la largeur du profil gaussien. En dérivant les expressions trouvées en première approximation, des relations allant de $v(t) \propto t^{-1.07}$ pour $\sigma = 0.5$ jusqu'à $v(t) \propto t^{-1.35}$ pour $\sigma = 2$ ont été trouvées.

Les dépendances en temps de la position pour un terme source étant une fonction marche sont plutôt semblables. Cependant, le biais étant présent pour la dernière situation en ce qui a trait aux positions aux temps faibles n'est plus présent, puisque la barrière s'établit dès le début de la simulation. Les pentes tracées pour chacun des cas sont très semblables aux pentes tracées dans le cas précédent, alors que la fonction marche d'une demi-largeur de 0.5 possède une relation $v(t) \propto t^{-1.1}$ approximativement, semblable

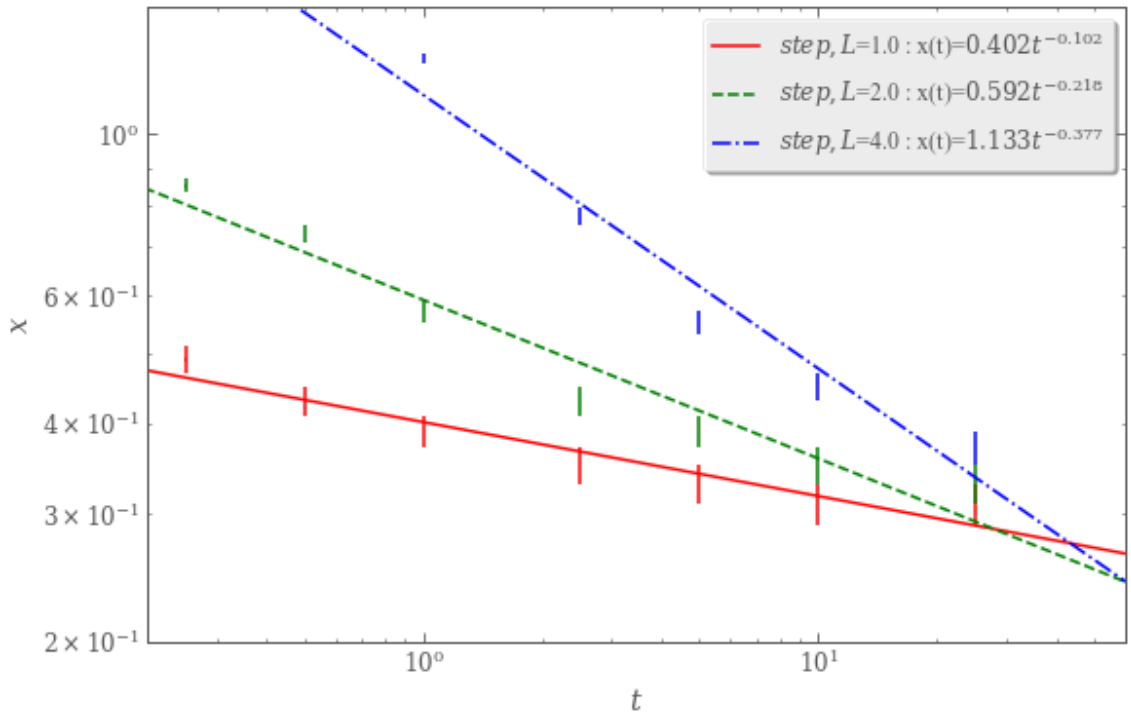

FIGURE 7 – Position du bris de pente en fonction du temps pour des simulations à H_0 constant à 14.4 en faisant varier la largeur de la fonction marche représentant le terme source.

à la valeur de $v(t) \propto t^{-1.07}$ trouvée précédemment pour un profil gaussien avec une demi-largeur à mi-hauteur également de 0.5. Les deux autres cas sont aussi semblables à leur cas homologue pour le profil gaussien, alors que les relations de vitesse sont de $v(t) \propto t^{-1.22}$ et de $v(t) \propto t^{-1.38}$ pour les fonctions marches de demi-largeur de 1.0 et 2.0, respectivement.

