TP info 14
Corrigé

I1- Euler vs RK2

1/ Voir cours

2/ Voir cours

3/ RK2 - Schéma numérique.

$$\begin{cases} y' = f(y,t) \\ y(0) = y_0 \end{cases}$$

$$y' = f(y,t) \Rightarrow \int_{t_i}^{t_{i+1}} y'\,dt = \int_{t_i}^{t_{i+1}} f(y,t)\,dt$$

$$y_{i+1} - y_i \qquad \approx f(y_{i+1/2}, t_{i+1/2}) = \text{cste}$$

$$\Rightarrow \boxed{y_{i+1} = y_i + f(y_{i+1/2}, t_{i+1/2})\,(t_{i+1} - t_i)}$$

Pas h : $t_{i+1} = t_i + h$

$t_{i+1/2} = t_i + \frac{h}{2}$

$y_{i+1/2}$? Méthode d'Euler explicite :

$$\boxed{y_{i+\frac{1}{2}} = y_i + f(y_i, t_i) \times \frac{h}{2}}$$

Implementation

Euler explicite

```
def euler(f,y0,a,b,h):
    n=(b-a)/h
    t=linspace(a,b,n,endpoint=True)
    y=zeros(len(t))
    y[0]=y0
    for i in range(0,len(y)-1):
        y[i+1]=y[i]+f(y[i],t[i])*h
    return t,y
```

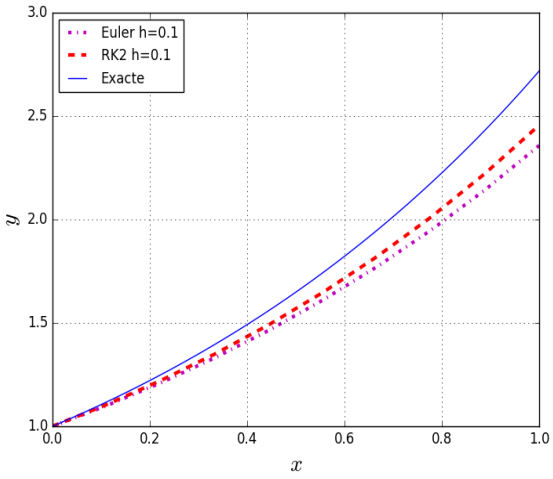
RK2

```
def RK2(f,y0,a,b,h):
    n=(b-a)/h
    t=linspace(a,b,n,endpoint=True)
    y=zeros(len(t))#y=[0,0,0,0,0.....0]
    y[0]=y0
    for i in range(0,len(y)-1):
        t_m=t[i]+h/2
        y_m=y[i]+f(y[i],t[i])*h/2
        y[i+1]=y[i]+f(y_m,t_m)*h
    return t,y
```

4/ Euler vs RK2

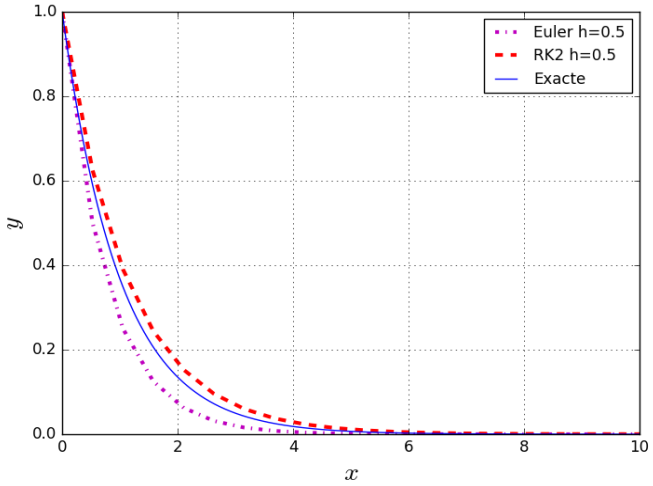
$$\begin{cases} y' = y \\ y(0) = y_0 \end{cases}$$

```
def f(y,t):
    return y
y0=1
h=1e-1

t,y_euler=euler(f,1,0,1,h)
t,y_rk2=RK2(f,1,0,1,h)
```



$$\begin{cases} y' = -y \\ y(0) = y_0 \end{cases}$$

```
def f(y,t):
    return -y
y0=1
h=0.5

t,y_euler=euler(f,1,0,10,h)
t,y_rk2=RK2(f,1,0,10,h)
```

