# La courbe brachistochrone

Lucille DELAPORTE

Candidat n °37477

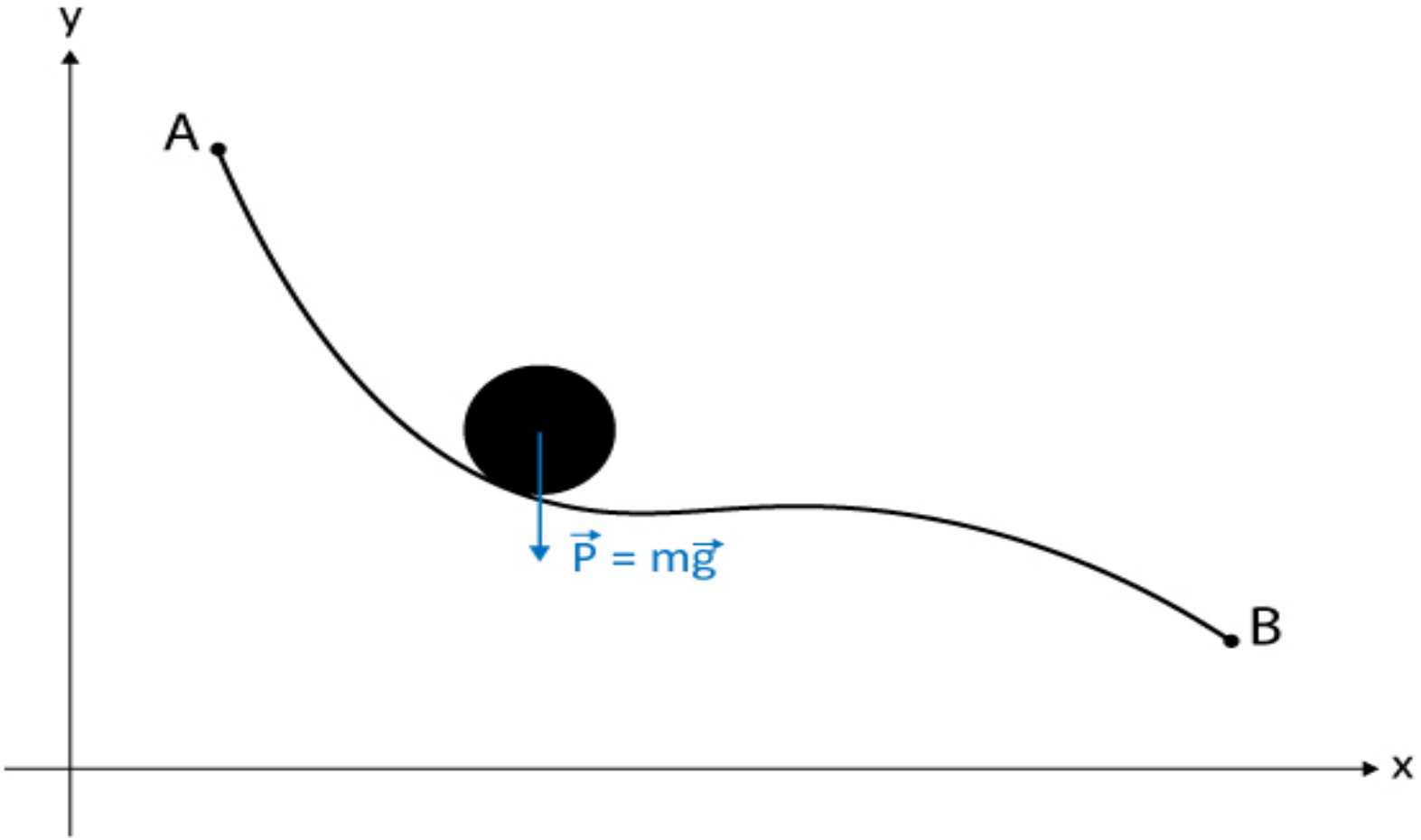


Figure – Illustration du problème

$$t = \int_{x_a}^{x_b} F(y, y')\,\mathrm{d}x \text{ où } F(y, y') = \sqrt{\frac{1+y'^2}{2g(y_0-y)}}$$

Théorème d'Euler-Lagrange : Soit $F(x, y(x), y'(x))$
$\frac{\partial F}{\partial y}(x, y(x), y'(x)) - \frac{\mathrm{d}}{\mathrm{dx}}\left(\frac{\partial F}{\partial z}(x, y(x), y'(x)\right) = 0$

Formule de Beltrami : Soit $F(x, y(x), y'(x))$ et $k \in \mathbb{R}$
$F - z\frac{\partial F}{\partial z}(x, y(x), y'(x)) = k$

