C.R. TP 13 : partie 2

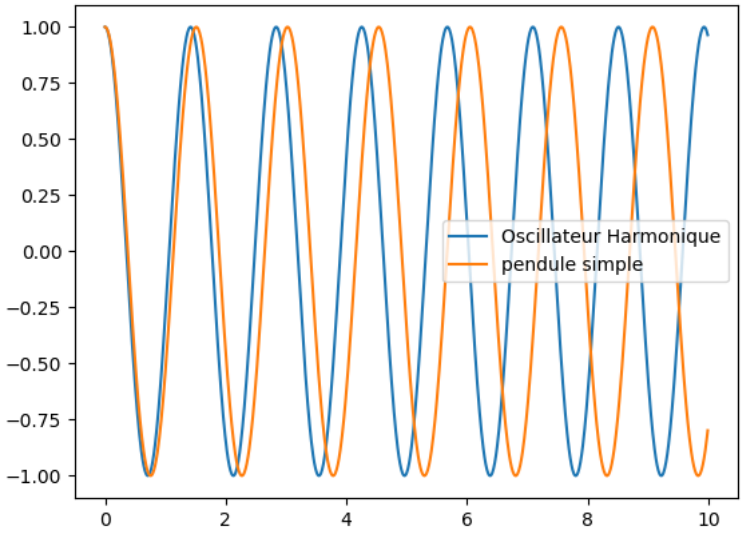
2 Oscillateurs

2.1 Comparaison pendule simple, Oscillateur harmonique

```
from scipy import integrate
pl.figure("comparaison pendule simple O.H.")
pl.clf()
g=9.81
l=0.5
omega0=pl.sqrt(g/l)
def F(Y,t):
    return [Y[1],-omega0**2*Y[0]]
t0=0
Y0=[1,0]# C.I.
t=pl.arange(0,10,0.01)
Y=integrate.odeint(F,Y0,t)

pl.plot(t,Y[:,0],label="Oscillateur Harmonique")
# code pour le pendule
def G(Y,t):
    return [Y[1],-omega0**2*pl.sin(Y[0])]
Z=integrate.odeint(G,Y0,t)
pl.plot(t,Z[:,0],label="pendule simple")
pl.show()
```

Avec le résultat pour un temps de $10s$.



Evidemment le décalage est important avec cette amplitude initiale.

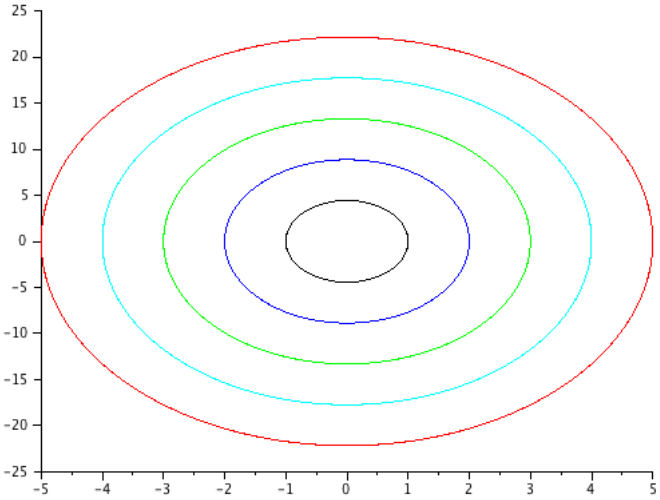
2.2 Portrait de phase

2.2.1 Pour l'O.H.

Pour obtenir un portrait de phase, on doit mettre θ en abscisse et θ' en ordonnée, ce qui s'obtient via un `pl.plot(Y[:,0],Y[:,1])`.