La seconde exploration numérique conduite est par-rapport aux valeurs de flux de chaleur en fonction du gradient de température pour une solution à l'équilibre possédant un effet barrière, ainsi qu'une autre solution où aucun effet barrière n'est observé à cause d'une trop faible valeur de l'amplitude du profil gaussien H_0 du terme source. Des simulations ont été conduites jusqu'à $t = 100$ pour être à l'équilibre pour les différentes combinaisons avec $H_0 \in \{2, 20\}$ et $\chi_1 \in \{2, 8\}$. Les valeurs de chi_1 ont été choisies de sorte qu'ils suivaient deux des courbes tracées à la figure 1 et les valeurs de H_0 ont été choisies car il y avait effet barrière lorsque $H_0 = 20$, et cet effet était absent lorsque $H_0 = 2$. Les résultats sont présentés à la figure 8.

Les points pour les solutions ne présentant pas d'effet barrière sont confinés dans la région où $-dT/dx < 0.4$ et restent dans le mode L. Par opposition, les points pour les solutions présentant un effet barrière avec $H_0 = 20$ s'étalent sur tout le domaine présenté et plusieurs des points de maille se retrouvent dans le mode H, signe qu'il y a confinement du plasma. Il est donc possible de voir directement sur ce graphique si il y a effet barrière pour une simulation donnée sans même voir les graphiques du

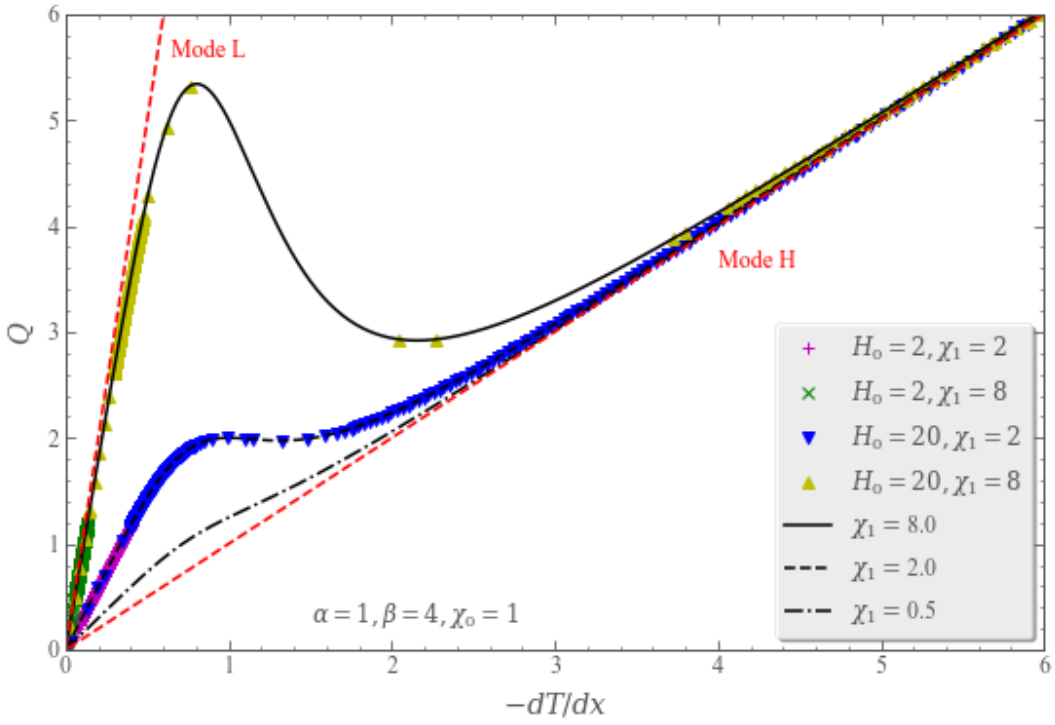

FIGURE 8 – Flux de chaleur Q en fonction du gradient de température pour des solutions à l'équilibre avec H_0 et 20 ($\chi_1 = 2$ et 8), et des solutions ne présentant pas d'effet barrière avec $H_0 = 2$ pour les mêmes valeurs de χ_1. Les figures pleine résolution et en couleur permettent de mieux déceler les différents points.

profil de température ou de l'évolution spatiotemporelle du gradient de température et de conductivité thermique totale, comme ceux présentés aux figures 2 et 3.