$$1 + y'^2 = -\frac{1}{2gk^2y}$$

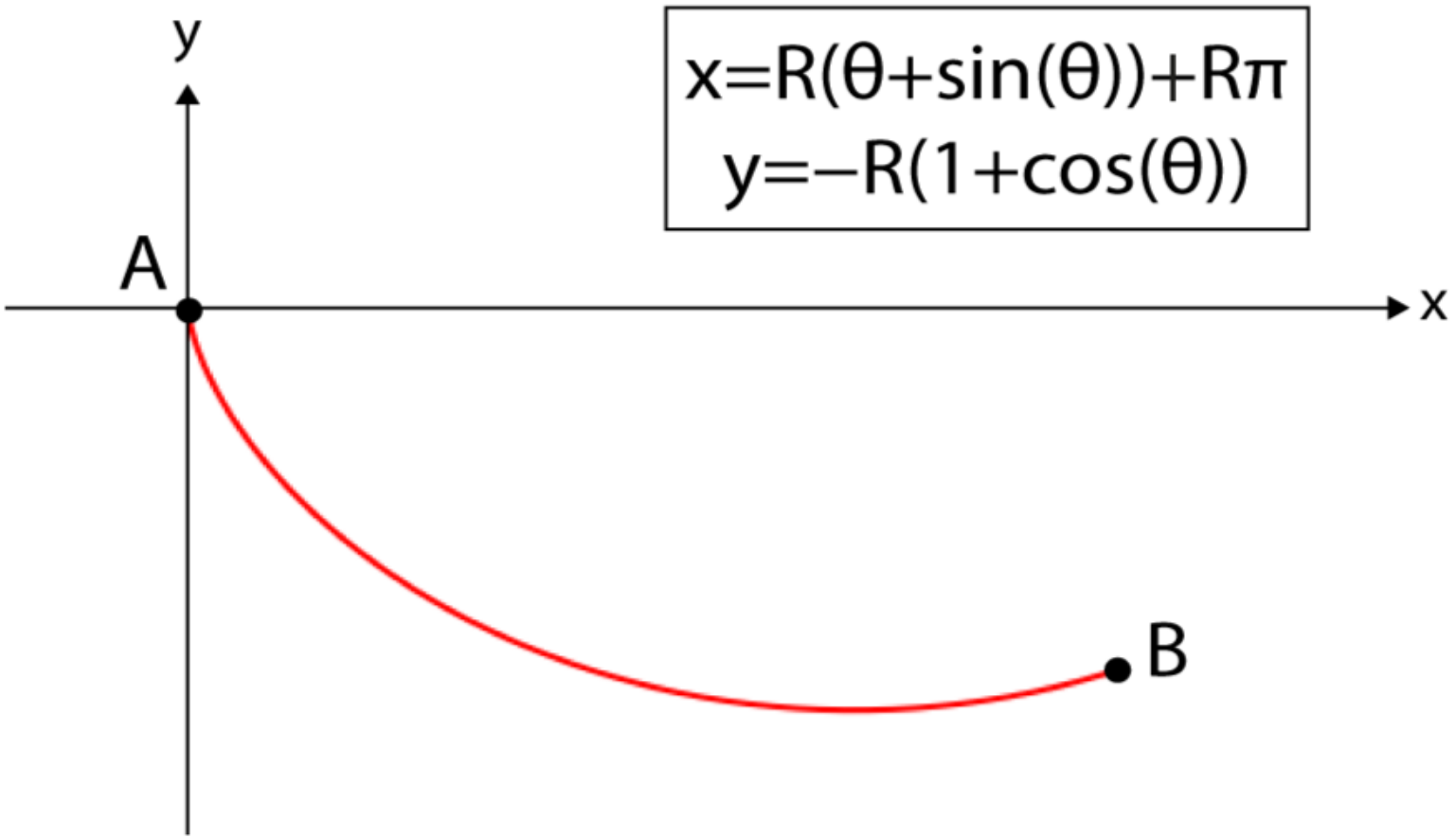


Figure – Illustration de la solution

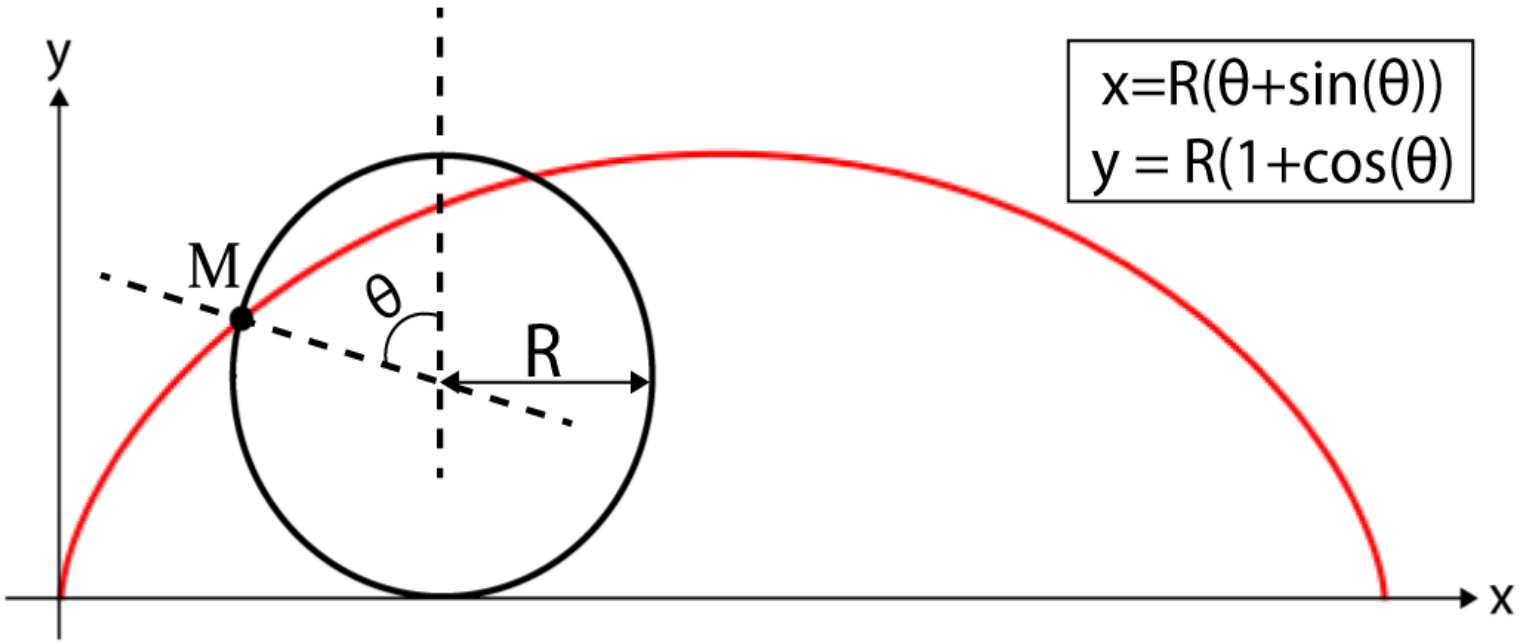


Figure – Cycloïde

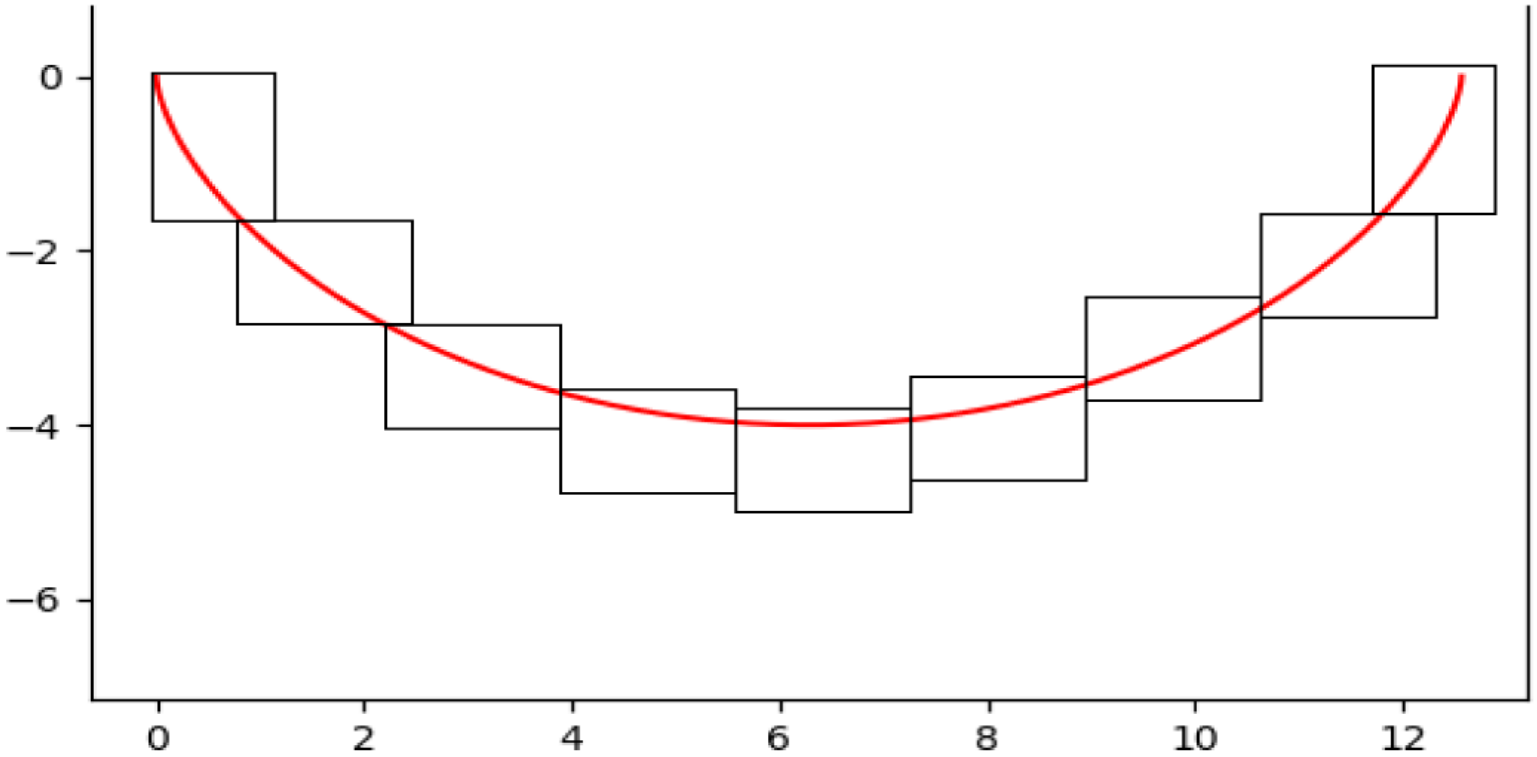