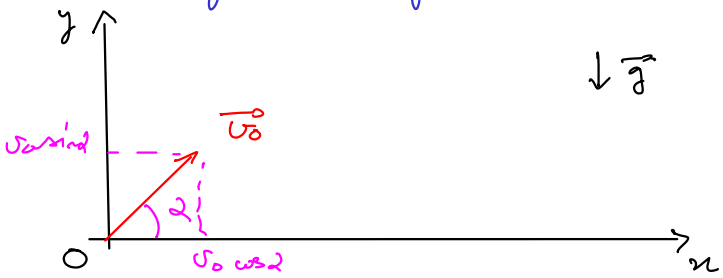


I2 - Tir de projectile

① Trajectoire d'un projectile soumis à une force de frottement linéaire : $\vec{f} = -\lambda \vec{v}$.

Equation du mvt :

$$\begin{cases} \ddot{x} = -\frac{\lambda}{m}\dot{x} \\ \ddot{y} = -\frac{\lambda}{m}\dot{y} - g \end{cases}$$



On pose $X = \dot{x}$ et $Y = \dot{y}$ pour déterminer le syst d'équations différentielles d'ordre 1 équivalent à ce système.

$$\begin{cases} \dot{x} = X \\ \dot{X} = -\frac{\lambda}{m}X \\ \dot{y} = Y \\ \dot{Y} = -\frac{\lambda}{m}Y \end{cases}$$

Paramètres :

```
g=9.81 # pesanteur
v0=10 # norme de la vitesse initiale
m=1 # masse
L=3.7 # coeff de frottement λ
tau=m/L # temps de relaxation.
alpha=pi/6 # angle de tir
init=[0,v0*cos(alpha),0,v0*sin(alpha)]# condition initiale
t=linspace(0,20*tau,10000)# durée sur laquelle on résout les équats du mvt.
```

Résolution des équa diff :

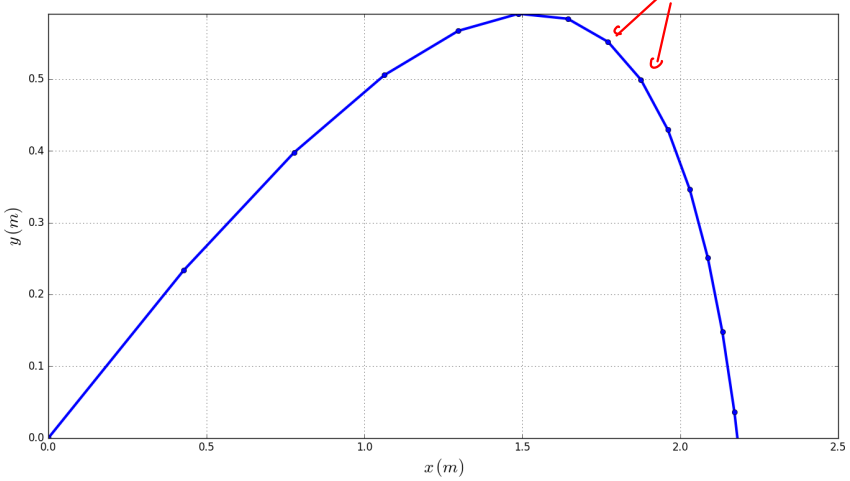
```
def resol(f,init,t):
    """
    Calcule la trajectoire du projectile satisfaisant au systeme d'equation implemente par f
    f: function, renvoie dxdt,dxdt**2,dydt,dydt**2
    init : list of float, conditions initiales [x(0),dxdt(0),y(0),dydt(0)]
    t : array, temps
    x,y : array of float, positions successives du projectile.
    """
    sol=odeint(f,init,t) # résolution.
    x=sol[:,0] ) # extraction des coordonnées
    y=sol[:,2]
    return x,y
```

```
def f(sol,t):
    x=sol[0]
    dxdt=sol[1]
    y=sol[2]
    dydt=sol[3]

    rhs0=dxdt
    rhs1=-L/m*dxdt
    rhs2=dydt
    rhs3=-L/m*dydt-g

    return [rhs0,rhs1,rhs2,rhs3]
```

implémentation des équations différentielles : calcul x_{i+1}, $X_{i+1} = \dot{x}_{i+1}$, y_{i+1}, $Y_{i+1} = \dot{y}_{i+1}$ à partir des 2d membres des équ diff (rhs0, rhs1, rhs2, rhs3) calculés à partir de t_i et t_{i+1} (via l'argument t) et x_i, $X_i = \dot{x}_i$, y_i et $Y_i = \dot{y}_i$ (via l'argument sol).

```
x,y=resol(f,init,t) # appel de la fonction graphe.
```