La méthode optimale pour avoir effet barrière semble donc être

4 Programme

4.1 Programme principal

```
# -*- coding: utf-8 -*-
"""
Title : Plasma confinement in a Tokamak

Author : Patrice Bechard

Date : February 20th, 2017
"""
#-----------------------Import Modules---------------------
import numpy as np
import matplotlib.pyplot as plt
```

```
import sys
import scipy as sp
import time
from itertools import cycle

plt.style.use('patrice')
print("Execution Start Time :",time.asctime())
start=time.time()            #time starts at beginning of execution

#-------------------------Functions-------------------------

def cond_initiales():
    """Setting initial conditions"""
    alpha=1
    beta=4
    chi0=1
    chi1=8
    init=[alpha,beta,chi0,chi1]     #to compute chi

    global dx,dt
    dx=0.02                         #gap between spatial points
    dt=5e-4                         #temporal step

    xrange=[-10,10]
    pos=np.linspace(xrange[0],xrange[1],int((xrange[1]-xrange[0])/dx))

    global tmax,nx
    tmax=25                         #maximum time
    nx=len(pos)                     #number of points in x

    temp=np.ones(nx)

    H0=14.4
    sigma=2
    values_H=[H0,sigma]

    return pos,temp,init,values_H

def confinement(pos,temp,init,values_H):

    a,b,c,r=[np.zeros(nx) for i in range(4)]    #initiate empty arrays
    temps=0

    timenum3=np.array([])                      #for the #3 exploration
    posnum3=np.array([])

    while temps<tmax:
        temps+=dt               #step forward in time
        temps=round(temps,6)    #having 'squared' time (no truncation bias)
        D=compute_D(temp,init)
```

```
    a,b,c=compute_abc(D)
    r=compute_r(r,temp,a,b,c,pos,values_H)
    r,a,b,c=cond_limites(r,a,b,c)
    tridiag(a,b,c,r,temp)

    if temps in [0.05,0.1,0.25,0.5,1.,2.5,5.,10.,25.,50.,100.]:

        plt.figure(0,figsize=(9,6))     #we plot figures like 2.16 and 2.17
        plt.plot(pos,temp,'k',lw=0.75)

        plt.figure(1,figsize=(9,6))

        dtdx=np.zeros(nx)
        for i in range(-1,nx-1):
            dtdx[i]=(temp[i-1]-temp[i+1])/(2*dx)    #derivative
        chi=compute_chi(temp,init)                  #chi

        plt.plot(pos,dtdx,'r',lw=0.75)
        plt.plot(pos,chi,'g--',lw=0.75)

        #num 3
        if temps>0.1:
            i=nx//2
            while chi[i]>4:i+=1
            timenum3=np.append(timenum3,[temps])
            posnum3=np.append(posnum3,[pos[i]])

        #num5
        if temps==tmax:                  #at equilibrium
            #plt.figure(4,figsize=(9,6))
            #plt.plot(-dtdx,-chi*dtdx,'.')
            f=open('datanum4_%s_chi%s.txt'%(str(values_H[0]),str(init[3])),'w')
            f.write('%s%s%s'%(-dtdx,'\n',-chi*dtdx))
            f.close()



plt.figure(0)
plt.plot(pos,compute_H(pos,values_H),'k--',lw=0.5)
plt.xlabel(r'$x$')
plt.ylabel(r'$T(x,t)$')
plt.axis([-10,10,0,max(temp)+5])
plt.savefig('fig_216_%d_chi%d.png'%(values_H[0],init[3]))

plt.figure(1)
plt.xlabel(r'$x$')
plt.ylabel(r'$dT(x,t)/dx$ , $\chi(x,t)$')
plt.axis([0,1.5,-0.25,10.25])
plt.savefig('fig_217%d_chi%d.png'%(values_H[0],init[3]))
```

```
    plt.figure(2,figsize=(9,6))
    sigtime=np.ones(len(timenum3))*dt
    sigpos=np.ones(len(timenum3))*dx
    f=open('num3data.txt','w')
    f.write('%s%s%s%s'%(str(timenum3),'\n',str(sigtime),'\n'))
    f.write('%s%s%s%s'%(str(posnum3),'\n',str(sigpos),'\n'))
    f.close()
    plt.loglog(timenum3,posnum3,'ko')
    plt.xlabel(r'$t$')
    plt.ylabel(r'$x$')

    plt.show()

    return

def cond_limites(r,a,b,c):

    r[0],r[-1]=1,1                      #limit conditions
    r[1],r[-2]=r[1]-a[1]*r[0],r[-2]+c[-2]*r[-1]
    a[1],b[0],c[0]=0,1,0                #changing the matrix
    a[-1],b[-1],c[-2]=0,1,0

    return r,a,b,c

