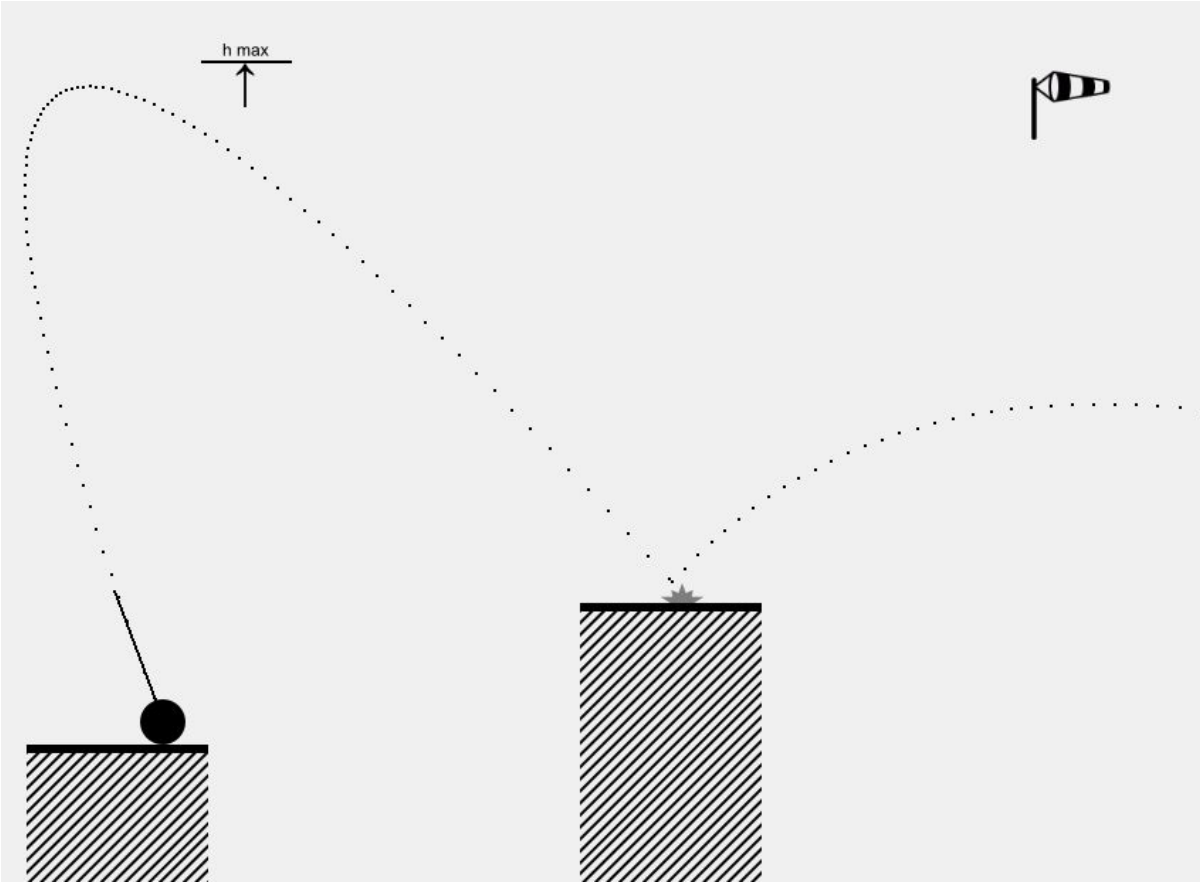
César Guillotel
3A 3DJV

12/2016

Calcul de trajectoire d'un projectile



Sommaire

- Vecteurs 2D

Afin d'intégrer facilement les formules de calculs de projection et autre, une classe Vecteur2D a été créée en Python, comportant les principaux opérateurs arithmétiques, permettant de manipuler des vecteurs comme des nombres.