Figure – Principe de l'impression

Figure – Photo de l'expérience

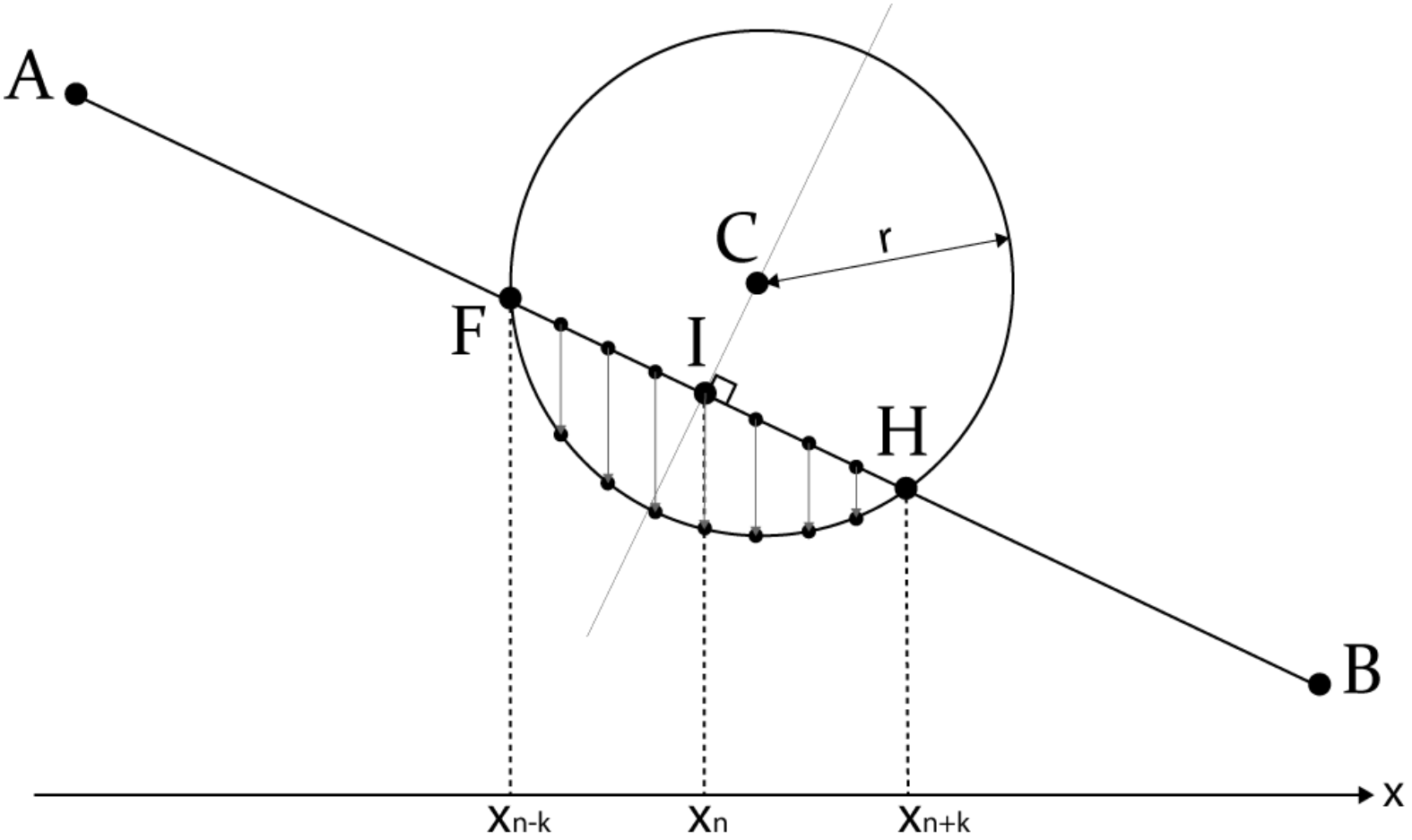


Figure – Première étape

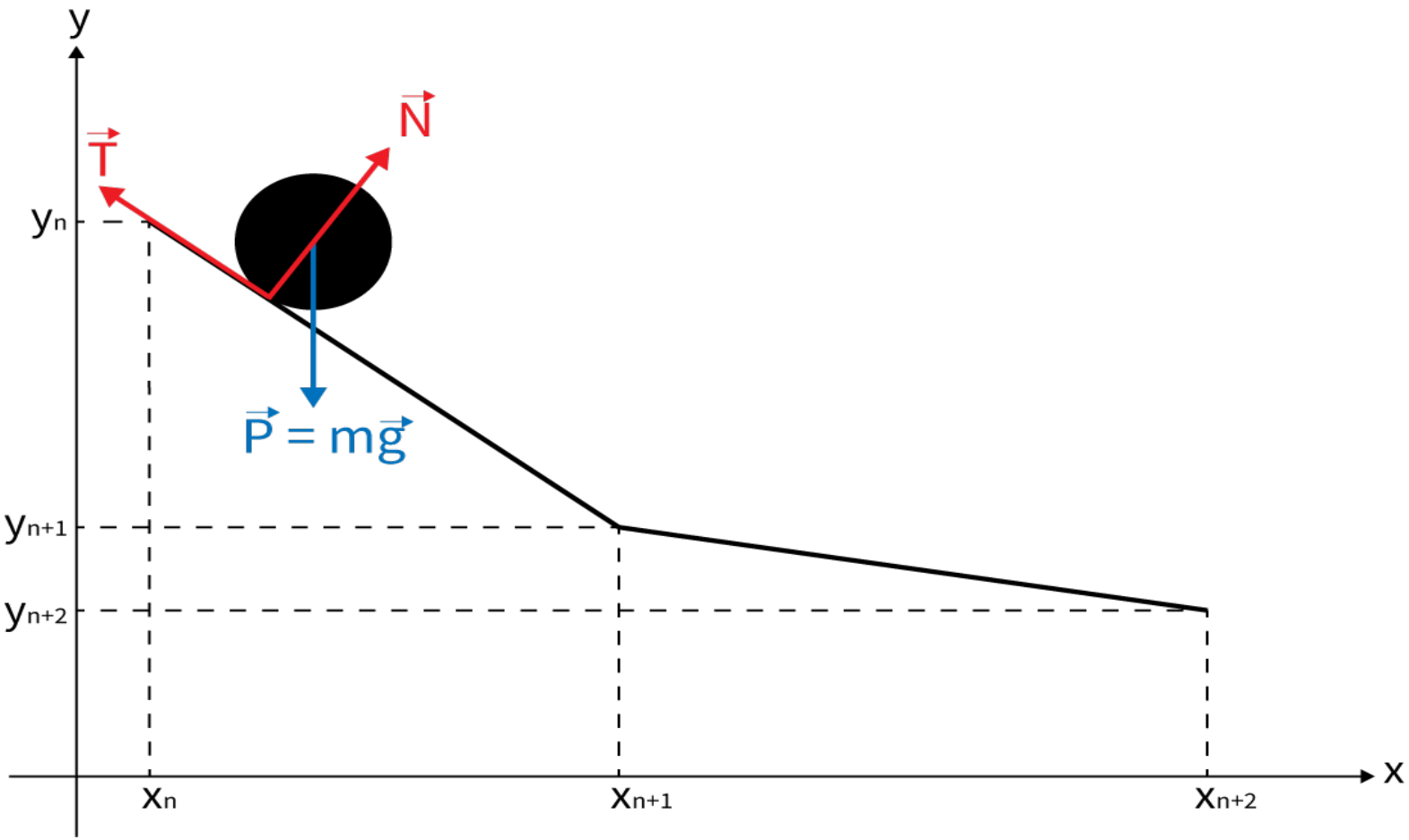


Figure – Principe du test

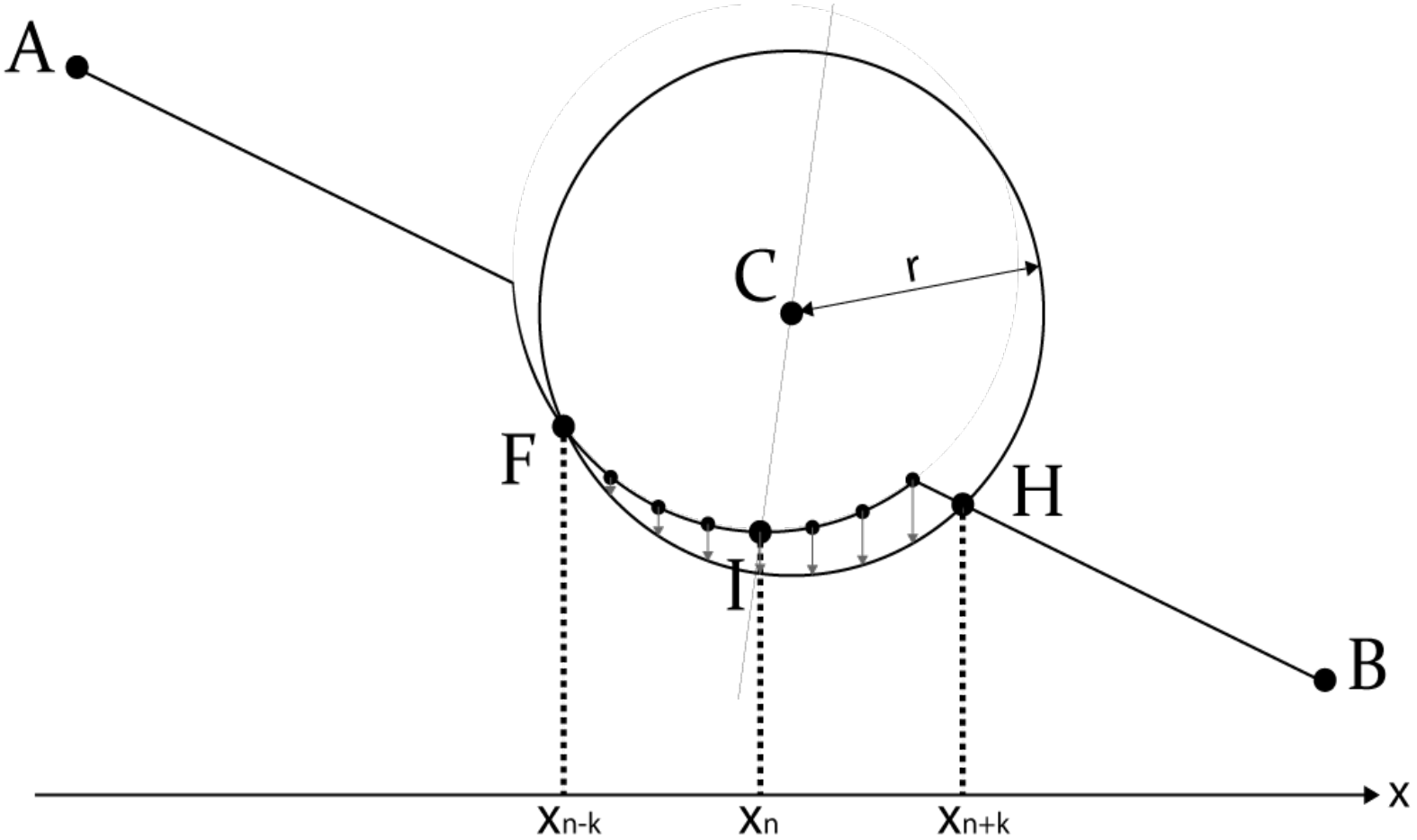