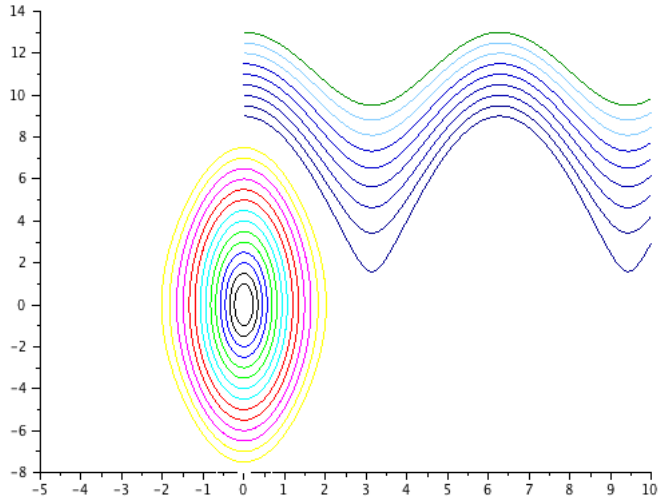
Pour tracer plusieurs courbes correspondant à différentes conditions initiales, une boucle `for` suffit. On détaille le code dans le paragraphe suivant pour le pendule simple.

Ici pour l'O.H. :



2.2.2 Portrait de phase du pendule simple

```
pl.figure("portrait de phase pendule")
pl.clf()
g=9.81
l=0.5
omega0=pl.sqrt(g/l)
def G(Y,t):
    return [Y[1],-omega0**2*pl.sin(Y[0])]
t0=0
t=pl.arange(0,3,0.01)
for yp0 in pl.arange(1,13,0.5):
    Y0=[0,yp0]# vecteur C.I. en bas, vitesse yp0
    Y=integrate.odeint(G,Y0,t)
    pl.plot(Y[:,0],Y[:,1])
 pl.show()
```



Sur le graphique, la *vitesse critique* qui permet de faire un tour complet est évaluée autour de entre 8 et $9\,rad.s^{-1}$.

Pour l'obtenir par le calcul, on considère l'intégrale première proportionnelle à l'énergie :

$E = \frac{1}{2}\theta'(t)^2 - \omega_0^2\cos(\theta(t))$.

Ici, vu la C.I. $\theta(0) = 0$, on a $E = \frac{1}{2}\theta'(0)^2 - \omega_0^2$.

Dans le cas d'un mouvement oscillant, la vitesse angulaire θ' s'annule au moment où le pendule est au sommet de sa trajectoire.

Dans le cas d'un mouvement de révolution complet, θ' ne s'annule plus.

Mathématiquement : $\theta'(t)^2 = \theta(0)^2 - 2\omega_0^2(1 - \cos(\theta(t)))$.

Comme $1 - \cos(\theta(t)) \in [0, 2]$, $\theta'(t)$ ne s'annule plus dès que $\theta(0)^2 > 4\omega_0^2$.

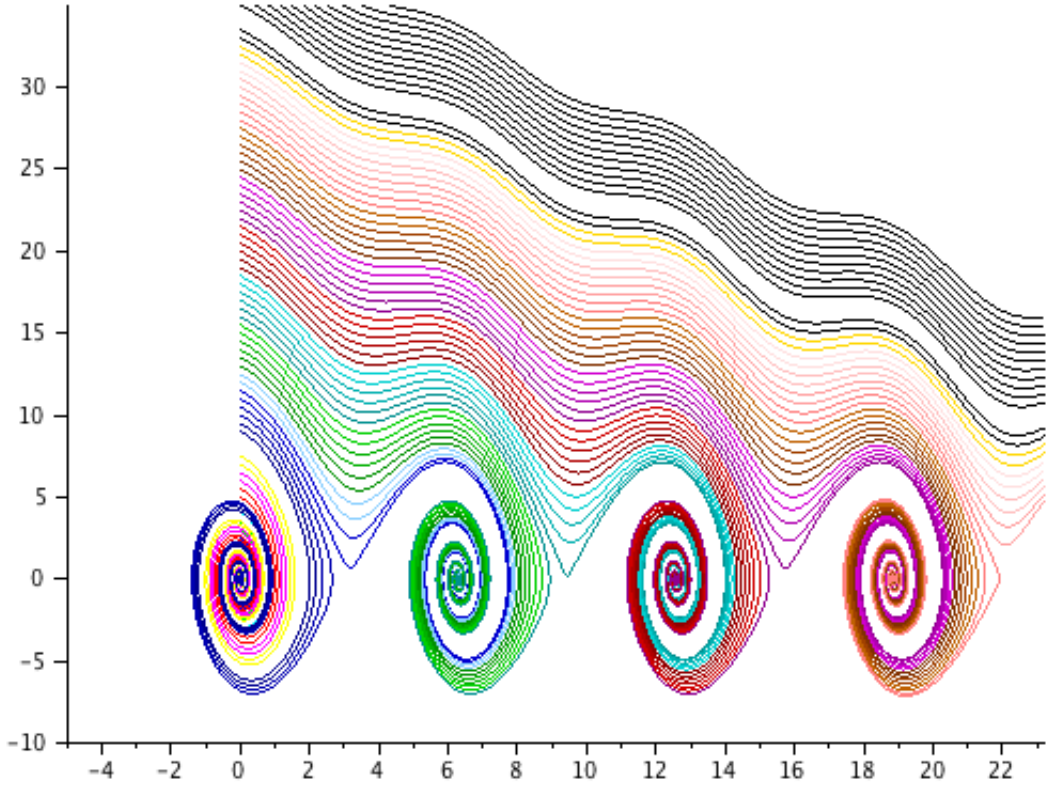
Donc la *vitesse critique* vaut $2\omega_0$. Ici on trouve donc $8.85 rad.s^{-1}$.

