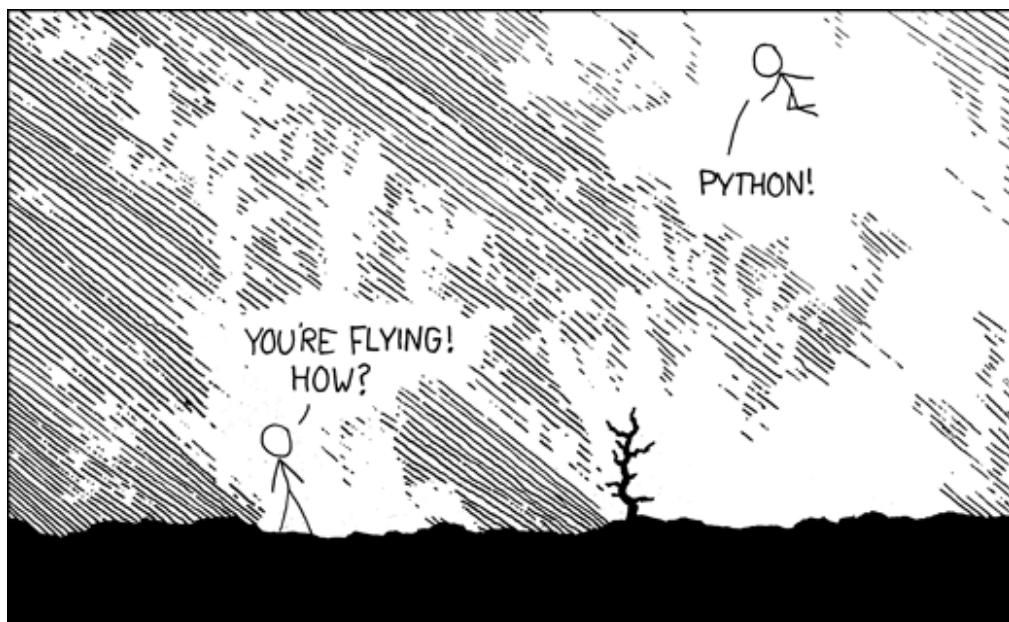


# Py4Phys I

Bloc Signal



I wrote 20 short programs in Python yesterday. It was wonderful.  
Perl, I'm leaving you.

xkcd.com

## Table des matières

<b>I Bloc Signal</b>	<b>1</b>
1 <i>S01</i> Oscillateur harmonique: isochonisme des oscillations . . . . .	3
2 <i>S01</i> Oscillateur harmonique: conservation de l'énergie . . . . .	4
3 <i>S02</i> Onde progressive: visualisations temporelles et spatiales . . . . .	6
4 <i>S02</i> Ondes progressives: animation d'une superposition . . . . .	9
5 <i>S03</i> Construction d'un diagramme de Fresnel . . . . .	12
6 <i>S03</i> Interférences . . . . .	16
7 <i>S03</i> Battements . . . . .	19
8 <i>S03</i> Ondes stationnaires . . . . .	20
9 <i>S03</i> Diffraction . . . . .	20
10 <i>S04</i> Lois de Descartes . . . . .	23
11 <i>S04</i> Réfraction dans une goutte: explication de l'arc-en-ciel . . . . .	23
12 <i>S05</i> Lentilles minces: construction graphique . . . . .	25
13 <i>S05</i> Distortion chromatique au passage d'une lentille . . . . .	25
14 <i>S05</i> Illustration de la règle des 4P: Plus Près, Plus Plat . . . . .	27
15 <i>S06</i> Construction d'un paquet d'onde pour la Mécanique Quantique . . . . .	30
16 <i>S07</i> Résolution électrique par pivot de Gauss . . . . .	31
17 <i>S08</i> Réseau électrique du premier ordre simple . . . . .	31
18 <i>S08</i> Réseau électrique complexe . . . . .	31
19 <i>S09</i> Oscillateur amorti en régime libre . . . . .	31
20 <i>S10</i> Oscillateur amorti en régime forcé . . . . .	33
21 <i>S11</i> Module pour dessiner un diagramme de Bode . . . . .	33
22 <i>S11</i> Filtre intégrateur . . . . .	34
23 <i>S11</i> Filtre Dérivateur . . . . .	35
24 <i>S11</i> Filtre bizarre . . . . .	36
25 <i>S11</i> Diagrammes en amplitude et échelle linéaire . . . . .	37
26 <i>S11</i> Filtres du second ordre . . . . .	39
<b>II Bloc Mécanique</b>	<b>42</b>
1 <i>M2</i> Exploration numérique de l'influence de la trainée . . . . .	42
2 <i>M2</i> Influence de la trainée: portrait de phase . . . . .	42
3 <i>M4</i> Module de génération de portrait de phase . . . . .	42
4 <i>M4</i> Pendule simple: non isochronisme des oscillations . . . . .	43
5 <i>M4</i> Pendule simple: portrait de phase . . . . .	43
6 <i>M4</i> Pendule simple: Oscillations amorties et portrait de phase . . . . .	43

7	$\mathcal{M}4$	Oscillateurs de Landau: effets non linéaires . . . . .	43
8	$\mathcal{M}4$	Oscillateur de Landau: Portrait de phase . . . . .	43
9	$\mathcal{M}5$	Mouvement hélicoïdal dans un champ magnétique . . . . .	43
10	$\mathcal{M}5$	Mouvement dans les champs E et B . . . . .	43
11		Papillon de Lorentz . . . . .	43
12		Pendule double: sensibilité aux conditions initiales . . . . .	43
13		Hypéron: rotation chaotique et section de Poincaré . . . . .	43
		<b>III Bloc Thermodynamique</b>	<b>50</b>
1	$T1$	Balles rebondissantes en boîte: illustration du facteur de Boltzmann . . . . .	50
2	$T1$	Particules dans une boîte: illustration du mouvement brownien . . . . .	57
3	$T1$	Particules libres dans une boîte: distribution des vitesses . . . . .	57
4	$T2$	Isothermes d'Andrews via CoolProp . . . . .	62
5	$T2$	Diagramme ( $P, v$ ) via CoolProp . . . . .	66
6	$T2$	Diagramme ( $P, T$ ) via CoolProp . . . . .	69
7	$T5$	Comparaison des isentropiques pour un gaz réel et pour un gaz parfait . . . . .	70
8	$T6$	Cycle de Carnot: gaz réel et gaz parfait . . . . .	70
9	$T6$	Résolution d'un exercice sur un cycle Diesel . . . . .	73
10	$T6$	Diagramme ( $P, h$ ) via CoolProp . . . . .	79
		<b>IV Bloc Induction</b>	<b>84</b>
1	$I1$	Tracé de lignes de champ magnétique . . . . .	84
2	$I2$	Champ tournant diphasé . . . . .	89
3	$I2$	Champ tournant triphasé . . . . .	89
4	$I4$	Couplage inductif de deux circuits . . . . .	89

Exemple I.1

## S01 Oscillateur harmonique: isochonisme des oscillations

```

1   '''
2   Simple résolution numérique de l'équation d'un oscillateur harmonique pour
3   illustrer l'isochronisme des oscillations quelle que soit l'amplitude de départ.
4   '''

5
6   import numpy as np          # Pour np.linspace
7   import scipy as sp           # Simple alias usuel
8   import scipy.integrate       # Pour l'intégration
9   import matplotlib.pyplot as plt # Pour les dessins

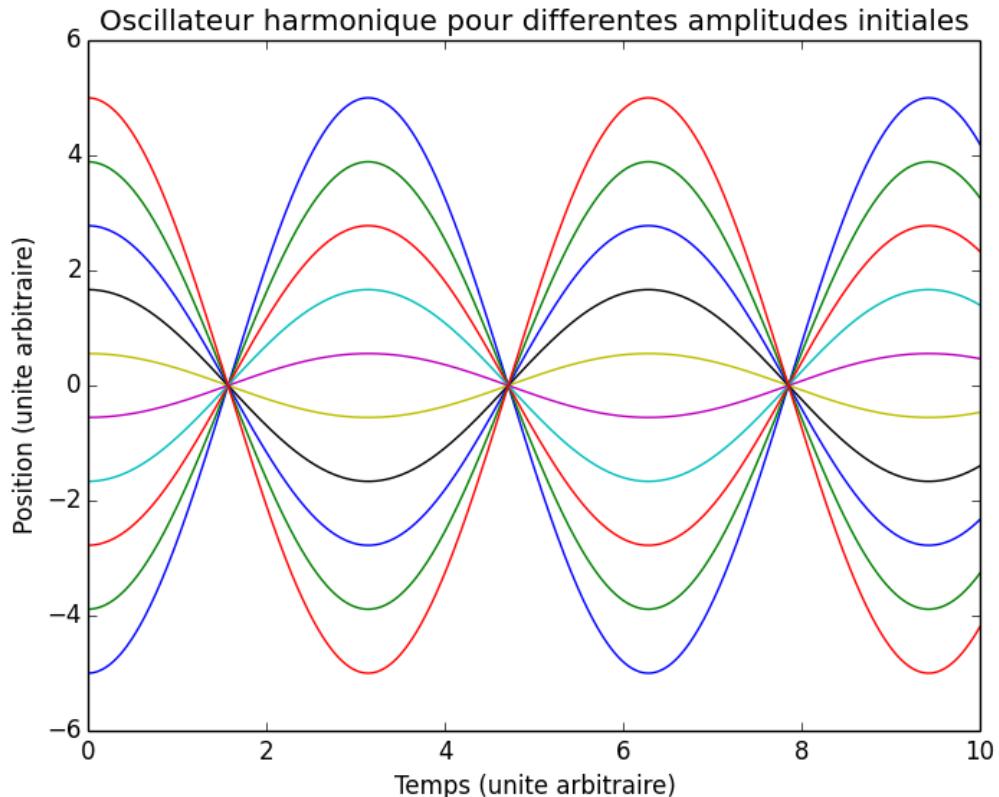
10
11 omega0 = 1      # On définit la pulsation propre

12
13 def equadiff(y,t):
14     '''Renvoie l'action du système  $dx/dt = vx$  et  $dvx/dt = -\omega_0^2 * x$ 
15     soit bien l'oscillateur harmonique  $x'' + \omega_0^2 * x = 0$ '''
16     x,vx = y                  # y contient position et vitesse
17     return [vx,- omega0**2 * x] # On renvoie un doublet pour  $[dx/dt, dvx/dt]$ 

18

```

```
19 nb_CI = 10 # Nombre de conditions initiales explorées
20
21 t = np.linspace(0,10,1000)           # Le temps total d'intégration
22 x0= np.linspace(-5,5,nb_CI)         # Les positions initiales choisies
23 v0= [0]*nb_CI                      # Les vitesses initiales choisies
24
25 for i in range(nb_CI):              # Pour chaque condition initiale
26     # L'intégration proprement dite
27     sol = sp.integrate.odeint(equadiff,[x0[i],v0[i]],t)
28     x = sol[:,0]                      # Récupération de la position
29     plt.plot(t,x)                   # et affichage
30
31 # Il ne reste que le traitement cosmétique
32
33 plt.title('Oscillateur harmonique pour différentes amplitudes initiales')
34 plt.ylabel('Position (unité arbitraire)')
35 plt.xlabel('Temps (unité arbitraire)')
36 plt.savefig('PNG/S01_oscillateur_harmonique_periode.png')
```



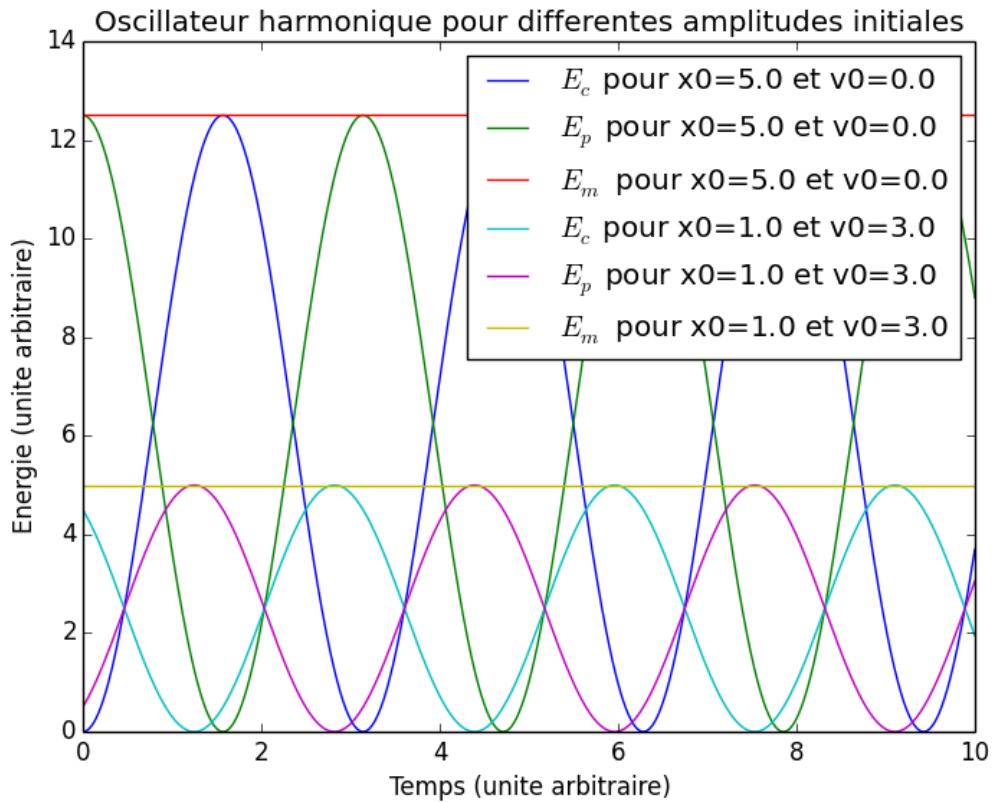
Exemple I.2

## S01 Oscillateur harmonique: conservation de l'énergie

1 , , ,

2 Illustration numérique de la conservation de l'énergie mécanique pour  
3 une équation d'oscillateur harmonique.

```
4   ''
5
6 import numpy as np          # Pour np.linspace
7 import scipy as sp           # Simple alias usuel
8 import scipy.integrate       # Pour l'intégration
9 import matplotlib.pyplot as plt # Pour les dessins
10
11 m = 1                      # Masse du mobile
12 k = 1                      # Constante de raideur du ressort
13 omega0 = (k/m)**0.5         # On définit la pulsation propre
14
15 def equadiff(y,t):
16     '''Renvoie l'action du système  $dx/dt = vx$  et  $dvx/dt = -omega0**2 * x$ 
17     soit bien l'oscillateur harmonique  $x'' + omega0**2 * x = 0$ '''
18     x,vx = y                  # y contient position et vitesse
19     return [vx,- omega0**2 * x] # On renvoie un doublet pour [dx/dt,dvx/dt]
20
21 nb_CI = 2 # Nombre de conditions initiales explorées
22
23 t = np.linspace(0,10,1000)      # Le temps total d'intégration
24 x0= np.linspace(5,1,nb_CI)      # Les positions initiales choisies
25 v0= np.linspace(0,3,nb_CI)      # Les vitesses initiales choisies
26
27 for i in range(nb_CI):          # Pour chaque condition initiale
28     # L'intégration proprement dite
29     sol = sp.integrate.odeint(equadiff,[x0[i],v0[i]],t)
30     x = sol[:,0]                 # Récupération de la position
31     v = sol[:,1]                 # et de la vitesse
32     Ec = 0.5*m*v**2             # Energie cinétique
33     Ep = 0.5*k*x**2             # Energie potentielle
34     Em = Ec + Ep                # Energie mécanique
35     lab = ' pour x0={} et v0={}'.format(round(x0[i],1),round(v0[i],1))
36     plt.plot(t,Ec,label='$E_c$'+lab) # Affichage Ec
37     plt.plot(t,Ep,label='$E_p$'+lab) # Affichage Ep
38     plt.plot(t,Em,label='$E_m$'+lab) # Affichage Em
39
40 # Il ne reste que le traitement cosmétique
41
42 plt.title('Oscillateur harmonique pour différentes amplitudes initiales')
43 plt.ylabel('Energie (unité arbitraire)')
44 plt.xlabel('Temps (unité arbitraire)')
45 plt.legend()
46 plt.savefig('PNG/S01_oscillateur_harmonique_energie.png')
```



### Exemple I.3

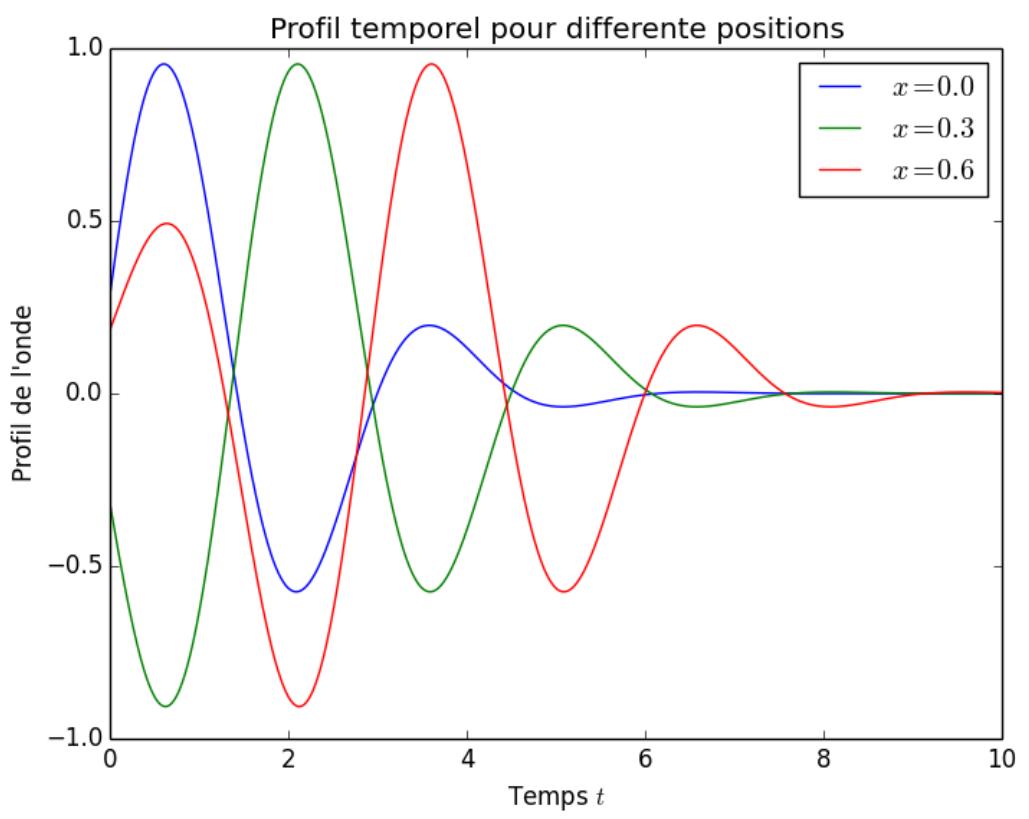
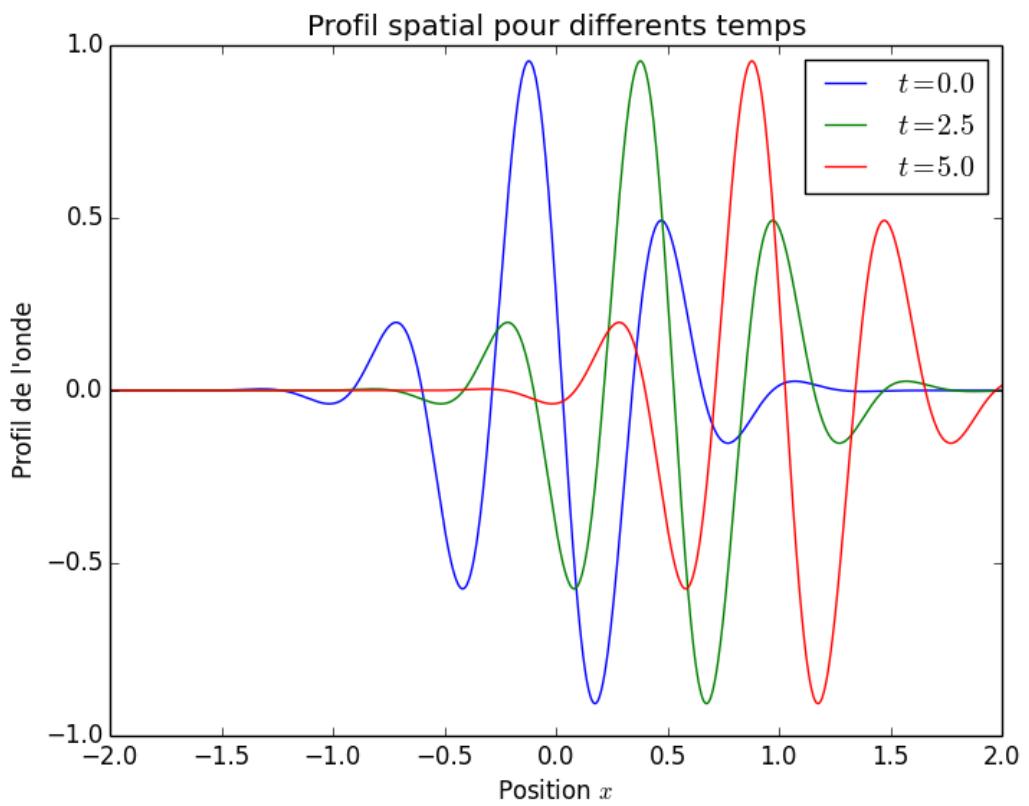
## S02 Onde progressive: visualisations temporelles et spatiales

```

1   """
2   Illustration du phénomène de propagation vers la droite d'une onde de forme
3   quelconque à la fois au cours du temps dans un profil spatial, et spatialement
4   dans un profil temporel.
5   """
6
7   import numpy as np          # Pour np.linspace, np.exp et np.cos
8   import matplotlib.pyplot as plt # Pour les dessins
9
10
11  def f(u,k=10):
12      '''Le profil de l'onde à propager: une gaussienne multipliée par un cosinus.'''
13      return np.exp(-3*u**2) * np.cos(k*u-5)
14
15  nb_points = 1000    # Le nombre de points d'échantillonnage du graphe
16  nb_courbes = 3       # Le nombre de courbes à représenter
17
18  # Tout d'abord la visualisation spatiale
19
20  x = np.linspace(-2,2,nb_points)    # Echantillonnage en position
21  t = np.linspace(0,5,nb_courbes)    # On regarde le profil à différents temps
22  c = 0.2 # Vitesse de propagation de l'onde

```

```
23
24 for ti in t:
25     fi = f(x-c*t) # Echantillonnage du profil pour les différents x
26     plt.plot(x,fi,label='$t={}$'.format(round(ti,1))) # Affichage
27
28 # La cosmétique
29
30 plt.title('Profil spatial pour differents temps')
31 plt.xlabel('Position $x$')
32 plt.ylabel("Profil de l'onde")
33 plt.legend()
34 plt.savefig('PNG/S02_onde_progressive_spatial.png')
35 plt.clf()
36
37 # Tout d'abord la visualisation spatiale
38
39 t = np.linspace(0,10,nb_points) # Echantillonnage en temps
40 x = np.linspace(0,0.6,nb_courbes) # On regarde le profil à différentes positions
41 c = 0.2 # Vitesse de propagation de l'onde
42
43 for xi in x:
44     fi = f(xi-c*t) # Echantillonnage du profil pour les différents t
45     plt.plot(t,fi,label='$x={}$'.format(round(xi,1))) # Affichage
46
47 # La cosmétique
48
49 plt.title('Profil temporel pour differente positions')
50 plt.xlabel('Temps $t$')
51 plt.ylabel("Profil de l'onde")
52 plt.legend()
53 plt.savefig('PNG/S02_onde_progressive_temporel.png')
54 plt.clf()
```



Exemple I.4

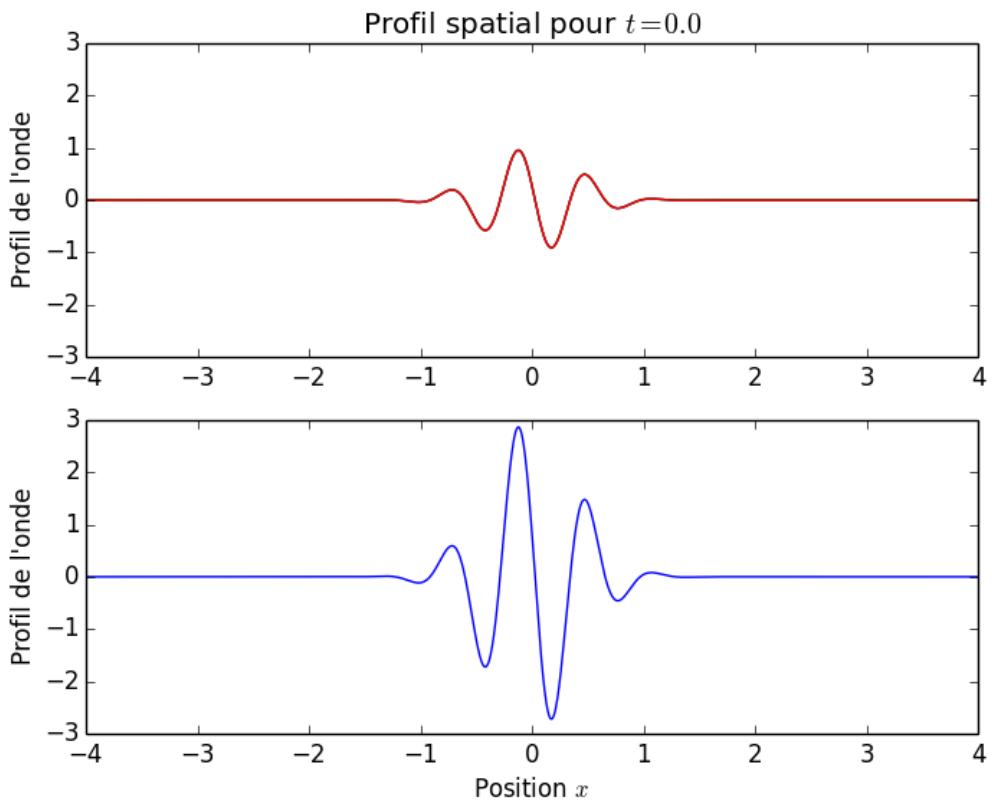
## S02 Ondes progressives: animation d'une superposition

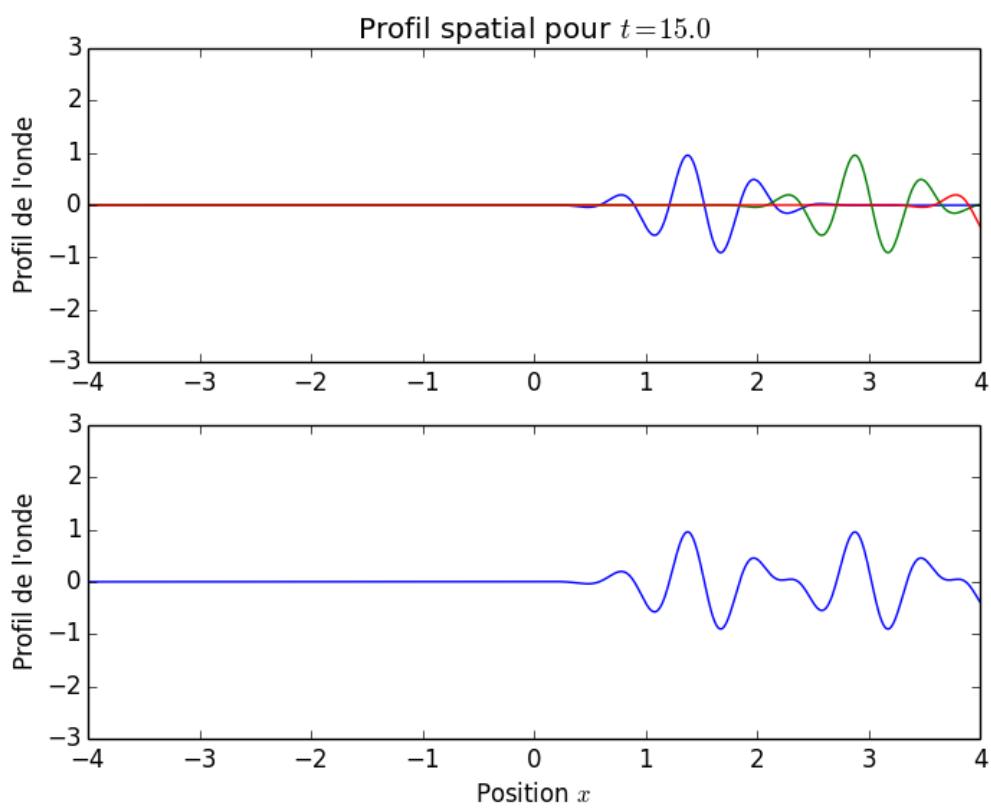
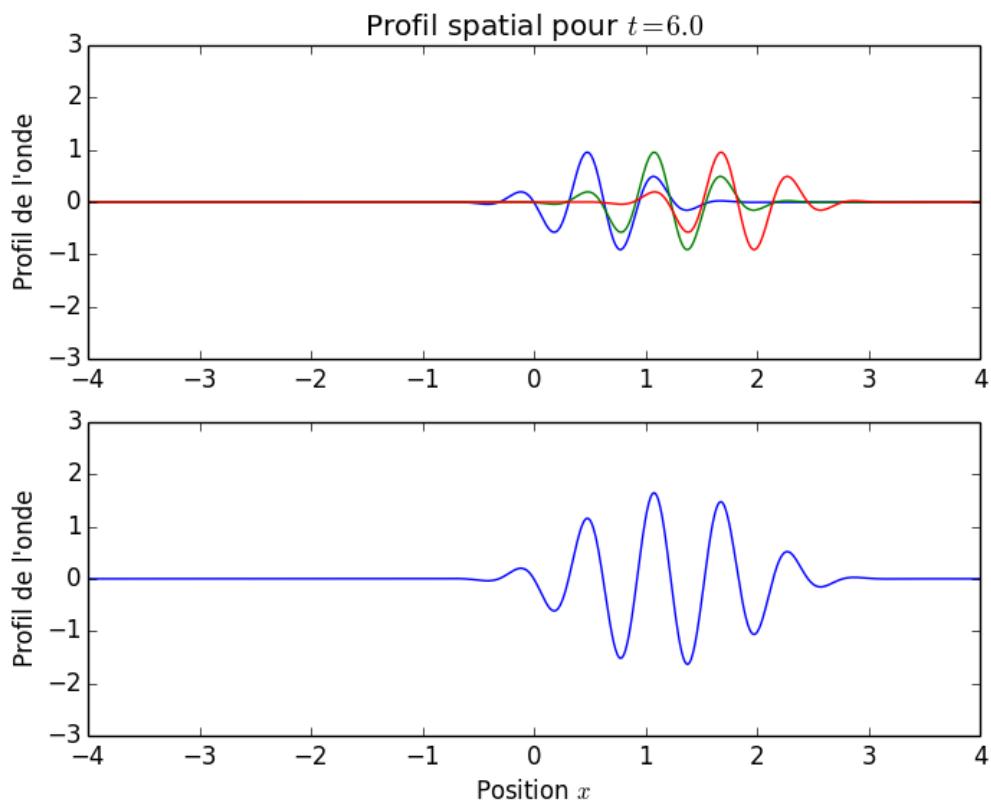
```
1   ''',
2   Animation montrant la propagation de trois ondes de profil identique mais de
3   vitesse différente ainsi que leur superposition. Peut servir à illustrer
4   qualitativement la notion d'étalement du paquet d'onde.
5   ''',
6
7   import numpy as np          # Pour np.linspace, np.exp et np.cos
8   import matplotlib.pyplot as plt # Pour les dessins
9
10
11  def f(u,k=10):
12      '''Le profil de l'onde à propager: une gaussienne multipliée par un cosinus.'''
13      return np.exp(-3*u**2) * np.cos(k*u-5)
14
15  nb_points = 1000    # Le nombre de points d'échantillonnage du graphe
16  nb_courbes = 3      # Le nombre de courbes à représenter
17  nb_images = 151     # Le nombre d'images à créer.
18  c = np.linspace(0.1,0.3,nb_courbes) # Les différentes vitesses de propagation
19
20  # On fait une visualisation spatiale
21
22  x = np.linspace(- 4, 4,nb_points)    # Echantillonnage en position
23  t = np.linspace(-15,15,nb_images)    # On regarde le profil à différents temps
24
25  base_name = 'PNG/S02_onde_progressive_superposition_'
26
27  for i,ti in enumerate(t):
28      plt.subplot(211)                  # La première sous-figure
29      plt.ylim(-3,3)                  # et ses limitations
30      plt.ylabel("Profil de l'onde")# ainsi que le label des ordonnées
31      plt.title('Profil spatial pour $t={}$.format(round(ti,2)))')
32      ftot = np.zeros(len(x))        # Initialisation du signal superposé
33      for cj in c:                  # On regarde chaque vitesse
34          fi = f(x-cj*ti) # Echantillonnage du profil pour les différents x
35          plt.plot(x,fi) # Affichage
36          ftot += fi       # On ajoute à l'onde totale
37      plt.subplot(212)                  # La seconde sous-figure
38      plt.ylim(-3,3)                  # et ses limitations
39      plt.ylabel("Profil de l'onde")# ainsi que le label des ordonnées
40      plt.plot(x,ftot)               # Affichage du signal superposé
41      plt.xlabel('Position $x$')    # et label des abscisses
42      fichier = base_name + '{:03d}'.format(i)
43      plt.savefig(fichier)         # On sauvegarde un fichier par temps
44      plt.clf()                   # et on nettoie pour le suivant
45
46  # Ne reste plus qu'à rassembler en un fichier gif à l'aide de convert
```

```
47
48 import subprocess
49
50 cmd = "convert -delay 1 -dispose Background +page {}".format(base_name + '*.png')
51 cmd+= " -loop 0 {}".format(base_name + 'film.gif')
52
53 print("Execution de la commande de conversion")
54 print(cmd)
55 p = subprocess.Popen(cmd, shell=True)
56 print("Fin de la commande de conversion")
```