Figure – Seconde étape

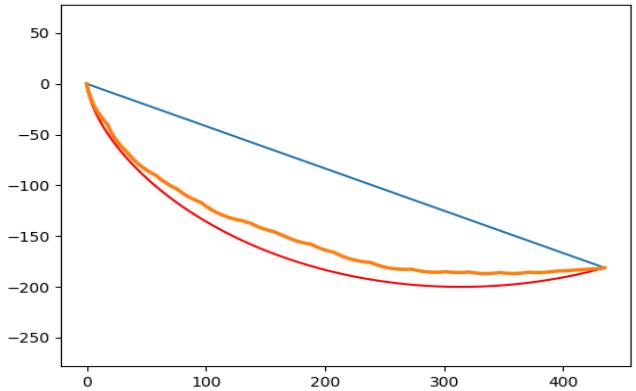


Figure - 5000 itérations

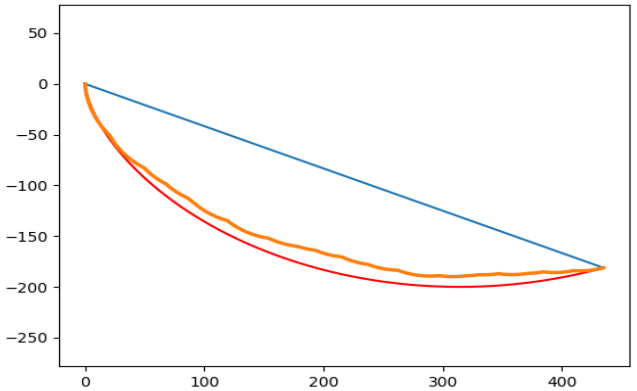


Figure - 100000 itérations

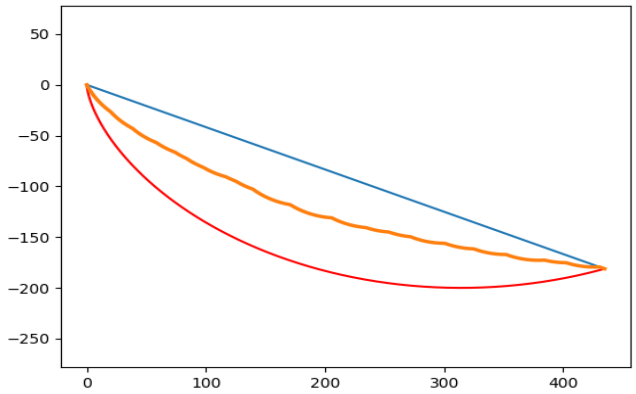


Figure - coefficient de 0,4