- **Opérateurs arithmétiques sur des vecteurs 2D**

La classe Vecteur2D comporte deux attributs : x et y. Les opérateurs +, -, *, / sont définis ci-dessous.

```
class Vecteur2 :

    def __init__(soi, x = 0., y = 0.) :
        soi.x = x
        soi.y = y

    def __add__(soi, vec) :
        vec2 = Vecteur2(soi.x + vec.x, soi.y + vec.y)
        return vec2

    def __sub__(soi, vec) :
        vec2 = Vecteur2(soi.x - vec.x, soi.y - vec.y)
        return vec2

    def __mul__(soi, a) :
        vec2 = Vecteur2(soi.x * a, soi.y * a)
        return vec2

    def __div__(soi, a) :
        vec2 = Vecteur2(soi.x / a, soi.y / a)
        return vec2
```

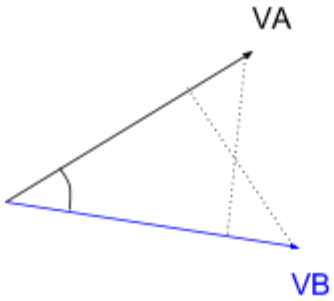
- **Produit vectoriel**

Le produit vectoriel renvoie un vecteur perpendiculaire au plan formé par deux vecteurs donnés. L'ordre d'entrée détermine le sens du vecteur de sortie.

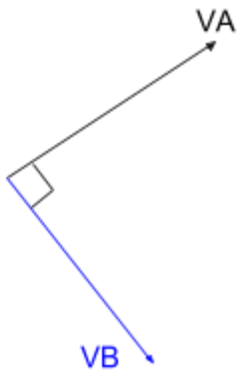
```
def __ixor__(soi, vec) : # Produit vectoriel ^
    vec2 = Vecteur2((soi.y * vec.x) - (soi.x * vec.y), (soi.x * vec.y) - (soi.y * vec.x))
    return vec2
```

- **Produit scalaire**

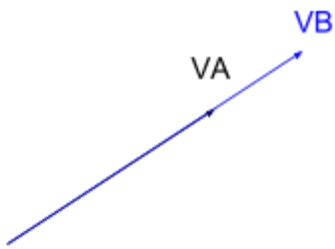
Le produit scalaire sert à calculer l'influence d'un vecteur sur un autre, par rapport à l'angle entre les deux.
Deux vecteurs perpendiculaires auront un produit scalaire nul (le premier vecteur ne va pas du tout dans le même sens que l'autre).
Tandis que deux mêmes vecteurs auront une influence maximale (égale au produit de leur longueur), car les deux vecteurs vont dans le même sens, multipliant ainsi leur force.



Produit non nul



Produit nul



Produit maximal

```
def scalaire(soi, vec) : # Produit scalaire
    return vec.x * soi.x + vec.y * soi.y
```

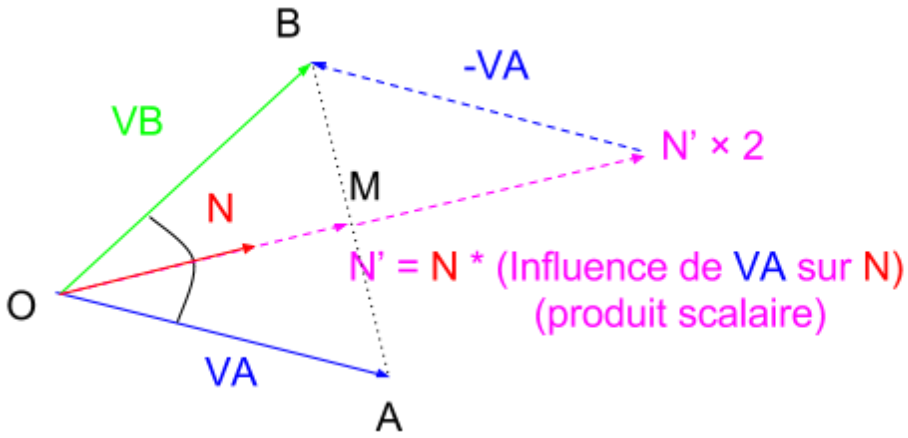
- **Symétrie d'un vecteur par rapport à une normale**

Pour calculer la symétrie VB d'un vecteur VA par rapport à une normale N, on remarque que le triangle OAB est isocèle en O. VB et B sont inconnus. Le but dans une premier temps est de calculer le milieu M de AB.
Pour cela, on calcule le produit scalaire de VA sur N, qui va donner l'intensité qu'a le vecteur VA sur N (voir ci-dessus), pour le multiplier à N et en déduire N', qui arrive entre A et B.