def compute_D(temp,init):
    chi=compute_chi(temp,init)
    D0,D1,D2=[np.zeros(nx) for i in range(3)]
    for j in range(-1,nx-1):            #using negative index to avoid IndexError
        D0[j]=(chi[j-1]+chi[j])
        D1[j]=-(chi[j-1]+2*chi[j]+chi[j+1])
        D2[j]=(chi[j]+chi[j+1])
    return [D0,D1,D2]

def compute_chi(temp,init):
    chi=np.zeros(nx)
    for j in range(-1,nx-1):            #using negative index to avoid IndexError
        dtdx=(temp[j-1]-temp[j+1])/(2*dx)
        chi[j]=init[2]+init[3]/(1+init[0]*(dtdx**init[1]))
    return chi

def compute_abc(D):
    coeff=dt/(4*dx*dx)
    a=-coeff*D[0]                       #computing the matrix coefficients
    b=1-coeff*D[1]
    c=-coeff*D[2]
    return a,b,c

def compute_H(pos,values_H):
    H=np.zeros(nx)                      #computing the source term
```

```
    for i in range(nx):                 #gaussian profile
        H[i]=values_H[0]*np.exp(-pos[i]*pos[i]/(values_H[1]*values_H[1]))
#    for i in range(nx):                #step fct profile
#        if abs(pos[i])<values_H[1]:
#            H[i]=values_H[0]
    return H

def compute_r(r,temp,a,b,c,pos,values_H):
    r=crank_nicolson(r,temp,a,b,c)      #computingthe RHS vector
    H=compute_H(pos,values_H)
    r+=(dt*H)
    return r

def crank_nicolson(r,u,a,b,c):          #matrix product
    r[0]=(2-b[0])*u[0]-c[0]*u[1]
    for i in range(1,nx-1):
        r[i]=-a[i]*u[i-1]+(2-b[i])*u[i]-c[i]*u[i+1]
    r[-1]=-a[-1]*u[-2]+(2-b[-1])*u[-1]
    return r

def fig2_15(init):
    """Figure 2.15 from notes"""
    dtdx=np.linspace(-6,0,601)
    lines = ['k','k--','k-.']
    linecycler=cycle(lines)

#    plt.figure(figsize=(9,6))

    for chi1 in [8,2,0.5]:
        chi=init[2]+chi1/(1+init[0]*dtdx**init[1])
        Q=-chi*dtdx
        plt.plot(-dtdx,Q,next(linecycler),label=r'$\chi_1 = $%.1f'%chi1)

    plt.plot(-dtdx,10*(-dtdx),'r--')
    plt.plot(-dtdx,1*(-dtdx),'r--')
    plt.xlabel(r'$-dT/dx$')
    plt.ylabel(r'$Q$')
    plt.axis([0,6,0,6])
    plt.legend(shadow=True,fancybox=True)
    plt.text(0.65, 5.6, 'Mode L',color='r')
    plt.text(4, 3.6, 'Mode H',color='r')
    plt.text(1.5, .25, r'$\alpha = 1, \beta = 4, \chi_0 = 1$')
    plt.savefig('verif_fig_215.png')
    plt.show()
    return

def integrale_hamiltonien(pos,values_H):
    """for #1 (not really interesting)"""

    echantillon=np.linspace(2,22,11)
```

```
    for i in echantillon:
        values_H[0]=i
        H=compute_H(pos,values_H)
        integrale=trapeze(pos,H)
        plt.plot(i,integrale,'k.')
        #plt.plot(pos,H)
    plt.show()

    return

def trapeze(x,y):
    """numerical integration"""
    total=0
    for i in range(len(x)-1):
        dx=x[i+1]-x[i]
        dy=0.5*(y[i+1]+y[i])
        total+=(dx*dy)
    return total


def tridiag(a,b,c,r,u):
    """Tridiagonal solver from Press et. al. (1992), section 2.4

    a[n], b[n], c[n] are the three diagonals of the matrix
    r[n]is the RHS vector
    u[n] is the solution the the n step at input, solution to n+1 at output
    """
    gam=np.zeros(nx)                #vecteur de travail
    bet=b[0]
    if bet == 0 : raise Exception("beta=0 1")
    u[0]=r[0]/bet
    for j in range(1,nx-1):         #decomposition LU
        gam[j]=c[j-1]/bet
        bet=b[j]-a[j]*gam[j]
        if bet == 0 : raise Exception("beta=0 2")
        u[j]=(r[j]-a[j]*u[j-1])/bet
    for j in range(nx-2,0,-1):      #backsubstitution
        u[j]-=gam[j+1]*u[j+1]
    return 0

#---------------------------Main----------------------------

pos,temp,init,values_H=cond_initiales()

#integrale_hamiltonien(pos,values_H)

#fig2_15(init)

i=0
```

```
confinement(pos,temp,init,values_H)

totaltime=time.time()-start
print("Total time : %d h %d m %d s"%(totaltime//3600,totaltime//60,totaltime%60))
```