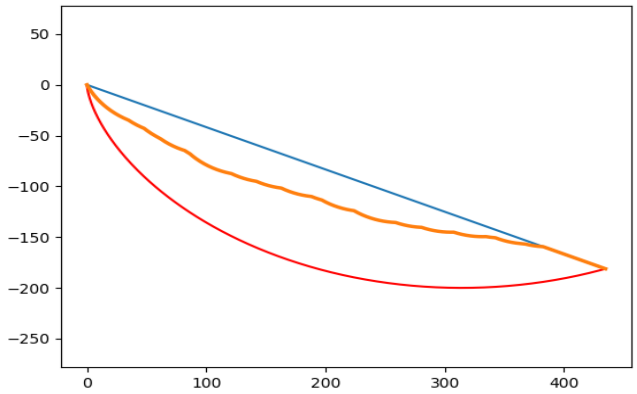


Figure - coefficient de 0,6

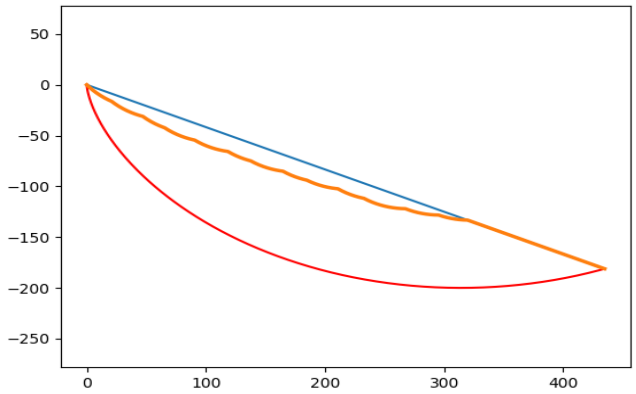


Figure - coefficient de 0,8

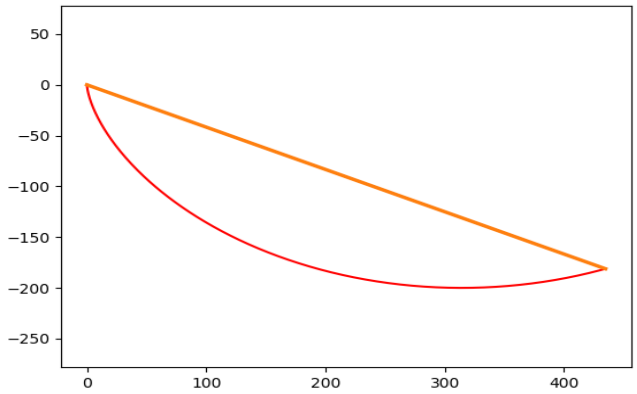


Figure - coefficient de 1,5

# Résolution analytique

$$F = \sqrt{\frac{1+y'^2}{2g(y-\mu y)}}$$

$$(1 + y'^2)(1 + \mu y') + 2(y - \mu x)y'' = 0$$

$$x = R[(\theta - sin(\theta)) + \mu(1 - cos(\theta))]$$
$$y = R[(1 - cos(\theta)) + \mu(\theta + sin(\theta))]$$