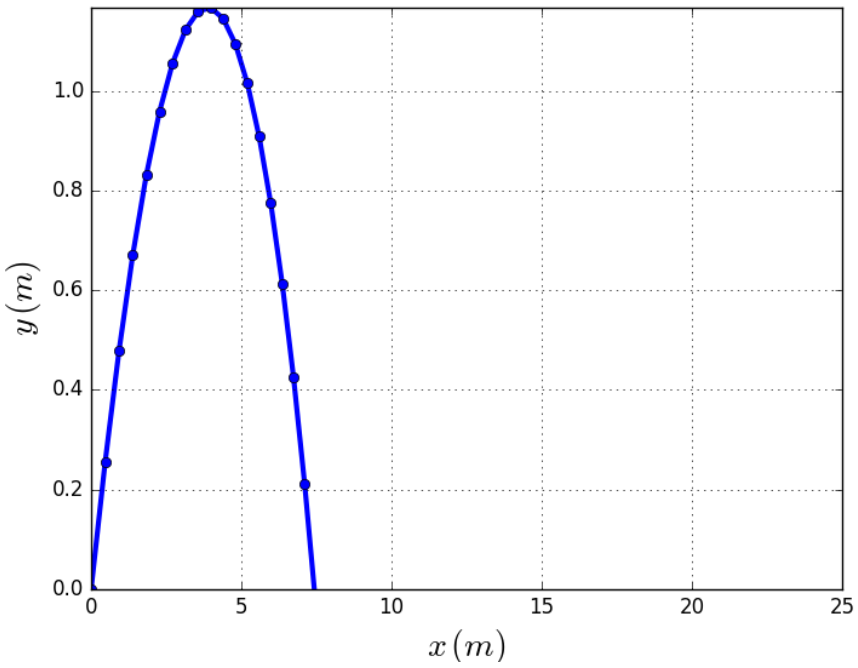
```
figure()
grid()
ylim(0,max(y))
xlabel(r'$x\,(m)$',fontsize=20)
ylabel(r'$y\,(m)$',fontsize=20)
plot(x,y,'bo-',lw=3)
```



N = 20 pts

La précision ↗ avec le nombre de points de calcul pour une durée donnée.

Test : $l = 0$ # pas de frottement



On retrouve bien une trajectoire parabolique comme attendu analytiquement.

$N = 20$ pts

② Portée ?

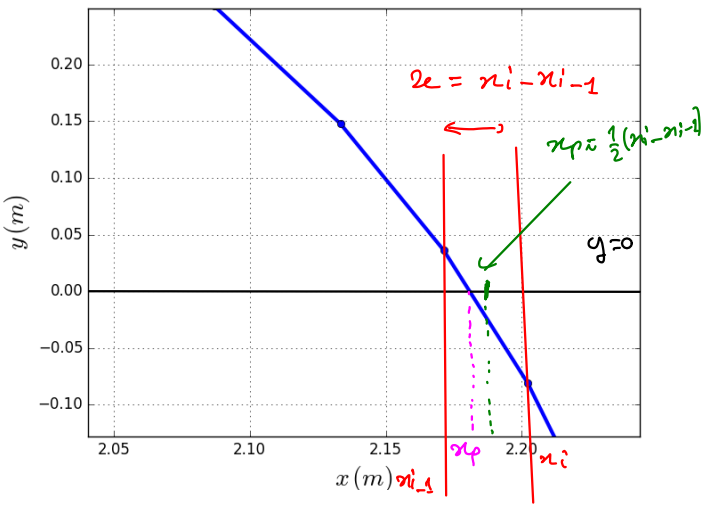
```
"""
Calcule la protee p du tir et evalue une erreur e sur cette portee.
x,y : list of float, coodonnees des points de la trajectoire
p : float, portee du tir (y(p)=0, x!=0)
e : float, erreur sur la portee
"""
n=len(y)
if y[n-1]>0:#trajectoire incomplete, sol non atteint
    return None
else:
    i=1
    while y[i]>0 and i<n:
        i=i+1
    p=1/2*(x[i]+x[i-1])
    e=1/2*(x[i]-x[i-1])
    return p,e
```