4.2 Programme secondaire : Exploration #3

```
# -*- coding: utf-8 -*-
"""
tokamak, num 3
"""

import matplotlib.pyplot as plt
import numpy as np
from itertools import cycle

#files to read

#files,analysis=['num3data_10.txt','num3data_save.txt','num3data_20.txt'],'H_0'
#files,analysis=['num3data_chi4.txt','num3data_save.txt','num3data_chi12.txt','num3data_chi16.txt
#files,analysis=['num3data_sig2.txt','num3data_save.txt','num3data_sig05.txt'],'\sigma'
files,analysis=['num3data_step05.txt','num3data_step1.txt','num3data_step2.txt'],'step, L'

plt.figure(figsize=(9,6))
plt.loglog()
color = ['r','g','b','m']       #we can iterate over plot properties
lines=['-','--','-.',':']

#values=[10,14.4,20]            #for legend
#values=[4,8,12,16]
#values=[2.0,1.0,0.5]
values=[1.0,2.0,4.0]

linecycler=cycle(color)
linecycler2=cycle(values)
linecycler3=cycle(lines)

for file in files:

    f=open(file)

    #read the messy data files
    timenum3=[float(x) for x in (f.readline()[:-2]).split(' ') if x.replace('.','').isdigit()]
    sigtime=[float(x) for x in (f.readline()[:-2]).split(' ') if x.replace('.','').isdigit()]
    posnum3=[float(x) for x in (f.readline()[:-2]).split(' ') if x.replace('.','').isdigit()]
    sigpos=[float(x) for x in (f.readline()[:-2]).split(' ') if x.replace('.','').isdigit()]

    """
    vitesse=[]
    for i in range(len(timenum3)-1):
```

```
        vit=np.abs((posnum3[i+1]-posnum3[i])/(timenum3[i+1]-timenum3[i]))
        vitesse.append(vit)

    plt.plot(timenum3[:-1],vitesse,'.')
    timenum3=timenum3[:-1]
    """
    c=next(linecycler)
    ligne=next(linecycler3)

    plt.errorbar(timenum3, posnum3, xerr=sigtime, yerr=sigpos, ls='',color=c)

    x,y=np.log10(timenum3),np.log10(posnum3)

    coefficients = np.polyfit(x, y, 1)          #fit for the data
    polynomial = np.poly1d(coefficients)
    varx=[1e-1,6e1]
    ys = 10**(polynomial(np.log10(varx)))
    coeff=10**coefficients[1]
    power=coefficients[0]

    plt.plot(varx, ys,ls=ligne,color=c,label='$%s $=%.1f : x(t)=$%.3ft^{%.3f}$'%(analysis,next(li


plt.xlabel(r'$t$')
plt.ylabel(r'$x$')
plt.axis([2e-1,6e1,2e-1,1.5e0])
plt.legend(fancybox=True,shadow=True)

plt.savefig('figures/num3_step.png')
plt.show()
```