# Méthode d'Euler

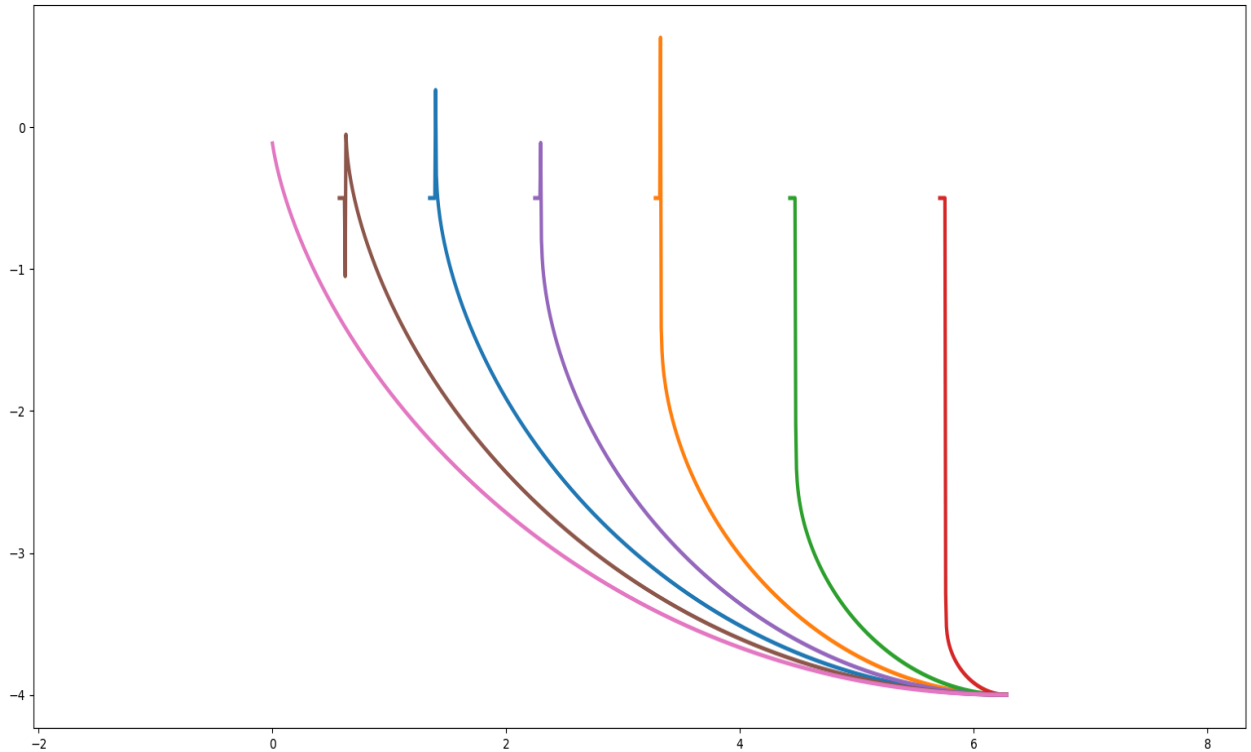


Figure – Résulats via la méthode d'Euler

```
## Presentation
# le but de ce programme est de mettre en place un
    systeme afin de retrouver la courbe
    brachistochrone qui minimise le temps de parcours
    en fonction de differents coefficients de
    frottements

## INTRO

from math import *
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import quad
from random import randint
import time

plt.close('all')

## FONCTIONS

def cycloide(R):
    # ce programme permet de determiner les listes x
        et y qui representent les coordonnees d'une
        cycloide de rayon R
    x, y = [], []
    t = np.arange(np.pi, tmax, pas)
    for i in range(len(t)):
```

```
22          x.append(R*(t[i] + np.sin(t[i]))-R*np.pi)
23          y.append(-R*(1 + np.cos(t[i])))
24      return x, y
25
26  def tracec(R,n,p):
27      # ce programme permet de tracer la cycloide, on
            arrete le tracer au pi eme element de x et y
28      x,y = cycloide(R)
29      x1, y1 = [], []
30      for i in range(p+1):
31          x1.append(x[i])
32          y1.append(y[i])
33      plt.plot(x1, y1,'red')
34      plt.grid()
35      return x1, y1
36
37  def droite(ax, ay, bx, by, n):
38      # ce programme permet de determiner les
            coordonnees du segment entre les points (ax,
            xy) et (bx, by)
39      x = np.linspace(ax, bx,n)
40      y = []
41      for i in range(len(x)):
42          y.append(((by - ay)/(bx-ax))*(x[i]-ax) +  ay)
43      return x, y
44
45  def traced(x, y, n):
```

```
46      # ce programme permet de tracer la droite
47      plt.plot(x,y)
48      plt.grid()
49
50  def longueur_courbe(p, xc, yc):
51      # ce programme calcule la longueur de la courbe
            sachant que la courbe est consideree comme
            une droite brisee
52      # p correspond au point niveau duquel on arr te
            le calcul, en effet ce point represente la
            fin de la courbe choisie
53      longueur = 0
54      for i in range(p-1):
55          petit = sqrt((yc[i+1] - yc[i])**2 + (xc[i+1]
                - xc[i])**2)
56          longueur += petit
57      return longueur
58
59  def acc_et_long(x, y):
60      # ce programme calcul les longueurs de chaque
            portion de courbe (ce sont des segments) et
            les accelerations sur chaque portion
61      # On suppose que l'acceleration est constante sur
            chaque portion
62      A, Longueur = [], []
63      g = 9.81
64      ltot = 0
```

```
65      for i in range(len(x)-1):
66          hyp = sqrt((y[i+1]-y[i])**2 +
                (x[i+1]-x[i])**2)
67          A.append(-g*(y[i+1] - y[i])/hyp)
68          Longueur.append(hyp)
69      return A, Longueur
70
71  def acc_et_long_frottements_solides(x, y, f):
72      # ce programme calcule les longueurs et
            accelerations sur chaque portion de courbe en
            utilisant les m mes approximation que dans
            le programme precedent
73      A, Longueur = [], []
74      g = 9.81
75      ltot = 0
76      for i in range(len(x)-1):
77          hyp = sqrt((y[i+1]-y[i])**2 +
                (x[i+1]-x[i])**2)
78          #print(-g*(y[i+1] - y[i])/hyp -
                (f*g*(x[i+1]-x[i])/hyp))
79          A.append(-g*(y[i+1] - y[i])/hyp -
                (f*g*(x[i+1]-x[i])/hyp))
80          Longueur.append(hyp)
81      return A, Longueur
82
83  def temps(x,y):
84      g = 9.81
```

```
    A, L = acc_et_long(x, y)
    t, T = 0, 0
    v = 0
    for i in range(1, len(A)-1):
        if A[i-1] == 0 and v != 0:
            t = L[i-1]/v
        else:
            t = (-v + sqrt(v**2 +
                2*L[i-1]*A[i-1]))/A[i-1]
        v = v + A[i-1]*t
        T += t
    return T

def temps_frottements_solides(x,y):
    g = 9.81
    A, L = acc_et_long_frottements_solides(x, y, f)
    t, T = 0, 0
    v = 0
    for i in range(1, len(A)-1):
        if A[i-1] == 0 and v != 0:
            t = L[i-1]/v
        else:
            t = (-v + sqrt(v**2 +
                2*L[i-1]*A[i-1]))/A[i-1]
        v = v + A[i-1]*t
        T += t
    return T
```

```
110
111 def test(x1, y1, y2):
112     return temps(x1, y2) < temps(x1, y1)
113
114 def test1(x, y, y1):
115     return temps_frottements_solides(x, y1) <
        temps_frottements_solides(x, y)
116
117 def algorithme(x, y):
118     tinit = time.time()
119     k = 20
120     r = 50
121     iter = 0
122     pomme = []
123     while iter < 20000 and time.time() - tinit < 30:
124         tps = temps_frottements_solides(x,y)
125         Y = y.copy()
126         i = randint(k,len(x)-k-1)
127         coeff = (y[i+k]-y[i-k])/(x[i+k]-x[i-k])
128         xi = 1/2 * (x[i-k] + x[i+k])
129         yi = 1/2 * (y[i-k] + y[i+k])
130         H = sqrt((xi - x[i-k])**2 + (yi - y[i-k])**2)
131         R = H + r
132         a = 1/(coeff**2) + 1
133         b = -(2*xi)/(coeff**2) - (2*yi)/coeff +
            2*y[i+k]/coeff - 2*x[i+k]
134         c = (xi**2)/(coeff**2) + yi**2 +
```

```
                2*(xi*yi)/coeff - 2*(y[i+k]*xi)/coeff
                -2*y[i+k]*yi + x[i+k]**2 - R**2 +
                y[i+k]**2
            if coeff <= 0:
                xc = (-b + sqrt(b**2 - 4*a*c))/(2*a)
            else:
                xc = (-b - sqrt(b**2 - 4*a*c))/(2*a)
            yc = (-1/coeff)*(xc - xi) + yi
            for p in range(i-k, i+k+1):
                b1 = -2*yc
                c1 = (xc - x[p])**2 + yc**2 - R**2
                y[p] = (-b1 - sqrt(b1**2 -4*c1))/2
            if test1(x, y, Y):
                y = Y.copy()
            # pomme.append(y)
            iter += 1
        return y #pomme


## MAIN PROGRAM
R = 50
n = 1000
tmax = 3*np.pi
pas = tmax/(n-1)
xc, yc = cycloide(R)
longueur_liste = len(xc)
p = int(0.60*len(xc))
```

```
159 ax, ay = 0, 0
160 coefdecreuse = 1
161 f = -0.5
162
163 xc1, yc1 = tracec(R, n, p)
164 bx, by = xc1[-1], yc1[-1]
165 xd, yd = droite(ax, ay, bx, by, n)
166 plt.plot(xd, yd)
167
168 longueur_droite1 = longueur_courbe(len(xd), xd, yd)
169 vitesse_droite1 = temps(xd, yd)
170 vitesse_droite_frot = temps_frottements_solides(xd,
        yd)
171 longueur_courbe1 = longueur_courbe(len(xc1), xc1, yc1)
172 vitesse_courbe1 = temps(xc1, yc1)
173 vitesse_courbe_frot = temps_frottements_solides(xc1,
        yc1)
174
175 #y2 = creuse(xd, yd, 1, 1, coefdecreuse)
176 #y3 = creuse_un_point(xd, yd, 4)
177 #yfinal = algorithme(xd, yd)
178
179 # print(xc[p], yc[p],
180 #       xd[-1], yd[-1])
181 # print(longueur_liste, p)
182 print(longueur_droite1, vitesse_droite1,
        vitesse_droite_frot)
```

```
183 print(longueur_courbe1, vitesse_courbe1,
        vitesse_courbe_frot)
184
185 plt.axis('equal')
186 #fig1 = tracec(R,n,p)
187 #fig2 = traced(xd, yd, n)
188 #fig3 = traced(xd, y2, n)
189 #for i in range(len(pomme)):
190 #fig4 = traced(xd, yfinal, n)
191 #plt.show()
```

```
1  ##  bibliotheques
2  import matplotlib.pyplot as plt
3  import numpy as np
4  from scipy.integrate import quad
5  plt.close("all")
6
7
8  ## fonctions
9
10 def cyclo(R, B, n = 12):
11     # R le rayon de la cycloide
12     # On veut tracer une courbe de cycloide et une
           droite
13
14     x,y,Bx,By,Ax,Ay = [],[],0,0,0,0
15     A = np.pi
```

```
16      d = []
17
18      # On rempli les listes des valeurs pour x et y en
            chaque point de la cycloide
19      a = np.arange(A,B,np.pi/n)
20      x = R * (a + np.sin(a))- R * A
21      y = -(R * (1 + np.cos(a)))
22
23      sommet = A
24      Ax = 0
25      Ay = 0
26
27      Bx = R * (B + np.sin(B)) - R * np.pi
28      By = -(R * (1 + np.cos(B)))
29
30      return Ax,Ay,Bx,By,x,y
31
32  def imprimercycloide(x,y):
33      sous_x, sous_y = [x[0]], [y[0]]
34      couleur = ['red','blue']
35      i = 1
36      a, b = 0, 0
37
38      for k in range(len(x)-1):
39          x_min, x_max = min(sous_x) , max(sous_x)
40          y_min, y_max = min(sous_y) , max(sous_y)
41
```

```
42          if x[k+1] >= x_min + 21 and b == 0:
43              a += 1
44              if y[k+1] <= y_max - 21 or x[k+1] >=
                    x_min + 29.7:
45                  # M = min(sous_y[0], sous_y[-1])
46                  fig = plt.figure(i,figsize=(w,h))
47                  plt.plot([x_min, x_min + 29.7],
                        [y_min, y_min + 21], 'o')
48                  plt.plot(sous_x, sous_y, couleur[i%2])
49                  plt.legend()
50                  plt.axis("equal")
51                  plt.grid()
52                  plt.show()
53                  i += 1
54                  sous_x = [x[k+1]]
55                  sous_y = [y[k+1]]
56                  a = 0
57          elif y[k+1] <=  y_max - 21 and a == 0:
58              b += 1
59              if x[k+1] >= x_min + 21 or y[k+1] <=
                    y_max - 29.7:
60                  # M = min(sous_x[0], sous_x[-1])
61                  fig = plt.figure(i,figsize=(w,h))
62                  plt.plot([-y_max , -y_max +
                        29.7],[x_min, x_min + 21], 'o')
63                  plt.plot(np.abs(np.array(sous_y)),
                        sous_x, couleur[i%2])
```

```
                    plt.legend()
                    plt.axis("equal")
                    plt.grid()
                    plt.show()
                    i += 1
                    sous_x = [x[k+1]]
                    sous_y = [y[k+1]]
                    b = 0
            sous_x.append(x[k])
            sous_y.append(y[k])
        fig = plt.figure(i,figsize=(w,h))
        plt.plot([x_min, x_min + 29.7], [y_min, y_min +
            21], 'o')
        plt.plot(sous_x, sous_y, couleur[i%2])
        plt.legend()
        plt.axis("equal")
        plt.grid()
        plt.show()


## Programme principal
w = 200  #inch
h = 100
R = 50
B = 3*np.pi
#B = (19/8)*np.pi

```

```
90 # Recuperation des points
91 Ax,Ay,Bx,By,x,y = cyclo(R, B, n = 500)
92 d = np.linspace(Ax,Bx,200)
93
94 # Trace
95 largeur_feuille = 21
96 longueur_feuille = 29.7
97 plt.plot(x,y,'r', label = 'arc de cycloide')
98 # plt.plot(d,
99 #          ((By - Ay)/(Bx - Ax))*d,
100 #           label = "droite")
101
102 # x = [0,10,30,40]
103 # y = [1,2,3,4]
104
105 imprimercycloide(x,y)
106
107 # sauvegarde
108
109 chemin = 'E:\\cours\\MPSI 3\\Diaporama\\Images\
110
111 # fig.savefig(chemin+'figure.png', dpi=100)
112
113 plt.axis("equal")
114 plt.show()
```