Bien sûr, on aurait pu tracer plus de courbes entre 8 et 9 pour préciser cette vitesse.

2.2.3 Pour le pendule simple amorti

Le code est identique à part la déclaration de la fonction.

En revanche, il est intéressant de prendre un temps plus long et davantage de vitesses initiales (plus grandes). Toutes les trajectoires finissent par s'enrouler autour d'un point, mais y mettent plus ou moins de temps suivant la vitesse initiale.



3 Etude d'un tir balistique avec différents frottements fluides

3.1 Le cas où il n'y a pas de frottement du tout : les paraboles et la parabole de sécurité

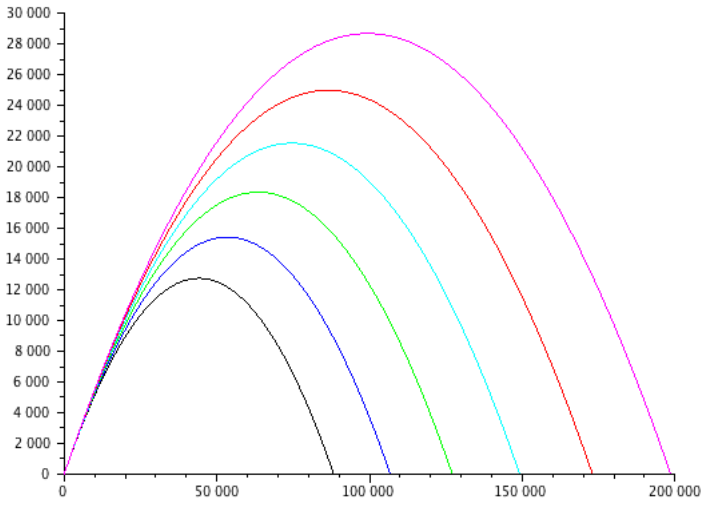
a) Par intégration : $\begin{cases} x(t) = v_0\cos(\alpha)t, \\ z(t) = -\frac{gt^2}{2} + v_0\sin(\alpha)t \end{cases}$

b) **Remarque :** On s'intéresse à la partie de la trajectoire obtenue avant que l'obus ne touche le sol, autrement dit (en supposant le sol plat), pour $z \geq 0$.

On peut donc délimiter l'intervalle de temps qu'on veut tracer en résolvant l'équation $z(t) = 0$ outre la solution évidente $t = 0$ a pour solution $t_f = \frac{2v_0\sin(\alpha)}{g}$.