L'animation complète peut être trouvée à l'adresse suivante (le lien devrait être cliquable):

[http://pcsi.kleber.free.fr/IPT/doc/S02\\_onde\\_progressive\\_superposition\\_film.gif](http://pcsi.kleber.free.fr/IPT/doc/S02_onde_progressive_superposition_film.gif)





Exemple I.5

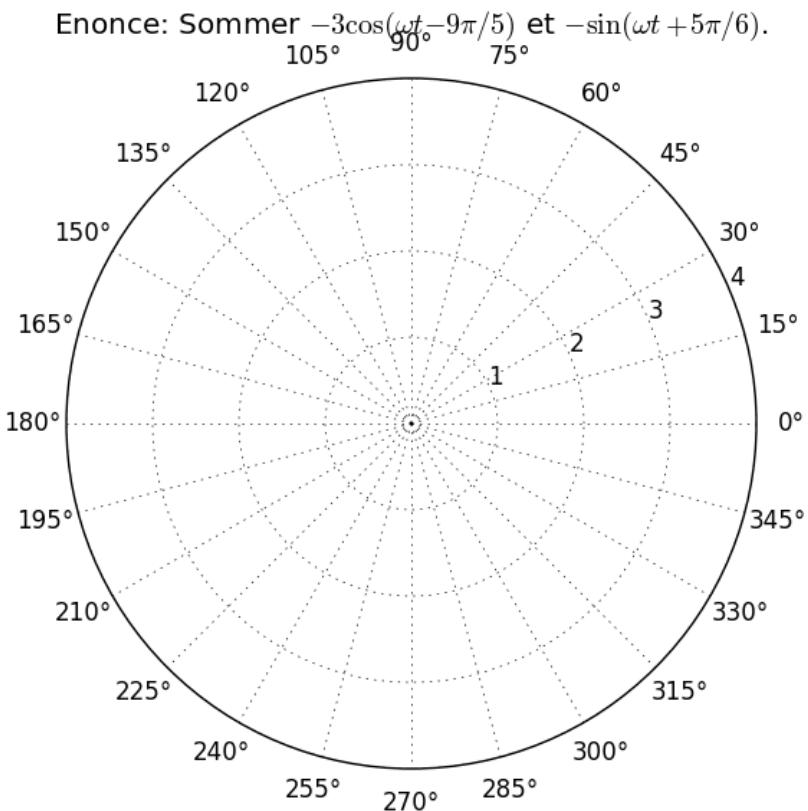
## S03 Construction d'un diagramme de Fresnel

```
1   ''',
2   Programme complet permettant la construction de diagrammes de Fresnel pour que
3   les élèves puissent s'entraîner. On génère à la fois un énoncé (avec les ondes
4   à sommer) et le corrigé correspondant. Le programme a été pensé pour pouvoir
5   sommer plus que deux ondes, même si au-delà de trois, cela devient
6   difficilement lisible (de plus, comme on ne doit pas dépasser le cadre, la
7   génération aléatoire peut prendre du temps avant de converger).
8
9   L'idée est de rassembler les divers énoncés sur certaines pages d'un document
10  LaTeX avec les corrigés correspondants sur la page suivant pour que les élèves
11  aient tout de suite le corrigé en vue.
12  ''',
13
14 from math import *                      # Pour sqrt, pi, cos et Cie
15 from cmath import *                     # Pour les complexes, notamment phase()
16 import random                          # Pour les tirages aléatoires
17 import matplotlib.pyplot as plt # Pour les dessins
18
19 # Décommenter pour débugger: afin que l'aléatoire soit toujours le même !
20 #random.seed(20)
21
22 # Les angles dont on prendra des multiples
23 angles_dict = {'\pi/{':pi.format(i):pi/i for i in range(2,7)}
24 angles = list(angles_dict.keys())
25 # Les tailles de flèches disponibles
26 tailles= [1,2,3]
27 max_size = 4 # Taille max pour l'affichage
28
29 def new_polar(z):
30     '''Bien sûr, il y a incohérence entre le couple (r,theta) renvoyé par les
31     complexes et l'attente de (theta,r) pour les coordonnées polaires de
32     matplotlib...'''
33     p = polar(z)
34     return tuple(reversed(p))
35
36 def ajoute_fleche(couple, ultra=False, origine = 0j, thin=False):
37     '''Rajoute une flèche correspond au complexe du couple (module,phase)
38     donné en argument. Paramètres optionnels:
39     * origine (complex): pour décaler l'origine ;
40     * ultra (bool): pour une trait plus gros.'''
41     if ultra : w,hw = 3,10                 # Veut-on un trait plus épais ?
42     elif thin: w,hw = 0.1,0.1              # ou plus fin ?
43     else      : w,hw = 1,7                # sinon une taille normale
44     c = rect(*couple)                   # Conversion en complexe
45     if abs(c) > max_size: raise        # Si on dépasse, c'est la fin !
46     plt.annotate(' ',xytext=new_polar(origine),
```

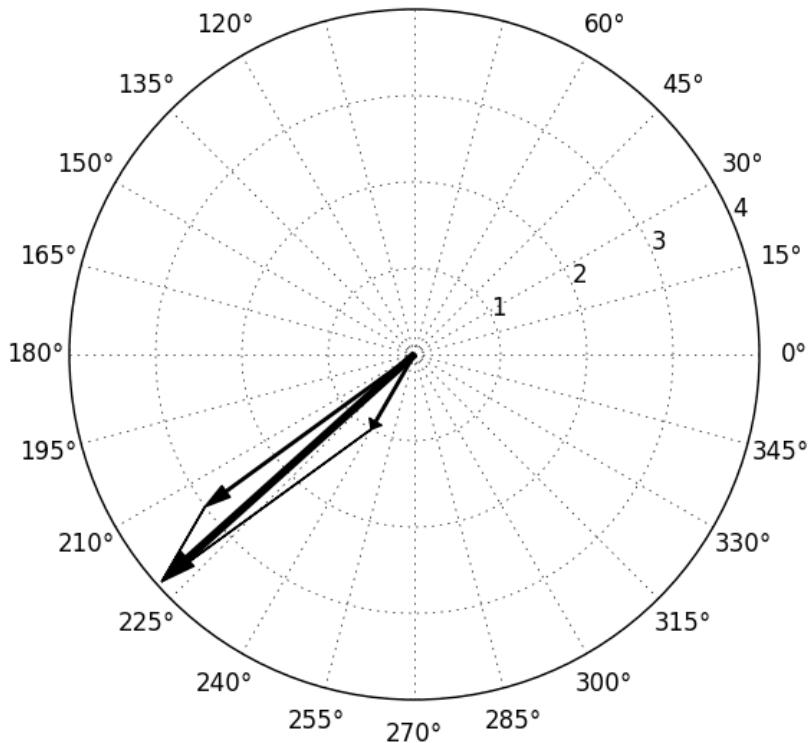
```
47                     xy=new_polar(origine+c),
48                     arrowprops=dict(width=w,facecolor='black',headwidth=hw))
49
50 def pointilles(liste_de_couples,reverse=False):
51     '''Construit les pointillés pour relier le départ à l'arrivée.'''
52     s = ''
53     ldc = liste_de_couples[:]
54     if reverse: ldc.reverse()
55     somme = rect(*ldc[0])
56     R,THETA = [ldc[0][0]],[ldc[0][1]]
57     for i in range(len(liste_de_couples)-1):
58         ajoute_fleche(ldc[i+1],origine=somme,thin=True)
59         somme += rect(*ldc[i+1])
60
61 def get_one(iterable):
62     '''Histoire de pouvoir récupérer un élément au hasard dans une liste.'''
63     nb = random.randint(0,len(iterable)-1)
64     return iterable[nb]
65
66 def enonce_et_corrige(nb_ondes=2,num_fichier = 1):
67     '''Tirage au sort de valeurs et construction d'un énoncé et du corrigé
68     correspondant. Renvoie un couple de nom des fichiers (énoncé,corrigé).'''
69     # Initialisation
70     ondes_enonce = [] # liste de strings
71     ondes_corrige= [] # liste de couples (norme,argument)
72     # Quelques tableaux utiles
73     sin_cos = [r'\cos',r'\sin']
74     signes = ['+','+', '+', '-'] # 1 -> +, -1 -> -
75     # Et c'est parti pour la sélection !
76     for i in range(nb_ondes): # Tirage des ondes
77         nb_phi = random.randint(0,10) # Le nombre multipliant l'angle
78         sincos = random.randint(0,1) # Tirage à pile ou phase pour sinus ou cosinus
79         signeA = (-1)**random.randint(0,1) # Signe aléatoire pour l'amplitude
80         signeP = (-1)**random.randint(0,1) # et pour la phase
81         A = signeA * get_one(tailles)
82         phi = get_one(angles)
83         # Le cas particulier où l'amplitude est unitaire
84         if A == 1: Astring = ''
85         elif A == -1: Astring = '_'
86         else :          Astring = str(A)
87         # Et si le facteur multiplicatif est nul ou unitaire
88         if nb_phi == 0: phi_string = ''
89         elif nb_phi == 1: phi_string = '{}{}'.format(signes[signeP],phi)
90         else :           phi_string = '{}{}{}'.format(signes[signeP],nb_phi,phi)
91         # Construction de la chaîne de l'énoncé
92         enonce = "{}{}(\omega t{})".format(Astring,sin_cos[sincos],phi_string)
93         ondes_enonce.append(enonce)
94         # Maintenant on calcule module et argument
95         module = A
```

```
96         # sin(x) = cos(x *-* pi/2)
97         argument = angles_dict[phi]*nb_phi*signeP - sincos*pi/2
98         ondes_corrigé.append((module,argument))
99     # Reste à faire le corrigé et l'énoncé
100    énonce = fabrique_énonce_fresnel(ondes_énonce,num_fichier)
101    try:    # Cas où cela dépasse du cadre
102        corrigé=fabrique_corrigé_fresnel(ondes_corrigé,ondes_énonce,num_fichier)
103    except: # Et on reessaie un coup !
104        return énonce_et_corrigé(nb_ondes)
105    # Sinon c'est bon.
106    return énonce,corrigé
107
108 def initialisation_graphe(ondes,énonce=True):
109     plt.clf()                  # Nettoyage, on commence un nouveau graphe
110     plt.axes(polar=True) # On initie un graphe en polaires
111     # Puis on définit le titre
112     if énonce: titre = 'Énonce: Sommer '
113     else      : titre = 'Corrigé: Sommer '
114     titre += ", ".join(['${}{}'.format(o) for o in ondes[:-1]])
115     titre += " et ${}{}".format(ondes[-1])
116     plt.title(titre)
117     # et finalement les grilles en distances
118     plt.rgrids([i+1 for i in range(max_size)])
119     plt.thetagrids([i*15 for i in range(360//15)]) # et en angles
120
121
122 def fabrique_énonce_fresnel(ondes,num_fichier):
123     '''À partir d'une série d'ondes sous forme de chaînes de caractère,
124     construit l'énoncé de la résolution avec Fresnel. Renvoie le nom du
125     fichier sauvegardé'''
126     initialisation_graphe(ondes)
127     fichier = 'PNG/{}ondes_énonce_{:03d}'.format(len(ondes),num_fichier)
128     plt.savefig(fichier)
129     return fichier
130
131 def fabrique_corrigé_fresnel(liste_des_couples,ondes_énonces,num_fichier):
132     '''Fabrication du corrigé pour les diagrammes de Fresnel correspondant à
133     une somme d'au moins deux ondes (mais potentiellement plus). La liste des
134     ondes se présente sous forme d'une liste de couples (norme,argument) à
135     chaque fois. Renvoie une chaîne correspondant au dessin TikZ final, ainsi
136     que l'amplitude et la phase (en degrés) pour possible affichage.'''
137     # Première étape: initialisation
138     initialisation_graphe(ondes_énonces,énonce=False)
139     # Puis, on ordonne par angle croissant
140     ldc = sorted(liste_des_couples, key=lambda c: (c[1]-pi)%(2*pi) )
141     point_final = 0
142     for c in ldc:
143         ajoute_fleche(c)          # Les flèches individuelles
144         point_final += rect(*c)  # On ajoute au total (en complexe)
```

```
145     ajoute_fleche(polar(point_final),ultra=True) # La flèche finale
146     # On rajoute aussi la construction en pointillés
147     pointilles(ldc)                      # dans un sens
148     pointilles(ldc,reverse=True)          # et dans l'autre
149     fichier = 'PNG/{}ondes_corrige_{:03d}'.format(len(ondes_enonces),num_fichier)
150     plt.savefig(fichier)
151     return fichier
152
153 # Exemple d'utilisation de tout ceci:
154 for i in range(5):
155     fichier_enonce,fichier_corrige = enonce_et_corrige(2,i)
156     # Ne reste plus qu'à faire quelque chose de ces noms de fichier, par
157     # exemple écrire un fichier .tex au vol...
```



Corrige: Sommer  $-3\cos(\omega t - 9\pi/5)$  et  $-\sin(\omega t + 5\pi/6)$ .



Exemple I.6

### S03 Interférences

```

1   """
2   Simulation d'un phénomène d'interférences à deux ondes circulaires (comme
3   des ronds dans l'eau). Les graphes du bas et de la droite représentent des
4   coupes respectivement à x et à y fixé.
5
6   On peut simuler une décroissance de l'onde en  $1/r$  (sauf au centre pour éviter
7   toute divergence) ou simplement avoir une fonction sinusoïdale en jouant sur
8   la fonction 'source'. En cas de changement, il faut penser à modifier les
9   valeurs  $v_{min}, v_{max}$  qui définissent les extrêmes du code couleur adopté.
10  """
11
12 import numpy as np          # Pour les facilités de calcul
13 import matplotlib.pyplot as plt # Pour les dessins
14
15 def source(x,y,t,x0=0,y0=0,phi=0):
16     '''La fonction représentant notre source située en (x0,y0)'''
17     k,w,epsilon = 5,1,1          # Quelques constantes
18     r = np.sqrt((x-x0)**2+(y-y0)**2) # La distance à la source
19     u = k*r - w*t + phi        # La variable de déplacement
20     #res = np.sin(u)
21     res = np.sin(u)/(r+epsilon) # Ou décroissance de l'amplitude...
22     res[u > 0] = 0.0           # Pour s'assurer qu'à  $t < 0$ , il n'y a pas d'onde

```

```
23     return res
24
25 ext = 6.0          # Les limites de la fenêtre d'étude
26 pos = 3.5          # Positions des sources symétriquement selon x
27 phi = np.pi/2      # Déphasage de la deuxième source
28 tmin,tmax = 0,60   # L'intervalle de temps d'étude
29 dt = 0.1           # et le pas
30 xcut = 1            # Les plans de coupe en x
31 ycut = 2            # et en y
32 vmin,vmax=-0.5,0.5 # Les valeurs extrêmes de l'amplitude
33 dx,dy = 0.05,0.05   # Resolution
34 x = np.arange(-ext,ext, dx) # Axe en x
35 y = np.arange(ext,-ext,-dy) # et en y
36 X,Y = np.meshgrid(x,y)    # pour produire la grille
37
38 # Pour définir correctement les limites de la fenêtre.
39 xmin, xmax, ymin, ymax = np.amin(x), np.amax(x), np.amin(y), np.amax(y)
40 extent = xmin, xmax, ymin, ymax
41
42 base_name = 'PNG/S03_interferences_' # Le nom par défaut
43
44
45 i = 0                # Initialisation du compteur
46 for t in np.arange(tmin,tmax,dt): # On boucle sur le temps
47     i += 1               # Incrémentation du compteur
48     print(t)             # Un peu de feedback
49     Z1 = source(X,Y,t,-pos,0) # La première source
50     Z2 = source(X,Y,t, pos,0,phi) # et la seconde
51
52 # Ouverture de la figure et définition des sous-figures
53 plt.figure(figsize=(8,7.76))
54 ax1= plt.subplot2grid((3,3),(0,0),colspan=2, rowspan=2)
55 plt.title('Interferences a deux sources, $t={}$.format(round(t,1)))
56 plt.ylabel('$y$')
57 plt.imshow(Z1+Z2,interpolation='bilinear',extent=extent,cmap='jet',vmin=vmin,vmax=vmax)
58
59 # Pour visualiser les deux plans de coupes
60 plt.plot([-ext,ext],[ycut,ycut],'-k')
61 plt.plot([xcut,xcut],[-ext,ext],'-k')
62
63 # La figure du bas
64 ax2= plt.subplot2grid((3,3),(2,0),colspan=2,sharex=ax1)
65 plt.xlabel('$x$')
66 plt.ylabel('Intensite\nSection $y={}$.format(ycut)')
67 plt.ylim((0,vmax**2))
68 plt.plot(x,(source(x,ycut,t,-pos,0)+source(x,ycut,t,pos,0,phi))**2)
69
70 # et celle de la droite
71 ax3= plt.subplot2grid((3,3),(0,2),rowspan=2,sharey=ax1)
```

```

72     plt.xlabel('Intensite')
73     plt.xlim((0,vmax**2))
74     plt.title('Section $x={}$'.format(xcut))
75     plt.plot((source(xcut,y,t,-pos,0)+source(xcut,y,t,pos,0,phi))**2,y)
76     plt.savefig(base_name + '{:04d}.png'.format(i))
77     plt.close()

78
79 # Ne reste plus qu'à rassembler en un fichier mpeg à l'aide de convert puis de
80 # ppmtoy4m et mpeg2enc (paquet mjpegtools à installer sur la machine)

81
82 import os

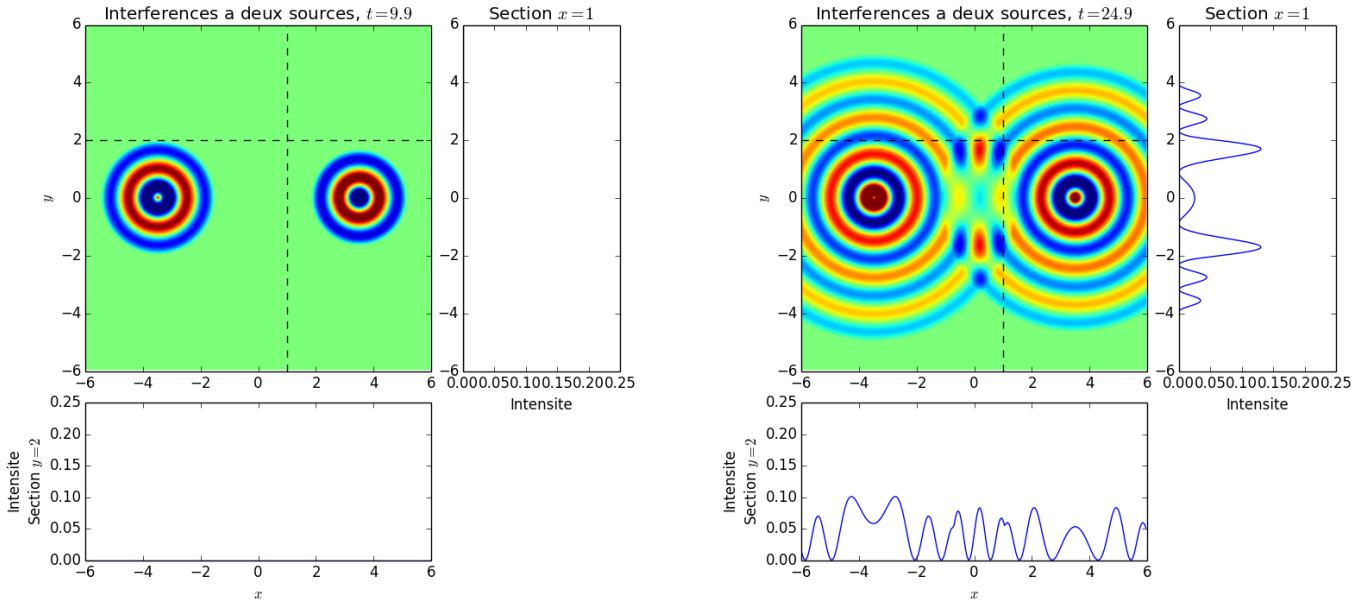
83
84 cmd = '(for f in ' + base_name + '*png ; '
85 cmd+= 'do convert -density 100x100 $f -depth 8 -resize 700x500 PNM:- ; done)'
86 cmd+= ' | ppmtoy4m -S 420mpeg2'
87 cmd+= ' | mpeg2enc -f1 -b 12000 -q7 -G 30 -o {}film.mpeg'.format(base_name)

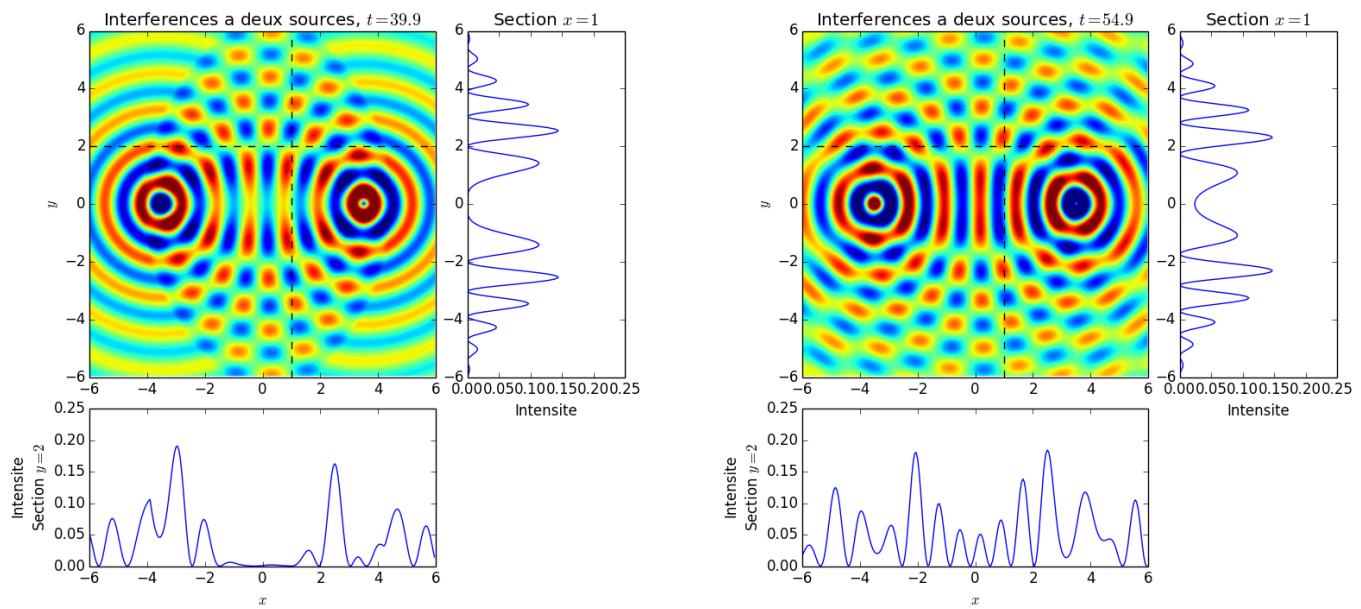
88
89 print("Execution de la commande de conversion")
90 print(cmd)
91 os.system(cmd)
92 print("Fin de la commande de conversion")

```

L'animation est disponible à l'adresse

[http://pcsi.kleber.free.fr/IPT/doc/S03\\_interferences\\_film.mpeg](http://pcsi.kleber.free.fr/IPT/doc/S03_interferences_film.mpeg)





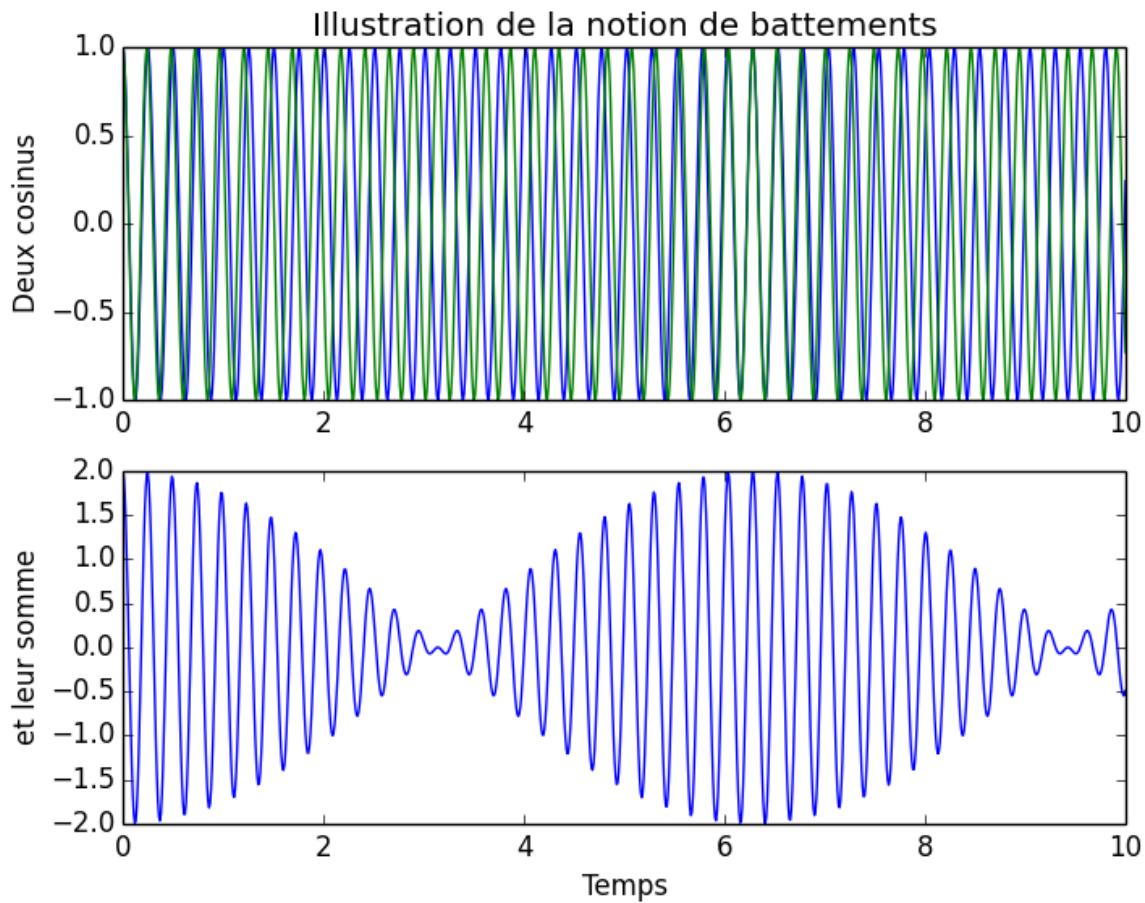
Exemple I.7

### S03 Battements

```

1   """
2   Simple illustration de la notion de battements par superposition de deux ondes
3   sinusoïdales de pulsation proches.
4   """
5
6   import numpy as np
7   import matplotlib.pyplot as plt
8
9
10  t = np.linspace(0,10,1000)      # Echantillonage en temps
11  w1 = 25                         # La pulsation du premier signal
12  w2 = w1*1.04                     # La pulsation du second signal
13  S1 = np.cos(w1*t)                # Le premier signal proprement dit
14  S2 = np.cos(w2*t)                # Le second signal proprement dit
15  S3 = S1+S2                      # La somme des deux
16
17  plt.subplot(211)                  # La figure du dessus contient
18  plt.title('Illustration de la notion de battements')    # le premier signal et
19  plt.plot(t,S1)                   # le second signal.
20  plt.plot(t,S2)
21  plt.ylabel('Deux cosinus')
22  plt.subplot(212)                  # La figure du dessous
23  plt.plot(t,S3)                   # contient leur somme
24  plt.ylabel('et leur somme')
25  plt.xlabel('Temps')
26  plt.savefig('PNG/S03_battements.png')

```



Exemple I.8

### S03 Ondes stationnaires

Exemple I.9

### S03 Diffraction

```
1   """
2   Simulation d'un phénomène de diffraction par une ouverture rectangulaire après
3   arrivée d'une onde plane inclinée d'un certain angle theta0 par rapport à la
4   normale.
5
6   Le graphe du bas représente la coupe en intensité (amplitude au carré) sur un
7   écran situé à y fixé.
8
9   """
10
11 import numpy as np          # Pour les facilités de calcul
12 import matplotlib.pyplot as plt # Pour les dessins
13
14 k,w,epsilon = 5,1,1          # Quelques constantes
15 c = w/k                      # La vitesse des ondes
16 tmin,tmax = 0,150            # L'intervalle de temps d'étude
```