On en déduit le Vecteur AM = N' - VA

On sait que AM = MB, donc OB = AM × 2

Donc :
VB = 2 × N' - VA
VB = 2 × N × (N • VA) - VA



La symétrie utilisée pour le TP concerne le rebond, le vecteur VA est inversé, on pose donc :

VB = VA - 2 × N × (N • VA)

```
def symetrie(soi, vec) :
    return Vecteur2(soi.x, soi.y) - (vec * 2.0 * soi.scalaire(vec));
```

- **Longueur du vecteur**

La longueur est calculée avec le théorème de Pythagore, en calculant l'hypoténuse du triangle dont les côtés perpendiculaires sont le segments X et Y du vecteur.

```
def getLongueur(soi) :
    return (soi.x ** 2 + soi.y ** 2) ** 0.5
```

- Résolution d’une équation du second degré

Certaines formules ci-dessous demandent la résolution d’une équation du second degré. Ci-dessous son implémentation en Python avec le calcul de Delta et les deux résultats renvoyés.

Paramètres d’entrée

Les facteurs a, b, c, correspondants à l’équation $ax^2 + bx + c = 0$

Valeurs de sortie

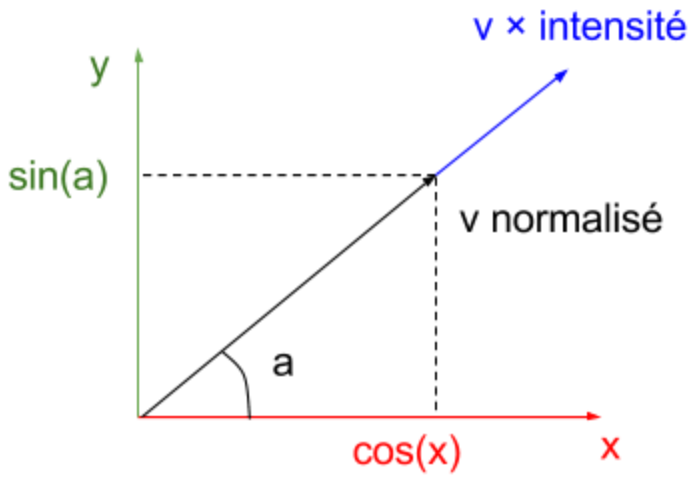
Les solutions x1 et x2 s’il y en a

Code Python

```
@staticmethod
def resoutEquationSecondDegre(a, b, c) :
    delta = (b ** 2) - (4. * a * c)
    if delta < 0 :
        return None, None
    racineDelta = delta ** .5
    x1 = (-b - racineDelta) / (2. * a)
    x2 = (-b + racineDelta) / (2. * a)
    return x1, x2
```

- Calcul du vecteur de tir à partir de l'angle et de la force

Afin de déterminer le vecteur de vitesse initial à partir d'un angle et une vitesse donnée, on calcule le cosinus de l'angle, qui va donner le vecteur X, et le sinus de l'angle : le vecteur Y. Multiplié par la vitesse en m/s, on obtient le vecteur vitesse.



Paramètres d'entrée

Angle en degré
Vitesse en m/s (puissance)

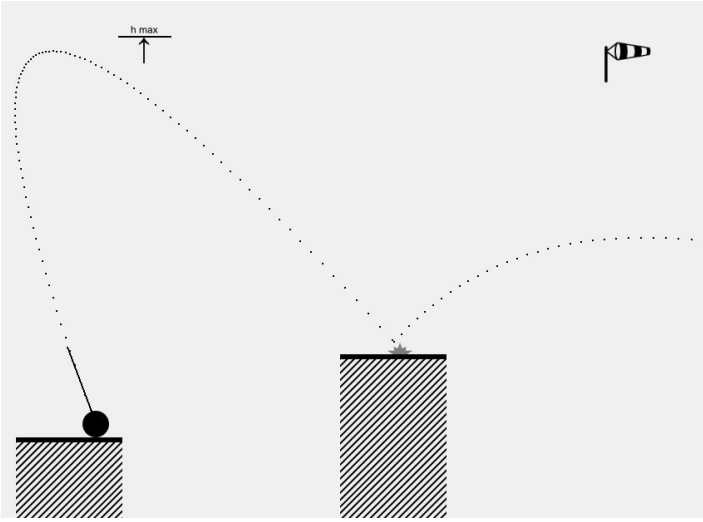
Valeur de sortie

Vecteur de vitesse initiale en m/s

Code Python

```
@staticmethod
def calculeVecteurVitesse(angle, puissance) :
    vec = Vecteur2(math.cos(angle), math.sin(angle))
    vec *= puissance
    return vec
```