Ainsi le code :

```
pl.figure("les paraboles")
pl.clf()
alpha=pl.pi/6
g=9.81
for v0 in range(1000,1600,100): # v0 nombre qui parcourt les 6 valeurs..
    tf=2*v0*pl.sin(alpha)/g
    t=pl.arange(0,tf,0.1) # tableau des t
    x=v0*pl.cos(alpha)*t
    z=-g*t*t/2+v0*pl.sin(alpha)*t
    pl.plot(x,z)
```

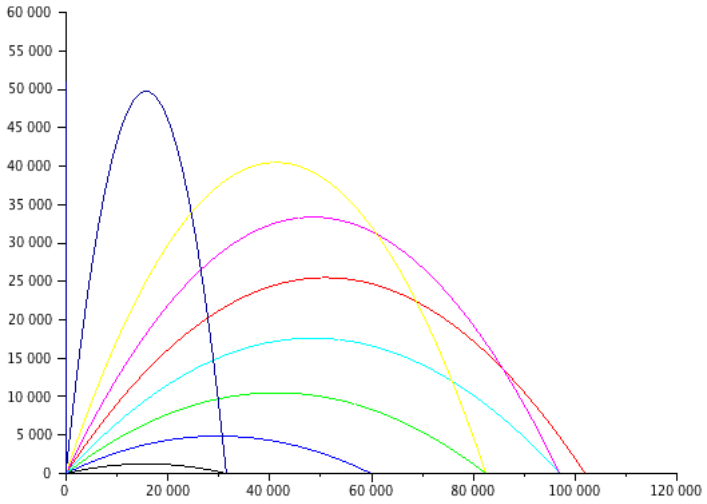


Donne le résultat :

Noter que la portée de nos canons va jusqu'à 200 km.

c) Cette fois v_0 est fixé et l'angle α varie.

```
pl.figure("angle qui varie")
pl.clf()
v0=1000
g=9.81
for i in range(1,11):
    alpha=i*pl.pi/20
    tf=2*v0*pl.sin(alpha)/g
    t=pl.arange(0,tf,0.1)
    x=v0*pl.cos(alpha)*t
    z=-g*t*t/2+v0*pl.sin(alpha)*t
    pl.plot(x,z)
```



Remarque : la norme v_0 du vecteur vitesse étant fixée, toutes les paraboles précédentes sont dans une zone du plan délimitée par ce qu'on appelle une *parabole de sécurité*. C'est cette parabole qui définit la zone « à l'abris des tirs ».

Excursion mathématique : comment calculer cette parabole ?

i) Remarquons d'abord qu'une trajectoire C_α peut aussi s'écrire comme le graphe Γ_a d'une fonction :

$$z = -\frac{g}{2v_0^2}x^2(1+a^2) + ax; \text{ en ayant posé } a = \tan(\alpha),$$

le nombre $a = \tan(\alpha)$ a un sens clair : c'est la pente du vecteur $\vec{v}(0)$.

ii) Prenons un point $M = (x_0, z_0)$ du plan. La C.N.S. sur (x_0, z_0) pour qu'il existe un a tel que (x_0, z_0) soit sur une courbe Γ_a est que le discriminant Δ de l'équation du second degré :

$$a^2(-\frac{g}{2v_0^2})x_0^2 + ax_0 - \frac{g}{2v_0^2}x_0^2 - z_0 = 0,$$

vérifie $\Delta \geq 0$.

Les points (x_0, z_0) tels que $\Delta = 0$ correspondent à la courbe de sécurité cherchée (bord du domaine où $\Delta \geq 0$). Les formules se simplifient : on peut faire tracer la courbe.

(ii) Pour l'équation du second degré d'inconnue a donnée par l'énoncé :

$$a^2(-\frac{g}{2v_0^2})x_0^2 + ax_0 - \frac{g}{2v_0^2}x_0^2 - z_0 = 0,$$

le discriminant Δ de cette équation d'inconnue a s'écrit :