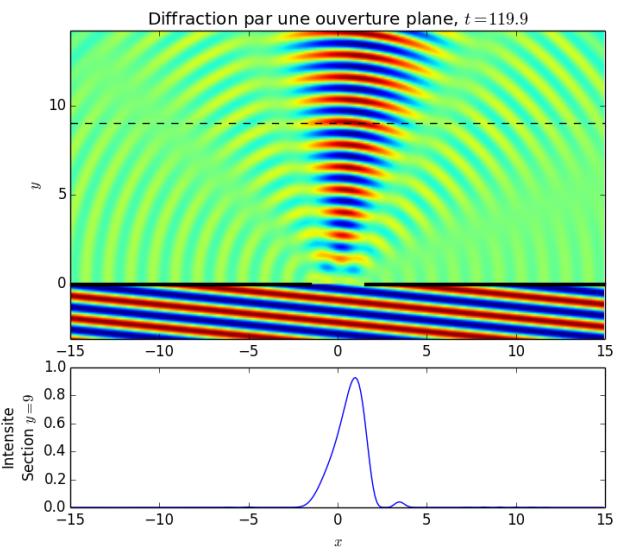
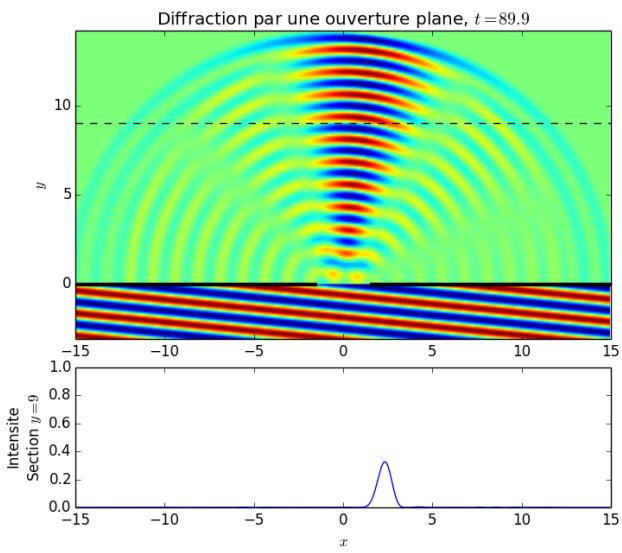
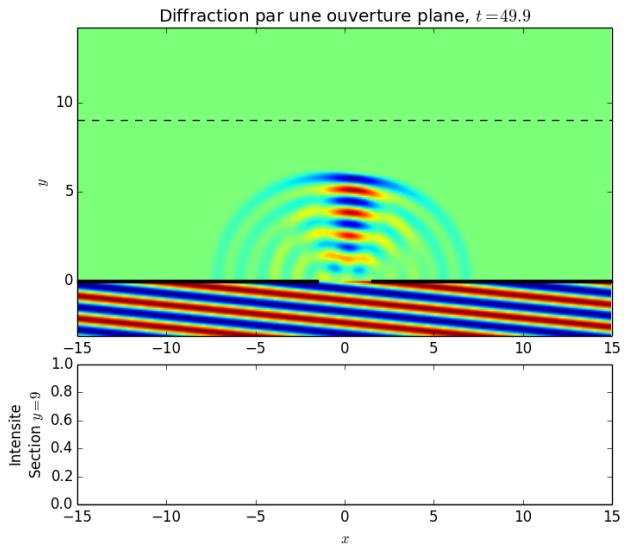
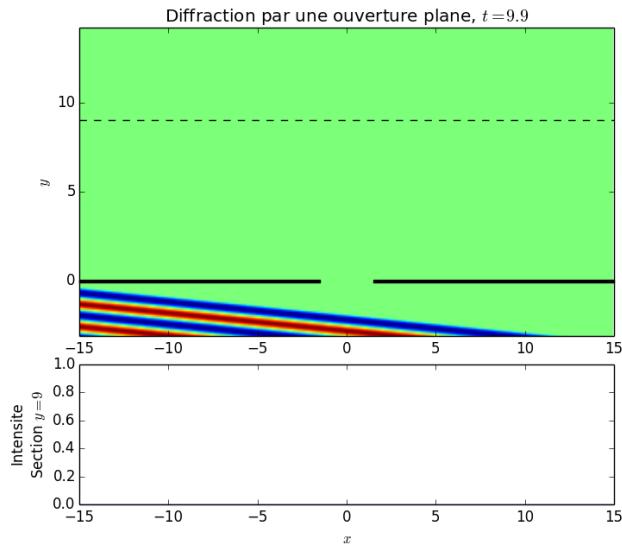
```
17 dt = 0.1                      # Le pas de temps
18 ycut = 9                        # Le plan de coupe en y
19 vmin,vmax=-1,1                 # Les valeurs extrêmes de l'amplitude
20 trou = 3                         # La taille du trou
21 theta= 0.1                      # L'angle d'incidence (en radians)
22 ext = 15.0                     # Les limites de la fenêtre d'étude
23 dx,dy = 0.1,0.1                 # Resolution
24 x = np.arange(-ext,ext,dx)      # Axe en x
25 y = np.arange(ext,-4,-dy)       # et en y (à l'envers du fait de imshow)
26 X,Y = np.meshgrid(x,y)          # pour produire la grille
27
28 # Pour définir correctement les limites de la fenêtre.
29 xmin, xmax, ymin, ymax = np.amin(x), np.amax(x), np.amin(y), np.amax(y)
30 extent = xmin, xmax, ymin, ymax
31
32 base_name = 'PNG/S03_diffraction_' # Le nom par défaut
33
34 def point_source(x,y,t,x0=0,y0=0,theta=0):
35     '''La fonction représentant une source située en (x0,y0) produite par un
36     front d'onde incliné de theta.'''
37     u0= front(x0,y0,t,theta)        # Le front au niveau de la source secondaire
38     r = np.sqrt((x-x0)**2+(y-y0)**2) # La distance à la source
39     u = u0 + k*r                   # La variable de déplacement
40                                         # (w*t est déjà dans le u0)
41     res = np.sin(u)                # Simple sinus
42     res[u > 0] = 0.0              # Le facteur n'est pas passé...
43     return res
44
45 def front(x,y,t,theta=0):
46     '''Définition de la ligne du front d'onde plane.
47     À t=0, le front d'onde passe au point (0,ymin).'''
48     return k*(np.sin(theta)*x + np.cos(theta)*(y-ymin)) - w*t
49
50
51 def onde_plane(x,y,t,theta=0):
52     '''Fonction représentative d'une onde plane faisant un angle theta avec
53     la normale. À t=0, le front d'onde passe au point (0,ymin).'''
54     u = front(x,y,t,theta)
55     res = np.sin(u)                # Simple sinus
56     res[u > 0] = 0.0              # Pour s'assurer qu'à t<0, il n'y a pas d'onde
57     return res
58
59 def superposition(x,y,t,largeur_trou,theta=0):
60     '''Fonction calculant automatiquement la superposition des ondes après
61     passage pour l'ouverture de largeur 'largeur_trou'.'
62     # On commence par mettre l'onde plane partout.
63     res = onde_plane(x,y,t,theta)
64     # Ensuite, on réfléchit et on corrige pour le valeurs de y > 0
65     x_trou = np.arange(-largeur_trou/2,largeur_trou/2,dx)
```

```
66     S = sum([point_source(x,y,t,xt,0,theta) for xt in x_trou])/len(x_trou)
67     res[y > 0] = S[y > 0]
68     print(t)      # Un tout petit peu de feedback
69     return res    # et on renvoie le résultat à afficher
70
71 i = 0                      # Initialisation du compteur
72 for t in np.arange(tmin,tmax,dt): # On boucle sur le temps
73     i += 1                    # Incrémentation du compteur
74     Z = superposition(X,Y,t,trou,theta)
75
76     # Calcul à part pour la section de coupe.
77     x_trou = np.arange(-trou/2,trou/2,dx)
78     Zcut = (sum([point_source(x,ycut,t,xt,0,theta) for xt in x_trou])/len(x_trou))**2
79
80     # Ouverture de la figure et définition des sous-figures
81     plt.figure(figsize=(8,6.9))
82     ax1= plt.subplot2grid((3,2),(0,0),colspan=2, rowspan=2)
83     plt.title('Diffraction par une ouverture plane, $t={}$'.format(round(t,1)))
84     plt.ylabel('$y$')
85     plt.xlim((xmin,xmax))
86     plt.ylim((ymin,ymax))
87     plt.imshow(Z,interpolation='bilinear',extent=extent,cmap='jet',vmin=vmin,vmax=vmax)
88
89     plt.annotate(' ',xytext=(-ext,0),xy=(-trou/2,0),
90                  arrowprops=dict(facecolor='black',width=2,frac=0,headwidth=2))
91     plt.annotate(' ',xytext=( ext,0),xy=( trou/2,0),
92                  arrowprops=dict(facecolor='black',width=2,headwidth=2,frac=0))
93
94     plt.plot([-ext,ext],[ycut,ycut],'-k')
95
96     # La figure du bas
97     ax2= plt.subplot2grid((3,2),(2,0),colspan=2,sharex=ax1)
98     plt.xlabel('$x$')
99     plt.ylabel('Intensité\nSection $y={}$'.format(ycut))
100    plt.ylim((0,vmax**2))
101    plt.plot(x,Zcut**2)
102
103    plt.savefig(base_name + '{:04d}.png'.format(i))
104    plt.close()
105
106    # Ne reste plus qu'à rassembler en un fichier mpeg à l'aide de convert puis de
107    # ppmtoy4m et mpeg2enc (paquet mjpegtools à installer sur la machine)
108
109    import os
110
111    cmd = '(for f in ' + base_name + '*png ; '
112    cmd+= 'do convert -density 100x100 $f -depth 8 -resize 700x500 PNM:- ; done)'
113    cmd+= ' | ppmtoy4m -S 420mpeg2'
114    cmd+= ' | mpeg2enc -f1 -b 12000 -q7 -G 30 -o {}film.mpeg'.format(base_name)
```

```
115  
116 print("Execution de la commande de conversion")  
117 print(cmd)  
118 os.system(cmd)  
119 print("Fin de la commande de conversion")
```

L'animation est disponible à l'adresse

[http://pcsi.kleber.free.fr/IPT/doc/S03\\_diffraction\\_film.mpeg](http://pcsi.kleber.free.fr/IPT/doc/S03_diffraction_film.mpeg)



Exemple I.10

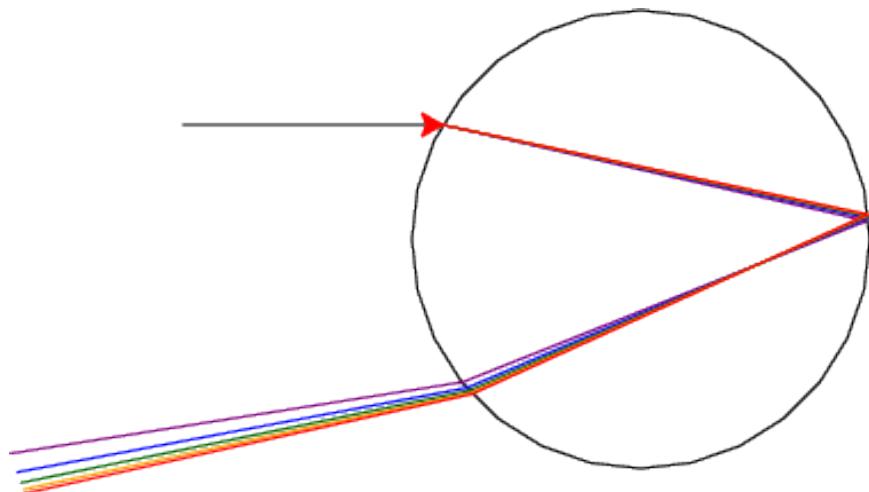
## S04 Lois de Descartes

Exemple I.11

## S04 Réfraction dans une goutte: explication de l'arc-en-ciel

```
1   """
2   Programme conçu par Tom Morel (PCSI, lycée Jean Jaurès) pour visualiser
3   l'influence de la couleur du rayon incident lors de sa réfraction à
4   l'intérieur d'une goutte d'eau. Pour alléger les calculs, on approxime un peu
5   les expressions des cordes dans la goutte, mais cela reste parfaitement
6   correct au tracé.
7   """
8
9
10  from turtle import *    # Pour le dessin à l'écran
11  from math import *      # Pour les fonctions mathématiques
12
13 def Cauchy(x):
14     '''Fonction donnant l'indice du milieu en suivant la loi de Cauchy en
15     fonction de la longueur d'onde x donnée en mètres.'''
16     A=1.58
17     B=171*1e-16
18     n=A+(B/x**2)
19     return n
20
21 # On commence par les déclarations
22 R =100          # Rayon du cercle
23 yi=R/2          # Coordonnée y du premier rayon
24 xi=-2*R        # Coordonnée x du premier rayon
25 longueur=[400,500,600,700,800]           # Différentes longueurs d'onde
26 pal=['purple','blue','dark green','orange','red'] # et les couleurs associées
27 i= asin(yi/R)    # angle d'incidence du premier rayon (en radian)
28
29 # On démarre le dessin
30 up()            # On lève le crayon
31 goto(0,-R)       # On va au point de coordonnée (0,-R)
32 down()          # On pose le crayon
33 circle(R,360)    # On dessine le cercle
34 up()            # On relève le crayon
35 goto(xi,yi)      # On va au point de départ du rayon lumineux
36 down()          # On repose le crayon
37 goto(-sqrt(R*R-yi*yi),yi) # Et on va jusqu'à toucher la goutte d'eau
38
39 def imprime_ecran(n):
40     """ Récupère ce qui est affiché à l'écran. """
41     base_name = 'PNG/S04_arc_en_ciel_turtle'
42     getscreen().getcanvas().postscript(file=base_name + "{:02d}.eps".format(n))
43
44
45 delay(40)        # On ralentit un peu Speedy Gonzales...
46 # À présent, on va boucler sur les couleurs que l'on veut représenter
47 for j in range(len(longueur)):
48     color(pal[j])          # On change la couleur du trait
49     n= Cauchy(longueur[j]*1e-9) # Valeur de l'indice optique en fonction de lambda
```

```
50      r=asin(sin(i)/n)          # Angle de réfraction (en radian)
51      right((i-r)*180/pi)       # Tourner à droite d'un angle (i-r) en degré
52      forward(190)             # On avance (à peu près) de la distance adéquate
53      right(180-(2*r*180/pi))   # On tourne de pi-2*r à droite
54      forward(190)             # On réavance (à peu près) de la distance adéquate
55      right((i-r)*180/pi)       # Même configuration qu'à l'aller
56      forward(200)             # On sort de la goutte
57      up()                     # On relève le crayon
58      goto(-sqrt(R*R-yi*yi),yi) # Et on retourne au point de départ
59      # Ne reste qu'à tourner à l'envers pour se remettre dans l'axe
60      left(2*(i-r)*180/pi+180-(2*r*180/pi))
61      down()                   # et on repose le crayon
62      imprime_ecran(j)         # On prends une petite photo pour la route
63
64  # L'important est bien sûr de montrer le dessin se construire en direct, mais
65  # si on veut en conserver une trace, on peut utiliser ce hack:
66
67  base_name = 'PNG/S04_arc_en_ciel_turtle'
68
69  ts = getscreen()
70  ts.getcanvas().postscript(file=base_name + ".eps")
71
72  import os # Pour pouvoir appeler BBcut et convert
73  os.system('~/BBcut {}.eps'.format(base_name))           # Pour tailler la bounding box
74  os.system('convert {}.eps {}.png'.format(base_name)) # Pour convertir en png
```



Exemple I.12

## S05 Lentilles minces: construction graphique

Exemple I.13

## S05 Distortion chromatique au passage d'une lentille

1     '''

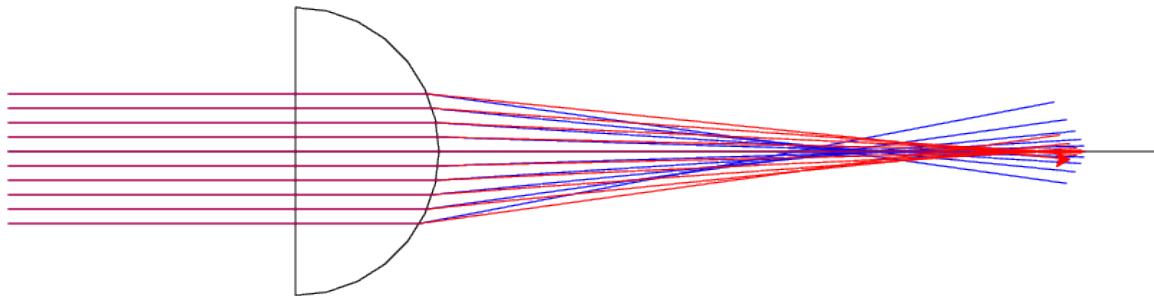
2     Programme conçu par Tom Morel (PCSI, lycée Jean Jaurès) pour visualiser

```
3 l'influence de la couleur du rayon incident sur la distance focale d'une
4 lentille sphérique
5 ''
6
7 from math import *      # Pour les fonctions mathématiques
8 from turtle import *     # Pour les dessins à l'écran (appelé après math pour radians())
9
10
11 def Cauchy(x):
12     '''Fonction donnant l'indice du milieu en suivant la loi de Cauchy en
13     fonction de la longueur d'onde x donnée en mètres.'''
14     A=1.2
15     B=171*1e-16
16     n=A+(B/x**2)
17     return n
18
19 N= 10                      # nombre de rayons à dessiner
20 xi= - 200                     # coordonnée xi de départ
21 yi= -50                       # coordonnée yi de départ
22 longueur=[400,800]            # différentes longueurs d'onde
23 pal=['blue','red']             # et les couleurs associées
24 R = 100                        # Le rayon de courbure de la lentille
25
26 up()                          # On soulève le crayon
27 goto(0,R)                     # On va en haut à gauche de la lentille
28 right(90)                    # On tourne pour commencer à descendre
29 down()                         # On pose le crayon
30 forward(2*R)                  # On dessine le côté plat de la lentille
31 left(90)                      # On se remet dans l'axe
32 circle(R,180)                 # On trace le cercle sur 180 degrés
33 up()                           # On soulève
34 goto(-200,0)                  # Pour aller tracer l'axe optique
35 down()                         # On pose le stylo
36 right(180)                    # Retour dans l'axe
37 forward(800)                  # et tracé effectif
38
39 radians()                     # On passe les angles en radians.
40
41 for j in range(len(longueur)): # On boucle sur les couleurs
42     color(pal[j])              # Sélection de la couleur
43     n=Cauchy(longueur[j]*1e-9) # On récupère l'indice optique
44     yi=-R/2                    # Ordonnée initiale
45     for i in range(N):         # On boucle sur les rayons à dessiner
46         up()                   # On lève le crayon
47         goto(-200,yi)           # On se place
48         down()                 # et c'est parti !
49         goto(sqrt(100**2-yi**2),yi) # On traverse jusqu'à la face sphérique
50         alpha=asin(yi/100)       # Angle d'incidence sur la face de sortie
51         theta=asin(n*yi/100)     # Angle après réfraction dans l'air
```

```

52     right(theta-alpha)      # On tourne de l'angle de déviation
53     forward(450)           # On complète le tracé
54     left(theta-alpha)      # et on se remet dans l'axe
55     yi=yi+(R/N)           # Définition de la prochaine ordonnée
56
57 # L'important est bien sûr de montrer le dessin se construire en direct, mais
58 # si on veut en conserver une trace, on peut utiliser ce hack:
59
60 base_name = 'PNG/S05_distortion_chromatique'
61
62 ts = getscreen()
63 ts.getcanvas().postscript(file=base_name + ".eps")
64
65 import os # Pour pouvoir appeler BBcut et convert
66 os.system('./BBcut {}.eps'.format(base_name))          # Pour tailler la bounding box
67 os.system('convert {}.eps {}.png'.format(base_name)) # Pour convertir en png

```



Exemple I.14

## S05 Illustration de la règle des 4P: Plus Près, Plus Plat

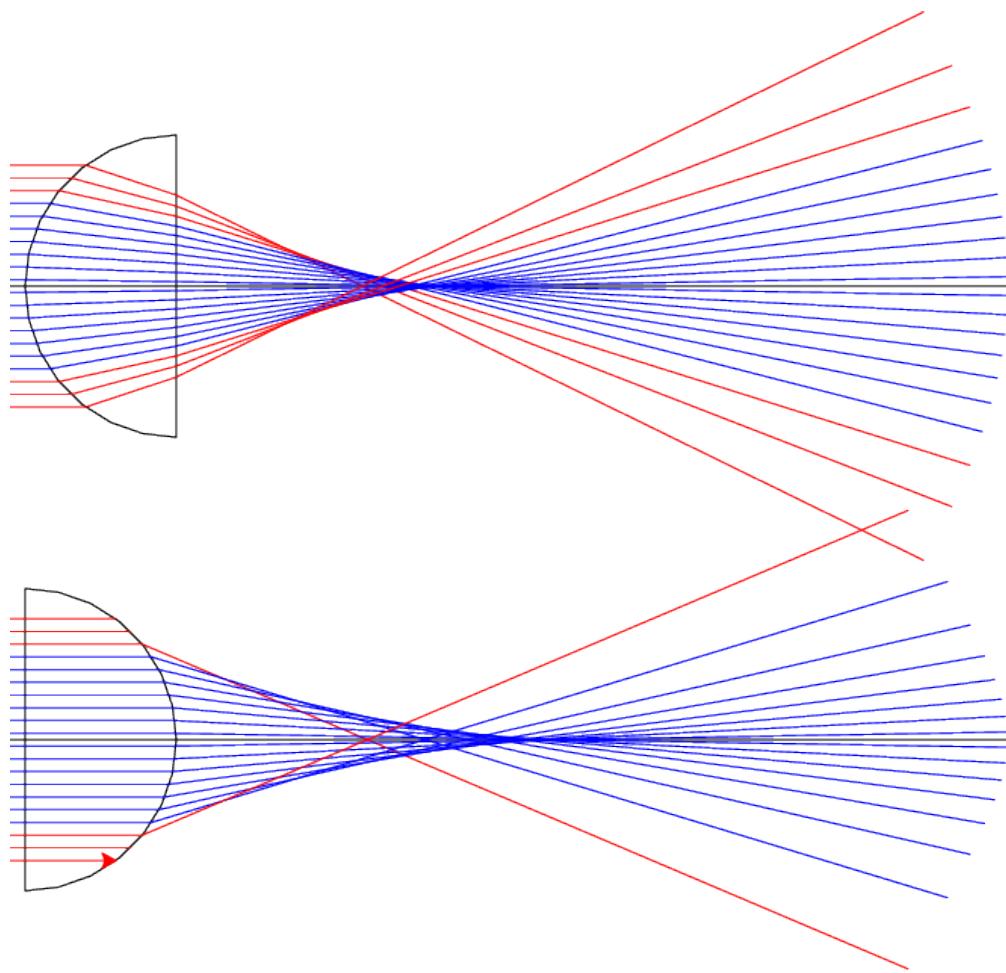
```

"""
1 Programme conçu par Tom Morel (PCSI, lycée Jean Jaurès) pour visualiser
2 l'influence de la position du côté plat pour la qualité d'une image. On se
3 rend compte que lorsque le côté plat est du côté où l'objet/image est au plus
4 près (règle des 4P), les aberrations géométriques sont moins prononcées.
5 Effets de bords intéressants:
6 * la lentille étant non-symétrique, le centre optique n'est pas au milieu
7 (d'où le décalage observé pour le foyer dans les deux positions).
8 * dans le 2e cas, certains rayons ne peuvent pas ressortir de la lentille du
9 fait du phénomène de réflexion totale.
10 """
11
12
13 from math import *    # Pour les calculs
14 from turtle import *   # Pour la tortue du LOGO
15
16 n = 1.4                # Indice choisi pour le verre
17 N = 20                  # Nombre de rayons à dessiner
18 xi= - 400               # Coordonnée xi de départ des rayons
19 yi= 80                  # Coordonnée relative yi de départ du premier rayon
20

```

```
21 R = 100          # Rayon de courbure de la lentille
22 decal = 1.5*R    # Moitié du décalage vertical entre les deux images
23 ylim = 60        # À partir de quand colorie-t-on les rayons en rouge
24
25 # Dessin de la première lentille
26 up()             # On lève le crayon
27 goto(xi+2*R,-R+decal) # Position de départ en bas à droite
28 left(90)         # On va vers le haut
29 down()           # On pose le crayon
30 forward(2*R)     # Tracé de la partie plane
31 left(90)         # On se positionne vers la gauche
32 circle(R,180)    # et on fait le demi-cercle
33 up()             # On lève le crayon
34 goto(xi,decal)   # Préparation de l'axe optique
35 down()           # On pose le crayon
36 forward(800)     # et on le trace
37
38 # Dessin de la seconde lentille
39 up()             # On lève le crayon
40 goto(xi+R,-R-decal) # Position de départ en bas à gauche
41 left(90)         # On va vers le haut
42 down()           # On pose le crayon
43 forward(2*R)     # Tracé de la partie plane
44 right(90)        # On se positionne vers la droite
45 circle(-R,180)   # et on fait le demi-cercle
46 up()             # On lève le crayon
47 goto(xi,-decal)  # Préparation de l'axe optique
48 right(180)       # Demi-tour droite !
49 down()           # On pose le crayon
50 forward(800)     # et on le trace
51
52 radians()         # À partir d'ici, on passe en radians pour les angles
53
54 for i in range(N): # Boucle sur les rayons à tracer
55     up()             # On lève le crayon
56     goto(xi,yi+decal) # pour se mettre au point de départ
57     down()           # puis on le repose pour commencer le tracé
58     alpha=asin(yi/R) # Angle par rapport à la normale 1ère interface
59     beta=asin(yi/(n*R)) # Angle de réfraction 1ère interface air/verre
60     xa=-R*cos(alpha)-2*R # Là où on va toucher la 1ère interface
61     if abs(yi)>=ylim: # On met les rayons extrêmes
62         color('red')   # en rouge
63     else:              # et les autres
64         color('blue')  # en bleu
65     goto(xa,yi+decal) # Allons jusqu'au contact avec la lentille
66     # Un petit calcul pour la position du contact avec la 2e interface
67     yb=yi+(xa+2*R)*tan(alpha-beta)
68     goto(-2*R,yb+decal) # On y va !
69     # Et on calcule l'angle de réfraction en sortie de cette 2e interface
```

```
70     gamma=asin(n*sin(alpha-beta))
71     right(gamma)      # La tortue étant toujours horizontale, on tourne de cet angle
72     forward(550)       # On avance tout droit
73     left(gamma)        # et on se remet à l'horizontale pour le tracé suivant
74
75     up()               # On passe à présent au second schéma avec un levé de crayon
76     goto(xi,yi-decal) # pour se mettre au point de départ
77     down()             # et on repose le crayon.
78     alpha=asin(yi/R)   # Angle par rapport à la normale 2ère interface
79     xa= R*cos(alpha)-3*R # Contact avec la 2ère interface (la partie plane est sans effet)
80     if abs(yi)>=ylim: # On met les rayons extrêmes
81         color('red')    # en rouge
82     else:               # et les autres
83         color('blue')   # en bleu
84     goto(xa,yi-decal) # On va jusqu'au contact
85     try:                # Attention, il est possible qu'il y ait réflexion totale
86         beta=asin(n*yi/R) # si ce calcul échoue...
87         right(-alpha+beta) # Si tout va bien, on tourne
88         forward(550)       # on avance
89         left(-alpha+beta) # et on se remet dans l'axe
90     except: pass         # Cas de la réflexion totale: on ne fait rien de plus
91
92     yi=yi-(160/(N-1))   # Passage au rayon suivant.
93
94 # L'important est bien sûr de montrer le dessin se construire en direct, mais
95 # si on veut en conserver une trace, on peut utiliser ce hack:
96
97 base_name = 'PNG/S05_gauss_4P'
98
99 ts = getscreen()
100 ts.getcanvas().postscript(file=base_name + ".eps")
101
102 import os # Pour pouvoir appeler BBcut et convert
103 os.system('./BBcut {}.eps'.format(base_name))           # Pour tailler la bounding box
104 os.system('convert {}{}.eps {}{}.png'.format(base_name)) # Pour convertir en png
```

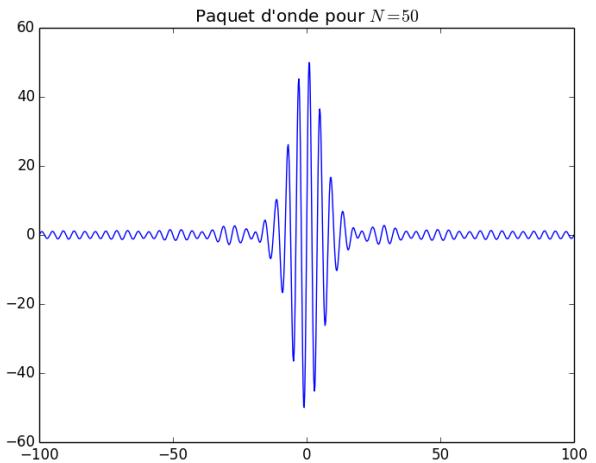
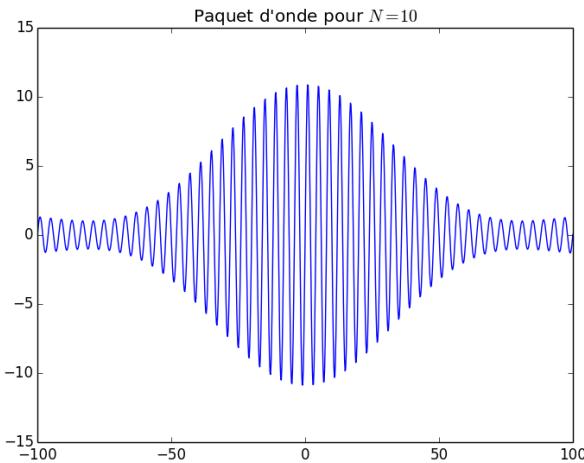


Exemple I.15

## S06 Construction d'un paquet d'onde pour la Mécanique Quantique

```
1  """ Implémentation proposée par Miriam Heckmann, PCSI3, Lycée Kléber """
2
3  import numpy as np          # Pour la fonction linspace, zeros et la trigonométrie
4  import matplotlib.pyplot as plt # Boîte à outils graphiques
5
6  def paquet_d_onde(N):
7      """Construction d'un paquet d'onde à N ondes"""
8      nb_points=1000           # Le nombre de points d'échantillonage
9      x=np.linspace(-100,100,nb_points) # Échantillonnage en position
10     y=np.zeros(nb_points)        # Création d'une liste de zéros
11
12    for i in range(N):
13        y += (1-i/N)*np.sin(np.pi*x/(2+i/100))+(1-i/N)*np.sin(np.pi*x/(2-i/100))
14    plt.plot(x,y)
15    plt.title("Paquet d'onde pour $N={}$".format(N))
16    plt.savefig('PNG/S06_paquet_d_ondes_MQ_N{:03d}.png'.format(N))
17    plt.clf()
18
19 paquet_d_onde(10)
```

20 paquet\_d onde(50)



Exemple I.16

## S07 Résolution électrique par pivot de Gauss

Exemple I.17

## S08 Réseau électrique du premier ordre simple

Exemple I.18

## S08 Réseau électrique complexe

Exemple I.19

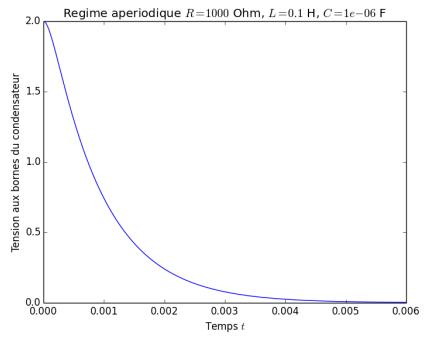
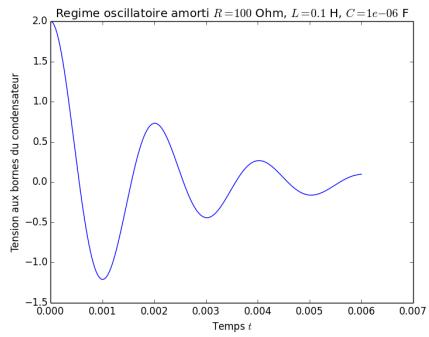
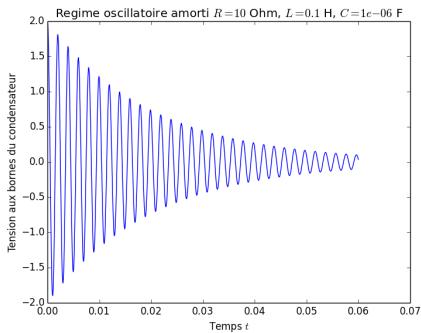
## S09 Oscillateur amorti en régime libre

```
1 """ Implémentation proposée par Miriam Heckmann, PCSI3, Lycée Kléber. """
2
3 import numpy as np          # Pour la fonction linspace, zeros et la trigonométrie
4 import scipy as sp           # Simple alias usuel
5 import scipy.integrate       # Pour l'intégration
6 import matplotlib.pyplot as plt # Boîte à outils graphiques
7
8 def RLC(R=1e2,L=0.1,C=1e-6,u0=2,du0=0,fichier=None,intervalle=None):
9     """
10     Réponse de la tension aux bornes du condensateur pour un circuit RLC
11     série. En cas d'absence du nom de fichier, le script fait un affichage
12     interactif du graphique à l'aide de plt.show(). De même, si l'intervalle
13     de temps n'est pas spécifié (sous forme d'un doublet), l'affichage se fait
14     de 0 à 3tau.
15     """
16     omega0=1/np.sqrt(L*C)      # Pulsation propre
17     Q=(np.sqrt(L/C))/R         # Facteur de qualité
18     tau=2*Q/omega0             # Temps de relaxation
```

```

20     if Q > 0.5: tit='Régime oscillatoire amorti'
21     if Q == 0.5: tit='Régime critique'
22     if Q < 0.5: tit='Régime aperiodique'
23
24     tit += ' $R={} \$ Ohm, $L={} \$ H, $C={} \$ F'.format(R,L,C)
25
26     def equadiff(y,t):
27         u,vu = y                                # y contient tension et dérivée de tension
28         return [vu , - omega0**2 * u - 2*vu/tau]  # On renvoie un doublet pour [du/dt, dv/dt]
29
30     nb_points=1000                            # Le nombre de points d'un graphe
31     if intervalle:
32         a,b = intervalle
33         t = np.linspace(a,b,nb_points)
34     else:
35         t=np.linspace(0,3*tau,nb_points)        # La fonction linspace crée une liste
36
37     plt.clf()
38
39     sol = sp.integrate.odeint(equadiff,[u0,du0],t) # Intégration proprement dite
40     u = sol[:,0]                                 # Récupération de la position
41     plt.plot(t,u)                               # et affichage
42
43     plt.title(tit)                             # annotations
44     plt.ylabel('Tension aux bornes du condensateur')
45     plt.xlabel('Temps $t$')
46
47     if fichier:
48         plt.savefig(fichier)                  # Sauvegarde dans un fichier
49     else:
50         plt.show()                           # Affichage à l'écran
51
52 RLC(R=10,fichier='PNG/S09_oscillateur_amorti_libre_R00010.png')
53 RLC(R=100,fichier='PNG/S09_oscillateur_amorti_libre_R00100.png')
54 RLC(R=1000,fichier='PNG/S09_oscillateur_amorti_libre_R01000.png',intervalle=(0,0.006))

```



Exemple I.20

## S10 Oscillateur amorti en régime forcé

Exemple I.21

## S11 Module pour dessiner un diagramme de Bode

Voici un petit module qui sert pour dessiner les diagrammes de Bode des sections suivantes.