- Calcul de la trajectoire



- **Sans friction**

L'équation de calcul de trajectoire sans friction est la suivante :
$f(t) = p + v \times t + (g \times t^2) / 2$

Paramètres d'entrée

Vecteur position initiale en m (p)
Vecteur vitesse en m/s (v)
Temps en secondes (t)
Gravité + vent en $m \cdot s^{-2}$ (g)

Valeur de sortie

Vecteur de position du projectile à un instant t

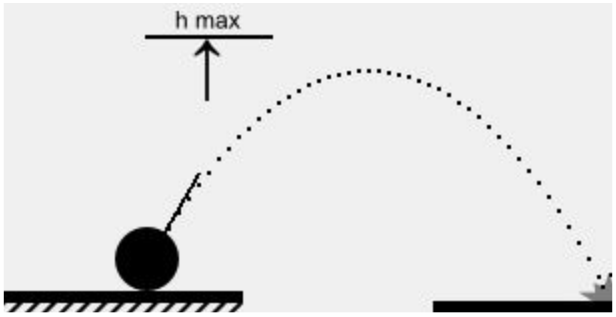
Code Python

```
@staticmethod
def formuleSansFriction(p, v, t, g) :
    return p + v * t + g * .5 * (t ** 2)
```

- **Implémentation du vent**

Les calculs comportent l'implémentation de la force du vent, notée sous forme d'un vecteur 2D. Celle-ci est additionnée à la force de gravitation.

- Calcul de la hauteur max



Le calcul de la hauteur max ne comporte que la composante Y des vecteurs, à partir de la vitesse initiale, la position initiale, la gravité et la force du vent.

Paramètres d'entrée

Position Y en m (p)
Vitesse Y en m/s (v)
Gravité Y + vent Y en $m \cdot s^{-2}$ (g)

Valeur de sortie

Hauteur max en m

Code Python

```
@staticmethod
def formuleCalculeHauteurMax(p, v, g) :
    return p.y + ((v.y ** 2) / (2. * -g.y))
```

- Calcul du temps à l'impact (sans friction)

L'équation de calcul de trajectoire renvoyant la position à un instant t, le calcul du temps à l'impact doit vérifier que l'équation de trajectoire = la position d'impact à un instant t.
Il suffit ensuite d'isoler le temps t de l'équation de trajectoire.

$pi = p + v \times t + (g \times t^2) / 2$

Avec

Position d'impact Y en m (pi)
Position de départ Y en m (p)
Vitesse Y en m/s (v)
Gravité Y + vent Y en $m \cdot s^{-2}$ (g)
Temps d'impact en sec (t)

On résout l'équation du second degré pour trouver les solutions de t :