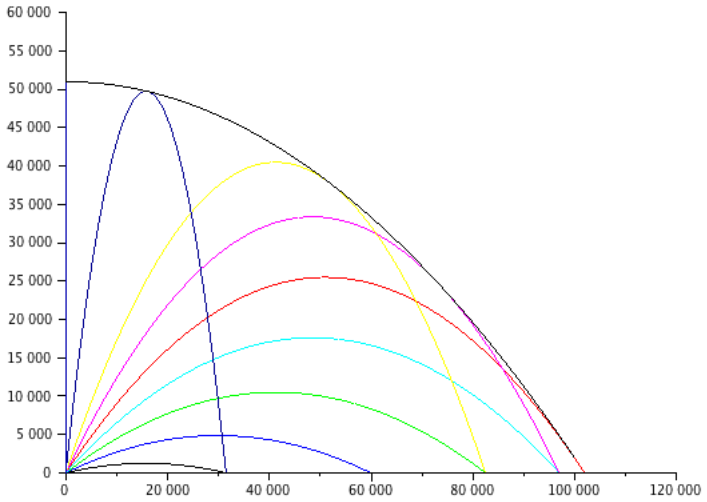
$$\Delta = x_0^2 - (\frac{g}{v_0^2}x_0^2 + 2z_0)\frac{gx_0^2}{v_0^2}.$$

N.B. La constante g/v_0^2 est homogène à l'inverse d'une longueur ce qui aide à comprendre ce calcul. On pose $g/v_0^2 = 1/L$.

Alors $\Delta = 0 \Leftrightarrow z_0 = \frac{L}{2} - \frac{1}{2L}x_0^2$, équation d'une parabole.

On peut alors la tracer en rajoutant le code suivant après la boucle `for`

```
L=v0**2/g
x=pl.arange(0,100000,10) # borne un peu grossière qu'on peut améliorer
z=L/2-x*x/(2*L)
pl.plot(x,z)
```



3.2 Frottement fluide proportionnel à la vitesse

Par intégration des équations pour la vitesse données par l'énoncé, avec les C.I. $x(0) = 0$ et $z(0) = 0$, on obtient : $\begin{cases} x(t) = \tau v_0 \cos(\alpha)(1 - e^{-t/\tau}), \\ z(t) = \tau(v_0 \sin(\alpha) + g\tau)(1 - e^{-t/\tau}) - g\tau t. \end{cases}$

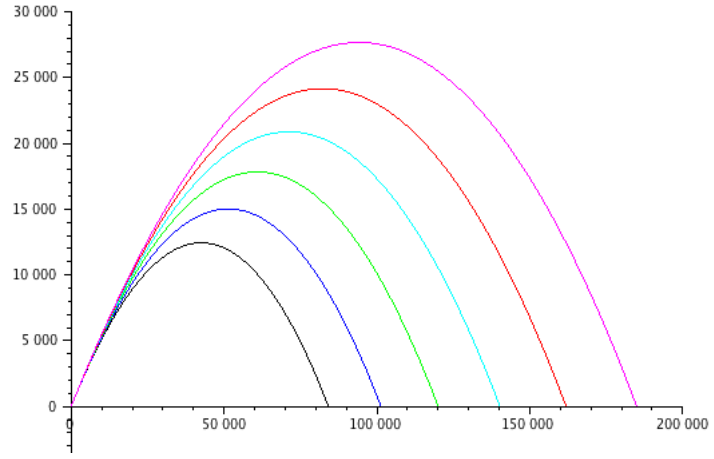
Là encore on peut délimiter le domaine du temps qui nous intéresse très précisément, en ne gardant que les t tels que $z(t) \geq 0$.

Cette fois comme l'équation est *transcendante* on la fait résoudre numériquement avec la fonction `fsolve` du module `scipy.optimize`.