On cherche la 1ère valeur de $y_i < 0$. $y = 0$ est compris entre y_i et y_{i-1} donc la portée est comprise entre x_i et x_{i-1}

Résultat :

p=2.186~~7578816646924~~ ± 0.015~~487898232985797~~

La précision sur la portée ↗ avec le nombre de pts de calcul de la trajectoire



③ Angle α_m pour lequel la portée est maximale à λ et v_0 fixés.

```
def liste_portee(f,t,v0,angle):
    """
    Calcule la portee pour les differents angle de tir de la liste alpha
    f : function, equations differentielles du mouvement
    t : array, temps
    v0 : float, norme de la vitesse initiale
    l_p: list of float, les portee pour les differentes valeurs de alpha
    """
    l_p=[]
    for alpha in angle:
        init=[0,v0*np.cos(alpha),0,v0*np.sin(alpha)]
        x,y=resol(f,init,t)
        p=portee(x,y)[0]
        l_p.append(p)
    return l_p
```

On calcule la portée pour un ensemble de valeur de l'angle de tir α.

```
def maxi(l):
    """
    Renvoie le max de la liste l et sa position dans la liste
    m : float, maximum de la liste
    index : int, position de m dans la liste
    """
    n=len(l)
    m=l[0]
    index=0
    i=0
    while i<n:
        if l[i]>m:
            m=l[i]
            index=i
        i=i+1
    return m,index
```

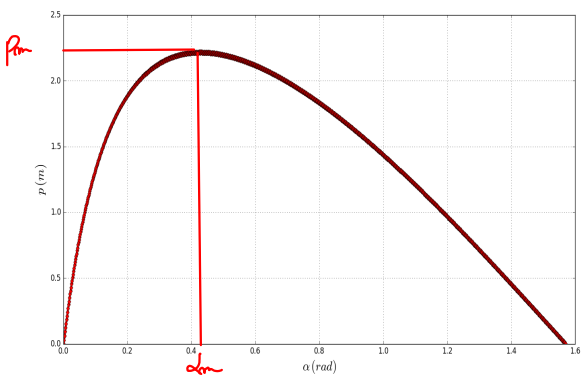
```
angle=linspace(0,pi/2,1000,endpoint=True) # angles pour lesquels on calcule la portee
l=liste_portee(f,t,v0,angle);             # les portee correspondant à ces angles.

figure()
grid()
xlabel(r'$\alpha\,(rad)$')
ylabel(r'$p\,(m)$')
plot(angle,l,'rd-',lw=3)

pm,im=maxi(l)      # portée max
alpham=angle[im]   # angle pour lequel la portée est max
```

portée p en fonction de l'angle de tir.

Résultats

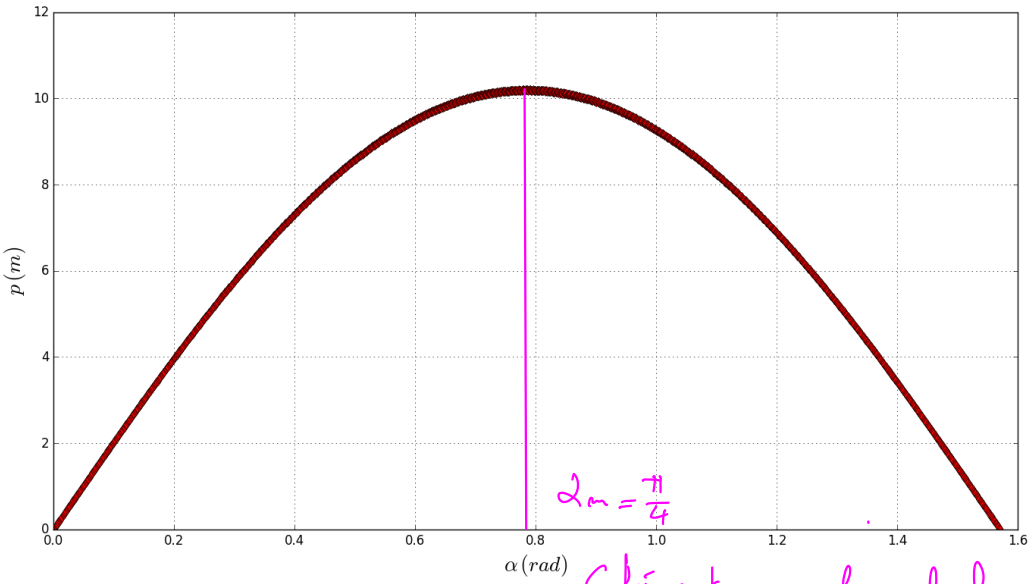
2.21532398065 0.429256653869

p_m $\alpha_m \neq \frac{\pi}{4}$ (ss frottement)