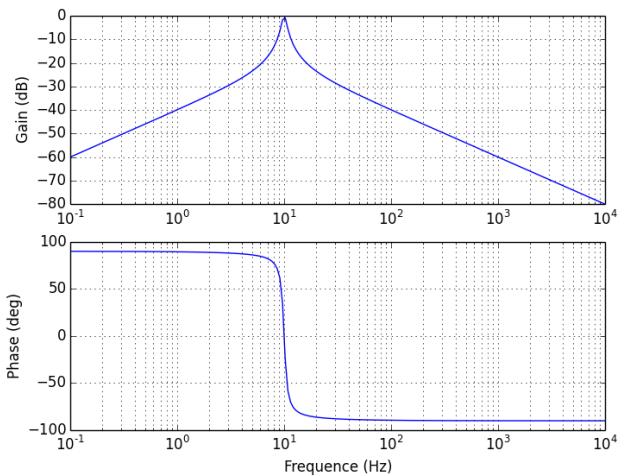
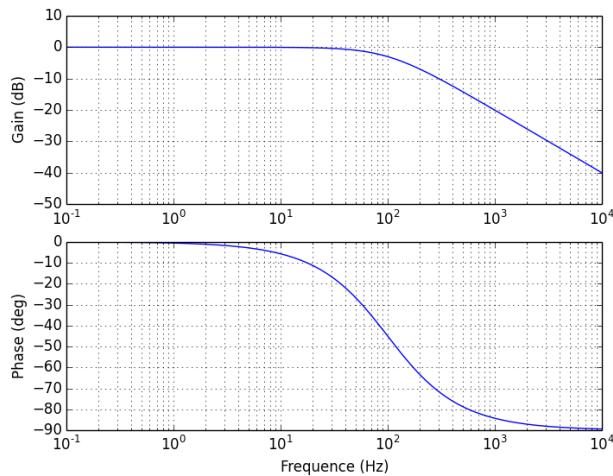
```
1   """
2   Petit module pour rassembler la procédure de tracé de diagramme de Bode pour
3   pouvoir se concentrer uniquement sur ce qu'il y a autour.
4   """
5
6   import matplotlib.pyplot as plt # Pour les dessins
7
8   def diag_bode(f,GdB,phase,out_file,titre=None):
9       '''Dessine un diagramme de bode quand on donne la fréquence, le gain et la
10      phase correspondants. Le résultat est écrit dans le fichier 'out_file'.'''
11      plt.figure()                      # Ouverture de la figure
12      plt.subplot(211)                   # La première sous-figure
13      if titre: plt.title(titre)        # Rajout du titre si demandé
14      plt.semilogx(f, GdB)             # Graphique semi-log en x pour le gain
15      plt.grid(which='both')           # On rajoute la grille
16      plt.ylabel(r'Gain (dB)')         # Label vertical
17      plt.subplot(212)                  # La seconde sous-figure
18      plt.semilogx(f, phase)          # Graphique semi-log en x pour la phase
19      plt.ylabel(r'Phase (deg)')       # Label en y
20      plt.xlabel(r'Fréquence (Hz)')    # et label en x commun
21      plt.grid(which='both')           # On rajoute la grille
22      plt.savefig(out_file)           # Sauvegarde du fichier
23      plt.close()                     # et fermeture de la figure
24
25  # Le module signal possède une fonction "bode" dédiée que l'on va utiliser
26  from scipy import signal
27
28  def second_ordre(f0,Q,filename='defaut.png',type='PBs',f=None):
29      '''Petite fonction pour faciliter l'utilisation de la fonction "bode" du
30      module "signal" quand on s'intéresse à des filtres du 2e ordre. Il suffit
31      de donner la fréquence propre f0 et le facteur de qualité Q pour obtenir
32      ce que l'on veut. Autres paramètres:
33      * filename: le nom du fichier ('defaut.png' par défaut)
34      * type: le type du filtre, à choisir parmi 'PBs' (passe-bas),
35      'PBd' (passe-bande) et 'PHt' (passe-haut). On peut aussi définir soi-même
36      le numérateur sous forme d'une liste de plusieurs éléments, le degré le
37      plus haut donné en premier. NB: le '1.01' des définitions est juste là
38      pour améliorer sans effort le rendu graphique.
39      * f: les fréquences à échantillonner (si None, la fonction choisit
40      d'elle-même un intervalle adéquat).
41      '''
```

```
42     den = [1./f0**2,1./(Q*f0),1]          # Le dénominateur de la fonction de transfert
43     if type == 'PBs': num = [1.01]           # Le numérateur pour un passe-bas
44     elif type == 'PBd': num = [1.01/(Q*f0),0] # pour un passe-bande
45     elif type == 'PHt': num = [1.01/f0**2,0,0]# pour un passe-haut
46     else: num = type                      # sinon, c'est l'utilisateur qui le définit
47     s1 = signal.lti(num,den)                # Définition de la fonction de transfert
48     f, GdB, phase = signal.bode(s1,f)      # Obtention des valeurs adéquates
49     diag_bode(f,GdB,phase,filename)        # Dessin du diagramme proprement dit
```

Exemple I.22

## S11 Filtre intégrateur

```
1   """
2   Pour un exercice de reconnaissance d'un filtre qui puisse jouer le rôle
3   d'intégrateur dans un certain intervalle de fréquences. On en génère un du
4   premier ordre (passe-bas) et un du second ordre (passe-bande).
5   """
6
7   from scipy import signal          # Pour les fonctions 'lti' et 'bode'
8   import numpy as np               # Pour np.logspace
9
10  # Ceci est un filtre passe-bas donc potentiellement intégrateur à HF
11
12  num = [1]                         # Polynôme au numérateur (-> 1)
13  den = [0.01,0.999]                 # Polynôme au dénominateur (-> 0.01*jw + 0.999)
14
15  f = np.logspace(-1,4,num=200)      # L'intervalle de fréquences considéré (échelle log)
16  s1 = signal.lti(num,den)           # La fonction de transfert
17  f,GdB,phase = signal.bode(s1,f)  # Fabrication automatique des données
18
19  from bode import diag_bode       # Pour générer le diagramme de Bode
20
21  # Appel effectif à la fonction dédiée.
22  diag_bode(f,GdB,phase,'PNG/S11_integrateur.png')
23
24  # Ceci est un filtre passe-bande du second ordre (intégrateur à HF)
25
26  num2 = [0.01,0]                   # Numérateur (-> 0.01*jw)
27  den2 = [10**-2,0.01,1]            # Dénominateur (-> 0.01*(jw)**2 + 0.01*jw + 1)
28
29  f = np.logspace(-1,4,num=200)      # Intervalle de fréquences en échelle log
30  s2 = signal.lti(num2,den2)         # Fonction de transfert
31  f,GdB,phase = signal.bode(s2,f)  # Fabrication des données
32  diag_bode(f,GdB,phase,'PNG/S11_integrateur2.png') # et du diagramme
```



Exemple I.23

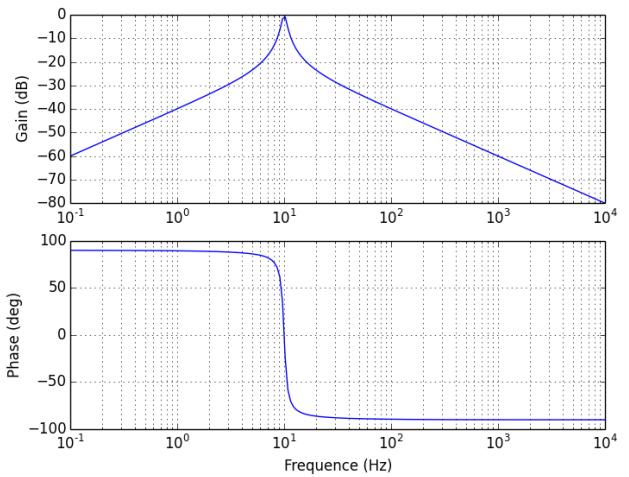
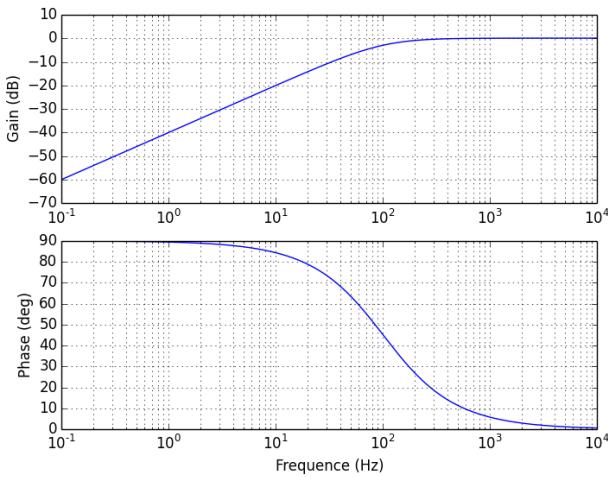
## S11 Filtre Déivateur

```

1   """
2   Pour un exercice de reconnaissance d'un filtre qui puisse jouer le rôle
3   de déivateur dans un certain intervalle de fréquences. On en génère un du
4   premier ordre (passe-bas) et un du second ordre (passe-bande).
5   """
6
7   from scipy import signal           # Pour les fonctions 'lti' et 'bode'
8   import numpy as np                 # Pour np.logspace
9
10  # Ceci est un filtre passe-haut donc potentiellement déivateur à BF
11
12 num = [0.01,0]                     # Polynôme au numérateur (-> 0.01*jw )
13 den = [0.0099,1]                   # Polynôme au dénominateur (-> 0.0099*jw + 1)
14
15 f = np.logspace(-1,4,num=200)      # L'intervalle de fréquences considéré (échelle log)
16 s1 = signal.lti(num,den)           # La fonction de transfert
17 f,GdB,phase = signal.bode(s1,f)   # Fabrication automatique des données
18
19 from bode import diag_bode       # Pour générer le diagramme de Bode
20
21 # Appel effectif à la fonction dédiée.
22 diag_bode(f,GdB,phase,'PNG/S11_derivateur.png')
23
24 # Ceci est un filtre passe-bande du second ordre (déivateur à BF)
25
26 num2 = [0.01,0]                   # Numérateur (-> 0.01*jw )
27 den2 = [10**-2,0.01,1]             # Dénominateur (-> 0.01*(jw)**2 + 0.01*jw + 1)
28
29 f = np.logspace(-1,4,num=200)      # Intervalle de fréquences en échelle log
30 s2 = signal.lti(num2,den2)         # Fonction de transfert
31 f,GdB,phase = signal.bode(s2,f)   # Fabrication des données

```

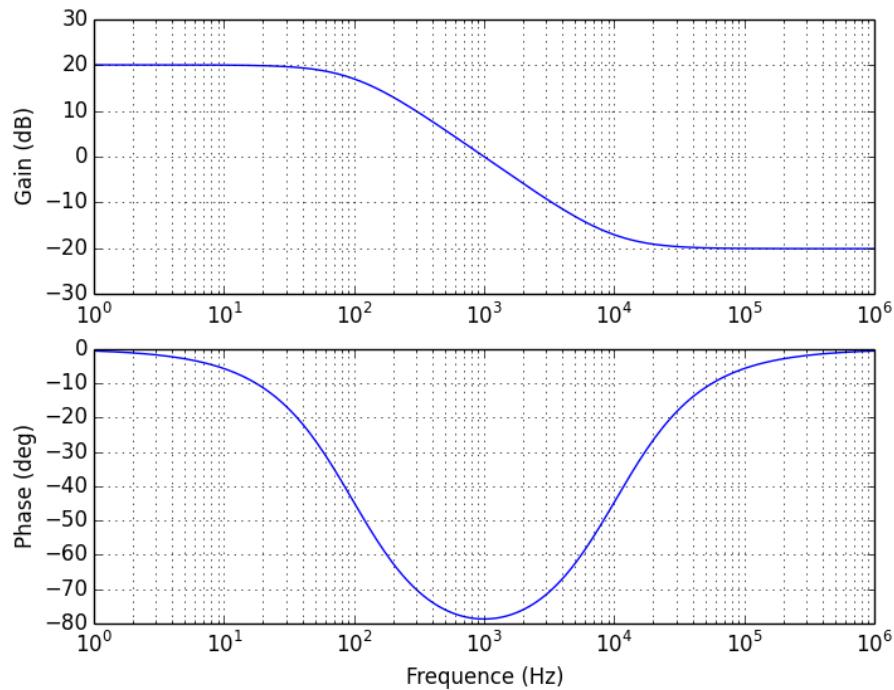
```
32 diag_bode(f,GdB,phase,'PNG/S11_derivateur2.png') # et du diagramme
```



— Exemple I.24 —

## S11 Filtre bizarre

```
1   ''
2   Un filtre "bizarre" dont il faut trouver la fonction de transfert sachant
3   qu'il est du premier ordre.
4   ''
5
6   from scipy import signal # Pour lti et bode
7   import numpy as np        # Pour l'échantillonnage
8
9   num = [0.001,10.1]        # Numérateur (-> 0.001*jw + 10.1)
10  den = [0.0101,1]          # Dénominateur (-> 0.0101*jw + 1 )
11
12
13  # Extraction des données
14  f = np.logspace(0,6,num=200)
15  s1 = signal.lti(num,den)
16  f, GdB, phase = signal.bode(s1,f)
17
18  # Et préparation du diagramme
19  from bode import diag_bode
20  diag_bode(f,GdB,phase,'PNG/S11_filtre_bizarre.png')
```



Exemple I.25

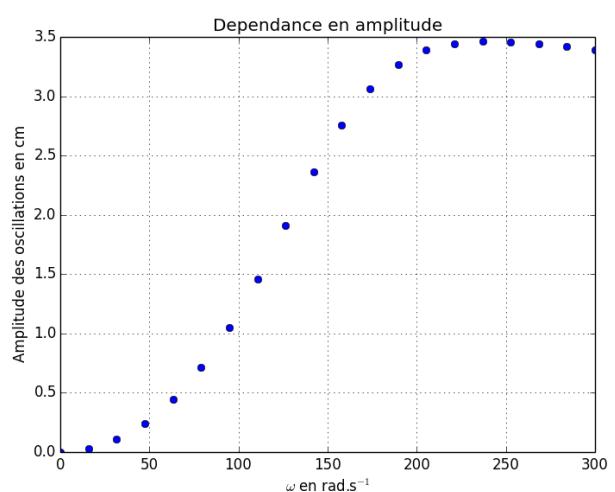
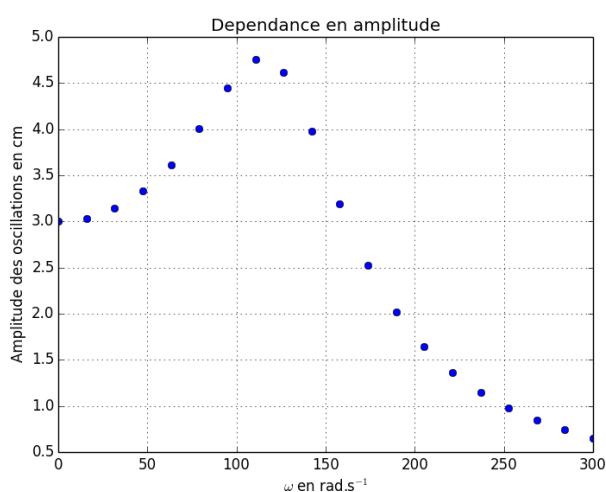
## S11 Diagrammes en amplitude et échelle linéaire

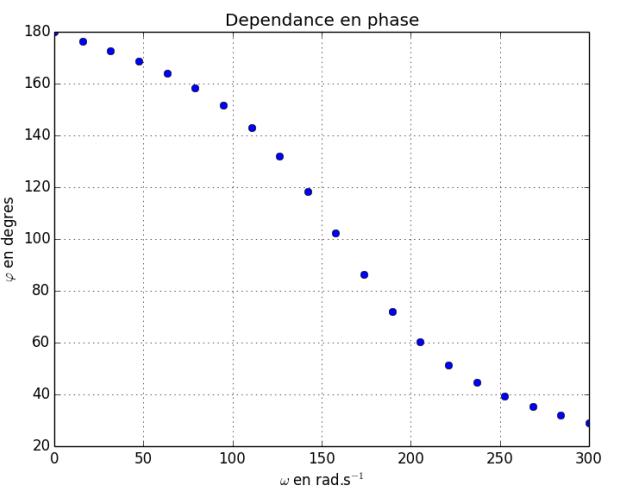
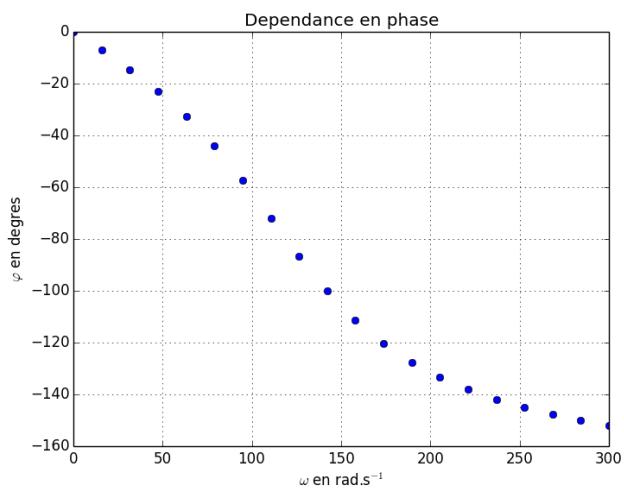
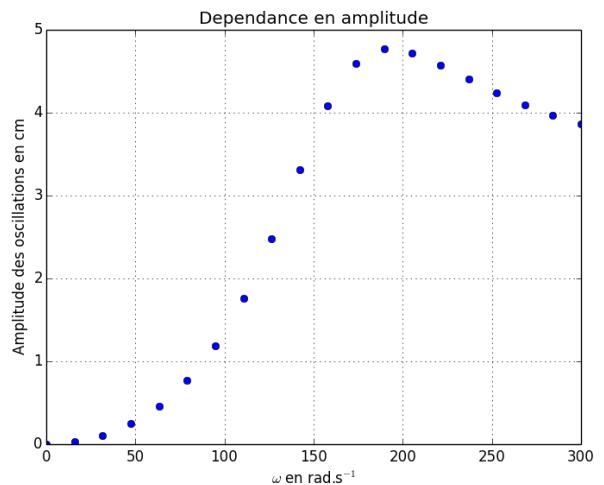
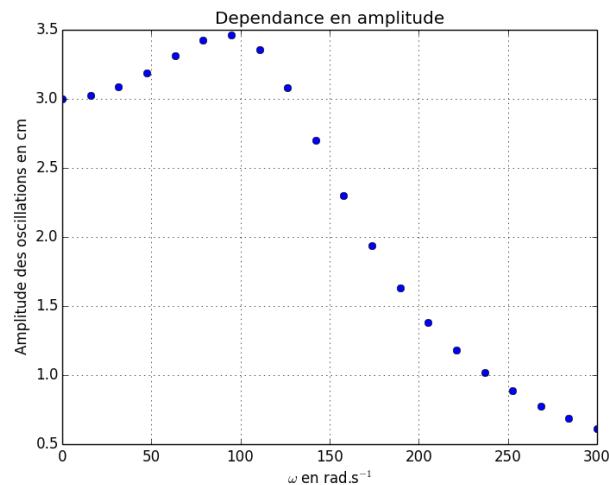
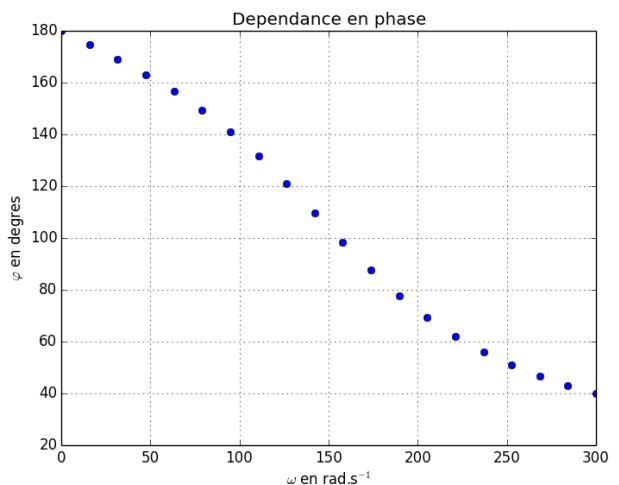
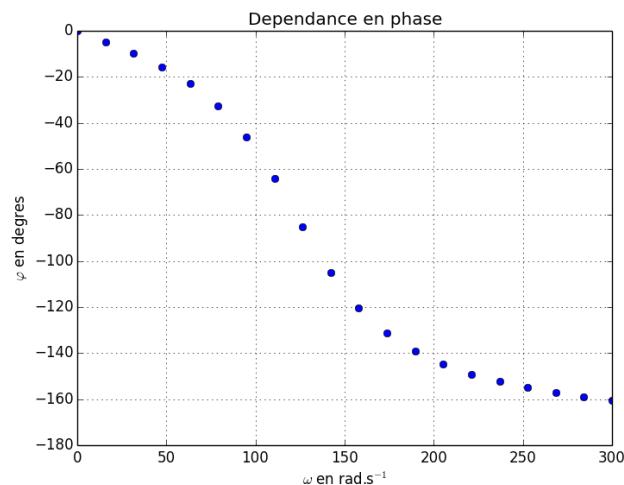
```
1   ''''  
2   À nouveau pour une question de cours: des diagrammes en amplitudes et échelle  
3   linéaire en pulsation. Le but pour les élèves est de savoir retrouver  $Q$  et  $w_0$   
4   à partir de ces données (comme en TP).  
5   ''''  
6  
7   import numpy as np                      # Pour l'échantillonage  
8   import matplotlib.pyplot as plt          # Pour les dessins  
9   import math                             # Fonctions mathématiques  
10  import cmath                            # et calculs en complexes  
11  
12  A = 3                                 # Amplitude globale des mesures  
13  wmin = 0.1                            # Pulsation minimale échantillonnée  
14  wmax = 300                            # Pulsation maximale  
15  nb_points = 20                         # nombre de points de "mesure"  
16  w = np.linspace(wmin,wmax,nb_points)    # Échantillonage effectif  
17  
18  def H(w,w0,Q):  
19      '''Fonction de transfert pour un passe bas du second ordre'''  
20      return A/(1-(w/w0)**2 + 1j*w/(Q*w0))  
21  
22  
23  def Z0(w,w0,Q):  
24      '''Amplitude de la fonction de transfert.'''  
25      return abs(H(w,w0,Q))
```

```

26
27 def phi(w,w0,Q):
28     '''Phase de la fonction de transfert (en degrés).'''
29     return cmath.phase(H(w,w0,Q))*180/math.pi
30
31 def amplitude_et_phase(w0,Q,base_fichier):
32     '''Le dessin proprement dit'''
33     plt.plot(w,[phi(wi,w0,Q) for wi in w], 'o') # Échantillonage de la phase
34     plt.xlabel(r'$\omega$ en rad.s$^{-1}$') # Légende en abscisse
35     plt.ylabel(r'$\varphi$ en degrés') # Légende en ordonnée
36     plt.title('Dependance en phase') # Titre
37     plt.grid(which='both') # La grille
38     plt.savefig(base_fichier + '_phase.png') # Sauvegarde
39     plt.clf() # Nettoyage
40     plt.plot(w,[Z0(wi,w0,Q) for wi in w], 'o') # On repart sur l'amplitude
41     plt.xlabel(r'$\omega$ en rad.s$^{-1}$') # Légende en abscisse
42     plt.ylabel(r'Amplitude des oscillations en cm') # Légende en ordonnée
43     plt.title('Dependance en amplitude') # Titre
44     plt.grid(which='both') # La grille
45     plt.savefig(base_fichier + '_amplitude.png') # Sauvegarde
46     plt.clf() # Nettoyage
47
48 # Appel effectif
49 amplitude_et_phase(130,1.5,'PNG/S11_trouver_w0_et_Q_01')
50 amplitude_et_phase(130,1.0,'PNG/S11_trouver_w0_et_Q_03')
51
52 def H(w,w0,Q):
53     '''Nouvelle fonction de transfert -> passe haut cette fois'''
54     return -A*(w/w0)**2/(1-(w/w0)**2 + 1j*w/(Q*w0))
55
56 # Comme H a été redéfinie, c'est automatiquement celle appelée par Z0 et phase
57 amplitude_et_phase(170,1.5,'PNG/S11_trouver_w0_et_Q_04')
58 amplitude_et_phase(170,1.0,'PNG/S11_trouver_w0_et_Q_02')

```





Exemple I.26

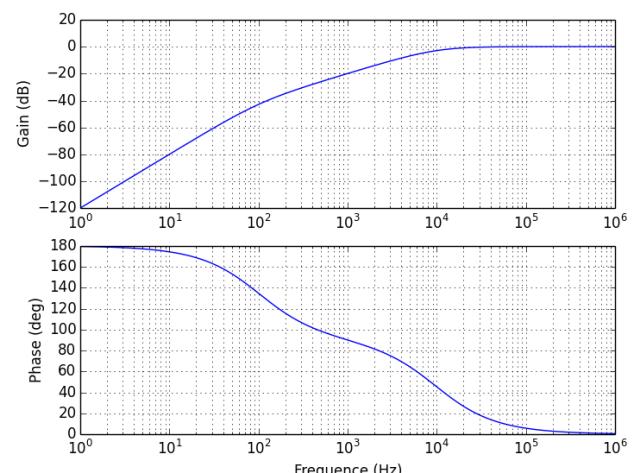
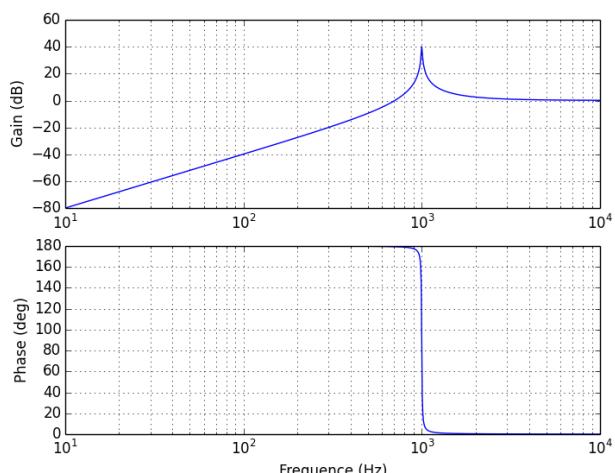
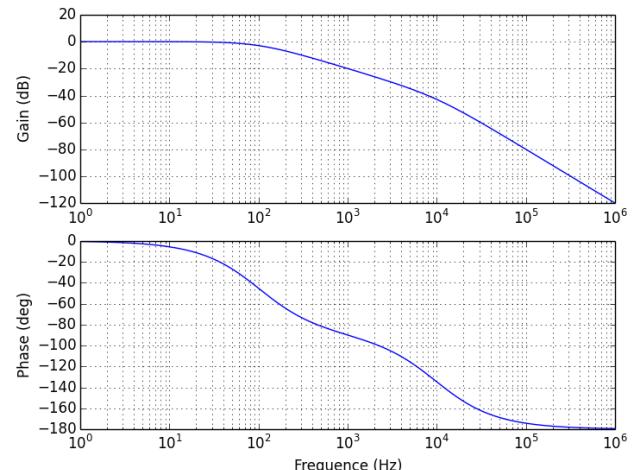
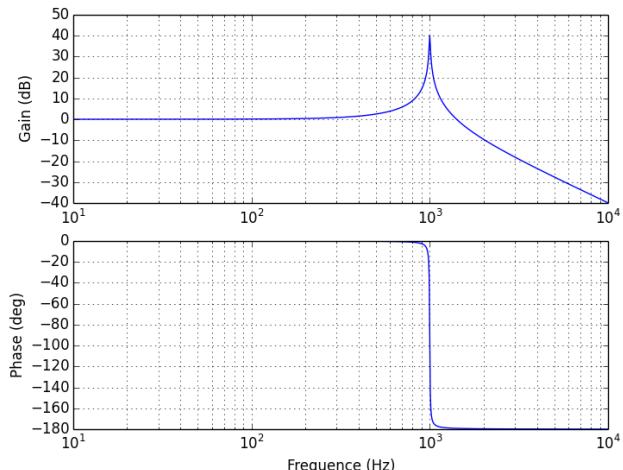
## S11 Filtres du second ordre

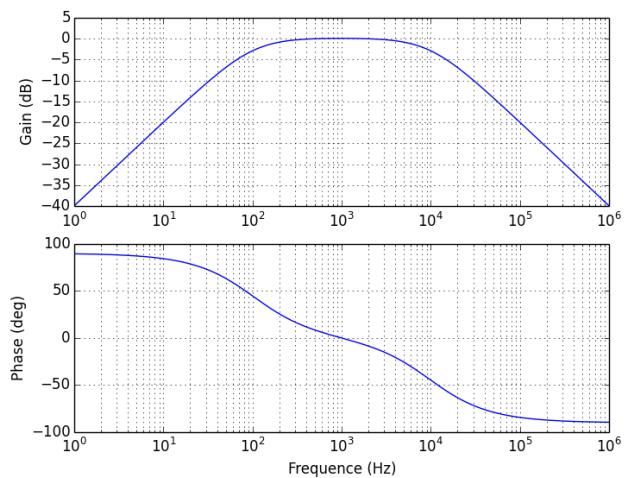
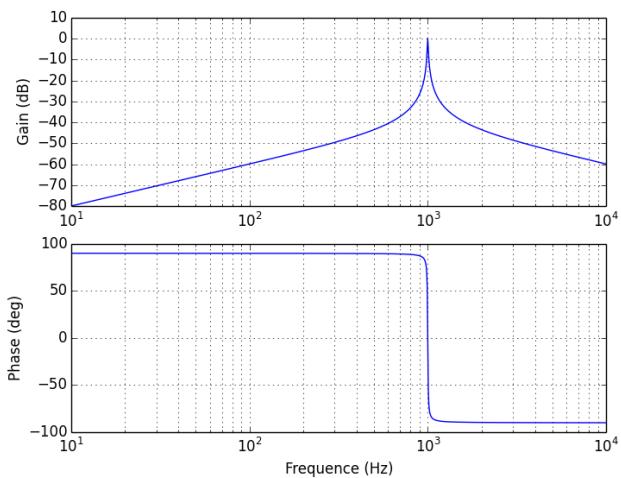
- 1    Exemple de génération de filtres du second ordre (ici pour une question de cours [QDC pour les intimes]).
- 2    Exemple de génération de filtres du second ordre (ici pour une question de cours [QDC pour les intimes]).
- 3    Exemple de génération de filtres du second ordre (ici pour une question de cours [QDC pour les intimes]).
- 4    Exemple de génération de filtres du second ordre (ici pour une question de cours [QDC pour les intimes]).

```

5
6 import numpy as np # Pour np.logspace
7 from bode import diag_bode, second_ordre # Pour les diagrammes
8
9 f100 = np.logspace(1,4,num=1000) # Echantillonnage en f pour Q=100
10 f0p1 = np.logspace(0,6,num=200) # Pareil pour Q=0.1
11
12 # Deux passe-bas, puis deux passe-hauts et enfin deux passe-bandes
13 second_ordre(10**3, 100, 'PNG/S11_filtres_QDC_PBs_Q100.png',f=f100)
14 second_ordre(10**3,1/10, 'PNG/S11_filtres_QDC_PBs_Q0_1.png',f=f0p1)
15 second_ordre(10**3, 100, 'PNG/S11_filtres_QDC_PHt_Q100.png',f=f100,type='PHt')
16 second_ordre(10**3,1/10, 'PNG/S11_filtres_QDC_PHt_Q0_1.png',f=f0p1,type='PHt')
17 second_ordre(10**3, 100, 'PNG/S11_filtres_QDC_PBd_Q100.png',f=f100,type='PBd')
18 second_ordre(10**3,1/10, 'PNG/S11_filtres_QDC_PBd_Q0_1.png',f=f0p1,type='PBd')

```





# Py4Phys II

## Bloc Mécanique

Exemple II.1

### M2 Exploration numérique de l'influence de la trainée

Exemple II.2

### M2 Influence de la trainée: portrait de phase

Exemple II.3

### M4 Module de génération de portrait de phase

```
1  """
2  Module de génération automatique de portrait de phase et d'animations
3  correspondantes..
4  """
5
6 def portrait_de_phase(x,vx,titre='Portrait de phase',
7     xlabel='$x$',ylabel='$v_x$',file=None,position=True,xlim=None,ylim=None):
8     """
9     Représentation de vx en fonction de x pour les différentes trajectoires
10    données en entrée (x et vx sont des tableaux de tableaux).
11    Si 'file' est précisé, on enregistre dans le fichier correspond, sinon on
12    affiche à l'écran.
13    Si 'position' est True, on affiche sous forme de rond le dernier point de
14    la trajectoire.
15    Si 'xlim' ou 'ylim' sont spécifiés, ils définissent les bords du graphe.
16    Sinon, c'est matplotlib qui choisit tout seul.
17    """
```

Exemple II.4

**M4 Pendule simple: non isochronisme des oscillations**

Exemple II.5

**M4 Pendule simple: portrait de phase**

Exemple II.6

**M4 Pendule simple: Oscillations amorties et portrait de phase**

Exemple II.7

**M4 Oscillateurs de Landau: effets non linéaires**

Exemple II.8

**M4 Oscillateur de Landau: Portrait de phase**

Exemple II.9

**M5 Mouvement hélicoïdal dans un champ magnétique**

Exemple II.10

**M5 Mouvement dans les champs E et B**

Exemple II.11

**Papillon de Lorentz**

Exemple II.12

**Pendule double: sensibilité aux conditions initiales**

Exemple II.13

**Hypérion: rotation chaotique et section de Poincaré**

Hypérion, un satellite de Saturne, est le seul objet connu du système solaire à être actuellement dans une phase de rotation chaotique. J'ai découvert cette étrangeté au travers de l'excellent bouquin « Dieu joue-t-il aux dés » de Ian Stewart qui y consacre un bon chapitre et demi de son livre.