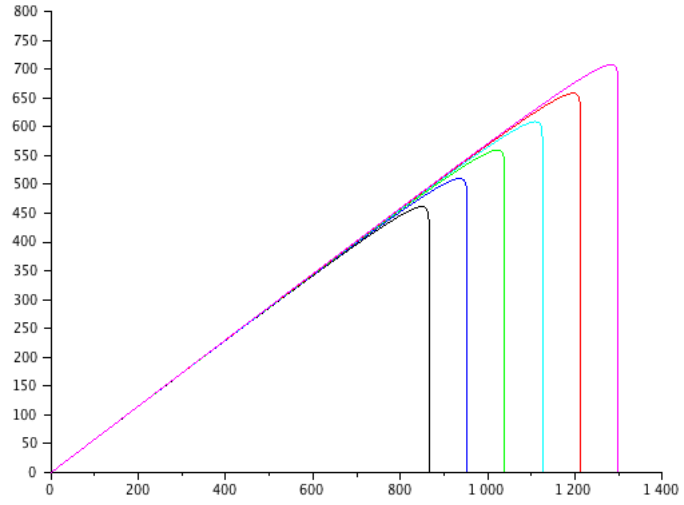
a) En faisant varier la vitesse de 100 à 500, avec le code :

```
import scipy.optimize as scp# pour le fsolve
pl.figure("frottement fluide pp à la vitesse")
pl.clf()
g=9.81
k=0.1
for v0 in range(1000,1600,100):# vitesse en m.s^-1
    alpha=pl.pi/6
    m=140
    tau=m/k
    def f(t):
        return tau*(v0*pl.sin(alpha)+g*tau)*(1-pl.exp(-t/tau))-g*t*tau
    tf=scp.fsolve(f,100) # pour avoir le temps correspondant au point d'impact
    # la valeur germe 100 a été estimée par un premier
    # tracé
    t=pl.linspace(0,tf,50)
    x=tau*v0*pl.cos(alpha)*(1-pl.exp(-t/tau))
    z=f(t)
    pl.plot(x,z)
```

On obtient des courbes très semblables au cas sans frottement, elles vont juste *moins loin* que dans le cas du 2.1. a).

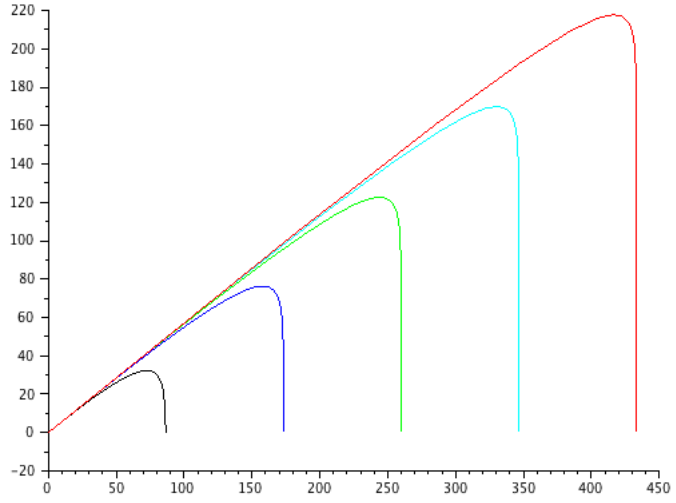


b) Au a) τ est très grand $\tau = 1400s$ ce qui explique que le terme en $\frac{\vec{v}}{\tau}$ qui correspond au terme de frottement, influence peu l'allure du mouvement. On recommence ici avec $\tau = 1$.



Remarque : par rapport aux paraboles, la courbe reste plus « droite » au début, puis tombe plus « brusquement ».

En réduisant un peu la vitesse initiale, on voit mieux la courbe :



3.3 Là où on a besoin de `ode` : frottements fluides plus élevés

En fait, ce sont ces frottements proportionnels à v^2 qui sont utilisés pour modéliser les trajectoires d'obus... (dès que la vitesse d'un objet dans l'air est assez grande, l'écoulement autour de cet objet est *turbulent* est modélisé comme dans la partie 1.2. sur le saut de Baumgartner.

a) **Détermination de la fonction F en question.**

Comme $X'(t) = \begin{pmatrix} x'(t) \\ z'(t) \\ x''(t) \\ z''(t) \end{pmatrix}$, et qu'on veut $\begin{pmatrix} x'(t) \\ z'(t) \\ x''(t) \\ z''(t) \end{pmatrix} = F(\begin{pmatrix} x(t) \\ z(t) \\ x'(t) \\ z'(t) \end{pmatrix}, t)$, on définit donc :

$$F(\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}, t) = \begin{pmatrix} x_3 \\ x_4 \\ -\lambda x_3\sqrt{x_3^2+x_4^2} \\ -\lambda x_4\sqrt{x_3^2+x_4^2} - g \end{pmatrix}.$$