4.3 Programme secondaire : Exploration #4

```
# -*- coding: utf-8 -*-
"""
tokamak num 4
"""


import numpy as np
import matplotlib.pyplot as plt
import sys
from itertools import cycle

def fig2_15(init):
    """Figure 2.15 from notes"""
    dtdx=np.linspace(-6,0,601)
    lines = ['k','k--','k-.']
    linecycler=cycle(lines)

    for chi1 in [8,2,0.5]:
```

```
        chi=init[2]+chi1/(1+init[0]*dtdx**init[1])
        Q=-chi*dtdx
        plt.plot(-dtdx,Q,next(linecycler),label=r'$\chi_1 = $%.1f'%chi1)

    plt.plot(-dtdx,10*(-dtdx),'r--')
    plt.plot(-dtdx,1*(-dtdx),'r--')
    plt.xlabel(r'$-dT/dx$')
    plt.ylabel(r'$Q$')
    plt.axis([0,6,0,6])
    plt.legend(shadow=True,fancybox=True)
    plt.text(0.65, 5.6, 'Mode L',color='r')
    plt.text(4, 3.6, 'Mode H',color='r')
    plt.text(1.5, .25, r'$\alpha = 1, \beta = 4, \chi_0 = 1$')
    plt.savefig('yaya.png')
    plt.show()
    return

#----------------------------MAIN-------------------------

files=['datanum4_2_chi2.txt','datanum4_2_chi8.txt','datanum4_20_chi2.txt','datanum4_20_chi8.txt']
count=0
plt.figure(figsize=(9,6))

#we plot data for each data file

for file in files:
    count+=1
    f=open(file)
    data=(f.read())
    time=str(data).replace('\n','').split('][')
    for i in range(len(time)):
        time[i]=time[i].replace(']','')

    time1=time[0].split(' ')
    values1=time[1].split(' ')

    time2=[]
    values2=[]
    for i in time1:
        try:
            time2.append(float(i))
        except:
            pass
    for i in values1:
        try:
            values2.append(float(i))
        except:
            pass

    print(len(time2))
```

```
    print(len(values2))
    if count==1:
        plt.plot(time2,values2,'m+',label=r'$H_0=2,\chi_1=2$')
    elif count==2:
        plt.plot(time2,values2,'gx',label=r'$H_0=2,\chi_1=8$')
    elif count==3:
        plt.plot(time2,values2,'bv',label=r'$H_0=20,\chi_1=2$')
    else:
        plt.plot(time2,values2,'y^',label=r'$H_0=20,\chi_1=8$')

alpha=1
beta=4
chi0=1
chi1=8
init=[alpha,beta,chi0,chi1]     #to compute chi

fig2_15(init)
```

Références

[1] Charbonneau, P., *PHY3075 - Modélisation numérique en physique, Chapitre 2 : Équations différentielles partielles*, Université de Montréal, Montréal, 2017, 38p.