Précision ↗ avec le nbre de pts du calcul des traj. et le nombre d'angle α pour lesquels on calcule p

Test : $L = 0$ (pas de frottement)



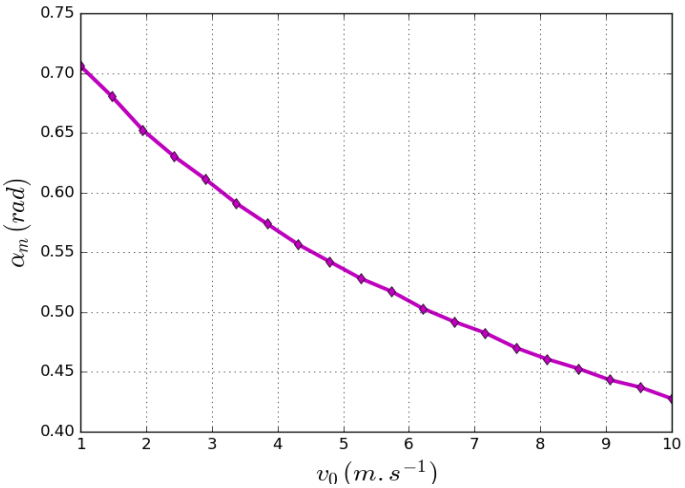
Cohérent avec le calcul analytique : rassurant !

④ Influence de v_0 sur l'angle de portée max α_m.
On veut tracer $\alpha_m = f(v_0)$

```
angle=linspace(0,pi/2,1000,endpoint=True) # angles
vinit=linspace(1,100,20,endpoint=True) # valeurs de v0 pour lesquels on cherche alpha_m.
liste_angle_portee_max=[] # alpha_m tq portée max pour chaque valeur de v0
for v0 in vinit:
    L=liste_portee(f,t,v0,angle)
    pm,im=maxi(L)
    alpham=angle[im]
    liste_angle_portee_max.append(alpham)

figure()
grid()
ylabel(r'$\alpha_m\,(rad)$')
xlabel(r'$v0\,(m.s^{-1})$')
plot(vinit,liste_angle_portee_max,'md-',lw=3)
```

graphe

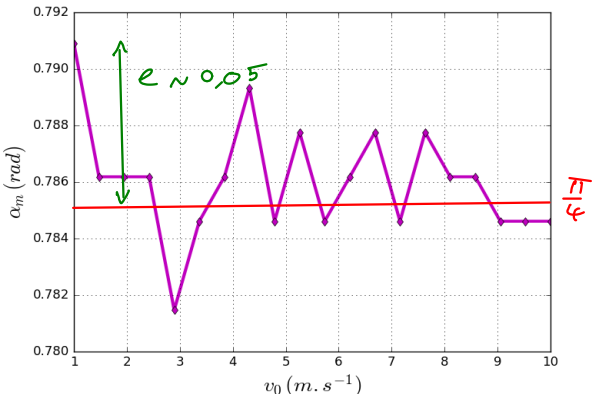
Résultat



Il semblerait que plus la vitesse initiale du projectile est élevé, plus il faut le lancer à l'horizontale pour obtenir une portée maximale.

20 valeurs de v_0, 1000 angles entre 0 et $\frac{\pi}{2}$, 10^5 pts sur la durée $10 \times \tau$.

Test : $L = 0$ # pas de frottements



$\alpha_m \simeq \frac{\pi}{4}$ $\forall$ v_0 ce qui est conforme à la résolution analytique. En supposant les erreurs numériques du même ordre de grandeur en présence de frottements, on constate que ces erreurs sont faibles. ~ 1%

Les résultats sont sensibles :
- au nbre de pts de calcul de la trajectoire.
- au nbre d'angles pour lesquels la portée est évaluée.
- au nbre de valeur de v_0 pour lesquels α_m est évalué.