En PYTHON, le code correspondant est avec $\lambda = 1$ et $g = 9,81$

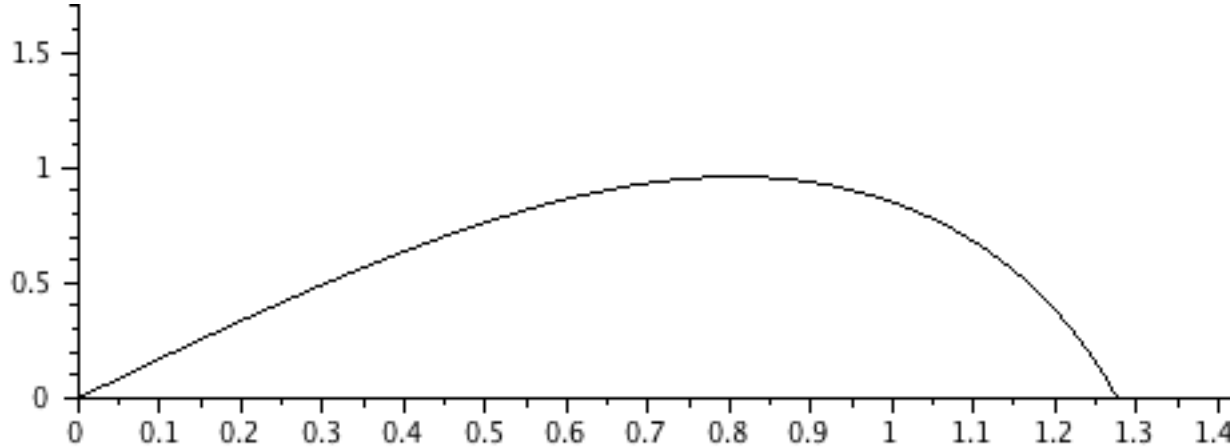
```
def f(X,t):
    return (X[2],X[3],-pl.sqrt(X[2]**2+X[3]**2)*X[2],-pl.sqrt(X[2]**2+X[3]**2)*X[3]-9.81)
```

b) **Résolution du système (S) ci-dessus avec `odeint`.**

```
v0=10
alpha=pl.pi/3
X0=[0,0,v0*pl.cos(alpha),v0*pl.sin(alpha)]# vecteur C.I.
t=pl.arange(0,10,0.01)
X=integrate.odeint(f,X0,t)
```

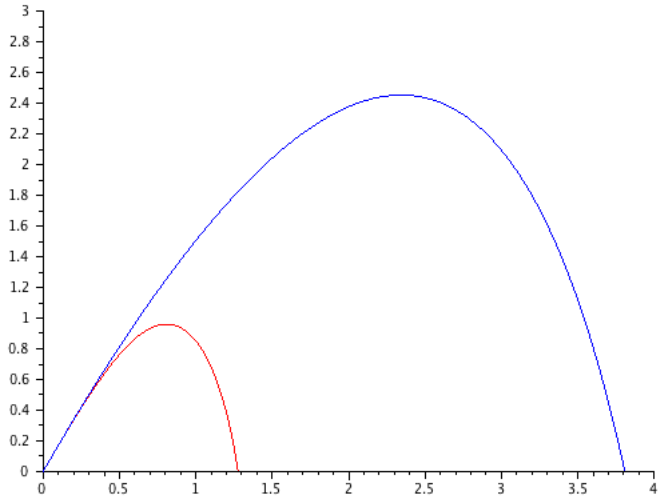
c) **Tracé de la courbe paramétrée $t \mapsto (x(t), z(t))$.**

```
pl.figure("frottement v^2")
pl.clf()
pl.plot(X[:,0],X[:,1])
# on arrange pour s'arrêter au sol
pl.xlim(0,1.5)
pl.ylim(0,3)
pl.show()
```



Tracé des deux courbes simultanées : frottement en v et en v^2 avec $\lambda = 1$

Ici, la comparaison des deux tracés n'est pas très intéressante : mieux vaudrait un choix de constante différent pour les deux frottements.


3
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.5
1
1.5
2
2.5
3
3.5
4