$0 = (g / 2) \times t^2 + v \times t + p - pi$

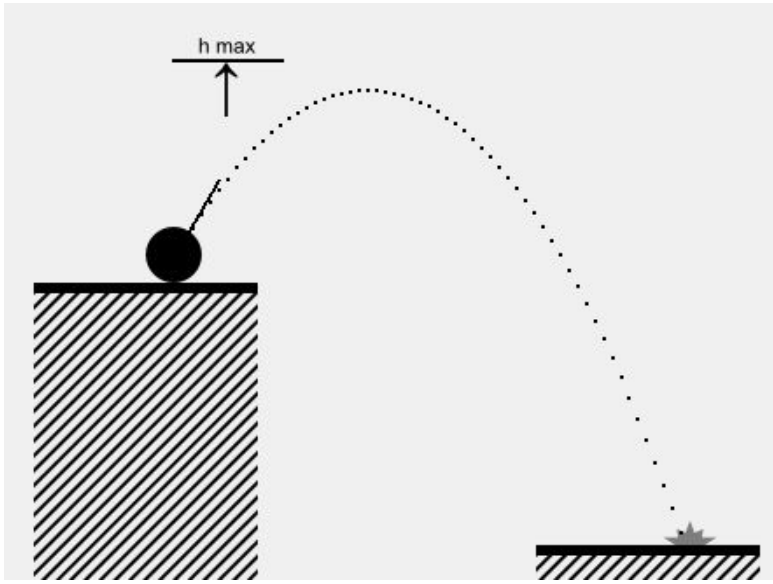
Valeur de sortie

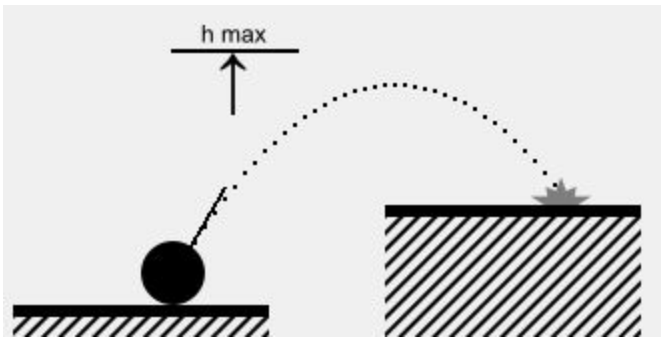
Les temps d'impact t1 et t2 en sec. L'un est négatif car la courbe de trajectoire coupe la droite définie par f(x) = pi à deux endroits, la valeur pertinente est le t positif car on évolue avec un temps qui augmente.

Exemple de code Python pour une position d'impact Y donnée

```
t1, t2 = soi.resoutEquationSecondDegre(.5 * g.y, v.y, p.y - pi.y)
```

- Calcul de la hauteur d’un sol pour une collision à une longueur donnée (sans friction)





Pour savoir à quelle hauteur il faut placer un sol pour avoir une collision à une distance X donnée, on utilise la formule de calcul de trajectoire avec le temps d’impact calculé précédemment.

$h = p + v \times t + (g \times t^2) / 2$

Avec

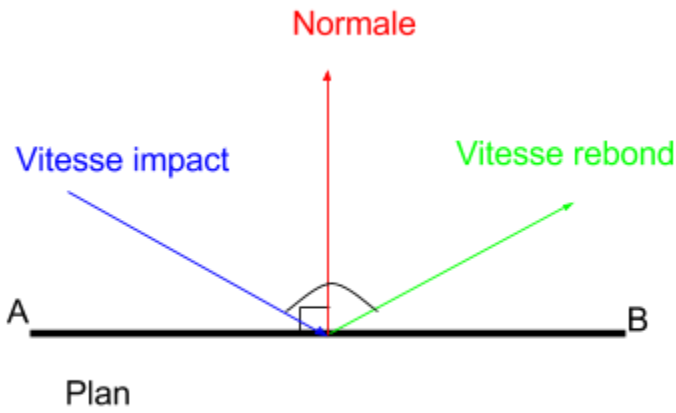
Position de départ Y en m (p)
Vitesse Y en m/s (v)
Gravité Y + vent Y en $m \cdot s^{-2}$ (g)
Temps d’impact en sec (t)

Valeur de retour

Position Y du sol en m (h)

- Calcul du rebond

Pour calculer le vecteur de vitesse au rebond, on calcule :



La vitesse à l'impact

Elle est égale à la vitesse de départ + les forces × le temps d'impact.

vi = f × ti + v

Avec

Vecteur vitesse impact en m/s
Forces (gravité + vent) en $m \cdot s^{-2}$
Temps impact en sec
Vecteur vitesse de départ en m/s

La normale au plan de collision

Elle est la perpendiculaire au plan de collision.

AB = B - A
nor = (-AB.y, AB.x)

Avec

Vecteur normale (nor)
Vecteur point A du plan de collision
Vecteur point B du plan de collision
Vecteur AB

La vitesse de rebond

C’est la symétrie de la vitesse d’impact par rapport à la normale du plan de collision, multipliée par un coefficient d'absorption.

vr = vi.symétrie(nor) × a

(Voir Vecteur symétrie, page 5)

vr = vi - 2 × nor × (vi • nor) × a

Avec

Vecteur de rebond (vr)
Vecteur d’impact (vi)
Vecteur normale (nor)
Coefficient d’absorption (a)