**Figure 1. Hyperion**

Plus de détails peuvent être trouvés dans la publication de Wisdom, Peale et Mignard<sup>1</sup> dont une version peut être trouvée en ligne à l'adresse suivante:

<http://web.mit.edu/wisdom/www/hyperion.pdf>

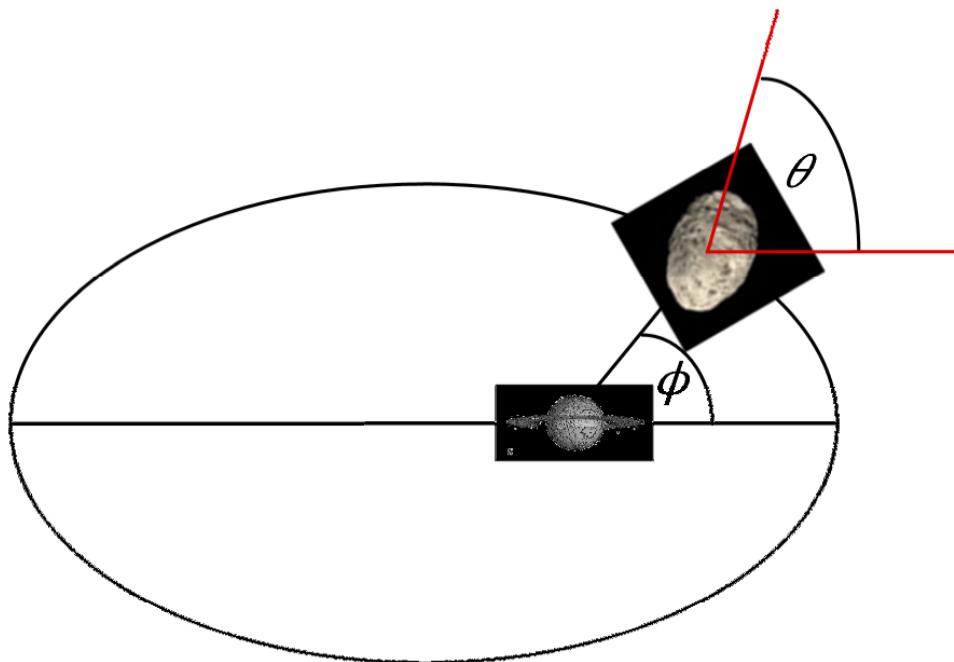
L'idée est que la forme patatoïdale d'Hypérion fait que sa rotation sur lui-même varie fortement d'une orbite sur l'autre, induisant une variation chaotique de sa luminosité (du fait qu'en plus une face est plus sombre que l'autre).

Le lien suivant est peut-être un peu plus lisible, notamment en ce qui concerne la définition des angles impliqués dans le problème:

[http://www.physics.udel.edu/~jim/PHYS460\\_660\\_13S/solarsystem/Chaotic%20rotation%20of%20Hyperion.pdf](http://www.physics.udel.edu/~jim/PHYS460_660_13S/solarsystem/Chaotic%20rotation%20of%20Hyperion.pdf)

---

<sup>1</sup>Wisdom, J., Peale, S. J., & Mignard, F. 1984, Icarus, 58, 137



**Figure 2. Definition of the angles  $\theta$  and  $\phi$ .**

Ce dernier (dont la figure précédente est extraite) permet d'obtenir les équations du mouvement en fonction de l'angle orbital  $\phi$ . En définissant  $\theta$  l'angle de rotation d'Hypérion autour de son axe principal et  $\Omega = \dot{\theta}$  sa vitesse angulaire de rotation, on obtient

$$\begin{cases} \frac{d\theta}{d\phi} = \frac{r^2}{a^2} \Omega \\ \frac{d\Omega}{dt} \phi = - \frac{B - A}{C} \frac{3}{2(1 - e^2)} \frac{a}{r} \sin(2(\theta - \phi)) \end{cases}$$

où A, B, et C sont les valeurs des trois moments d'inertie principaux (avec  $A < B < C$ ) qui, pour Hypérion, sont tels que  $(B - A)/C \approx 0,265$ . On a aussi l'excentricité  $e$  de l'orbite qui apparaît, telle que  $e \approx 0,123$  pour Hypérion. Le rapport  $a/r$  du demi-grand axe de l'orbite elliptique (stable) d'Hypérion autour de Saturne est relié à l'angle  $\phi$  via la relation

$$(1 - e^2) \frac{a}{r} = 1 + e \cos \phi$$

ce qui permet de définir entièrement le système différentiel précédent. Mais maintenant, place au code (d'autres explications y sont données...)

```

1  """
2  Programme pour construire une section de Poincaré pour Hypérion. L'idée d'une
3  section de Poincaré est de représenter le couple (theta, Omega) à chaque fois
4  que l'on passe dans le plan phi=0. Il suffit donc de demander à scipy les
5  valeurs pour les multiples de 2pi et de tracer les trajectoires pour de
6  multiples conditions initiales. On reconnaîtra les trajectoires
7  quasi-périodiques par le fait que les points successifs tombent proches les
8  uns des autres et dessinent une courbe bien définie alors que pour les
9  trajectoires chaotiques, les points semblent se déposer "au hasard" sur toute
10 une surface de la section de Poincaré.
11 """
12

```

```
13 import numpy as np
14 import scipy as sp
15 import scipy.integrate
16 import matplotlib.pyplot as plt
17
18 # Les constantes de notre problème
19 e = 0.1                      # Excentricité de l'orbite d'Hypéron
20 BmAsC = 0.265                 # Valeur de B Moins A Sur C (B-A)/C pour Hypéron
21
22 def hyperion(y,phi):
23     """ Fonction définissant le système différentiel (en phi) régissant
24     l'évolution de theta et Omega pour Hypéron. """
25     theta,Omega = y
26     a_sur_r = (1 + e*np.cos(phi))/(1-e**2)
27     dtheta_sur_dphi = 1/a_sur_r**2 * Omega
28     dOmega_sur_dphi = - BmAsC * 3 / (2*(1-e**2)) * a_sur_r * np.sin(2*(theta-phi))
29     return [dtheta_sur_dphi,dOmega_sur_dphi]
30
31 def trajectoire(Omega0,n=200,dphi=2*np.pi):
32     """ Récupération d'une trajectoire dans la section de Poincaré. Renvoie un
33     triplet contenant les theta mesurés, les Omega et les phicorrespondants.
34     """
35     y0 = [0,Omega0]                      # Condition initiale (theta=0)
36     phi = np.arange(0,n*dphi,dphi) # n points répartis tous les 2pi
37     sol = sp.integrate.odeint(hyperion,y0,phi) # Intégration effective
38     theta = (sol[:,0]+np.pi)%(2*np.pi)-np.pi
39     Omega = sol[:,1]
40     return theta,Omega,phi
41
42 # Les conditions initiales regardées (qui bien sûr doivent dépendre de e et
43 # BmAsC pour bien délimiter les zones chaotiques [ici 0.2] des zones
44 # quasi-périodiques [toutes les autres])
45 L_Omega0 = [0,0.2,0.3,0.7,2.37,2.7,2.85]
46 for Omega0 in L_Omega0:
47     if Omega0 == 0.2: n = 5000    # Il faut plus de point pour la zone chaotique
48     else: n = 500
49     theta,Omega,phi = trajectoire(Omega0,n=n)
50     plt.plot(theta,Omega,'.',label='$\Omega_0=$'.format(Omega0))
51
52 plt.xlim((-np.pi,6))
53 plt.ylim((0,3))
54 plt.title('Hyperion: Section de Poincaré dans le plan $\phi=0[2\pi]$')
55 plt.xlabel('$\theta$')
56 plt.ylabel('$\dot{\Omega}=\dot{\theta}$')
57 plt.legend()
58 plt.savefig('PNG/M_hyperion.png')
59 plt.clf()
60
61 # Regardons aussi ce que donne une trajectoire quasi-periodique et une
```

```

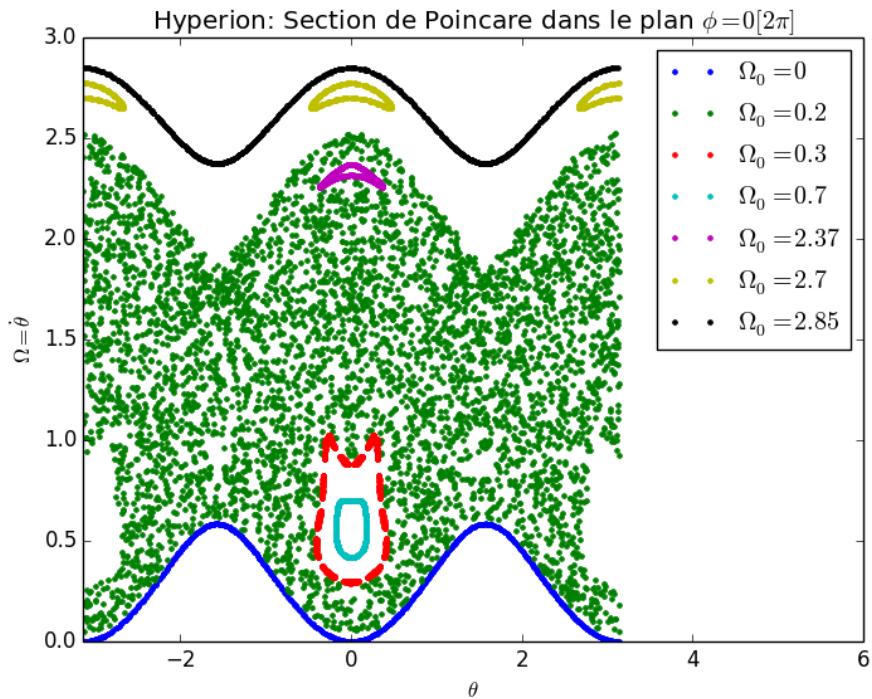
62 # trajectoire chaotique en observant la vitesse angulaire Omega en fonction de
63 # la position phi sur l'orbite
64 def plot_trajectoire_donnee(Omega0,phi_max,n,titre='',fichier=None):
65     theta,Omega,phi = trajectoire(Omega0,n=n,dphi=phi_max/n)
66     plt.plot(phi,Omega)
67     plt.title(titre)
68     plt.xlabel("Position $\\phi$ sur l'orbite")
69     plt.ylabel("Vitesse $\\Omega$ de rotation")
70     if fichier: plt.savefig(fichier)
71     else: plt.show()
72     plt.clf()
73
74 plot_trajectoire_donnee(0,100,1000,'Trajectoire quasi-periodique $\\Omega_0=0$',  

75                         'PNG/M_hyperion_0.png')
76 plot_trajectoire_donnee(0.2,400,1000,'Trajectoire chaotique $\\Omega_0=0.2$',  

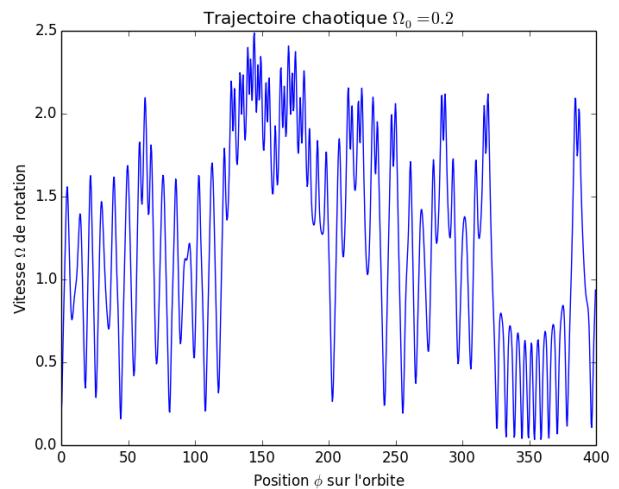
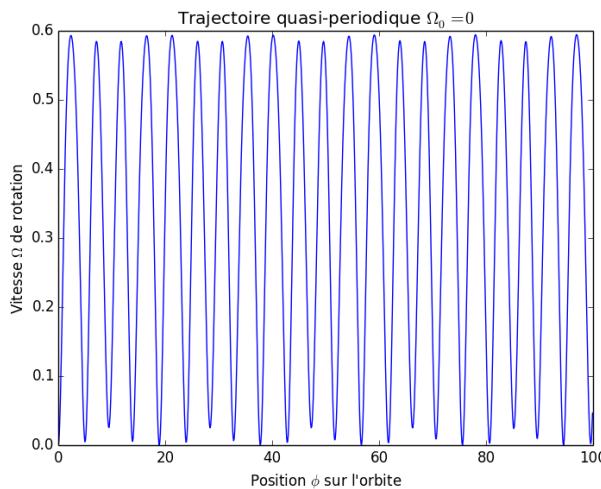
77                         'PNG/M_hyperion_0_2.png')

```

Et voici le résultat :



Une petite vision de ce que donnent des trajectoires pseudo-périodiques et chaotiques dans la paramétrisation choisie:



Bien sûr, il serait intéressant de faire une étude systématique en changeant progressivement la valeur de l'excentricité pour voir comment évoluent les zones chaotiques.

```

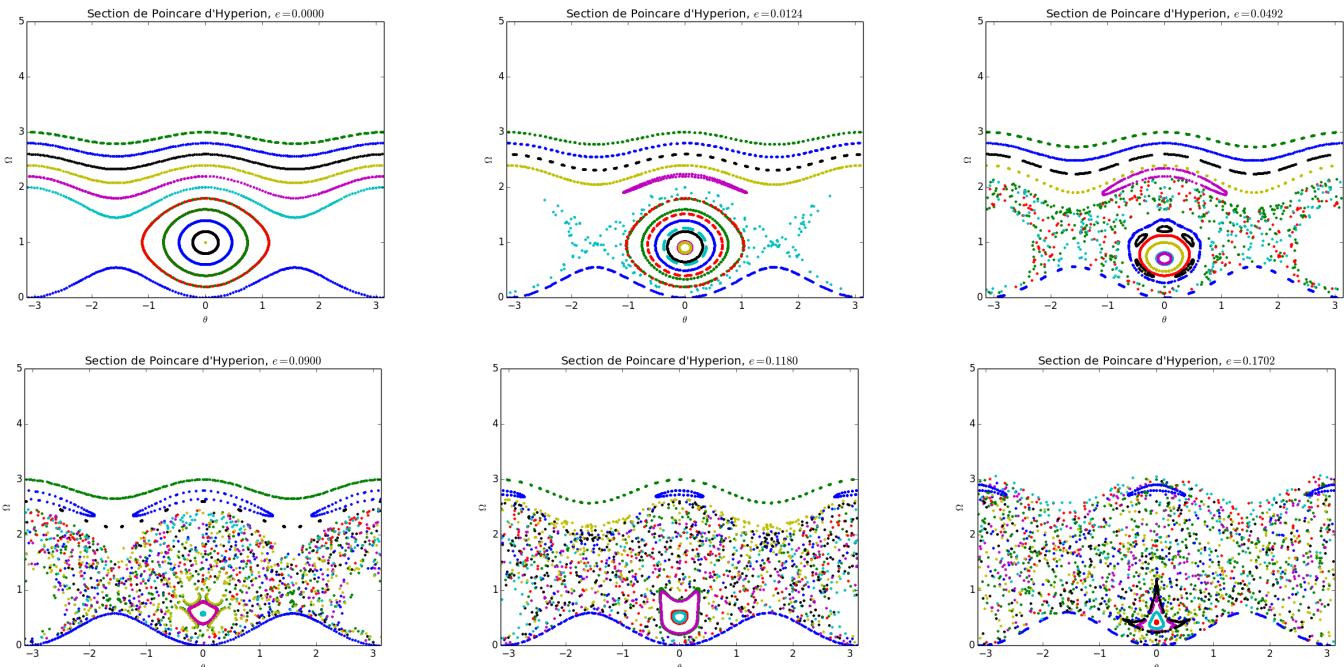
1 # Évolution des zones chaotiques quand on modifie progressivement
2 # l'excentricité de l'orbite.
3
4 def fait_diagramme(e,L_Omega0=np.arange(0,3.1,0.2),
5                     label=False,n=200,fichier=None):
6     """ Fait automatiquement la section de Poincaré pour l'excentricité e
7     fournie et en utilisant les valeurs initiales contenue dans L_Omega0 pour
8     les vitesses angulaires. N'affiche les label que si 'label' est à True.
9     On peut aussi changer le nombre 'n' de points d'échantillonnage.
10    Si le nom du fichier est donné, on y sauvegarde le résultat. Sinon, on
11    l'affiche à l'écran.
12    """
13    for Omega0 in L_Omega0:
14        theta,Omega,phi = trajectoire(Omega0,n=n)
15        plt.plot(theta,Omega,'.',label='$\backslashOmega_0={}$'.format(Omega0))
16        if label: plt.xlim(-np.pi,6) ; plt.legend()
17        else:     plt.xlim(-np.pi,np.pi)
18        plt.ylim(0,5)
19        plt.xlabel('$\backslashtheta$')
20        plt.ylabel('$\backslashOmega$')
21        plt.title("Section de Poincaré d'Hyperion, $e={:.4f}$".format(e))
22        if fichier: plt.savefig(fichier)
23        else:       plt.show()
24        plt.clf()
25
26 base_name = 'PNG/M_hyperion_e'
27
28 for e in np.arange(0,0.2,0.0002):
29     print('e={}'.format(e))
30     fichier = base_name + '{:.4f}'.format(e).replace('.','_')
31     fait_diagramme(e,fichier=fichier)
32
33 # Ne reste plus qu'à rassembler en un fichier mpeg à l'aide de convert puis de

```

```
34 # ppmtoy4m et mpeg2enc (paquet mjpegttools à installer sur la machine)
35
36 import os
37
38 cmd = '(for f in ' + base_name + '*png ; '
39 cmd+= 'do convert -density 100x100 $f -depth 8 -resize 600x600 PNM:- ; done)'
40 cmd+= ' | ppmtoy4m -S 420mpeg2'
41 cmd+= ' | mpeg2enc -f1 -b 12000 -q7 -G 30 -o {}film.mpeg'.format(base_name)
42
43 print("Execution de la commande de conversion")
44 print(cmd)
45 os.system(cmd)
46 print("Fin de la commande de conversion")
```

Et voici le résultat en images:

[http://pcsi.kleber.free.fr/IPT/doc/M\\_hyperion\\_efilm.mpeg](http://pcsi.kleber.free.fr/IPT/doc/M_hyperion_efilm.mpeg)



# Py4Phys III

## Bloc Thermodynamique

Exemple III.1

### T1 Balles rebondissantes en boîte: illustration du facteur de Boltzmann

```
1      """
2
3  Code servant à simuler un ensemble de balles rebondissantes soumises à la
4  gravité. Le code est adapté de celui proposé par le cours de l'ENS Ulm:
5  "Statistical Mechanics: Algorithms and Computations" sur le site coursera.org.
6  cf https://class.coursera.org/smac-001
7
8  L'idée est de faire une simulation "event-driven", c'est-à-dire que les
9  équations du mouvement sont connues entre deux chocs, il suffit donc de
10 déterminer la date du prochain choc et d'utiliser les positions et vitesses
11 connues après un choc pour calculer facilement les positions entre deux chocs.
12
13 Attention, comme on ne sait pas à l'avance qui va rencontrer qui, l'algorithme
14 est quadratique avec le nombre de particules considérées.
15
16 """
17
18
19 import os, math, pylab
20 import numpy as np
21 import numpy.random
22
23 N_sur_4 = 100    # On prend un multiple de 4
24 N = 4*N_sur_4   # pour les couleurs
25 output_dir = "PNG/T1_balles_rebondissantes_en_boite_movie"
26 colors = ['r', 'b', 'g', 'orange']*N
27 sigma = 0.002          # Rayon des particules
28 singles = [(i,j) for i in range(N) for j in range(2)]  # L'ensemble des particules (en
29 pairs = [(i,j) for i in range(N) for j in range(i+1,N)] # L'ensemble des paires
30
31 t = 0.0              # Temps initial
```

```
32 dt = 0.02                                # dt=0 corresponds to event-to-event animation
33 n_steps = 1000                            # Nombre d'étapes
34
35
36 def wall_time(pos_a, vel_a, sigma, g):
37     """Fonction qui détermine le prochain choc d'une particule avec un mur."""
38     del_t = float('inf')
39     #print(pos_a,vel_a,g)
40     if g == 0: # Cas d'un axe sans gravité
41         if vel_a > 0.0:
42             del_t = (1.0 - sigma - pos_a) / vel_a
43         elif vel_a < 0.0:
44             del_t = (pos_a - sigma) / abs(vel_a)
45     else:
46         Delta1= vel_a**2 - 2*g*(pos_a - (1-sigma))
47         Delta2= vel_a**2 - 2*g*(pos_a - sigma)
48         tpossibles = []
49         if Delta1 >= 0:
50             tpossibles += [1/g*(-vel_a - Delta1**0.5), 1/g*(-vel_a + Delta1**0.5)]
51         if Delta2 >= 0:
52             tpossibles += [1/g*(-vel_a - Delta2**0.5), 1/g*(-vel_a + Delta2**0.5)]
53         #print(tpossibles)
54         tpossibles = [t for t in tpossibles if t > 1e-5] # On ne garde que les positifs
55         #print(tpossibles)
56         if len(tpossibles)>0:
57             del_t = min(tpossibles)
58         #print(del_t)
59     return del_t
60
61 def pair_time(pos_a, vel_a, pos_b, vel_b, sigma):
62     """
63     Fonction qui détermine le temps du prochain choc d'une particule avec une autre. Magie de la gravité: les termes quadratiques disparaissent ! On se ramène donc au cas des particules libres.
64     """
65
66     del_x = [pos_b[0] - pos_a[0], pos_b[1] - pos_a[1]]
67     del_x_sq = del_x[0] ** 2 + del_x[1] ** 2
68     del_v = [vel_b[0] - vel_a[0], vel_b[1] - vel_a[1]]
69     del_v_sq = del_v[0] ** 2 + del_v[1] ** 2
70     scal = del_v[0] * del_x[0] + del_v[1] * del_x[1]
71     Upsilon = scal ** 2 - del_v_sq * (del_x_sq - 4.0 * sigma ** 2)
72     if Upsilon > 0.0 and scal < 0.0:
73         del_t = - (scal + math.sqrt(Upsilon)) / del_v_sq
74     else:
75         del_t = float('inf')
76
77     return del_t
78
79 def min_arg(l):
80     """Récupère à la fois le minimum d'une liste et l'indice correspondant à ce minimum
```

```
81     return min(zip(l, range(len(l)))))

82

83 g = np.array([0,-1])

84

85 def compute_next_event(pos, vel):
86     """ Détermination du prochain "événement", c'est-à-dire l'instant de ce
87     choc et la particule (ou la paire) correspondante. À noter que l'on stocke
88     toutes ces infos dans un seul indice (cf disjonction de cas dans
89     compute_new_velocities)."""
90     #print('-'*70)
91     wall_times = [wall_time(pos[k][l], vel[k][l], sigma, g[l]) for k, l in singles]
92     pair_times = [pair_time(pos[k], vel[k], pos[l], vel[l], sigma) for k, l in pairs]
93     return min_arg(wall_times + pair_times)

94

95 def compute_new_velocities(pos, vel, next_event_arg):
96     """Calcul des nouvelles vitesses"""
97     if next_event_arg < len(singles): # Cas d'un choc avec le mur
98         collision_disk, direction = singles[next_event_arg]
99         vel[collision_disk][direction] *= -1.0 # seule la vitesse sur cet axe est modifiée
100    else:                                # Cas d'un choc entre deux particules de même vitesse
101        a, b = pairs[next_event_arg - len(singles)]
102        del_x = [pos[b][0] - pos[a][0], pos[b][1] - pos[a][1]]
103        abs_x = math.sqrt(del_x[0]**2 + del_x[1]**2)
104        e_perp = [c / abs_x for c in del_x]
105        del_v = [vel[b][0] - vel[a][0], vel[b][1] - vel[a][1]]
106        scal = del_v[0] * e_perp[0] + del_v[1] * e_perp[1]
107        for k in range(2):
108            vel[a][k] += e_perp[k] * scal
109            vel[b][k] -= e_perp[k] * scal

110

111 img = 0
112 if not os.path.exists(output_dir): os.makedirs(output_dir)

113

114 def snapshot(t, pos, vel, colors, X, Y, arrow_scale=.2):
115     """ La routine qui s'occupe des tracés graphiques."""
116     global img
117     nbmax = 80 # Limite verticale des histogrammes
118     nb_bins = 20 # Nombre de bins pour les histogrammes
119     # Quelques déclarations de tailles
120     pylab.subplots_adjust(left=0.10, right=0.90, top=0.90, bottom=0.10)
121     pylab.gcf().set_size_inches(12, 12*2/3)

122

123     # Le premier sous-plot: carré de 2x2
124     ax1 = pylab.subplot2grid((2,3),(0,0),colspan=2, rowspan=2)
125     pylab.setp(pylab.gca(), xticks=[0, 1], yticks=[0, 1])
126     pylab.plot(X,Y,'k') # On y met le trajet de la dernière particule
127     pylab.xlim((0,1)) # On doit astreindre les côtés horizontaux
128     pylab.ylim((0,1)) # et verticaux
129     # Boucle sur les points pour rajouter les cercles colorés
```

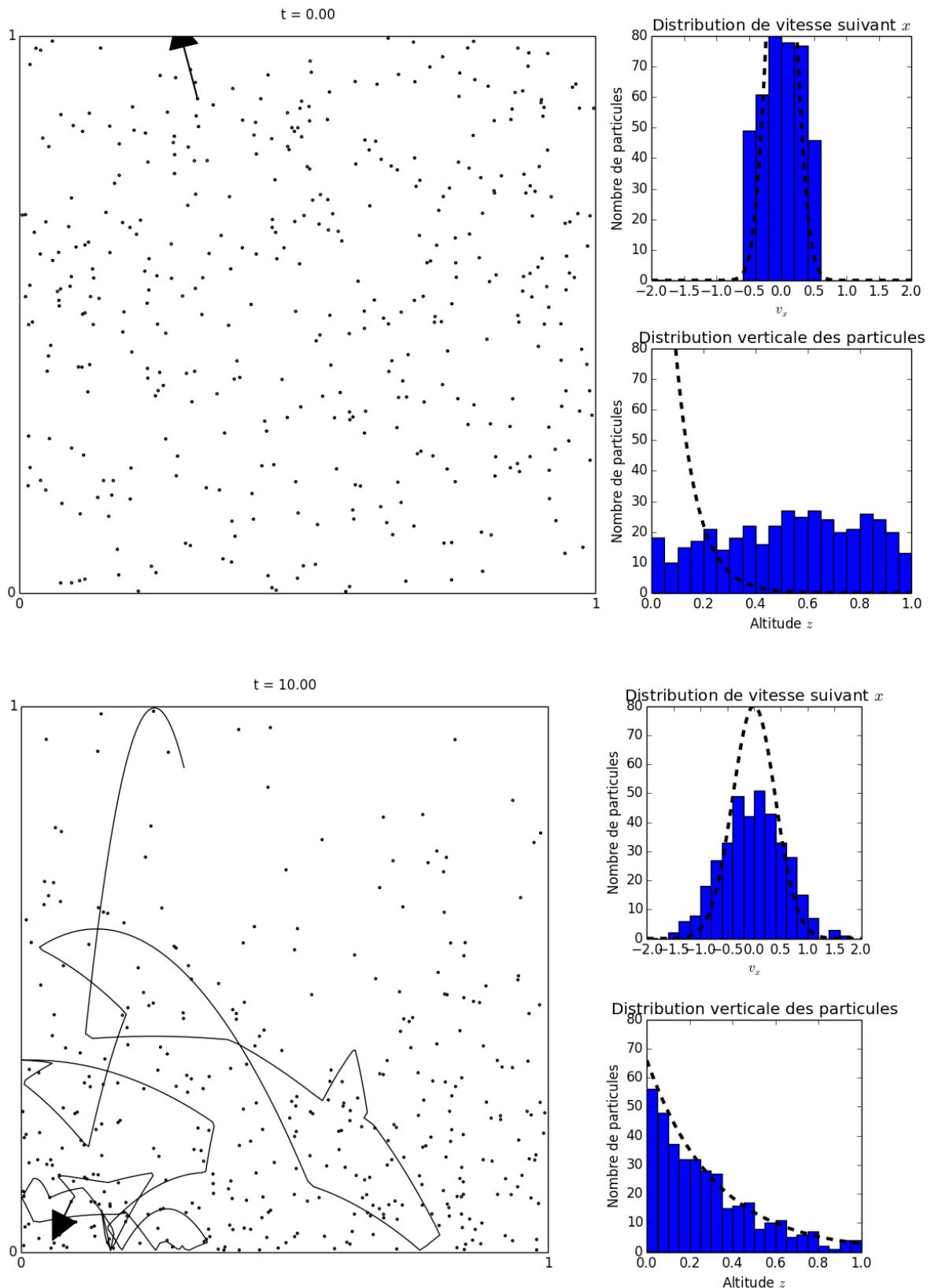
```

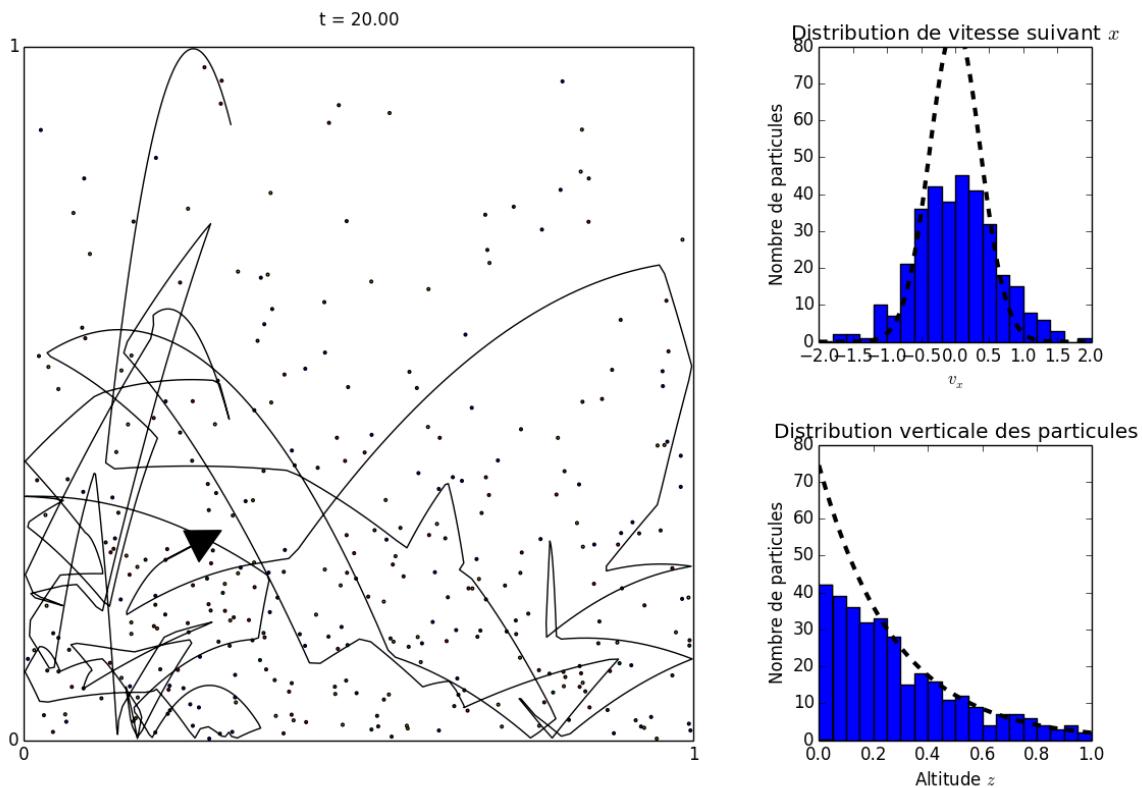
130     for (x, y), c in zip(pos, colors):
131         circle = pylab.Circle((x, y), radius=sigma, fc=c)
132         pylab.gca().add_patch(circle)
133         # La dernière particule a droit à son vecteur vitesse
134         dx,dy = vel[-1] * arrow_scale
135         pylab.arrow( x, y, dx, dy, fc="k", ec="k", head_width=0.05, head_length=0.05 )
136         pylab.text(.5, 1.03, 't = %.2f' % t, ha='center')
137
138         # Second sous-plot: histogramme de la projection suivant x des vitesses
139         ax2 = pylab.subplot2grid((2,3),(0,2),colspan=1, rowspan=1)
140
141         # Récupération des vitesses et des positions pour les particules encore
142         # dans le champ d'étude (au cas où certaines se seraient échappées [bug à
143         # corriger, mais on fait avec...])
144         vitesses = np.array([vel[k] for k in range(N) if 0 <= pos[k][1] <=1 and 0 <= pos[k][0] <= nb_max])
145         positions= np.array([pos[k] for k in range(N) if 0 <= pos[k][1] <=1 and 0 <= pos[k][0] <= nb_max])
146         nb_part = vitesses.shape[0]                                # Nombre de particules encore dans le champ
147         r = (-2,2)                                              # Intervalle de vitesses regardé
148         pylab.hist(vitesses[:,0],bins=nb_bins,range=r) # Dessin de l'histogramme
149         # Récupération des centres de positionnement des histogrammes
150         hist_data,bin_edges = np.histogram(vitesses[:,0],bins=nb_bins,range=r)
151         bin_centers = (bin_edges[:-1] + bin_edges[1:])/2
152         pylab.xlim(r)                                         # On impose l'abscisse
153         pylab.ylim((0,nbmax))                                 # et l'ordonnée
154         pylab.ylabel("Nombre de particules")                 # Légende verticale
155         pylab.xlabel("$v_x$")                                # et horizontale
156         pylab.title('Distribution de vitesse suivant $x$') # Titre
157         # La distribution est sensée être gaussienne
158         v = np.linspace(r[0],r[1],100)                      # Echantillonnage en vitesses
159         #Dv= bin_centers[1] - bin_centers[0]
160         #dispx = np.sum(vel[:,0]**2)/N
161         disp2 = np.sum(vitesses**2)/(2*N)                  # Dispersion (carrée) des vitesses (avec
162         A = N / (sum(np.exp(-(bin_centers)**2/disp2))) # Normalisation de la gaussienne
163         # Représentation graphique de la gaussienne en pointillés
164         pylab.plot(v,A*np.exp(-v**2/disp2), '--k', linewidth=3.0)
165
166         # Troisième sous-plot: histogramme de répartition des particules selon l'altitude
167         ax3 = pylab.subplot2grid((2,3),(1,2),colspan=1, rowspan=1)
168         r = (0,1)                                              # Intervalle de hauteurs regardé
169         pylab.hist(positions[:,1],bins=nb_bins,range=r) # Dessin de l'histogramme
170         pylab.xlim(r)                                         # On impose l'abscisse
171         pylab.ylim((0,nbmax))                                 # et l'ordonnée
172         pylab.ylabel("Nombre de particules")                 # Légende verticale
173         pylab.xlabel("Altitude $z$")                         # et horizontale
174         pylab.title("Distribution verticale des particules")
175         # Récupération des centres de positionnement des histogrammes
176         hist_data,bin_edges = np.histogram(positions[:,1],bins=nb_bins,range=r)
177         bin_centers = (bin_edges[:-1] + bin_edges[1:])/2
178         A = N/(sum(np.exp(g[1]*bin_centers/disp2)))# Calcul de la normalisation

```

```
179     z = np.linspace(r[0],r[1],100)           # Échantillonage des z
180     # Affichage de la courbe exponentielle théorique
181     pylab.plot(z, A * np.exp(g[1]*z/disp2), '-k', linewidth=3.0)
182     pylab.tight_layout() # Pour ajuster un peu les bords
183
184     pylab.savefig(os.path.join(output_dir, '{:04d}.png'.format(img)))
185     img += 1
186
187 def check_position():
188     """ Une routine pour s'assurer que les particules ne se chevauchent pas au
189     départ. Il peut se passer un certain temps avant que l'on trouve une
190     configuration adéquate. """
191     continue_condition = True # Condition de non-arrêt
192     c = 0                      # Compteur
193     d2= 4*sigma**2             # Distance (carrée) de sécurité
194     while continue_condition:
195         c += 1
196         if c%100 == 0:          # Un peu de feedback
197             print(c,'trials to get initial conditions and still trying... ')
198         pos = np.random.random((N,2))*(1-2*sigma) + sigma
199         k = 0
200         for (i,j) in pairs:    # Les vérifications sur toutes les paires
201             if sum((pos[i]-pos[j])**2) > d2: k+= 1
202             else:
203                 if c%100 == 0: print(i,j)
204                 break
205             if k == len(pairs): continue_condition = False
206     print("Let's compute some physics !")
207     return pos
208
209 # Le début du programme proprement dit
210
211 pos = check_position()           # Sélection des positions
212 vel = 0.5*(np.random.random((N,2))*2 - 1) # et des vitesses
213 X,Y = [pos[-1][0]], [pos[-1][1]] # La dernière particule va être suivie à la loupe
214 next_event, next_event_arg = compute_next_event(pos, vel) # On calcule la première étape
215 snapshot(t, pos, vel, colors, X, Y) # et on prend une première photo.
216 for step in range(n_steps):      # On boucle
217     if dt:                      # Cas normal,
218         next_t = t + dt          # on avance de dt
219     else:                       # Sinon,
220         next_t = t + next_event # c'est qu'on veut regarder choc après choc
221     while t + next_event <= next_t:# Début des calculs jusqu'à la prochaine sortie
222         t += next_event          # On avance
223         # On met à jour les position
224         pos += [vel[k] * next_event + g/2 * next_event**2 for k in range(N)]
225         # Et les vitesses
226         vel += [g * next_event for k in range(N)]
227         # Ainsi que les vitesses des particules ayant été "choquées"
```

```
228         compute_new_velocities(pos, vel, next_event_arg)
229         # On calcule le prochain évènement.
230         next_event, next_event_arg = compute_next_event(pos, vel)
231         remain_t = next_t - t           # S'il est après la mise à jour,
232         # On met à jour les position pour le snapshot
233         pos += [vel[k] * remain_t + g/2 * remain_t**2 for k in range(N)]
234         # Et les vitesses
235         vel += [g * remain_t for k in range(N)]
236         t += remain_t                 # On arrive au temps voulu
237         next_event -= remain_t       # et on corrige du temps restant
238         X.append(pos[-1][0])        # Suivi x de la dernière particule
239         Y.append(pos[-1][1])        # Ainsi que Y
240         snapshot(t, pos, vel, colors, X, Y) # Souriez pour la photo
241         print('time',t)             # et un peu de feedback
242
243 # Ne reste plus qu'à rassembler en un fichier mpeg à l'aide de convert puis de
244 # ppmtoy4m et mpeg2enc (paquet mjpegtools à installer sur la machine)
245
246 import os
247
248 cmd = '(for f in ' + output_dir + '/*png ; '
249 cmd+= 'do convert -density 100x100 $f -depth 8 -resize 600x600 PNM:- ; done)'
250 cmd+= ' | ppmtoy4m -S 420mpeg2'
251 cmd+= ' | mpeg2enc -f1 -b 12000 -q7 -G 30 -o {}/film.mpeg'.format(output_dir)
252
253 print("Execution de la commande de conversion")
254 print(cmd)
255 os.system(cmd)
256 print("Fin de la commande de conversion")
```





Exemple III.2

## T1 Particules dans une boîte: illustration du mouvement brownien

Exemple III.3

## T1 Particules libres dans une boîte: distribution des vitesses

```

1     """
2
3     Code servant à simuler un gaz parfait soumis uniquement aux chocs entre
4     molécules. Le code est adapté de celui proposé par le cours de l'ENS Ulm:
5     "Statistical Mechanics: Algorithms and Computations" sur le site coursera.org.
6     cf https://class.coursera.org/smac-001
7
8     L'idée est de faire une simulation "event-driven", c'est-à-dire que les
9     équations du mouvement sont connues entre deux chocs, il suffit donc de
10    déterminer la date du prochain choc et d'utiliser les positions et vitesses
11    connues après un choc pour calculer facilement les positions entre deux chocs.
12
13    Attention, comme on ne sait pas à l'avance qui va rencontrer qui, l'algorithme
14    est quadratique avec le nombre de particules considérées.
15
16     """
17
18
19    import os, math, pylab
20    import numpy as np

```

```
21 import numpy.random
22
23 N_sur_4 = 20    # On prend un multiple de 4
24 N = 4*N_sur_4  # pour les couleurs
25 output_dir = "PNG/T1_particules_en_boite_libre_movie"
26 colors = ['r', 'b', 'g', 'orange']*N
27
28 def wall_time(pos_a, vel_a, sigma):
29     """Fonction qui détermine le prochain choc d'une particule avec un mur."""
30     if vel_a > 0.0:
31         del_t = (1.0 - sigma - pos_a) / vel_a
32     elif vel_a < 0.0:
33         del_t = (pos_a - sigma) / abs(vel_a)
34     else:
35         del_t = float('inf')
36     return del_t
37
38 def pair_time(pos_a, vel_a, pos_b, vel_b, sigma):
39     """Fonction qui détermine le temps du prochain choc d'une particule avec une autre.
40     del_x = [pos_b[0] - pos_a[0], pos_b[1] - pos_a[1]]
41     del_x_sq = del_x[0] ** 2 + del_x[1] ** 2
42     del_v = [vel_b[0] - vel_a[0], vel_b[1] - vel_a[1]]
43     del_v_sq = del_v[0] ** 2 + del_v[1] ** 2
44     scal = del_v[0] * del_x[0] + del_v[1] * del_x[1]
45     Upsilon = scal ** 2 - del_v_sq * (del_x_sq - 4.0 * sigma ** 2)
46     if Upsilon > 0.0 and scal < 0.0:
47         del_t = - (scal + math.sqrt(Upsilon)) / del_v_sq
48     else:
49         del_t = float('inf')
50     return del_t
51
52 def min_arg(l):
53     """Récupère à la fois le minimum d'une liste et l'indice correspondant à ce minimum
54     return min(zip(l, range(len(l))))
55
56 def compute_next_event(pos, vel):
57     """ Détermination du prochain "événement", c'est-à-dire l'instant de ce
58     choc et la particule (ou la paire) correspondante. À noter que l'on stocke
59     toutes ces infos dans un seul indice (cf disjonction de cas dans
60     compute_new_velocities)."""
61     wall_times = [wall_time(pos[k][l], vel[k][l], sigma) for k, l in singles]
62     pair_times = [pair_time(pos[k], vel[k], pos[l], vel[l], sigma) for k, l in pairs]
63     return min_arg(wall_times + pair_times)
64
65 def compute_new_velocities(pos, vel, next_event_arg):
66     """Calcul des nouvelles vitesses"""
67     if next_event_arg < len(singles): # Cas d'un choc avec le mur
68         collision_disk, direction = singles[next_event_arg]
69         vel[collision_disk][direction] *= -1.0 # seule la vitesse sur cet axe est modifiée
```

```
70         else:                                # Cas d'un choc entre deux particules de même mas
71             a, b = pairs[next_event_arg - len(singles)]
72             del_x = [pos[b][0] - pos[a][0], pos[b][1] - pos[a][1]]
73             abs_x = math.sqrt(del_x[0] ** 2 + del_x[1] ** 2)
74             e_perp = [c / abs_x for c in del_x]
75             del_v = [vel[b][0] - vel[a][0], vel[b][1] - vel[a][1]]
76             scal = del_v[0] * e_perp[0] + del_v[1] * e_perp[1]
77             for k in range(2):
78                 vel[a][k] += e_perp[k] * scal
79                 vel[b][k] -= e_perp[k] * scal
80
81     img = 0
82     if not os.path.exists(output_dir): os.makedirs(output_dir)
83
84 def snapshot(t, pos, vel, colors, X, Y, arrow_scale=.2):
85     """ La routine qui s'occupe des tracés graphiques. """
86     global img
87     nbmax = 20
88     pylab.subplots_adjust(left=0.10, right=0.90, top=0.90, bottom=0.10)
89     pylab.gcf().set_size_inches(12, 12*2/3)
90     # Le premier sous-plot: carré de 2x2
91     ax1 = pylab.subplot2grid((2,3),(0,0),colspan=2, rowspan=2)
92     pylab.setp(pylab.gca(), xticks=[0, 1], yticks=[0, 1])
93     pylab.plot(X,Y, 'k') # On y met le trajet de la dernière particule
94     pylab.xlim((0,1))    # On doit astreindre les côtés horizontaux
95     pylab.ylim((0,1))    # et verticaux
96     # Boucle sur les points pour rajouter les cercles colorés
97     for (x, y), c in zip(pos, colors):
98         circle = pylab.Circle((x, y), radius=sigma, fc=c)
99         pylab.gca().add_patch(circle)
100    dx,dy = vel[-1] * arrow_scale # La dernière particule a droit à son vecteur vitesse
101    pylab.arrow( x, y, dx, dy, fc="k", ec="k", head_width=0.05, head_length=0.05 )
102    pylab.text(.5, 1.03, 't = %.2f' % t, ha='center')
103    # Second sous-plot: histogramme de la projection suivant x des vitesses
104    ax2 = pylab.subplot2grid((2,3),(0,2),colspan=1, rowspan=1)
105    r = (-2,2) # Intervalle de vitesses regardé
106    pylab.hist(vel[:,0], bins=20, range=r)
107    pylab.xlim(r)
108    pylab.ylim((0,nbmax))
109    pylab.ylabel("Nombre de particules")
110    pylab.xlabel("$v_x$")
111    # Troisième sous-plot: histogramme de la norme des vitesses
112    ax3 = pylab.subplot2grid((2,3),(1,2),colspan=1, rowspan=1)
113    r = (0,2) # Intervalle de vitesses regardé
114    pylab.hist(np.sqrt(np.sum(vel**2, axis=1)), bins=20, range=r)
115    pylab.xlim(r)
116    pylab.ylim((0,nbmax))
117    pylab.ylabel("Nombre de particules")
118    pylab.xlabel("$||\vec{v}||$")
```

```
119     pylab.savefig(os.path.join(output_dir, '{:04d}.png'.format(img)))
120     img += 1
121
122 def check_position():
123     """ Une routine pour s'assurer que les particules ne se chevauchent pas au
124     départ. Il peut se passer un certain temps avant que l'on trouve une
125     configuration adéquate. """
126     continue_condition = True # Condition de non-arrêt
127     c = 0                      # Compteur
128     d2= 4*sigma**2              # Distance (carrée) de sécurité
129     while continue_condition:
130         c += 1
131         if c%100 == 0:          # Un peu de feedback
132             print(c,'trials to get initial conditions and still trying... ')
133         pos = np.random.random((N,2))*(1-2*sigma) + sigma
134         k = 0
135         for (i,j) in pairs:    # Les vérifications sur toutes les paires
136             if sum((pos[i]-pos[j])**2) > d2: k+= 1
137             else:
138                 if c%100 == 0: print(i,j)
139                 break
140         if k == len(pairs): continue_condition = False
141     print("Let's compute some physics !")
142     return pos
143
144 sigma = 0.01                      # Rayon des particules
145 singles = [(i,j) for i in range(N) for j in range(2)] # L'ensemble des particules (en
146 pairs = [(i,j) for i in range(N) for j in range(i+1,N)] # L'ensemble des paires
147 pos = check_position()            # Sélection des positions
148 vel = np.random.random((N,2))*2 - 1# et des vitesses
149 X,Y = [pos[-1][0]], [pos[-1][1]]   # La dernière particule va être suivie à la loupe
150
151 t = 0.0                            # Temps initial
152 dt = 0.02                          # dt=0 corresponds to event-to-event animation
153 n_steps = 1000                     # Nombre d'étapes
154 next_event, next_event_arg = compute_next_event(pos, vel) # On calcule la première étape
155 snapshot(t, pos, vel, colors, X, Y) # et on prend une première photo.
156 for step in range(n_steps):        # On boucle
157     if dt:                         # Cas normal,
158         next_t = t + dt             # on avance de dt
159     else:                          # Sinon,
160         next_t = t + next_event    # c'est qu'on veut regarder choc après choc
161     while t + next_event <= next_t:# Début des calculs jusqu'à la prochaine sortie
162         t += next_event           # On avance
163         pos += vel * next_event   # On met à jour les position
164         # Ainsi que les vitesses des particules ayant été "choquées"
165         compute_new_velocities(pos, vel, next_event_arg)
166         # On calcule le prochain évènement.
167         next_event, next_event_arg = compute_next_event(pos, vel)
```

```

168     remain_t = next_t - t           # S'il est après la mise à jour,
169     pos += vel * next_event       # On met à jour les position pour le snapshot
170     t += remain_t                # On arrive au temps voulu
171     next_event -= remain_t      # et on corrige du temps restant
172     X.append(pos[-1][0])        # Suivi x de la dernière particule
173     Y.append(pos[-1][1])        # Ainsi que Y
174     snapshot(t,pos,vel,colors,X,Y) # Souriez pour la photo
175     print('time',t)             # et un peu de feedback

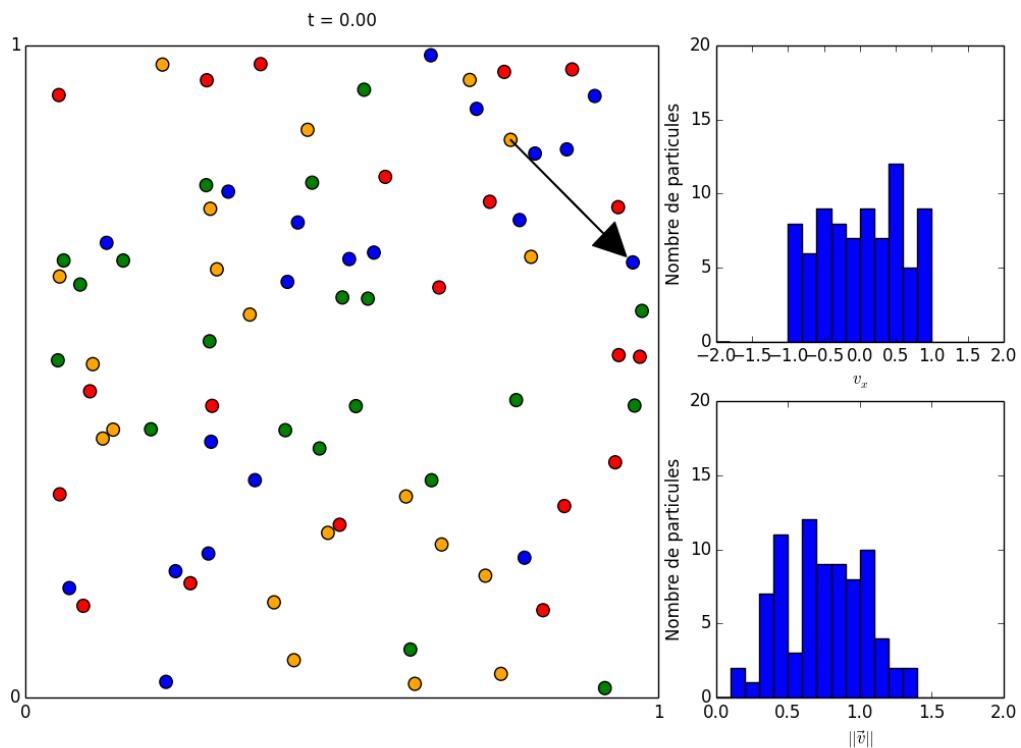
176
177 # Ne reste plus qu'à rassembler en un fichier mpeg à l'aide de convert puis de
178 # ppmtoy4m et mpeg2enc (paquet mjpegtools à installer sur la machine)

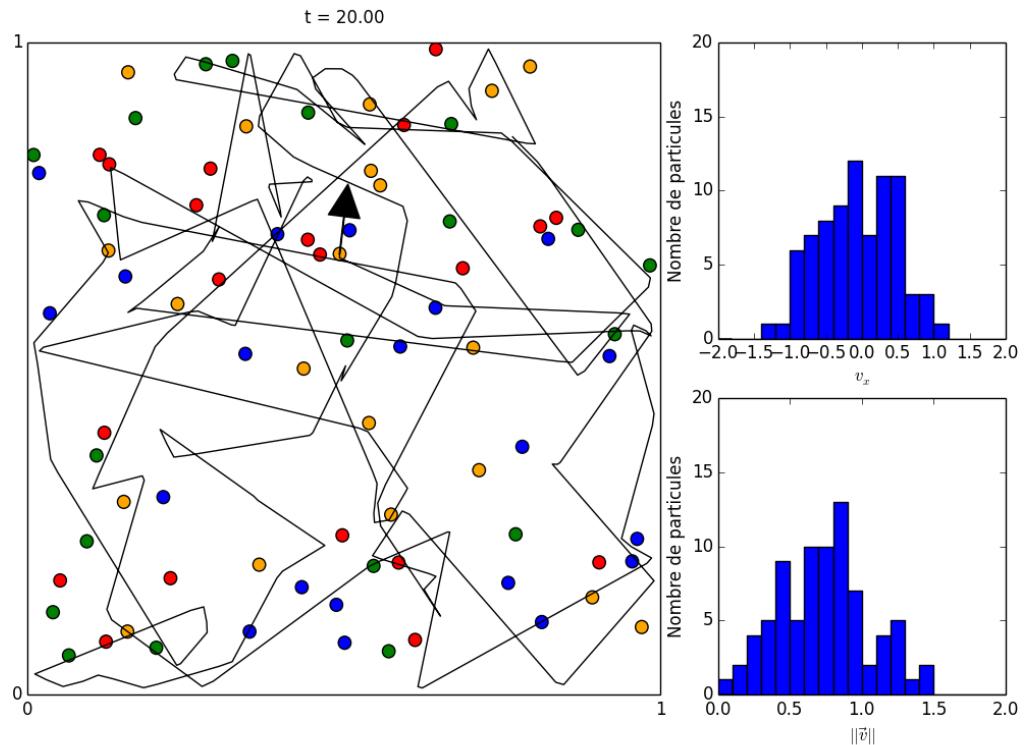
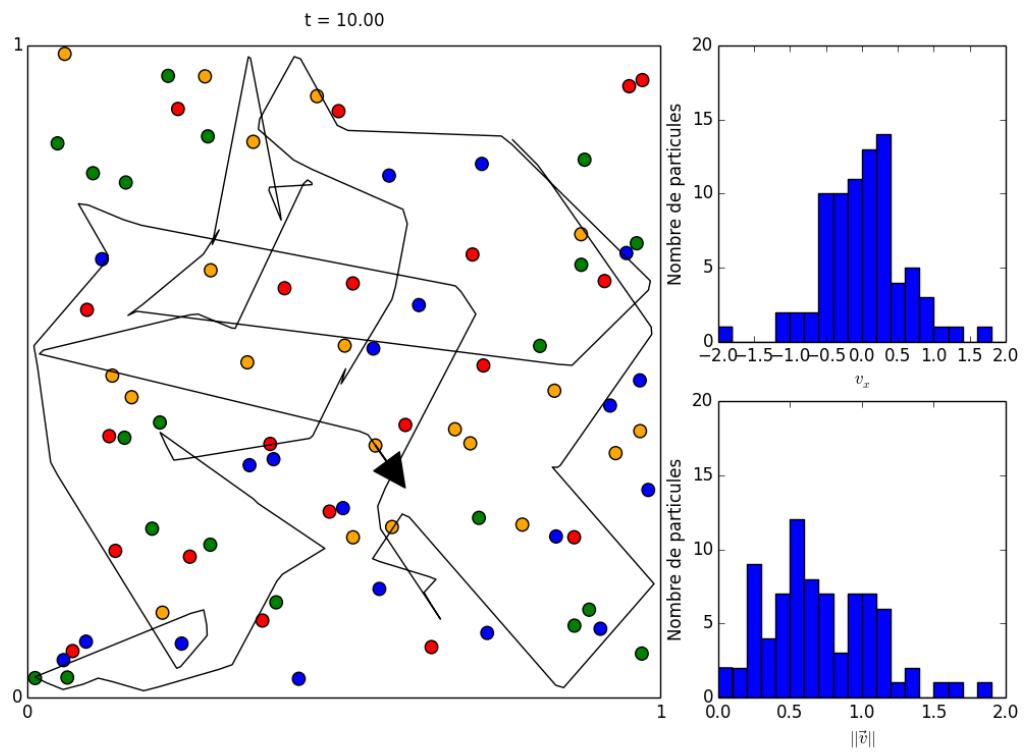
179
180 import os

181
182 cmd = '(for f in ' + output_dir + '/*png ; '
183 cmd+= 'do convert -density 100x100 $f -depth 8 -resize 600x600 PNM:- ; done)'
184 cmd+= ' | ppmtoy4m -S 420mpeg2'
185 cmd+= ' | mpeg2enc -f1 -b 12000 -q7 -G 30 -o {}/film.mpeg'.format(output_dir)

186
187 print("Execution de la commande de conversion")
188 print(cmd)
189 os.system(cmd)
190 print("Fin de la commande de conversion")

```



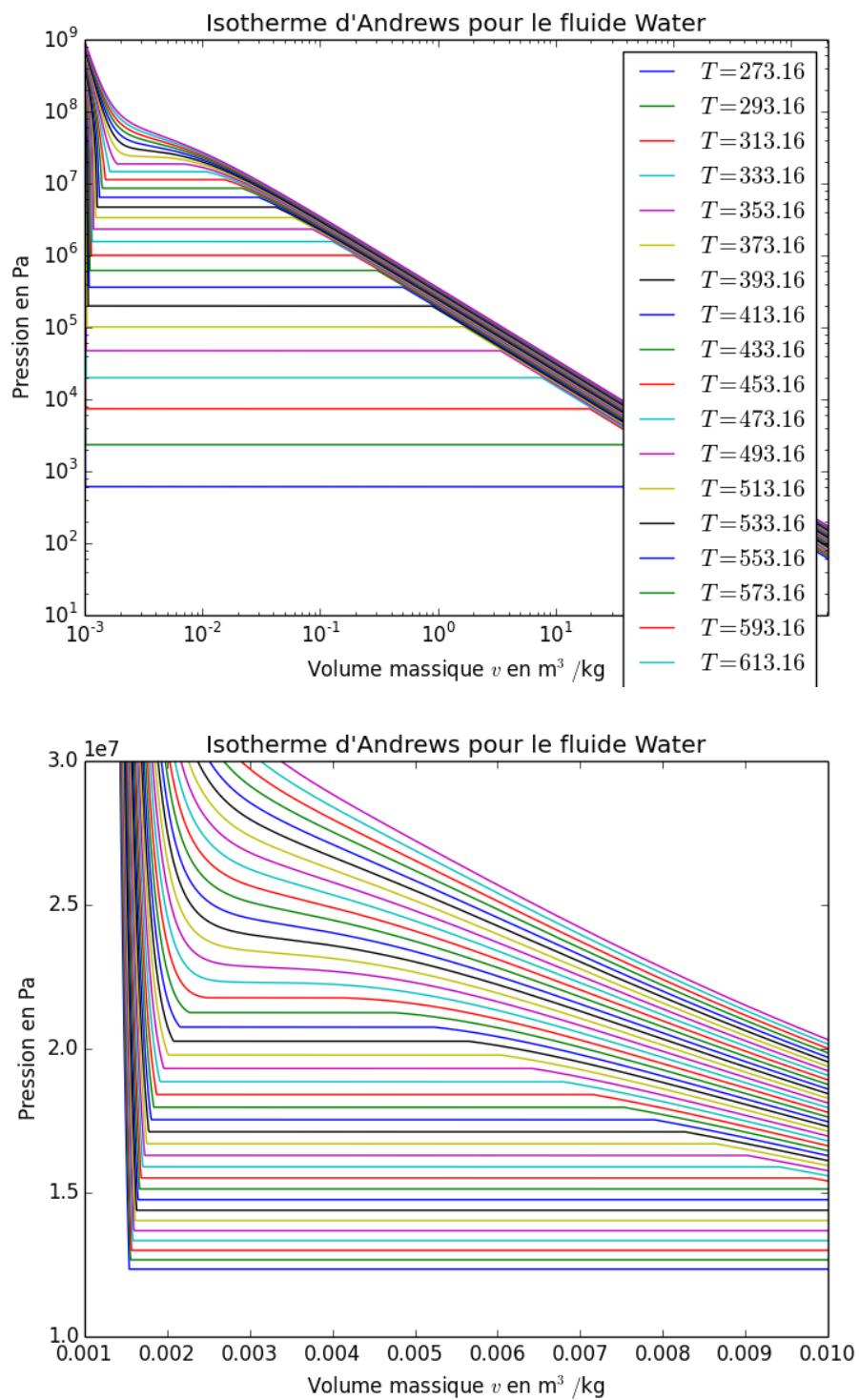


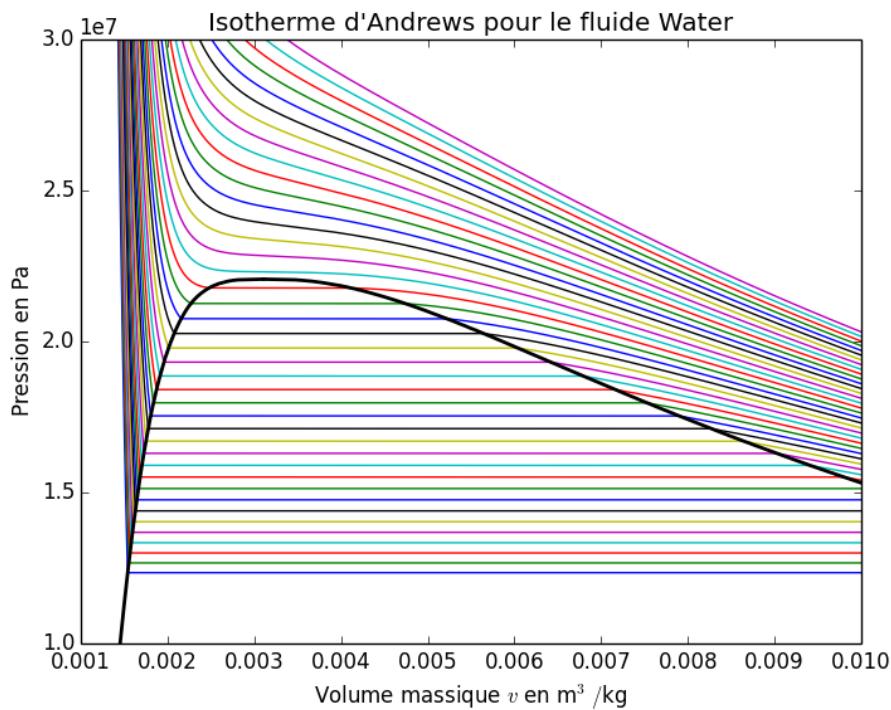
Exemple III.4

## T2 Isothermes d'Andrews via CoolProp

```
1 import numpy as np          # Les outils mathématiques
2 import CoolProp.CoolProp as CP # Les outils thermodynamiques
3 import matplotlib.pyplot as plt # Les outils graphiques
4
5 def isothermes_d_andrews(fluide,dico={}):
6     """ Dessine les isothermes d'Andrews pour le fluide demandé avec des
7     choix par défaut qui peuvent être "overridden" en spécifiant ceux à
8     changer dans le dictionnaire 'dico'. Les options disponibles sont:
9     * 'vmin' et 'vmax' pour définir les limites des échantillonnages en volume
10    massique. Par défaut vtripleL et vtripleG * 10
11    * 'Prange' pour l'intervalle de pression affiché
12    * 'T': une liste des températures pour lesquelles il faut tracer
13    l'isotherme.
14    * 'titre': le titre à donner au graphique.
15    * 'fichier': le nom du fichier dans lequel enregistrer la figure.
16    * 'logx': Booléen indiquant si on veut un axe logarithmique en abscisse
17    * 'logy': Booléen indiquant si on veut un axe logarithmique en ordonnée
18    * 'legend': Booléen indiquant si on veut rajouter les légendes
19    * 'saturation': Booléen indiquant si on veut rajouter la courbe de
20    saturation au tracé (défaut à False)
21 """
22 Pcritique = CP.PropsSI(fluide,'pcrit') # Pression
23 Tcritique = CP.PropsSI(fluide,'Tcrit') # et température critique
24 Ptriple = CP.PropsSI(fluide,'ptriple') # Pression
25 Ttriple = CP.PropsSI(fluide,'Ttriple') # et température au point triple
26 # On récupère les volumes massiques via les 'densités' (ie masses
27 # volumiques) données par CoolProp
28 vtripleL = 1/CP.PropsSI('D','P',Ptriple,'Q',0,fluide)
29 vtripleG = 1/CP.PropsSI('D','P',Ptriple,'Q',1,fluide)
30 vcritique= 1/CP.PropsSI('D','P',Pcritique,'T',Tcritique,fluide)
31 # L'ensemble des valeurs par défaut.
32 DEFAUTS = {'vmin':vtripleL, 'vmax':vtripleG*10,
33             'Prange': None,
34             'T': np.arange(Ttriple,Tcritique*1.2,20),
35             'titre': "Isotherme d'Andrews pour le fluide {}".format(fluide),
36             'fichier': 'PNG/T2_reseau_d_isothermes_coolprop_{}.png'.format(fluide),
37             'logx': True, 'logy': True, 'legend': True,
38             'saturation': False}
39 DEFAUTS.update(dico)      # Mise à jour des valeurs par défaut via 'dico'
40 # L'échantillonnage sera différent
41 if DEFAUTS['logx']:      # si l'axe est logarithmique
42     v=np.logspace(np.log10(DEFAUTS['vmin']),np.log10(DEFAUTS['vmax']),1000)
43 else:                     # ou simplement linéaire
44     v=np.linspace(DEFAUTS['vmin'],DEFAUTS['vmax'],1000)
45 for Ti in DEFAUTS['T']:   # Tracé des différentes isothermes
46     P = CP.PropsSI('P','T',Ti,'D',1/v,fluide)
47     plt.plot(v,P,label='$T={}$'.format(Ti))
48 if DEFAUTS['saturation']: # Tracé de la courbe de saturation
49     P_sat= np.linspace(Ptriple,Pcritique,1000)
```

```
50         v_eb    = 1/CP.PropsSI('D','P',P_sat,'Q',0,fluide)
51         v_rosee= 1/CP.PropsSI('D','P',P_sat,'Q',1,fluide)
52         plt.plot(v_eb,P_sat,'k',linewidth=2.0)
53         plt.plot(v_rosee,P_sat,'k',linewidth=2.0)
54     if DEFAUTS['Prange']: plt.ylim(DEFAUTS['Prange']) # Intervalle vertical
55     plt.xlim((DEFAUTS['vmin'],DEFAUTS['vmax']))      # Intervalle horizontal
56     if DEFAUTS['logx']: plt.xscale('log')             # Echelle log en x
57     if DEFAUTS['logy']: plt.yscale('log')             # Echelle log en y
58     if DEFAUTS['legend']: plt.legend()                # Rajout des légendes
59     plt.xlabel('Volume massique $v$ en m$^3/kg$')   # Légende en abscisse
60     plt.ylabel('Pression en Pa')                     # Légende en ordonnée
61     plt.title(DEFAUTS['titre'])                      # Titre
62     plt.savefig(DEFAUTS['fichier'])                  # Enregistrement
63     plt.clf()                                       # Nettoyage
64
65 # Le fluide à étudier (à choisir parmi ceux donnés par CP.FluidsList())
66 fluide = 'Water'
67
68 # Le diagramme "par défaut"
69 isothermes_d_andrews(fluide)
70
71 # Les valeurs suivantes ont été choisies suite à l'observation du diagramme
72 # par défaut. Il faudra certainement changer les valeurs si vous modifiez le
73 # fluide
74 dico = {'Prange':(1e7,3e7),
75         'fichier':'PNG/T2_reseau_d_isothermes_coolprop_{}_lin.png'.format(fluide),
76         'logx':False, 'logy': False,
77         'vmin': 1e-3, 'vmax':1e-2,
78         'T': [600 + i*2 for i in range(40)],
79         'legend': False}
80 isothermes_d_andrews(fluide,dico)
81
82 # Le même en rajoutant la courbe de saturation
83 dico['saturation'] = True
84 dico['fichier'] = 'PNG/T2_reseau_d_isothermes_coolprop_{}_lin_sat.png'.format(fluide)
85 isothermes_d_andrews(fluide,dico)
```





Exemple III.5

## T2 Diagramme ( $P, v$ ) via CoolProp

```

1 import numpy as np          # Les outils mathématiques
2 import CoolProp.CoolProp as CP # Les outils thermodynamiques
3 import matplotlib.pyplot as plt # Les outils graphiques
4
5 """
6 Fabrication d'un diagramme ( $P, v$ ) avec CoolProp, ce qui n'est pas possible
7 en natif car ils travaillent en masse volumique et non en volume massique par
8 défaut.
9 """
10
11 def diagramme_Pv(fluide,dico={}):
12     """ Dessine le diagramme Pv pour le fluide demandé avec des
13     choix par défaut qui peuvent être "overridden" en spécifiant ceux à
14     changer dans le dictionnaire 'dico'. Les options disponibles sont:
15     * 'vmin' et 'vmax' pour définir les limites des échantillonnages en volume
16     massique. Par défaut utripleL et utripleG * 10
17     * 'Prange' pour l'intervalle de pression affiché
18     * 'T': une liste des températures pour lesquelles il faut tracer
19     l'isotherme. (défaut à None)
20     * 'x': une liste des titres en vapeur pour lesquels il faut tracer la
21     courbe isotitre (défaut à np.linspace(0,1,11))
22     * 'titre': le titre (textuel, hein...) à donner au graphique.
23     * 'fichier': le nom du fichier dans lequel enregistrer la figure.
24     * 'logx': Booléen indiquant si on veut un axe logarithmique en abscisse
25     * 'logy': Booléen indiquant si on veut un axe logarithmique en ordonnée

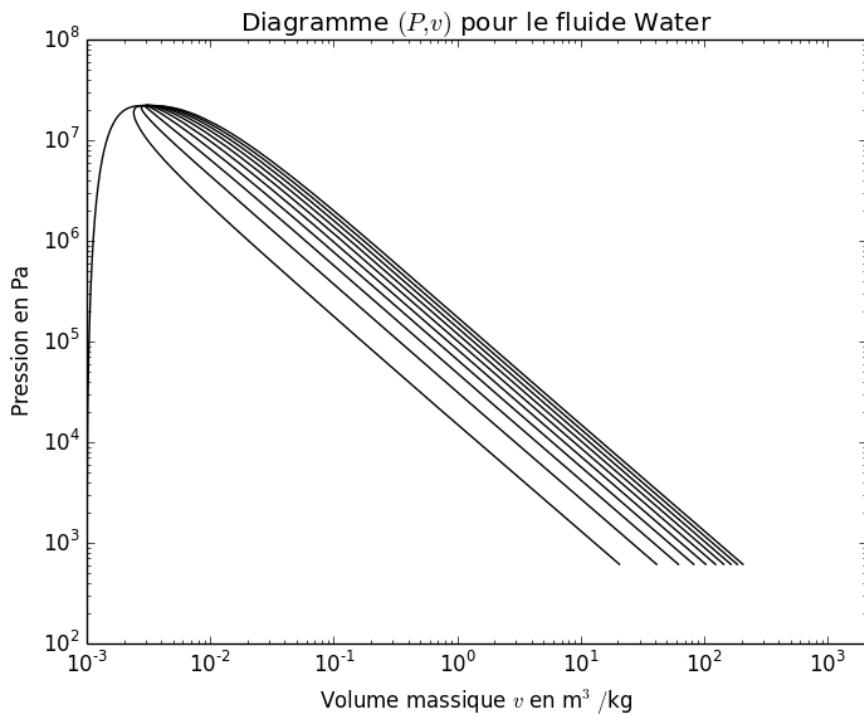
```

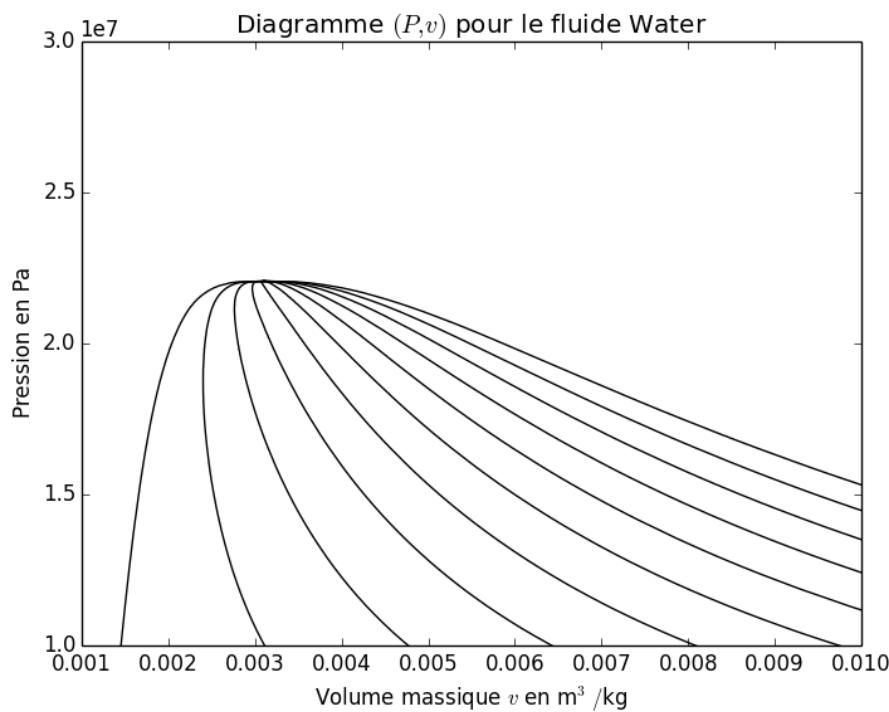
```

26      * 'legend': Booléen indiquant si on veut rajouter les légendes
27      * 'saturation': Booléen indiquant si on veut rajouter la courbe de
28      saturation au tracé (défaut à True)
29      """
30
31      Pcritique = CP.PropsSI(fluide, 'pcrit')    # Pression
32      Tcritique = CP.PropsSI(fluide, 'Tcrit')    # et température critique
33      Ptriple = CP.PropsSI(fluide, 'ptriple')   # Pression
34      Ttriple = CP.PropsSI(fluide, 'Ttriple')   # et température au point triple
35      # On récupère les volumes massiques via les 'densités' (ie masses
36      # volumiques) données par CoolProp
37      vtripleL = 1/CP.PropsSI('D', 'P', Ptriple, 'Q', 0, fluide)
38      vtripleG = 1/CP.PropsSI('D', 'P', Ptriple, 'Q', 1, fluide)
39      vcritique= 1/CP.PropsSI('D', 'P', Pcritique, 'T', Tcritique, fluide)
40      P_sat= np.linspace(Ptriple,Pcritique,1000)
41      # L'ensemble des valeurs par défaut.
42      DEFAUTS = { 'vmin':vtripleL, 'vmax':vtripleG*10,
43                  'Prange': None,
44                  'T': None, 'x': np.linspace(0,1,11),
45                  'titre': "Diagramme $(P,v)$ pour le fluide {}".format(fluide),
46                  'fichier': 'PNG/T2_diagramme_Pv_coolprop_{}.png'.format(fluide),
47                  'logx': True, 'logy': True, 'legend': False,
48                  'saturation': False}
49      DEFAUTS.update(dico)          # Mise à jour des valeurs par défaut via 'dico'
50      # L'échantillonnage sera différent
51      if DEFAUTS['logx']:         # si l'axe est logarithmique
52          v=np.logspace(np.log10(DEFAUTS['vmin']),np.log10(DEFAUTS['vmax']),1000)
53      else:                      # ou simplement linéaire
54          v=np.linspace(DEFAUTS['vmin'],DEFAUTS['vmax'],1000)
55      if DEFAUTS['T'] != None:
56          for Ti in DEFAUTS['T']:  # Tracé des différentes isothermes
57              P = CP.PropsSI('P', 'T', Ti, 'D', 1/v, fluide)
58              plt.plot(v,P,label='$T={}{}'.format(Ti))
59      if DEFAUTS['x'] != None:
60          for xi in DEFAUTS['x']:  # Tracé des courbes isotrope
61              vxi = 1/CP.PropsSI('D', 'P', P_sat, 'Q', xi, fluide)
62              plt.plot(vxi,P_sat,'k',label='$x={}{}'.format(xi))
63      if DEFAUTS['saturation']: # Tracé de la courbe de saturation
64          v_eb   = 1/CP.PropsSI('D', 'P', P_sat, 'Q', 0, fluide)
65          v_rosee= 1/CP.PropsSI('D', 'P', P_sat, 'Q', 1, fluide)
66          plt.plot(v_eb,P_sat,'k',linewidth=4.0)
67          plt.plot(v_rosee,P_sat,'k',linewidth=4.0)
68      if DEFAUTS['Prange']: plt.ylim(DEFAUTS['Prange']) # Intervalle vertical
69      plt.xlim((DEFAUTS['vmin'],DEFAUTS['vmax']))        # Intervalle horizontal
70      if DEFAUTS['logx']: plt.xscale('log')               # Echelle log en x
71      if DEFAUTS['logy']: plt.yscale('log')               # Echelle log en y
72      if DEFAUTS['legend']: plt.legend()                 # Rajout des légendes
73      plt.xlabel('Volume massique $v$ en m$^3/$kg')     # Légende en abscisse
74      plt.ylabel('Pression en Pa')                      # Légende en ordonnée
75      plt.title(DEFAUTS['titre'])                       # Titre

```

```
75     plt.savefig(DEFAUTS['fichier'])                      # Enregistrement
76     plt.clf()                                              # Nettoyage
77
78 # Le fluide à étudier (à choisir parmi ceux donnés par CP.FluidsList())
79 fluide = 'Water'
80
81 # Le diagramme "par défaut"
82 diagramme_Pv(fluide)
83
84 # Les valeurs suivantes ont été choisies suite à l'observation du diagramme
85 # par défaut. Il faudra certainement changer les valeurs si vous modifiez le
86 # fluide
87 dico = {'Prange':(1e7,3e7),
88         'fichier':'PNG/T2_diagramme_Pv_coolprop_{}_.lin.png'.format(fluide),
89         'logx':False, 'logy': False,
90         'vmin': 1e-3, 'vmax':1e-2}
91 diagramme_Pv(fluide,dico)
```

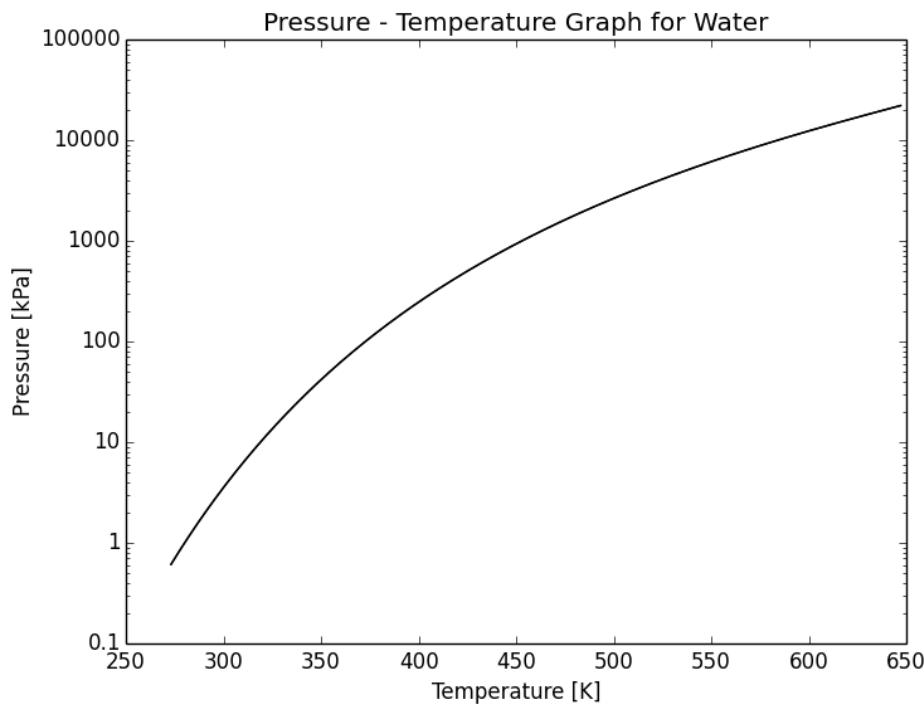




Exemple III.6

## T2 Diagramme (P, T) via CoolProp

```
1  """
2  Fabrication simple d'un diagramme PT avec CoolProp. Malheureusement, on n'a
3  accès qu'à la partie "fluide" du diagramme donc l'équilibre liquide/vapeur,
4  mais c'est déjà pas mal.
5  """
6
7  import matplotlib.pyplot as plt
8  from CoolProp.Plots import PropsPlot
9
10 fluid = 'Water'          # Le fluide choisi (plus dans CoolProp.CoolProp.Fluids)
11 pt_plot = PropsPlot(fluid, 'PT') # Le type de diagramme
12 plt.yscale('log')          # Échelle logarithmique en pression
13 pt_plot._draw_graph()      # Dessin du graphe obligatoire avant sauvegarde
14 plt.savefig('PNG/T2_diagramme_PT_coolprop_{}.png'.format(fluid))
```



Exemple III.7

### T5 Comparaison des isentropiques pour un gaz réel et pour un gaz parfait

Exemple III.8

### T6 Cycle de Carnot: gaz réel et gaz parfait

```
1     """
2     Le but de ce script est de tracer un cycle de Carnot (deux isothermes et deux
3     isentropiques) pour un gaz (a priori l'air, mais on peut le modifier) à la
4     fois à partir de données réelles (via CoolProp) et dans la modélisation d'un
5     gaz parfait (le coefficient gamma de Laplace étant aussi obtenu à partir de
6     CoolProp). On suppose que le cycle se fait de la manière suivante (on
7     numérote les point "à l'informaticienne" de 0 à 3):
8     * Compression isotherme de (P0,TF) à (P1,TF)
9     * Compression isentropique de (P1,TF) à (Pmax,TC)
10    * détente isotherme de (Pmax,TC) à (P3,TC)
11    * détente isentropique de (P3,TC) à (P0,TF)
12    À noter que si jamais Pmax est trop faible, commence par une détente isotherme
13    à TF et on continuera par une compression isotherme à TC
14    """
15
16    import numpy as np          # Les outils mathématiques
17    import CoolProp.CoolProp as CP  # Les outils thermodynamiques
18    import matplotlib.pyplot as plt # Les outils graphiques
19
20    # Les valeurs réglables
21    R = 8.314      # Constante des gaz parfaits
```

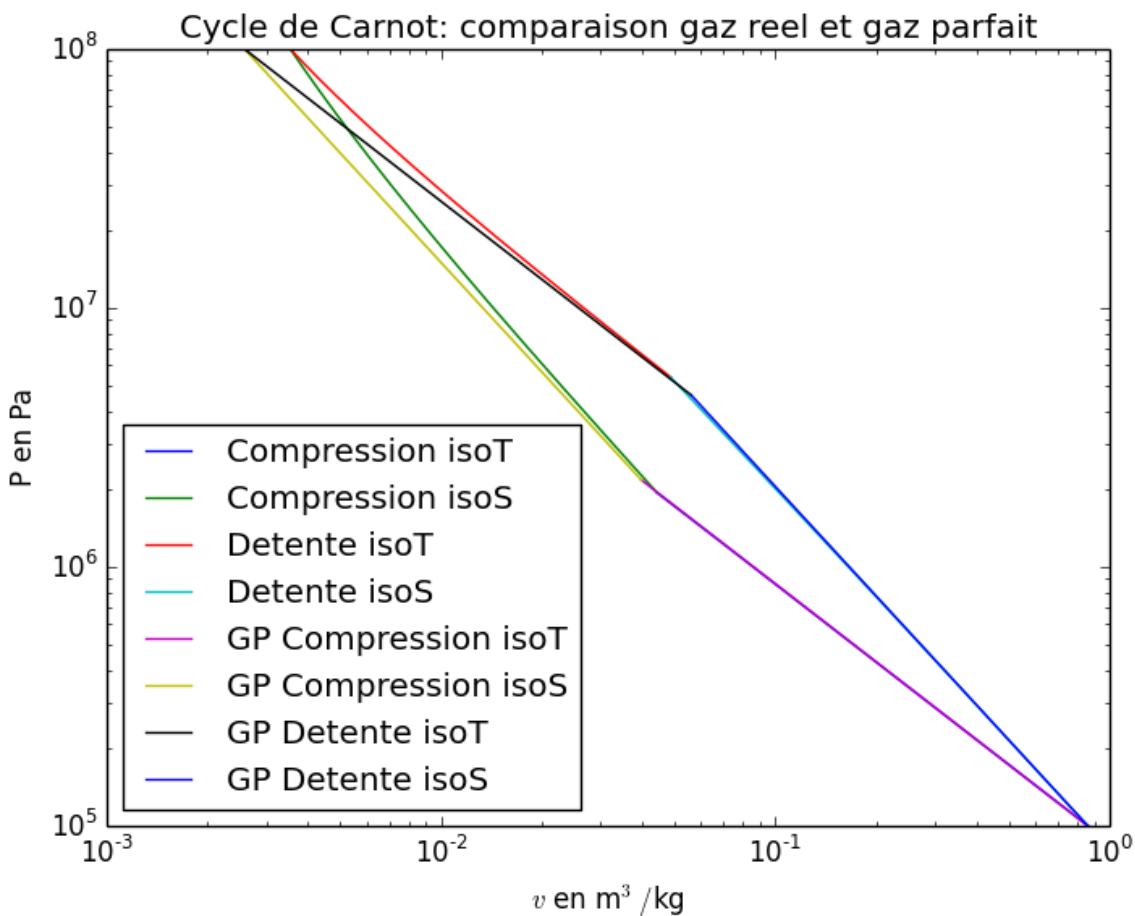
```
22 gaz = 'Air'      # Type de gaz
23 TF  = 300        # Température de l'isotherme "froide"
24 TC  = 900        # Température de l'isotherme "chaude"
25 P0  = 1e5         # Pression (a priori) la plus faible (et de départ)
26 Pmax= 1e8         # Pression (a priori) la plus importante
27
28 # Calcul du coefficient de Laplace (en le supposant inchangé sur tout le cycle)
29 cP  = CP.PropsSI('C','T',TF,'P',P0,gaz)
30 cV  = CP.PropsSI('O','T',TF,'P',P0,gaz)
31 gamma = cP/cV
32 # et de la masse molaire (NB: pour eux le "SI" de M, c'est le kg/kmol...)
33 M   = CP.PropsSI(gaz,'molemass')*1e-3
34
35 # Un peu de feedback pour l'utilisateur:
36 print('Gaz choisi:',gaz)
37 print('Masse molaire:',M,'kg/mol')
38 print('gamma:',gamma)
39
40 # Calcul des positions intermédiaires réelles
41 S3 = CP.PropsSI('S','T',TF,'P',P0,gaz)      # Entropie de la détente isentropique
42 S1 = CP.PropsSI('S','T',TC,'P',Pmax,gaz)    # Entropie de la compression isentropique
43 P1 = CP.PropsSI('P','T',TF,'S',S1,gaz)       # Pression au point 1
44 P3 = CP.PropsSI('P','T',TC,'S',S3,gaz)       # Pression au point 3
45
46 # On échantillonne à présent les pressions sur les différents chemins...
47 nb_points = 1000
48 P01 = np.linspace(P0,P1,nb_points)
49 P12 = np.linspace(P1,Pmax,nb_points)
50 P23 = np.linspace(Pmax,P3,nb_points)
51 P30 = np.linspace(P3,P0,nb_points)
52
53 # ...pour calculer les volumes massiques correspondants (comme d'habitude,
54 # CoolProp fournit la masse volumique (densité "D") et non le volume massique
55 # donc il faut passer à l'inverse).
56 v01 = 1/CP.PropsSI('D','P',P01,'T',TF,gaz) # Compression isotherme
57 v12 = 1/CP.PropsSI('D','P',P12,'S',S1,gaz) # Compression isentropique
58 v23 = 1/CP.PropsSI('D','P',P23,'T',TC,gaz) # Détente isotherme
59 v30 = 1/CP.PropsSI('D','P',P30,'S',S3,gaz) # Détente isentropique
60
61 def infos_point(nb,P,T,v):
62     print('Infos pour le point {0}: T={1}K, v={3} m^3/kg, P={2} bar'.format(nb,T,round(P,
63
64 def travail(L_P,L_v):
65     W = 0
66     for (P,v) in zip(L_P,L_v):
67         W -= np.trapz(P,v)
68     return W
69
70 # On donne du feedback:
```

```
71 print('Cas réel: ')
72 infos_point(0,P0,TF,v01[0])
73 infos_point(1,P1,TF,v12[0])
74 infos_point(2,Pmax,TC,v23[0])
75 infos_point(3,P3,TC,v30[0])
76 W = travail([P01,P12,P23,P30],[v01,v12,v23,v30])
77 print('Travail total sur le cycle:',round(W/1e3,2),'kJ/kg')
78
79 # Reste à représenter le tout
80 plt.plot(v01,P01,label='Compression isoT')
81 plt.plot(v12,P12,label='Compression isoS')
82 plt.plot(v23,P23,label='Detente isoT')
83 plt.plot(v30,P30,label='Detente isoS')
84 plt.legend()
85 plt.yscale('log')
86 plt.xscale('log')
87 #plt.show()
88
89 # Maintenant, faisons quelques calculs théoriques.
90 # On peut calculer les pressions  $P_1$  et  $P_3$  grâce aux relations de Laplace sur
91 # les deux isentropiques
92 P3 = P0 * (TF/TC)**(gamma/(1-gamma))
93 P1 = Pmax*(TC/TF)**(gamma/(1-gamma))
94
95 # On échantillonne à présent les pressions sur les différents chemins...
96 nb_points = 1000
97 P01 = np.linspace(P0,P1,nb_points)
98 P12 = np.linspace(P1,Pmax,nb_points)
99 P23 = np.linspace(Pmax,P3,nb_points)
100 P30 = np.linspace(P3,P0,nb_points)
101
102 # ... pour calculer les volumes massiques à l'aide des lois de Laplace ou des
103 # gaz parfaits.
104 v01 = R*TF/(M*P01)
105 v12 = v01[-1] * (P1/P12)**(1/gamma)
106 v23 = R*TC/(M*P23)
107 v30 = v23[-1] * (P3/P30)**(1/gamma)
108
109 # Reste à représenter le tout
110 plt.plot(v01,P01,label='GP Compression isoT')
111 plt.plot(v12,P12,label='GP Compression isoS')
112 plt.plot(v23,P23,label='GP Detente isoT')
113 plt.plot(v30,P30,label='GP Detente isoS')
114
115 plt.title('Cycle de Carnot: comparaison gaz reel et gaz parfait')
116 plt.xlabel('$v$ en $m^3/kg$')
117 plt.ylabel('P en Pa')
118 plt.legend(loc='lower left')
119 plt.savefig('PNG/T6_cycle_de_carnot_reel_et_GP_{}.png'.format(gaz))
```

```

120
121 # On donne du feedback:
122 print('Cas Gaz Parfait:')
123 infos_point(0,P0,TF,v01[0])
124 infos_point(1,P1,TF,v12[0])
125 infos_point(2,Pmax,TC,v23[0])
126 infos_point(3,P3,TC,v30[0])
127 W = travail([P01,P12,P23,P30],[v01,v12,v23,v30])
128 print('Travail total sur le cycle:',round(W/1e3,2),'kJ/kg')

```

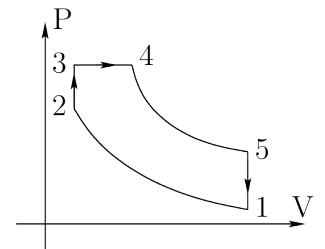


— Exemple III.9 —

## T6 Résolution d'un exercice sur un cycle Diesel

L'exercice à résoudre est le suivant.

Un cycle Diesel classique est composé de deux adiabatiques, d'une isobare et d'une isochore ; dans les moteurs Diesel actuels (dits à double combustion), le cycle de l'air est représenté ci-contre ( $P_1 = 1,0$  bar,  $T_1 = 293$  K,  $P_{\max} = 65$  bar et  $T_{\max} = T_4 = 2173$  K). On suppose que l'air est un gaz parfait diatomique, de masse molaire  $M = 29 \text{ g.mol}^{-1}$ . On donne le taux de compression:  $a = V_1/V_2 = 19$ . Toutes les transformations seront supposées mécaniquement réversibles.



1. Représenter le cycle Diesel classique dans un diagramme (P, V).
2. Déterminer les types de transformations pour le cycle Diesel à double combustion.

3. Exprimer, en fonction de  $\gamma$  et des températures  $T_1$ ,  $T_2$ ,  $T_3$ ,  $T_4$  et  $T_5$  des points correspondants sur le diagramme, le rendement  $r$  du moteur Diesel à double combustion.
4. Déterminer numériquement  $T_2$ ,  $T_3$  et  $T_5$ . En déduire la valeur de  $r$ .
5. Quel est, en kJ, le transfert thermique reçue par 1 kg d'air au cours de l'évolution entre les points 2 et 4 ? Quelle est le transfert thermique reçue entre les points 5 et 1 ? En déduire le travail fourni par 1 kg d'air au milieu extérieur au cours d'un cycle.

Essayons d'utiliser CoolProp pour le résoudre à la fois de manière théorique et en suivant les propriétés de l'air en tant que gaz réel.

```
1  """
2  Le but de ce script est la résolution d'un exercice concernant un cycle Diesel
3  à double combustion (cf fichier py4phys.pdf pour le détail de l'exercice)
4  """
5
6  import numpy as np          # Les outils mathématiques
7  import CoolProp.CoolProp as CP # Les outils thermodynamiques
8  import matplotlib.pyplot as plt # Les outils graphiques
9
10 P,T,s,v = {},{},{},{}
11
12 # Les données de l'énoncé
13 R = 8.314      # Constante des gaz parfaits
14 gaz = 'Air'    # Type de gaz
15 T[1]= 293
16 P[1]= 1e5
17 P[3]= 65e5
18 P[4]= P[3]
19 T[4]= 2173
20 a = 19
21
22 # Calcul du coefficient de Laplace (en le supposant inchangé sur tout le cycle)
23 cP = CP.PropsSI('C','T',T[1], 'P',P[1],gaz)
24 cV = CP.PropsSI('O','T',T[1], 'P',P[1],gaz)
25 gamma = cP/cV
26 # et de la masse molaire (NB: pour eux le "SI" de M, c'est le kg/kmol...)
27 M = CP.PropsSI(gaz,'molemass')*1e-3
28
29 # Un peu de feedback pour l'utilisateur:
30 print('Gaz choisi:',gaz)
31 print('Masse molaire:',M,'kg/mol')
32 print('gamma:',gamma)
33
34 # Fonction dichotomique utile
35 def find_P_from_v_s(v,s,Pstart,Pstop,eps=1e-6):
36     """ Retrouver P par dichotomie à partir du volume massique et de
37     l'entropie massique. """
38     v1 = 1/CP.PropsSI('D','P',Pstart,'S',s,gaz)
39     v2 = 1/CP.PropsSI('D','P',Pstop,'S',s,gaz)
40     while abs(v2-v1)/v > eps:
```

```

41         Pm = (Pstart+Pstop)/2.0
42         vm = 1/CP.PropsSI('D','P',Pm,'S',s,gaz)
43         if (vm-v)*(v2-v) < 0: Pstart,v1 = Pm,vm
44         else: Pstop,v2 = Pm,vm
45     return Pm
46
47
48
49 # Calculs des points intermédiaires
50 v[1]= 1/CP.PropsSI('D','P',P[1],'T',T[1],gaz) # Inverse de densité
51 v[2]= v[1]/a # Facteur de compression
52 s[1]= CP.PropsSI('S','T',T[1],'P',P[1],gaz) # Entropie correspondante
53 s[2]= s[1] # Isentropique
54 P[2]= find_P_from_v_s(v[2],s[2],P[1],P[3]) # Récupération pression (faut ruser)
55 T[2]= CP.PropsSI('T','P',P[2],'S',s[2],gaz) # Température correspondante
56 v[3]= v[2] # Isochore
57 T[3]= CP.PropsSI('T','P',P[3],'D',1/v[3],gaz) # dont on connaît la pression finale
58 v[4]= 1/CP.PropsSI('D','P',P[4],'T',T[4],gaz) # Isobare à T connue
59 s[4]= CP.PropsSI('S','P',P[4],'T',T[4],gaz) # et calcul de l'entropie correspondante
60 s[5]= s[4] # Isentropique
61 v[5]= v[1] # Dernière isochore
62 P[5]= find_P_from_v_s(v[5],s[5],P[1],P[3]) # Ruse sioux pour la pression
63 T[5]= CP.PropsSI('T','P',P[5],'D',1/v[5],gaz) # et obtention de la T correspondante
64
65
66 # On échantillonne à présent les pressions sur les différents chemins...
67 nb_points = 100
68 P12 = np.linspace(P[1],P[2],nb_points)
69 P23 = np.array([P[2],P[3]])
70 P34 = np.array([P[3],P[4]])
71 P45 = np.linspace(P[4],P[5],nb_points)
72 P51 = np.array([P[5],P[1]])
73
74 # ...pour calculer les volumes massiques correspondants (comme d'habitude,
75 # CoolProp fournit la masse volumique (densité "D") et non le volume massique
76 # donc il faut passer à l'inverse).
77 v12 = 1/CP.PropsSI('D','P',P12,'S',s[1],gaz) # Compression isentropique
78 v23 = [v[2],v[3]] # Compression isochore
79 v34 = [v[3],v[4]] # Déntente isobare
80 v45 = 1/CP.PropsSI('D','P',P45,'S',s[4],gaz) # Déntente isentropique
81 v51 = [v[5],v[1]] # Déntente isochore
82
83 def infos_point(nb,P,T,v):
84     print('Infos pour le point {0}: T={1} K, v={3} m^3/kg, P={2} bar'.format(nb,round(T,
85
86 def travail(L_P,L_v):
87     W = 0
88     for (P,v) in zip(L_P,L_v):
89         W -= np.trapz(P,v)

```

```
90     return W
91
92 def calcule_Delta(f,i,j):
93     """ Calcule la variation de la fonction d'état f (à choisir entre 'U',
94     'H', 'S' ou 'G') entre les points i et j à partir des valeurs (supposées
95     connues) de température et de pression. """
96     fi = CP.PropsSI(f,'P',P[i],'T',T[i],gaz)
97     fj = CP.PropsSI(f,'P',P[j],'T',T[j],gaz)
98     return fj-fi
99
100
101 # On donne du feedback:
102 print('Cas réel:')
103 for i in range(1,6):
104     infos_point(i,P[i],T[i],v[i])
105 L_P = [P12,P23,P34,P45,P51]
106 L_v = [v12,v23,v34,v45,v51]
107 W = travail(L_P,L_v)           # Calcul du travail sur tout le cycle
108 print('Travail total sur le cycle:',round(W/1e3,2),'kJ/kg')
109 Q23 = calcule_Delta('U',2,3)    # Q pour une isochore
110 Q34 = calcule_Delta('H',3,4)    # Q pour une isobare
111 print('Transfert thermique reçu sur 2->3:',round(Q23/1e3,2),'kJ/kg')
112 print('Transfert thermique reçu sur 3->4:',round(Q34/1e3,2),'kJ/kg')
113 # Calcul du rendement
114 print('Rendement total: r= ',-W/(Q23+Q34))
115
116 # Reste à représenter le tout
117 for i in range(len(L_v)):
118     plt.plot(L_v[i],L_P[i]/1e5,label='{}$\\rightarrow${}',reel'.format(i+1,(i+1)%5+1))
119
120
121 # Maintenant, faisons quelques calculs théoriques.
122 # D'abord les volumes massiques des deux premiers points:
123 v[1] = R*T[1]/(M*P[1])
124 v[2] = v[1]/a
125 # Pressions et Température en 2 s'obtiennent via la loi de Laplace
126 P[2] = P[1] * (v[1]/v[2])**gamma
127 T[2] = T[1] * (v[1]/v[2])**gamma_1
128 # Ensuite, on fait une isochore dont on connaît la pression d'arrivée
129 v[3] = v[2]
130 T[3] = M*P[3]*v[3]/R
131 # Après, c'est au tour de l'isobare dont on connaît la température finale
132 v[4] = R*T[4]/(M*P[4])
133 # Finalement, on refait une isentropique jusqu'à atteindre v[1]
134 v[5] = v[1]
135 P[5] = P[4] * (v[4]/v[5])**gamma
136 T[5] = T[4] * (v[4]/v[5])**gamma_1
137
138 # On échantillonne à présent les pressions sur les différents chemins...
```

```
139 nb_points = 100
140 P12 = np.linspace(P[1],P[2],nb_points)
141 P23 = np.array([P[2],P[3]])
142 P34 = np.array([P[3],P[4]])
143 P45 = np.linspace(P[4],P[5],nb_points)
144 P51 = np.array([P[5],P[1]])

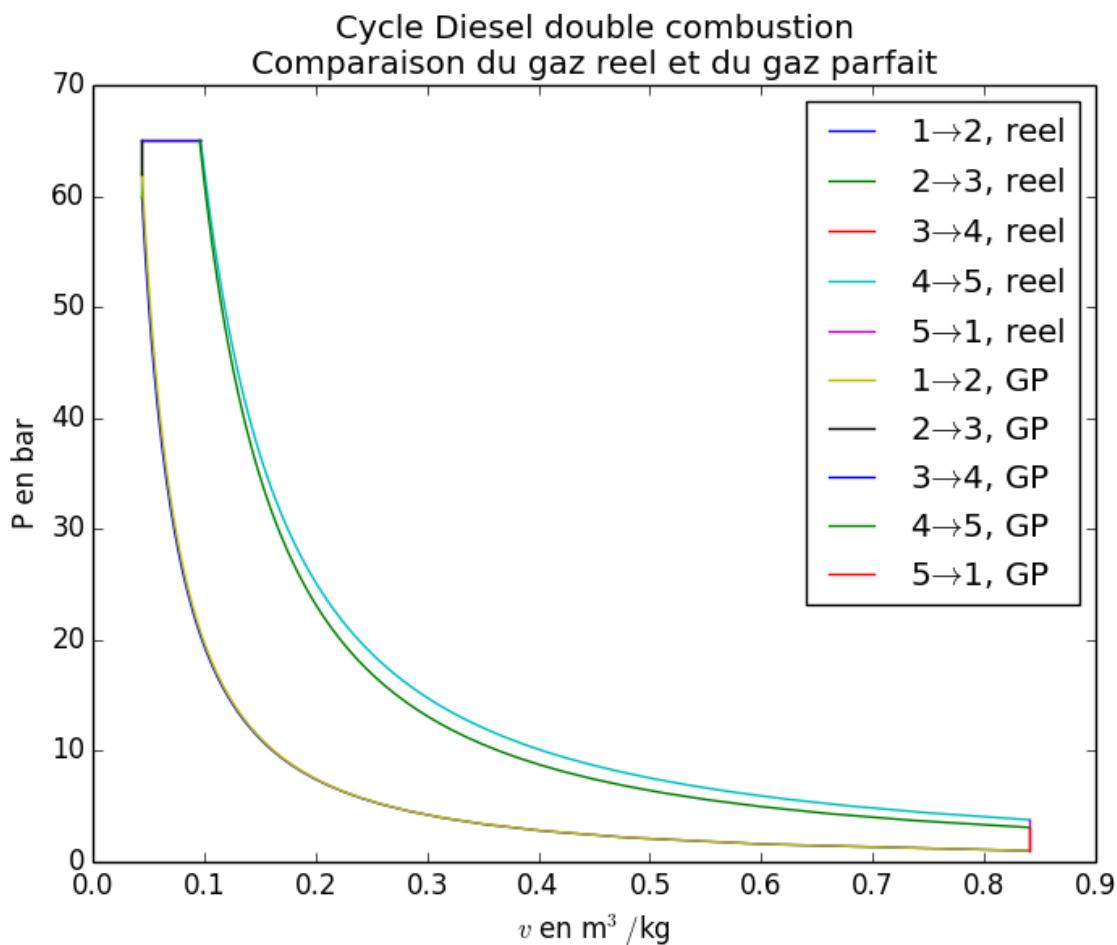
145
146 # ...pour calculer les volumes massiques correspondants
147 v12 = v[1]*(P[1]/P12)**(1/gamma)                      # Compression isentropique
148 v23 = [v[2],v[3]]                                       # Compression isochore
149 v34 = [v[3],v[4]]                                       # Dé-tente isobare
150 v45 = v[4]*(P[4]/P45)**(1/gamma)                      # Dé-tente isentropique
151 v51 = [v[5],v[1]]                                       # Dé-tente isochore

152
153 # On donne du feedback:
154 print('Cas Gaz parfait:')
155 for i in range(1,6):
156     infos_point(i,P[i],T[i],v[i])
157 L_P = [P12,P23,P34,P45,P51]
158 L_v = [v12,v23,v34,v45,v51]
159 W = travail(L_P,L_v)          # Calcul du travail total
160 print('Travail total sur le cycle:',round(W/1e3,2),'kJ/kg')
161 Q23 = cV*(T[3]-T[2])          # Q sur une isochore
162 Q34 = cP*(T[4]-T[3])          # Q sur une isobare
163 print('Transfert thermique reçu sur 2->3:',round(Q23/1e3,2),'kJ/kg')
164 print('Transfert thermique reçu sur 3->4:',round(Q34/1e3,2),'kJ/kg')

165
166 print('Rendement total: r=',-W/(Q23+Q34))

167
168 # Reste à représenter le tout
169 for i in range(len(L_v)):
170     plt.plot(L_v[i],L_P[i]/1e5,label='{}$\backslash$to${}$_{}$, GP'.format(i+1,(i+1)%5+1))
171 plt.legend()
172 plt.xlabel('$v$ en $m^3/kg$')
173 plt.ylabel('P en bar')
174 plt.title("""Cycle Diesel double combustion
175 Comparaison du gaz réel et du gaz parfait""")

176
177 plt.savefig('PNG/T6_resolution_cycle_diesel.png')
```



Et voici la sortie à l'écran du programme:

```
Gaz choisi: Air
Masse molaire: 0.02896546 kg/mol
gamma: 1.40194967423
Cas réel:
Infos pour le point 1: T=293.0 K, v=0.8407 m^3/kg, P=1.0 bar
Infos pour le point 2: T=904.7 K, v=0.0442 m^3/kg, P=59.9 bar
Infos pour le point 3: T=981.5 K, v=0.0442 m^3/kg, P=65.0 bar
Infos pour le point 4: T=2173.0 K, v=0.097 m^3/kg, P=65.0 bar
Infos pour le point 5: T=1110.2 K, v=0.8407 m^3/kg, P=3.8 bar
Travail total sur le cycle: -865.17 kJ/kg
Transfert thermique reçu sur 2->3: 64.87 kJ/kg
Transfert thermique reçu sur 3->4: 1446.82 kJ/kg
Rendement total: r= 0.572323227974
Cas Gaz parfait:
Infos pour le point 1: T=293.0 K, v=0.841 m^3/kg, P=1.0 bar
Infos pour le point 2: T=956.9 K, v=0.0443 m^3/kg, P=62.1 bar
Infos pour le point 3: T=1002.4 K, v=0.0443 m^3/kg, P=65.0 bar
Infos pour le point 4: T=2173.0 K, v=0.096 m^3/kg, P=65.0 bar
Infos pour le point 5: T=908.1 K, v=0.841 m^3/kg, P=3.1 bar
Travail total sur le cycle: -763.96 kJ/kg
Transfert thermique reçu sur 2->3: 32.64 kJ/kg
Transfert thermique reçu sur 3->4: 1177.79 kJ/kg
```

Rendement total:  $r = 0.631150489954$

Exemple III.10

## T6 Diagramme ( $P, h$ ) via CoolProp

```
1 """
2 Fabrication d'un diagramme ( $P, h$ ) avec les iso-choses adéquates.
3 """
4
5 import numpy as np           # Les outils mathématiques
6 import CoolProp.CoolProp as CP # Les outils thermodynamiques
7 import CoolProp.Plots as CPP  # Les outils thermographiques
8 import matplotlib.pyplot as plt # Les outils graphiques
9
10
11 print(CP.FluidsList())      # Pour regarder les fluides disponibles
12 fluide= 'Water'             # Le choix du fluide
13 Plogscale = True            # Axe en pression logarithmique ?
14 iso_T = True                # Veut-on des isothermes ?
15 iso_X = True                # et les isotitres ?
16 iso_S = True                # et les isentropiques ?
17 iso_V = True                # et les isochores ?
18
19 # Données pour les isothermes
20 dT = 20                      # Incrément de températures
21 Ttriple = CP.PropsSI(fluide,'Ttriple') # Valeur de la température au point triple
22 Tcrit = CP.PropsSI(fluide,'Tcrit')       # et au point critique
23 Tmin = int(Ttriple/10)*10 + 10          # Par défaut, on par près du point triple
24 val_T = np.arange(Tmin,1.5*Tcrit,dT)     # et on dépasse un peu le point critique
25 T_to_show = list(range(2,len(val_T),2))  # Sélection des  $T$  à afficher (mettre None pour
26
27 # Données pour les isotitres
28 val_x = np.linspace(0.1,0.9,9)           # Les valeurs des isotitres
29
30 # Données pour les isentropiques
31 ds = 0.5e3
32 striple_x0 = CP.PropsSI('S','Q',0,'T',Ttriple,fluide) # Entropie triple à gauche
33 striple_x1 = CP.PropsSI('S','Q',1,'T',Ttriple,fluide) # Entropie triple à droite
34 val_s = np.arange(striple_x0,striple_x1*1.2,ds)       # Valeurs à tracer
35 s_to_show = list(range(2,len(val_s),2))                 # et à afficher
36
37 # Données pour les isochores (réparties de manière logarithmique par défaut)
38 vcrit = 1/CP.PropsSI(fluide,'rhocrit')                  # Volume massique critique
39 exp_min = int(np.floor(np.log10(vcrit)))+1              # Puissance de 10 proche
40 vtriple_x1 = 1/CP.PropsSI('D','Q',1,'T',Ttriple,fluide) # Point triple à droite
41 exp_max = int(np.ceil(np.log10(vtriple_x1)))-1          # Puissance de 10 proche
42 # Les valeurs à prendre
43 val_v = [a * 10**b for a in [1,2,5] for b in range(exp_min,exp_max+1)]
```

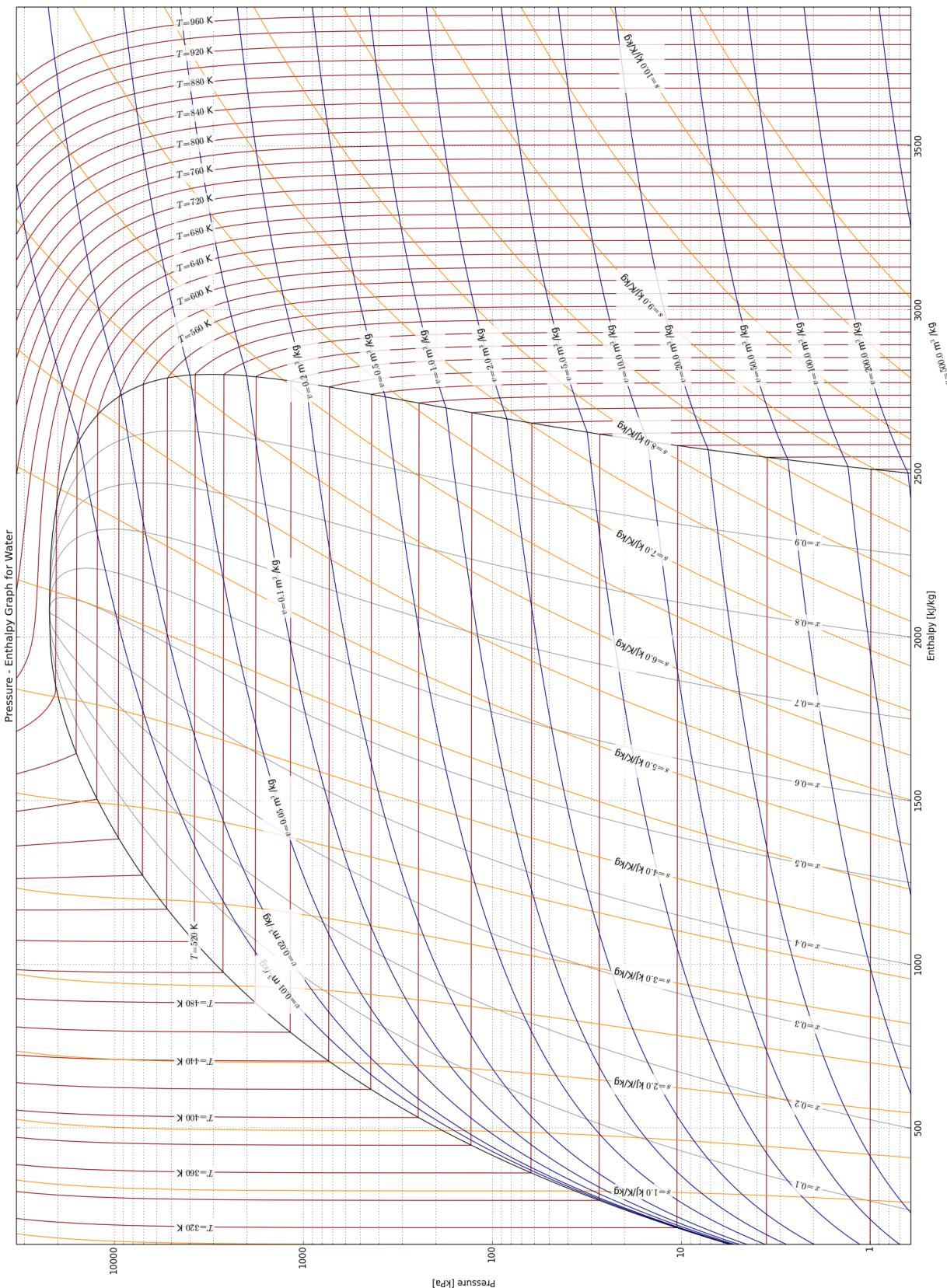
```

44 v_to_show = None                                # On les affiche toutes.
45
46 # Quelques constantes
47 UNITS = {'T': 'K', 'Q': '', 'S': 'kJ/K/kg', 'V': 'm$^3$/kg'}
48 LABEL = {'T': 'T', 'Q': 'x', 'S': 's', 'V': 'v'}
49 COLOR_MAP = {'T': 'Darkred',
50             'P': 'DarkCyan',
51             'H': 'DarkGreen',
52             'V': 'DarkBlue',
53             'S': 'DarkOrange',
54             'Q': 'black'}
55
56 # On prépare un format pour impression sur A3 ou presque (dimensions en pouces)
57 plt.figure(figsize=(30,21))
58
59 def place_label(x,y,label,indice=None,cotan=False,color='k'):
60     """ Routine qui se débrouille pour mettre un label semi-transparent au
61     niveau de la courbe données par ses coordonnées x et y. Si on sait que le
62     label sera presque vertical avec possibilité de dépasser 90°, on peut
63     utiliser cotan=True pour corriger (considération purement esthétique).
64     'indice' correspond à la position dans les tableaux x et y où devra
65     s'afficher le label demandé. """
66     print(x[0],y[0],label) # un peu de feedback pour savoir ce qu'on calcule
67     N = len(x)//2          # Emplacement par défaut
68     if indice: N=indice    # sauf si l'utilisateur impose la valeur
69     xi,xf = plt.xlim()      # Les limites en x du graphe
70     yi,yf = plt.ylim()      # Pareil en y
71     Xsize = xf - xi        # La largeur
72     # Pour la hauteur et la pente, cela dépend si les ordonnées sont en repère
73     # logarithmique ou non.
74     if Plogscale:
75         Ysize = np.log10(yf) - np.log10(yi)
76         a = (np.log10(y[N+1])-np.log10(y[N-1]))/(x[N+1]-x[N-1]) * Xsize/Ysize
77     else:
78         Ysize = yf - yi
79         a = (y[N+1]-y[N-1])/(x[N+1]-x[N-1]) * Xsize/Ysize
80     bbox = plt.gca().get_window_extent() # Récupération de la taille de la figure
81     a *= bbox.height / bbox.width       # Correction de la pente avec la taille
82     rot = np.degrees(np.arctan(a))       # Calcul de l'angle de rotation
83     if cotan:                           # Si on dépasse la verticale
84         rot = 90 - np.degrees(np.arctan(1/a))
85     t = plt.text(x[N],y[N],label,      # On met le texte au bon endroit
86                 ha='center',va='center',color=color,rotation = rot) # Avec la bonne rotation
87     # On se débrouille pour que la "boîte" d'écriture soit semi-transparente
88     t.set_bbox(dict(facecolor='w',edgecolor='None',alpha=0.8))
89
90 def fait_isolignes(type,valeurs,position=None,nb_points=1000,to_show=None,round_nb = 0 )
91     """ S'occupe du calcul et du tracé des isolignes. """
92     if not(to_show):                      # Valeurs par défauts:

```

```
93     to_show = list(range(len(valeurs))) # toutes !
94     Pmin,Pmax = plt.ylim()           # On regarde les
95     Hmin,Hmax = plt.xlim()          # limites du graphique
96     # Il y a un bug au niveau des unités du graphe et des points de
97     # représentation, d'où les 1e3 qui traînent un peu partout
98     # Par défaut, l'échantillonnage en  $P$  est linéaire
99     val_P = np.linspace(Pmin*1e3,Pmax*1e3,nb_points)
100    # Sinon, on se met en échelle log. (1e3 -> 3)
101    if Plogscale: val_P = 3+np.logspace(np.log10(Pmin),np.log10(Pmax),nb_points)
102    # Cas où les lignes ne vont pas sur tout l'éventail des pression, on
103    # échantillonne en températures (car on ne peut pas directement
104    # échantillonner en enthalpie  $h$ )
105    Tmin = Ttriple
106    Tmax = CP.PropsSI('T','P',Pmax,'H',Hmax,fluide)
107    val_T = np.linspace(Tmin,Tmax,nb_points)
108    # Pour chacune des valeurs demandées,
109    for val,i in zip(valeurs,range(len(valeurs))):
110        if type == 'V': # Cas particulier des volumes massiques: échantillonnage
111            val_P = CP.PropsSI('P','T',val_T,'D',1/val,fluide) # en température
112            val_H = CP.PropsSI('H','T',val_T,'D',1/val,fluide) # et non en  $P$ 
113        else:           # Sinon, on utilise l'éventail des pression
114            val_H = CP.PropsSI('H','P',val_P,type,val,fluide)
115        if type == 'S': val /= 1e3 # Pour mettre en kJ/K/kg
116        if round_nb >0 : val = str(round(val,round_nb)) # Pour faire joli
117        else: val = str(int(round(val)))                 # là aussi...
118        label = '${}={} ${}'.format(LABEL[type],val,UNITS[type])
119        plt.plot(val_H,val_P,color=COLOR_MAP[type])      # Affichage courbe
120        if i in to_show: # Ainsi que du label s'il fait partie de la liste
121            place_label(val_H,val_P,label,int(position*nb_points))
122
123    # Le programme proprement dit commence ici.
124
125    ph_plot = CPP.PropsPlot(fluide,'Ph')    # On demande gentiment le plot de base
126    if Plogscale: plt.yscale('log')          # Passage en log( $P$ )
127
128    if iso_x: # Les lignes isotites sont un peu spéciales, donc ont leur code propre
129        ph_plot.draw_isolines('Q',val_x)    # Tracé des lignes isotites
130        # Récupération de la liste des isotites.
131        isoQ = CPP.Plots.IsoLines(fluide,'Ph','Q').get_isolines(val_x)
132        for line in isoQ:                  # Rajout des label
133            label = line['label'] + line['unit']
134            x,y = line['x'],line['y']
135            place_label(x,y,label,indice=len(x)//20)
136
137    # Ici, on fait toutes les autres isolines (le boulot a été fait plus haut)
138    if iso_T: fait_isolines('T',val_T,position=0.8,to_show=T_to_show)
139    if iso_s: fait_isolines('S',val_s,position=0.3,to_show=s_to_show,round_nb=3)
140    if iso_v: fait_isolines('V',val_v,position=0.25,to_show=v_to_show,round_nb=3)
141
```

```
142 plt.grid(which='both') # Rajout de la grille
143 ph_plot._draw_graph() # On oblige le dessin avant la sauvegarde
144 plt.savefig('PNG/T6_diagramme_Ph_coolprop_{}.png'.format(fluide))
```



# Py4Phys IV

## Bloc Induction

Exemple IV.1

### I1 Tracé de lignes de champ magnétique

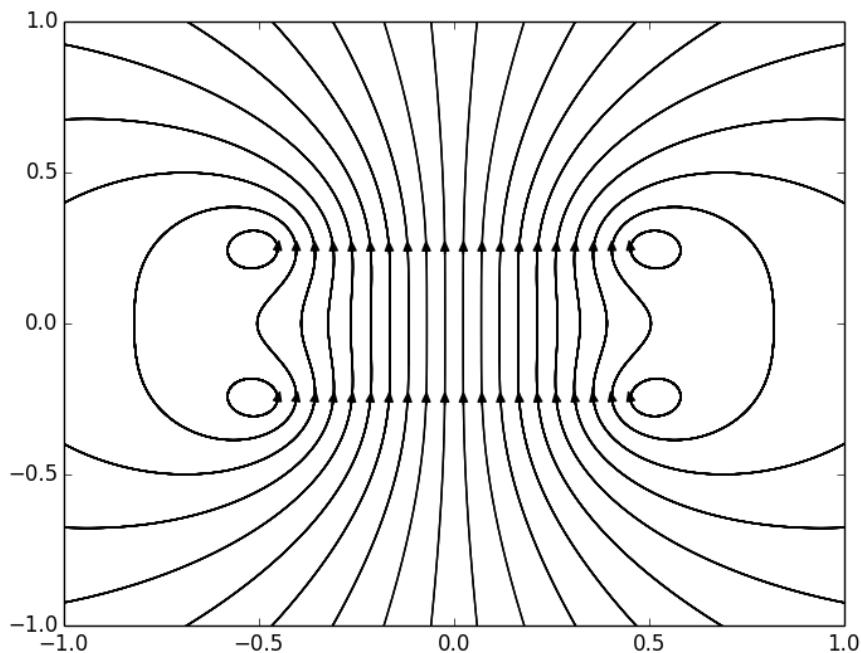
```
1  """
2  Created on Fri May 31 10:33:06 2014
3
4  @author: Sylvain Condamin, adapté d'un code de Thierry Pré.
5
6  On prend mu0/(2*Pi) =1
7  Ce programme permet de tracer les lignes de champ magnétique d'un ensemble de
8  lignes orthogonales au plan, de spires dont l'axe est dans le plan, et de
9  moments magnétiques dans le plan. Il faut commencer par créer un objet
10 Diagramme, puis ajouter les différents objets magnétiques à l'aide des
11 méthodes dédiées, puis utiliser la méthode draw pour tracer le diagramme.
12 Cf. description des méthodes, et exemple final pour plus d'information.
13 """
14
15 import numpy as np
16 import scipy.special as sp
17 import matplotlib.pyplot as pl
18 from scipy.integrate import odeint
19
20
21
22 class Diagramme(object):
23     """ La classe Diagramme s'occupe de tout ce qui concerne les tracés."""
24     def __init__(self,size=1,title="",numpoints=1000,k=1):
25         """
26             Routine d'initialisation d'un diagramme de lignes de champ:
27             * size est la taille de la fenêtre utilisée. (x et y varient de -size à +size)
28             * numpoints est le nombre de points utilisé pour le tracé.
29             * Le coefficient k permet de régler la longueur des lignes
30                 tracées. Le défaut k=1 assure normalement que les lignes bouclent
31                 correctement, mais il est possible de le réduire pour améliorer le temps
32                 de calcul.
```

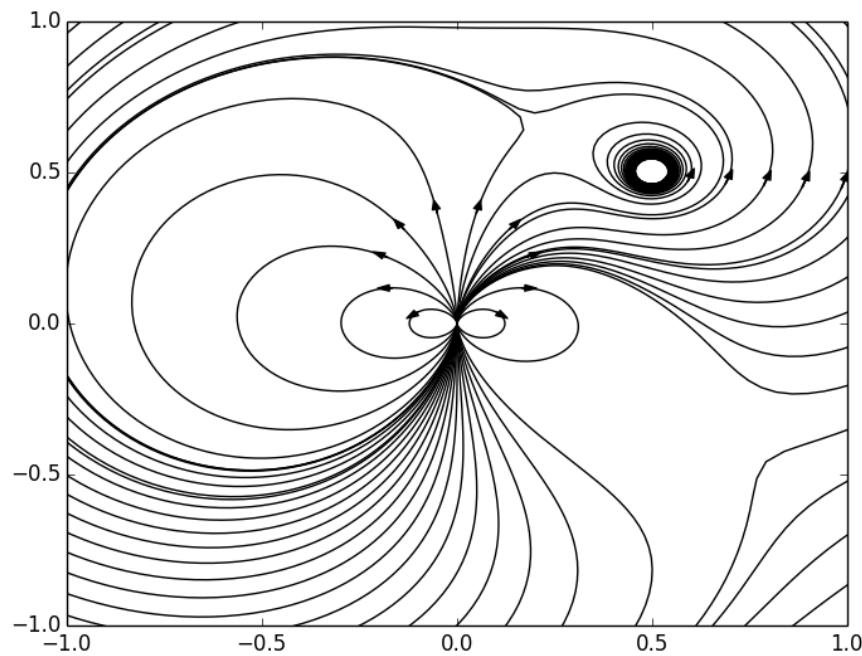
```
33      """
34      self.size = float(size)
35      self.title=title
36      self.numpoints=numpoints
37      self.objects=[]
38      self.startpoints=[]
39      self.maxint=0 # Ceci correspond au temps d'intégration à utiliser pour
39                  # le tracé des courbes.
40      self.k=k
41
42
43  def addLine(self,x,y,I=1,points=20):
44      """
45      Permet d'ajouter une ligne de courant infinie, orthogonale au
46      plan du tracé. Par défaut cela ajoute des points de départ de
47      lignes de champ.
48      """
49      self.objects.append(Line(x,y,I))
50      for i in range(points):
51          self.startpoints.append([x+2*(i+1)*self.size/points,y])
52      self.maxint = pow(pow(2*np.pi,3)*abs(I)*self.size**2+self.maxint**3,1/3)
53
54  def addSpire(self,x,y,a,I=1,theta=0,points=20):
55      """
56      Permet de définir une spire: x et y désignent le centre, a le
57      rayon, I l'intensité, et theta l'angle que fait l'axe
58      de la spire par rapport à la verticale. Par défaut cela ajoute
59      des points de départ de lignes de champ.
60      """
61      self.objects.append(Spire(x,y,a,I,theta))
62      for i in range(points):
63          l = -a+2*a*float(1+i)/(1+points)
64          self.startpoints.append([x+l*np.cos(theta),y+l*np.sin(theta)])
65      self.maxint += pow(100*abs(I)*np.pi*a*a+self.maxint**3,1/3)
66
67  def addDipole(self,x,y,m,theta=0,points=20):
68      """
69      Permet d'ajouter un dipôle magnétique m, à la position x et
70      y. theta désigne l'angle que fait la spire par rapport à la
71      verticale. Par défaut cela ajoute des points de départ de
72      lignes de champ.
73      """
74      self.objects.append(Dipole(x,y,m,theta))
75      s = self.size/5
76      x0 = x-s*np.sin(theta)
77      y0 = y + s*np.cos(theta)
78      for i in range(points):
79          phi = theta+2*(i+1)*np.pi/(1+points)
80          self.startpoints.append([x0+s*np.sin(phi),y0-s*np.cos(phi)])
81      self.maxint += pow(1000*abs(m)+self.maxint**3,1./3.)
```

```
82
83     def addStartPoints(self,x,y):
84         """
85             Permet d'ajouter des points de départ de lignes de champ.
86         """
87         self.startpoints.append([x,y])
88
89     def B(self,P):
90         Bx = 0
91         By = 0
92         for magnet in self.objects:
93             Bx += magnet.B(P)[0]
94             By += magnet.B(P)[1]
95         return [Bx,By]
96
97     def draw(self,file=None):
98         """
99             Trace l'ensemble des lignes de champ passant par les points de départ
100            stockés dans Startpoints. Si un nom de fichier est donné, enregistre
101            la figure dans le fichier mais n'affiche rien à l'écran
102        """
103
104     def fun(P,t):
105         B = self.B(P)
106         Bx = B[0]
107         By = B[1]
108         B = np.sqrt(Bx*Bx+By*By)
109         return [Bx/pow(B,4./3.),By/pow(B,4./3.)]
110
111     t = np.linspace(0,self.k*self.maxint,self.numpoints/2)
112     t2 = - t
113     for P0 in self.startpoints:
114         sol = odeint(fun,P0,t)
115         x = sol[:,0]
116         y = sol[:,1]
117         pl.plot(x,y,'-',color='k')
118         sol = odeint(fun,P0,t2)
119         x = sol[1:,0]
120         y = sol[1:,1]
121         pl.plot(x,y,'-',color='k')
122         pl.arrow(x[1],y[1],x[0]-x[1],y[0]-y[1],color='k')
123     pl.title(self.title)
124     pl.xlim([-self.size,self.size])
125     pl.ylim([-self.size,self.size])
126     if file:
127         pl.savefig(file)
128         pl.close()
129     else:
130         pl.show()
```

```
131
132 class MagneticObjects(object):
133     def __init__(self):
134         pass
135
136     def B(self,P):
137         raise NotImplementedError()
138
139 class Line(MagneticObjects):
140     def __init__(self,x,y,I):
141         self.x0 = x
142         self.y0 = y
143         self.I = I
144
145     def B(self,P):
146         x = P[0]- self.x0
147         y = P[1]- self.y0
148         return([-self.I*y/(x*x+y*y),self.I*x/(x*x+y*y)])
149
150 class Spire(MagneticObjects):
151     def __init__(self,x,y,a,I,theta):
152         self.x0 = x
153         self.y0 = y
154         self.a = a
155         self.I = I
156         self.theta = theta
157
158     def B(self,P):
159         a = self.a
160         theta = self.theta
161         x = P[0] - self.x0
162         y = P[1] - self.y0
163         r = x*np.cos(theta)+y*np.sin(theta)
164         z = -x*np.sin(theta)+y*np.cos(theta) # On se ramène à des
165         # coordonnées cylindriques par rapport à la spire. Pour la
166         # suite des calculs, voir l'article de T.Pré
167         # http://www.udppc.asso.fr/bupdoc/textes/fichierjoint/918/0918D119.zip
168         k= 4.*abs(r)*a/((a+abs(r))**2+z**2)
169         Kk=sp.ellipk(k)
170         Ek=sp.ellipe(k)
171         Br=self.I*(z/r)/np.sqrt((a+abs(r))**2+z**2)*(-Kk+(a**2+r**2+z**2)/((a-abs(r))**2+z**2))
172         Bz=(self.I/np.sqrt((a+abs(r))**2+z**2))*(Kk+((a**2-r**2-z**2)/((a-abs(r))**2+z**2))
173         return([Br*np.cos(theta)-Bz*np.sin(theta),Br*np.sin(theta)+Bz*np.cos(theta)])
174
175 class Dipole(MagneticObjects):
176     def __init__(self,x,y,m,theta):
177         self.x0 = x
178         self.y0 = y
179         self.m = m
```

```
180         self.theta = theta
181
182     def B(self,P):
183         m = self.m
184         theta = self.theta
185         x = P[0] - self.x0
186         y = P[1] - self.y0
187         r = np.sqrt(x*x+y*y)
188         Bx = (m/(2*r**3))*((-x*np.sin(theta)+y*np.cos(theta))*3*(x/r**2)+np.sin(theta))
189         By = (m/(2*r**3))*((-x*np.sin(theta)+y*np.cos(theta))*3*(y/r**2)-np.cos(theta))
190         return [Bx,By]
191
192 # Premier exemple: deux spires en configuration de Helmholtz
193 diag = Diagramme()
194 diag.addSpire(0,0.25,a=0.5,I=1)
195 diag.addSpire(0,-0.25,a=0.5,I=1)
196 diag.draw('PNG/I1_lignes_champ_magnetique_helmholtz.png')
197
198 # Deuxième exemple: un dipôle avec un fil perpendiculaire au plan de la figure.
199 diag = Diagramme()
200 diag.addLine(0.5,0.5,I=1)
201 diag.addDipole(0,0,m=1,points=10)
202 diag.draw('PNG/I1_lignes_champ_magnetique_fil_et_dipole.png')
```





Exemple IV.2

### I2 Champ tournant diphasé

Exemple IV.3

### I2 Champ tournant triphasé

Exemple IV.4

### I4 Couplage inductif de deux circuits